Stanisław J. Piestrak

Design of self-testing checkers for unidirectional error detecting codes

Oficyna Wydawnicza
Politechniki Wrocławskiej

Scientific Papers of the Institute of Technical Cybernetics
No. 92 of the Technical University of Wrocław No. 92

Monographs No. 24 1995

Stanisław J. PIESTRAK

Design of self-testing checkers for unidirectional error detecting codes

Prace Naukowe Instytutu Cybernetyki Technicznej
Politechniki Wrocławskiej **92**

Seria:
Monografie **24**

50 LAT NAUKI • WROCŁAW • 1945 – 1995

SIG·UNIVERS·ET·POLYTECHN·WRATISL.

Stanisław J. Piestrak

Design of self-testing checkers for unidirectional error detecting codes

Oficyna Wydawnicza Politechniki Wrocławskiej · Wrocław 1995

Recenzenci
Andrzej KRAŚNIEWSKI
Henryk KRAWCZYK

OFICYNA WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

ISSN 0324-9786

Nakład 150+55 egz. Ark. wyd. 8. Ark. druk. 7. Papier offset. kl. III, 70 g, B1.
Oddano do druku w listopadzie 1995 r. Druk ukończono w listopadzie 1995 r.
Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Zam. nr 729/95.

Prace Naukowe Instytutu Cybernetyki Technicznej
Nr 92 Politechniki Wrocławskiej Nr 92
Monografie Nr 24 1995

*Concurrent error detection, error detecting codes,
fault-tolerant computing,
reliable digital circuit,
self-checking circuits, self-testing checker,
unidirectional error detecting codes*

Stanisław J. Piestrak*

DESIGN OF SELF-TESTING CHECKERS FOR UNIDIRECTIONAL ERROR DETECTING CODES

Digital self-checking circuits wherein inputs, outputs, and internal states are encoded using error detecting codes (EDCs) allow for concurrent error detection of both permanent and temporary internal faults. For many years, it has been observed that many faults occurring in VLSI circuits, semiconductor memory systems, and optical disks cause unidirectional errors. In this monograph, a general approach for designing highly efficient hardware supporting the use of unidirectional error detecting codes (UEDCs) in fault-tolerant digital systems is presented. Included are the design methods of self-testing checkers (STCs) for nonsystematic UEDCs codes and both encoders and STCs for systematic UEDCs. The nonsystematic codes include the all-UED m-out-of-n codes and t-UED Borden codes. The systematic codes include: all-UED Berger and equivalent codes, t-UED codes, and burst-UED codes. The basic building blocks of all encoders and STCs presented here are parallel counters — which are entirely built of full- and half-adders, and multi-output threshold circuits — whose the most efficient version can be built using an identical simple cell. It is shown that the gate-level STCs proposed for all UEDCs considered here can be designed in a uniform way and that they can be made not only less complex and/or faster than existing designs but easily-testable as well. They also enjoy a highly regular structure composed of gates with low fan-in and fan-out, which makes them attractive for VLSI implementation.

*Technical University of Wrocław, Institute of Engineering Cybernetics, ul. Wybrzeże Wyspiańskiego, 50–370 Wrocław

LIST OF ABBREVIATIONS

ALU	—	arithmetic-logic unit
AUEDC	—	all-unidirectional error detecting code
BUEDC	—	burst-unidirectional error detecting code
CD	—	code-disjoint
CED	—	concurrent error detection
CLA	—	carry lookahead adder
CPA	—	carry-propagate adder
CSA	—	carry-save adder
EAC	—	end-around-carry
ECC	—	error-correcting code
EDC	—	error-detecting code
FA	—	full-adder
FS	—	fault-secure
FT	—	fault-tolerant
FTC	—	fault-tolerant computing
HA	—	half-adder
IF	—	inverter-free
LSB	—	least significant bit
MLB code	—	maximal length Berger code
m/n code	—	m-out-of-n code (constant-weight code)
mod	—	modulo
MSB	—	most significant bit
OSUC	—	optimal systematic unordered code
PLA	—	programmable logic array
RAM	—	random access memory
ROM	—	read only memory
s/z	—	stuck-at-z
SC	—	self-checking
SEC	—	single-error correcting
SFS	—	strongly fault-secure
SN	—	sorting network
ST	—	self-testing
STC	—	self-testing checker
TSC	—	totally self-checking
t-UEDC	—	t-unidirectional error detecting code
UEDC	—	unidirectional error detecting code
u-error	—	unidirectional error
VLSI	—	very large scale integration

LIST OF SYMBOLS

b	—	length of burst error
$b(K)$	—	maximum length of burst error detected by a code with K check bits
(c_1c_0)	—	two outputs of an STC
C	—	set of codewords
C_{2r}	—	2-rail code
$C_{(I,K)}$	—	Berger code with I information bits and K check bits
C_{IN}	—	input code space of a circuit **H** ($C_{IN} \subset \mathbf{X}$)
$C(n,t)$	—	n-bit t-UED Borden code
C_{OUT}	—	output code space of a circuit **H** ($C_{OUT} \subset \mathbf{Z}$)
f	—	fault in a circuit **H**
F	—	set of likely faults in a circuit **H**
$Ga(\cdot)$	—	number of gates in a circuit $\cdot$
$\mathbf{H}$	—	combinational circuit with n inputs
I	—	number of information bits
$In(\cdot)$	—	total number of gate inputs in a circuit $\cdot$
$J_X = (x_{I-1} \ldots x_0)$	—	information part of a codeword X of a systematic code
K	—	number of check bits ($K = n - I$)
$L(\cdot)$	—	number of gate levels in a circuit $\cdot$
n	—	length of codeword
$(n;q)$	—	parallel counter with n inputs and q outputs
p_0 (p_1)	—	number of 0s (1s) in a binary vector
$P_X = (s_{K-1} \ldots s_0)$	—	check part of a codeword X of a systematic code
t	—	multiplicity of unidirectional errors detected by a code
$\mathbf{T}(\cdot)$	—	test for a circuit $\cdot$
T^n	—	n-input n-output threshold circuit
T_i^n	—	n-variable threshold function with threshold i
$X = (x_{n-1} \cdots x_0)$	—	input vector of a circuit **H**; x_{n-1} (x_0) is MSB (LSB)
$\mathbf{X}$	—	input space of a circuit **H** (set of all 2^n input binary n-tuples)
X^e	—	input non-codeword, i.e. $X^e \in \{\mathbf{X} \setminus C_{IN}\}$
$\mathbf{Z}$	—	output space of **H** (set of all 2^s output binary s-tuples)

1. INTRODUCTION

The steady progress in microelectronics technology has made available VLSI components whose complexity is already counted in millions of transistors. However, despite significant improvement of reliability of all elementary components, reflected e.g. by their much lower failure rates, their large number required to build faster and more complex computer systems causes that the problem of failures cannot be ignored. Three types of behavior of a computer system which experiences component malfunctions can be distinguished: (i) it produces wrong data, (ii) it ceases to produce the intended data, or (iii) it does both at the same time. A spectrum of computer applications demanding both enhanced reliability and predictable behavior in case of failures has been growing continuously. The most notable examples are: (1) airline flight control (fly-by-wire), space applications, and real-time control of complex technological processes, that require continuous availability and absolute data integrity; (2) telephone switching networks, for which high availability is crucial; and (3) on-line transaction processing, such as airline reservation systems, banking, credit card verification, wherein data integrity is the overriding concern. In these critical applications, a computer outage during normal functioning may result in financial losses, massive inconvenience or loss of life. It is therefore desirable that the behavior of the computer system experiencing failures of its internal elements should be thoroughly thought over and implemented; e.g. the system should be provided with a special means to detect a failure, so that it can shut down all or part of the system, provide warnings or alarms, and possibly switch in back-up systems. In other words, the general objectives of a dependable computer system are: to prevent damage, to restore operation as quickly as possible, and to enhance required availability. However, modern computers are too sophisticated and failure mechanisms are too complex, so that any ad hoc approaches to design and implementation of a dependable system are infeasible. The only way is systematic implementation of proper dependability techniques at every level of computer architecture: from the lowest circuit-level to the highest system-level. It is therefore not surprising that dependable computing has evolved into a broad discipline that is related to very diverse areas of computer science and engineering, including VLSI logic design, computer organization and architecture, software engineering, and system design, which are supported by analytical and simulation techniques needed to perform validation, verification, and quantitative evaluation of dependability of computer systems.

1.1. Background and Motivation

Dependability is a global concept which subsumes the usual attributes of reliability (continuity of service), availability (readiness of usage), safety (avoidance of catastrophic consequences on the environment), and security (preservation of integrity and confidentiality) [85]. Fault avoidance and fault tolerance are two basic approaches to design of a dependable system. *Fault avoidance* is aimed at constructing a fault-free system, which can be achieved e.g. by using higher quality, thoroughly tested components. *Fault tolerance* is aimed at providing correct functional operation of the system even in the presence of faults whose occurrence is assumed inevitable. (Here we consider physical faults in hardware which accidentally occur within the system during its operation.) The major advantages of fault-tolerant (FT) techniques are a significant improvement in system availability and the feasibility of error-free computation, hence its potentially safer operation.

Dependability has been an important attribute since the early highly unreliable computers were constructed in the fifties [108], [109], [114], [7]. Despite continuous progress in manufacturing of digital components and assembling processes, the need for more and more dependable computation have continued until now, although the requirements towards dependability of digital systems changed in time, as the digital systems are expanding both in size and the number and variety of applications. This can be seen from a continuous flow of monographs on designing FT computers (given here in a chronological order of appearance): [210], [135], [181], [164], [171], [207], [78], [177], [159], [73], [165], [170], [189], [182].

The expectations towards dependability of many modern digital systems can be extremely high — the systems with availability even as high as 0.9999999 [182] and reliability of 10^{-9} failures per hour over the 10 hour mission time [159] are considered. That even so high expectations towards dependability of digital systems are not exaggerated, the following crashes of large scale systems caused by hardware faults of control computers which have been reported in the nineties, are bitter reminders.

1. Jan. 15, 1990 — the failure of the AT&T long-distance telephone network in the USA which lasted for nine hours [106], [183].

2. Feb. 1990 — the crash of a "fly by wire" Airbus A320 in Bangalore, India [183].

3. Sept. 17, 1991 — the seven and a half hour outage of the telecommunication network in the New York City [31].

4. June 26, 1993 — the thirty hours long failure of the banking card network in France affecting 40% of 21 mln owners of banking cards in France during the whole weekend [88].

Many other incidents related to computer system failures were extensively reported in [105].

Fault tolerance of a digital system always requires introducing some redundancy into a system. Generally, the types of redundancy are classified as:

1. Hardware redundancy;

2. Software redundancy;
3. Information redundancy; and
4. Time redundancy.

Although this classification is rather conventional as one type of redundancy (which is predominant) usually involves introducing some other types as well, it is still valuable as it indicates the type of redundancy which is prevalent. As an example, consider the information redundancy such as encoding of data with an error-detecting code (EDC) or an error-correcting code (ECC). Basically, it relies on introducing extra bits to any data unit, e.g. by encoding inputs and outputs of all circuits (and internal states—if a circuit is sequential) with EDCs or ECCs. However, it also involves hardware redundancy — in the form of encoders, decoders, extra datapath lines (to transmit) or memory bits (to store) these extra bits, as well as time redundancy — as the encoding and decoding of data may introduce some delay into a system.

Fault tolerance may be achieved by fault masking or fault detection followed by recovery. The former—more expensive—provides faster error correction while the latter requires less redundancy. The latter approach, which is of our concern, may involve duplication of a system (or its modules) with comparison and/or using EDCs, providing the circuitry with the possibility of on-line detection of internal faults, and appropriate design of recovery procedures. Since various classes of such circuits can be distinguished, we will use a generic term *self-checking (SC) circuits* to denote all of them. Self-checking design offers concurrent error detection (CED), i.e. error detection performed in real-time continuously with the system functioning, of both permanent and temporary (both transient and intermittent) faults. This technique supports achieving fault tolerance of a digital system for temporary faults, provided that the sequence of events: error detection, rollback recovery, and operation retry is sucessfully performed. This approach is particularly welcome in many digital systems, when we recall that: (1) nowadays more than 90% of faults which occur during system operation are temporary [182] and they are difficult to detect (if at all) by applying periodic software diagnostic test procedures; and (2) the use of EDCs and implementing circuits as SC is relatively inexpensive. However, it must be pointed out that using EDCs can be economically justified if and only if it results in significantly less overhead than duplication with comparison, which always introduces more than 100% redundancy: one extra module and a comparator. Also, one should not overlook that duplication usually involves significantly less design effort to be implemented than other SC techniques. Another important problem with designing and implementing any FT digital system is the presence of a *hardcore*, which can be loosely defined as that part of the hardware that must be functioning correctly in order to initiate any diagnostic functions. The probability of hardcore failure can be reduced by using the following techniques: (i) minimizing the amount of hardware in the hardcore, e.g. by implementing circuits as SC or FT; (ii) implementing the hardcore using highly reliable components; and (iii) providing mechanism to facilitate periodic off-line testing the hardcore. A more extensive discussion of the hardcore problem can be found in [78].

The following recovery functions are performed within a FT system that uses fault detection and recovery, after a fault occurs. The first step in tolerating a fault is to detect an error produced as a result of it as soon as it is produced, before it propagates through the system. Once the fault is detected, the diagnosis whether the fault is permanent or temporary can be performed, e.g. through rollback and retry. In case of temporary fault, it is likely that the fault will disappear and the system may resume its operation without repair. Otherwise, it is assumed that a fault is permanent and its location up to a replaceable component is necessary for a possible reconfiguration of the system, which then may resume its operation, possibly with degraded performance.

The need for the use of coding techniques for providing digital systems with hardware means for on-line verification that the system operates correctly is very apparent from the wealth of applications of digital systems in which error-free computation is one of the key requirements. Once an internal fault occurs in the system, then it is desirable to detect it as quickly as possible. On one hand, the ever growing complexity of VLSI chips with decreasing size of internal devices makes modern digital systems more and more vulnerable to temporary faults. On the other hand, the decreasing price of hardware (relative to the cost of software development) justifies the use of hardware error handling techniques. The industrial experience of respected computer manufacturers such as Tandem Computers and Sun Microsystems proves that relying solely on the high reliability of VLSI integrated circuits and run-time checks to ensure data integrity of general-purpose microprocessor-based systems has become insufficient [61]. It is estimated that on average one out of 10000 personal computers per month experiences data corruption due to internal failures, which go undetected by commonly used traditional techniques of on-line error detection which rely on validity checks, parity control of the datapaths only, and ECCs of memory. It was explicitely pointed out that the densly packed VLSI chips are already so susceptible to intermittent and transient faults, that the minimum of error protection, such as provided by an EDC, of most modules of a digital system becomes mandatory. Therefore, there is a strong motivation to support less expensive techniques of tolerating temporary faults that rely on control flow checking (for an excellent survey, see [190]) by significantly stronger methods such as SC design.

A number of different classes of error control codes have been constructed for

2. Software redundancy;
3. Information redundancy; and
4. Time redundancy.

Although this classification is rather conventional as one type of redundancy (which is predominant) usually involves introducing some other types as well, it is still valuable as it indicates the type of redundancy which is prevalent. As an example, consider the information redundancy such as encoding of data with an error-detecting code (EDC) or an error-correcting code (ECC). Basically, it relies on introducing extra bits to any data unit, e.g. by encoding inputs and outputs of all circuits (and internal states—if a circuit is sequential) with EDCs or ECCs. However, it also involves hardware redundancy — in the form of encoders, decoders, extra datapath lines (to transmit) or memory bits (to store) these extra bits, as well as time redundancy — as the encoding and decoding of data may introduce some delay into a system.

Fault tolerance may be achieved by fault masking or fault detection followed by recovery. The former—more expensive—provides faster error correction while the latter requires less redundancy. The latter approach, which is of our concern, may involve duplication of a system (or its modules) with comparison and/or using EDCs, providing the circuitry with the possibility of on-line detection of internal faults, and appropriate design of recovery procedures. Since various classes of such circuits can be distinguished, we will use a generic term *self-checking (SC) circuits* to denote all of them. Self-checking design offers concurrent error detection (CED), i.e. error detection performed in real-time continuously with the system functioning, of both permanent and temporary (both transient and intermittent) faults. This technique supports achieving fault tolerance of a digital system for temporary faults, provided that the sequence of events: error detection, rollback recovery, and operation retry is sucessfully performed. This approach is particularly welcome in many digital systems, when we recall that: (1) nowadays more than 90% of faults which occur during system operation are temporary [182] and they are difficult to detect (if at all) by applying periodic software diagnostic test procedures; and (2) the use of EDCs and implementing circuits as SC is relatively inexpensive. However, it must be pointed out that using EDCs can be economically justified if and only if it results in significantly less overhead than duplication with comparison, which always introduces more than 100% redundancy: one extra module and a comparator. Also, one should not overlook that duplication usually involves significantly less design effort to be implemented than other SC techniques. Another important problem with designing and implementing any FT digital system is the presence of a *hardcore*, which can be loosely defined as that part of the hardware that must be functioning correctly in order to initiate any diagnostic functions. The probability of hardcore failure can be reduced by using the following techniques: (i) minimizing the amount of hardware in the hardcore, e.g. by implementing circuits as SC or FT; (ii) implementing the hardcore using highly reliable components; and (iii) providing mechanism to facilitate periodic off-line testing the hardcore. A more extensive discussion of the hardcore problem can be found in [78].

The following recovery functions are performed within a FT system that uses fault detection and recovery, after a fault occurs. The first step in tolerating a fault is to detect an error produced as a result of it as soon as it is produced, before it propagates through the system. Once the fault is detected, the diagnosis whether the fault is permanent or temporary can be performed, e.g. through rollback and retry. In case of temporary fault, it is likely that the fault will disappear and the system may resume its operation without repair. Otherwise, it is assumed that a fault is permanent and its location up to a replaceable component is necessary for a possible reconfiguration of the system, which then may resume its operation, possibly with degraded performance.

The need for the use of coding techniques for providing digital systems with hardware means for on-line verification that the system operates correctly is very apparent from the wealth of applications of digital systems in which error-free computation is one of the key requirements. Once an internal fault occurs in the system, then it is desirable to detect it as quickly as possible. On one hand, the ever growing complexity of VLSI chips with decreasing size of internal devices makes modern digital systems more and more vulnerable to temporary faults. On the other hand, the decreasing price of hardware (relative to the cost of software development) justifies the use of hardware error handling techniques. The industrial experience of respected computer manufacturers such as Tandem Computers and Sun Microsystems proves that relying solely on the high reliability of VLSI integrated circuits and run-time checks to ensure data integrity of general-purpose microprocessor-based systems has become insufficient [61]. It is estimated that on average one out of 10000 personal computers per month experiences data corruption due to internal failures, which go undetected by commonly used traditional techniques of on-line error detection which rely on validity checks, parity control of the datapaths only, and ECCs of memory. It was explicitely pointed out that the densly packed VLSI chips are already so susceptible to intermittent and transient faults, that the minimum of error protection, such as provided by an EDC, of most modules of a digital system becomes mandatory. Therefore, there is a strong motivation to support less expensive techniques of tolerating temporary faults that rely on control flow checking (for an excellent survey, see [190]) by significantly stronger methods such as SC design.

A number of different classes of error control codes have been constructed for error protection of digital systems [11], [46], [64], [65], [134], [160], [165], [207]. For many years the research of error codes has been motivated primarily by the needs of error-free data transmission [11], [64], [134]. However, since the requirements imposed on error codes used to implement FT computer systems are essentially different, their principal attributes are highlighted now.

1. Information is handled in parallel in a computer. This calls for efficient and fast encoding and decoding algorithms.
2. Encoding and decoding circuitry introduces extra delay and it should not slow down the system operation significantly.
3. Hardware needed for encoding and decoding should be relatively simple compared to hardware of functional circuitry (which performs 'useful' computation), since it not only increases the overall cost of a system but also the likelihood of a hardware failure (which will however be detected before undetected errors will be spread through a system).
4. Anticipated errors vary from one module of a digital system to another. This will require a clever selection of encoding which is most efficient to handle a particular class of errors.
5. Application of error codes to protect data which is only transferred from one place to another or is stored is relatively easy. The task becomes generally difficult when information is modified in some way (e.g. addition of two operands with an ALU generates new information—a sum and an overflow signal), since the error code used must be preserved under such operations.
6. Encoders and decoders themselves also cannot be assumed to be fault-free, and therefore they also should be implemented as FT circuits.

As for the EDCs (which are of our interest), four general classes of EDCs can be distinguished: parity codes, 2-rail and duplication codes, arithmetic codes, and unidirectional EDCs (UEDCs). Basic theory of EDCs and their applications for designing SC circuits can be found in any textbook on FT hardware [73], [78], [159], [164], [165], [182], and [207], whereas the state-of-the-art surveys were given in [96] and [145]. Unfortunately, selection of only one EDC which would be the best suited for a particular digital system, aimed at reducing the overall number of encoders and checkers throughout the system, is infeasible. It is well known that the error codes that are suitable e.g. to protect the bus or the RAM system are not preserved by the arithmetic circuitry, and conversely, error codes that are suitable to protect arithmetic circuitry may be too expensive to be used for the bus and not powerful enough to be used in memory (which generally requires using an ECC). Many other examples of such conflicting requirements can also be given. Overall, error coding used to protect hardware of digital systems does not need to be sophisticated, and yet it should be capable of sufficiently protecting every module of a system at the minimum cost of extra hardware and/or delay.

Encoding data using EDCs and realization of circuits as SC may provide varying degree of protection against undetected faults and errors. The most stringent requirement formulated for SC design has been referred to as *the totally self-checking (TSC)* goal [188]:

the first erroneous output resulting from an internal fault of the digital circuit is detectable, i.e. it is a non-codeword.

This property is desirable in any digital system wherein increased confidence that a system is working correctly is of primary importance, i.e. it is preferable to stop the operation of a system rather than to allow it to produce an incorrect output.

The following advantages of TSC circuits are well known:

1. Any error caused by both permanent and temporary fault from a well defined class of likely faults is detected.
2. Errors are detected immediately after their first occurrence, which precludes the corruption of data in the system.
3. The amount of hardware in the hardcore of the computer system can be significantly reduced.
4. Recovery time of a system after error detection is significantly shortened, due to automatic fault detection and isolation. In particular, automatic replacement of a faulty module is feasible in a redundant and/or multiprocessor system.
5. Repair of a faulty system is facilitated and expedited.
6. Diagnostic software of a system can be significantly simplified, since SC design can be seen as a supplementary technique to off-line diagnosis.

The Electronic Switching System (ESS) by Bell Labs., built in the sixties, is the first example of a complex digital system for non-military and non-space applications wherein most modules were built as SC for high availability [10], [24], [78], [191], [199]. Many experimental SC processors and complex SC modules, most of which were partially protected by various UEDCs, have been proposed since then: [25], [37], [45], [56], [57], [95], [111], [121], [132], [169], [170], [180], [207], [211], [212], and [213]. Renewed growing interest in practical applications of SC circuits is exemplified by two most recent commercial products: the large mainframe IBM Enterprise System/9000 [184] and the on-board processor by Harris Corp. [63]. The modules of either system are protected against undetected errors by a combination of parity, duplication, residue mod 15, and m/n encodings, depending on the type of a module. Also, there are already known multiprocessor systems composed of up to 100,000 processors which use SC modules to provide for fault-tolerance and reconfiguration, see e.g. [29]. Clearly, the design of SC digital circuits is an important area of fault-tolerant computing (FTC), attractive not only from a theoretical but also from a practical point of view, which has experienced significant growth of interest in the computer industry in recent years.

Unfortunately, despite declining cost of hardware and many unquestionable advantages, SC circuits have been rather sparingly used in commercial computers, due to increase of chip area and extra delay, which are regarded too high by most computer manufacturers. Hence, the most important factors which could make SC circuits an economically feasible alternative for implementation of future FT systems are: (1) availability of EDCs with low redundancy but capable of detecting the most likely errors; (2) readily-available and efficient implementation methods of EDCs in digital hardware of any type; and (3) availability of fast and low-cost encoders and checkers.

1.2. The Scope of this Work

A general scheme of the TSC system (given in Fig. 1.1) consists of two types of blocks: a *functional circuit* that executes some 'useful' function, and two *checkers* whose only function in a digital system is to monitor whether the *inputs* entering and the *outputs* generated by a functional circuit are correct or not. The checkers are a pure overhead in the system: should there be no faults in the system, no checker would be needed. The first step in implementing a TSC system is to encode input and output data using some EDCs C_{IN} and C_{OUT}, respectively. C_{IN} (C_{OUT}) can be seen as a proper subset of all 2^n (2^s) binary input n-tuples (output s-tuples) selected in such a way that they have some common attribute which is easy to verify, so that every occurence of an input codeword $X \in C_{IN}$ and an output codeword $Z \in C_{OUT}$ is interpreted as correct operation of the system. Every occurence of a non-codeword $X^e \notin C_{IN}$ or $Z^e \notin C_{OUT}$ is a symptom of incorrect operation of the system, which should be promptly signaled by the checkers to some external control unit of the system. A special circuit called a *checker for a code* C_{IN} (and C_{OUT}) must be built to determine if the output of the circuit it checks is a codeword or not, by proper setting of its error indication signal. However, the simplest 1-output checker whose output is z whenever a

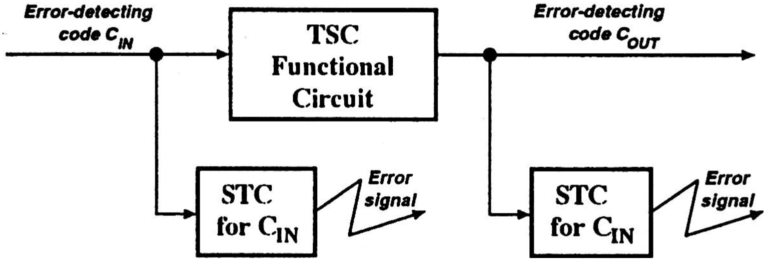


Figure 1.1. General scheme of the self-checking system

codeword is present, and $\overline{z}$ when a non-codeword is present, $z \in \{0,1\}$, contains a hardcore: any internal fault sticking its output at z would never be detected and the checker would loose its checking capabilities. (Note that any internal fault sticking its output at $\overline{z}$ is always detected.) This is one obvious reason that to upgrade the reliability of a checker, it should have at least two outputs encoded in an EDC. Usually, a checker output of (01) or (10) indicates correct operation, whereas (00) or (11) indicates an input error or an internal fault. Additionally, the 2-output checker should be designed and implemented in such a way that no realistic single fault of some set of faults F sticks its output at $(z\overline{z})$. However, even then the same undesirable effect of checker output stuck at $(z\overline{z})$ can be caused by the occurrence of the sequence of one latent fault followed by another fault, both from F. A checker implemented as self-testing (ST) for F is a circuit wherein the latter problem virtually does not occur, provided that all codewords from C_{IN} (C_{OUT}) occur frequently enough, so that every fault in F is detected before another fault from F occurs. To achieve the TSC goal, the functional circuit must be implemented as TSC for a set F of the most likely internal faults. In a TSC circuit, no fault in F can cause an undetectable error in normal operation and all faults in F are tested by incoming codewords from C_{IN}. In the latter case, for some codeword $X \in C_{IN}$ a fault causes a non-codeword output $Z^e \notin C_{OUT}$, which is detected by the output checker. Therefore, the system from Fig. 1.1 is free of hardcore, provided that there occur no faults other than those for which the functional circuit is TSC and the checkers ST. More formal and detailed presentation of these topics will be given in Section 3. This monograph deals with the design methods of combinational self-testing checkers (STCs) for several classes of UEDCs.

A *unidirectional error* (u-error) is a multiple error such that all erroneous bits are of either 0→1 or 1→0 type, but not both at the same time. Unidirectional errors have been observed as the result of power failures [160], the failures in byte-serial memory systems [52], [128], [206], [207], in regular VLSI circuitry such as ROM memory systems [45], [46], [101], [122], [160] and Programmable Logic Arrays (PLAs) [94], [123], [116], [213], in laser compact disks [46], [87], and many other [22], [24], [121], [122], [158], [160], [172]. Unidirectional errors may also occur due to single and multiple unidirectional faults occurring in *inverter-free (IF)* (i.e. using AND and OR gates only) combinational circuits using shared logic [35], [119], [185].

The u-errors of any multiplicity are detected by so called *unordered codes* such as, for example, m/n codes and Berger codes which were proved to be the optimal nonsystematic and systematic unordered codes in [44] and [9], respectively. Many other optimal systematic unordered codes equivalent to Berger codes were defined by Ashjaee and Reddy [5], [6], and Piestrak [151]. However, the data used in a digital system are in many cases organized in bytes and stored, transmitted or transformed by separate units operating on bytes. Therefore, provided that only single hardware failures can occur and are confined to independent units, instead of u-errors of any multiplicity only up to t u-errors are the most likely to occur (t is the byte length). Also, the multiplicity t of the output u-error in certain circuits using shared logic (e.g. a PLA [119]) can be inherently limited by their internal structure. As a result, instead of an unordered code a less redundant (and hence cheaper) t-UEDC can be used. The optimal (nonsystematic) t-UEDCs were proposed by Borden [14]. Systematic t-UEDCs were first proposed by Dong [38], and then improved by Bose and Lin [17] and Jha and Vora [71]. Further savings in the codeword length are possible when a burst error model is assumed. (A *burst error* of length b means that the erroneous bits are confined to a cluster of b adjacent bits.) Burst unidirectional error detecting codes (BUEDCs) were proposed by Bose [15] and Blaum [12]. Byte unidirectional error detecting codes were proposed by Bose and Lin [15], [18], and by Dunning *et al.* [40], [41]. Finally, let us mention for completeness about the abundant number of systematic codes capable of correcting d random (symmetric) errors and detecting all or t u-errors (d-EC/AUED and d-EC/t-UED codes) which have also been studied extensively in recent years, see e.g. [13] and [124], but are beyond the scope of this work. Since all the above mentioned codes have in common that they are *capable of detecting u-errors of a given multiplicity t*, henceforth we shall call any of them with a single generic term *unidirectional error detecting code (UEDC).*

Numerous applications of UEDCs to implement SC and FT digital modules, which have been reported continuously throughout years, include:

- Microprogram control units: [24], [34], [56], [78], [132], [191], [199], [207], and [211].
- Synchronous sequential circuits: [22], [32], [33], [35], [49], [55], [72], [82], [97], [102], [110], [127], [158], [175], [177], [195], [196].

- Asynchronous sequential circuits: [27], [50], [93], [100], [112], [126], [176], [177].
- PLAs: [19], [45], [94], [120], [123], [163], [209], [212] and semiconductor ROMs [46], [101], [160], [213]; and
- Arithmetic circuitry: arithmetic-logic units (ALUs) [91], multipliers and dividers [92], and floating-point arithmetic units [89].

The growing interest in u-error detecting and correcting codes is confirmed by their first commercial applications recently reported in [46]: in FT 4-Megabit VLSI ROMs for yield improvement and in large area laser compact disks for defect tolerance, both by NTT.

Among UEDCs, unordered codes deserve special attention. First, the circuits both combinational [111] and sequential [110] can be easily implemented to achieve the TSC goal, provided that unordered codes are used in all interfaces and all combinational circuitry is realized IF (which is quite natural when unordered codes are used) [93]. Secondly, such classes of self-timed (asynchronous) circuits as speed-independent and delay-insensitive circuits must use unordered codes for data encoding to enforce their correct operation independently of delays (in gates and wires, respectively) and the by-product is that they are also SC for some faults: [4], [201], [202], [203]. The most recent study of these topics we have presented in [156]. Finally, an extremely important practical application of certain m/n codes (called *balanced codes*) and circuitry supporting their use was revealed recently in [129] and [193], where they were used for noise reduction across VLSI input/output pins.

Nonsystematic UEDCs such as m/n codes and Borden codes have in common that checking whether a binary n-tuple X is a codeword can be done by determining the *weight of* X (i.e. the number of 1s in X). Similarly, all systematic UEDCs mentioned above have in common that the check bits which are responsible for providing the code with u-error detecting capabilities are generated on the basis of the weight of the remaining part of a codeword which is: (i) an information part — for a code which is UED only, and (ii) a codeword of some d-EC code — for a code which is d-EC/AUED or d-EC/t-UED. Thus, the availability of the efficient *counter of 1s* (i.e. a circuit that generates the weight of a binary vector) as well as any other combinational circuit that efficiently realizes an arbitrary logic function that depends on the weight of an input vector (e.g. a threshold circuit), is of crucial importance while designing circuitry supporting the use of all UEDCs.

To date, there have been several design methods for the STCs for almost all classes of UEDCs which, however, have not taken explicitely into account the common attribute such as the dependence of the checker functions on the weight

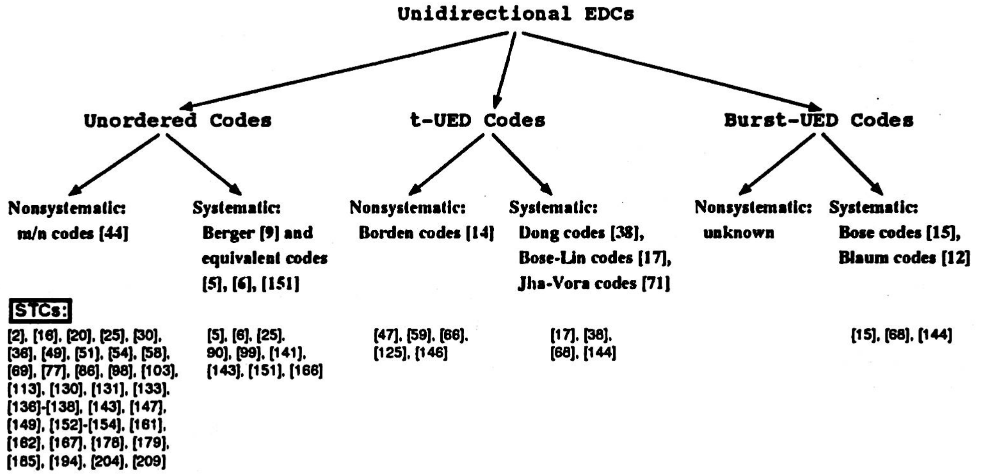


Figure 1.2. Survey of the most important UEDCs and design methods of STCs for these codes

of an input vector. This observation can be derived from the following general survey of the existing design methods for the STCs for UEDCs. Firstly, note that many STCs for m/n codes ($m > 1$) built using threshold circuits [2], [49], [54], [98], [167], [177], [178], [179], and [204] have used only two realizations of the multioutput threshold circuits: a prohibitively complex (for larger n) two-level version or less complex multilevel cellular version from [168]. Later the so called *completely bifurcated threshold circuits* were considered in [42], [136], and [137]. All these threshold circuits have the number of gates and gate inputs which is at least $O(n^2)$. This happened because the superb performance and testability of the multioutput threshold circuits implemented as sorting networks (SNs) was discovered only recently by the author [143], [148]. Secondly, there have been STCs for various classes of UEDCs using the counters of 1s which were built using the so called *parallel counters* (entirely composed of full-adders (FAs) and half-adders (HAs)) [6], [15], [17], [36], [68], [71], [99], [131], [133]. Only two realizations of parallel counters have been suggested to build encoders and decoders in all the literature on designing FT hardware and on systematic UEDCs with error-correcting properties (including the most recent works [68] and [124]): the basic realization given in [3] and its modification for easy testability suggested in [99]. Both employ a number of carry-propagate adders (CPAs) which introduce unnecessarily large delay. To date, significantly faster and in some cases less complex parallel counters that have been suggested in the voluminous literature on multipliers [26], [43], [76], [104], [192], [208] have clearly been overlooked. Finally, there have been some systematic t-UEDCs for which the STCs have not been constructed yet — see [17] and [71]. For readers' convenience, the classification of the most important UEDCs that will be considered here and the survey of design methods of STCs for these codes are shown in Fig. 1.2.

In this monograph, we aim to present systematic design for combinational STCs for UEDCs. It summarizes and extends the results on related topics that we have presented in several earlier works [136]–[156]. We shall show that the STCs for the most important known UEDCs can be designed by using two essentially different but complementary basic building blocks: a multi-output threshold circuit and a parallel counter. The design procedures will be presented independently of any particular technology of VLSI integrated circuits, since their fast development would easily make any technology-dependent considerations obsolete. Nevertheless, we shall concentrate on those realizations which take into account some attributes which are common for most modern VLSI technologies such as low fan-in and fan-out, repeatability of basic modules, regular structure, and easy testability. In this context, the emphasis will be given on the STCs which offer hardware efficiency and/or small delay.

This monograph is organized as follows. Section 2 introduces the fault and error models applicable to digital VLSI circuits. Then the basic characteristics of the EDCs are presented. A particular emphasis is on the thorough analysis of error detecting properties of various UEDCs and their definitions.

Section 3 contains a systematic presentation of the basic theory of SC combinational circuits. Special attention is given to the properties of STCs and the basic assumptions regarding the fault/error behavior of a SC digital system. It is concluded with the presentation of the so called 'normal checker' — a universal general structure of an STC suitable for any systematic EDC. Its structure justifies why both the encoders and STCs for systematic EDCs are considered here together: an encoder can be used as an integral part of an STC for the same code.

Section 4 presents the detailed survey of various counters of 1s which will be used as the basic building blocks in almost all circuits presented here. Two major classes of the counters of 1s are presented separately: parallel counters and the multi-output threshold circuits.

The next three sections present the design methods of the STCs for nonsystematic UEDCs and both encoders and STCs for systematic UEDCs. We have concentrated on the methods which provide the most hardware-efficient and fast circuits. The testability of these circuits is also analyzed in details.

Section 5 covers the design of the STCs for selected m/n codes — those which are the most important from a practical point of view. Included are the $m/2m$ and $m/(2m+1)$ codes (which are the optimal unordered codes as they offer the maximum capacity for a given codeword length n and are capable of detecting u-errors of any multiplicity), the $2/n$ codes, and the $1/n$ codes.

Section 6 presents the design of STCs for nonsystematic Borden codes, which are the optimal t-UEDCs. It is shown that the least complex albeit the fastest STCs can be built on the basis of two multioutput threshold circuits. We have derived the logic functions of an STC for any Borden code expressed in terms of the threshold functions.

Section 7 presents the design of both the encoders and the STCs for various classes of UEDCs. For all codes considered in this section, the two versions of both the encoders and the STCs are presented: one version is built on the basis of a parallel counter, whereas the other employs a multioutput threshold circuit. Subsequently are presented the encoders and STCs for: the optimal systematic unordered EDCs (i.e. the Berger codes and some codes equivalent to them), three classes of t-UEDCs, and the optimal BUEDCs. The STCs for two classes of systematic t-UEDCs are the first ever proposed.

The monograph is concluded with Section 8 containing summary and applications of the results given here. Also, suggestions for further advancement of the concepts presented here and some related open research problems are discussed.

2. ERROR DETECTING CODES (EDCs)

In this section we shall present the basic fault and error models, the main characteristics of EDCs, and the definitions of the most important classes of UEDCs in the order from the most to the least redundant.

2.1. Faults and Errors in Digital Circuits

Here we are concerned exclusively with digital circuits whose incorrect operation can be analyzed on the logic level (i.e. parametric faults such as the change of current drawn by the circuit or a delay fault are excluded). The terminology presented here is based on [85], [165], and [182].

Definition 2.1. *A* failure *occurs in a digital circuit when its behavior deviates from the specified behavior, i.e. when it is unable to perform its designed logic function.*

The failures are considered on a *physical level* of a circuit: e.g. a line shorted to the ground, two lines shorted together, a broken line, a shorted diode or transistor.

Definition 2.2. *A* fault *is an incorrect logic state of a line of a digital circuit resulting from failures of its components.*

The faults are considered at the *logical level* of abstraction. A given fault may result from various failures. The most commonly used is the stuck-at-z (s/z), $z \in \{0,1\}$, fault model which assumes that the logic value on an input or output line of a circuit is set to z, independently of the binary vector applied to the input of the circuit. There are also several faults which are typical either for a particular technology (e.g. stuck-open and stuck-on faults in CMOS) or for a particular class of circuits (e.g. adjacent line faults and crosspoint faults in PLAs and pattern-sensitive faults in RAMs), but none of them will be considered here.

Further classification of faults may include the multiplicity of faults (single and multiple faults) and the time of duration (temporary and permanent faults). Usually, and in this monograph too, it is preliminarily assumed that only single s/z faults occur. This assumption is justified because of two reasons: 1) only single faults are algebraically tractable without excessive computations; and 2) the test sets capable of detecting single s/z faults have been proved to have very high fault coverage of multiple faults as well [1], [80]. However, it will be shown later that most parts of many checkers considered here are tested exhaustively by applying codewords only, and therefore the checkers are actually ST for many multiple combinational faults as well. Temporary faults, which have been predominant in modern digital systems [182], can originate from the external physical environment — *transient faults*, or can result from internal (with respect to a circuit) causes — *intermittent faults*. Transient faults are those caused e.g. by power supply fluctuation, electromagnetic perturbations, temperature and humidity variations, air pollution, vibrations, and radiation. Intermittent faults result from the presence of rarely occurring combinations of conditions such as: changes in the parameters of a hardware component (e.g. effect of temperature variation, delay in timing due to parasitic capacitance), loose connections, pattern-sensitive faults in semiconductor RAM, or when a system load goes beyond a certain level. Although temporary faults do not damage the circuits involved, they cause errors which are extremely difficult to detect (due to their temporary nature) unless the CED techniques based on using EDCs and implementing the circuits as SC are used.

Definition 2.3. *An* error *is the occurrence of an incorrect logic state caused by a fault within a circuit, which may occur in some other place than a fault site.*

The errors are considered at the *informational level* of abstraction and are characterized by the register values. The types of errors which occur in modern memory, logic, and arithmetic VLSI circuits as well as in non-semiconductor memory systems are many and varied. They can be broadly classified as *symmetric* (single or multiple), multiple *unidirectional*, and multiple *asymmetric* errors. In the sequel we refer to the transition 0→1 as 0-error and 1→0 as 1-error.

Definition 2.4. *If both 0-errors and 1-errors occur in a received word with equal probability then the errors are called* symmetric.

Definition 2.5. *If both 1-errors and 0-errors can occur in received words, but in any particular word all errors are of one type, then they are called* unidirectional.

Definition 2.6. *If the probability of unidirectional z-errors is extremely small compared to unidirectional $\overline{z}$-errors, $z \in \{0, 1\}$, the errors are called* asymmetric.

Symmetric errors are quite common in many digital systems and hence their sources need not be specified here. u-errors have been observed as a result of permanent failures such as: power supply failure, stuck-at faults in shift register memories, faults typical for PLA and ROM, large-area digital compact disks [24], [191], [206], [128], [160], [94], [122], [45], and [46]. They also occur as a result of single (or multiple unidirectional) s/z faults in IF combinational circuits using shared logic. Finally, asymmetric errors occur in optical communication (spurious photons cannot be generated) and as a result of some failures in a class of ROMs [46], [65], [101]. Obviously, the analysis shows that unidirectional or asymmetric errors may occur: in a single line, in a single byte, in a part of the storage area, or as a burst of a particular length. In each case an appropriate code, capable of detecting a particular class of the most likely errors, should be used for protection.

Here we are concerened with u-errors only. The u-errors can be classified from the most to the least difficult to detect as follows: a) any multiplicity; b) up to t errors; c) burst errors of length b; and d) byte errors.

Definition 2.7. *A* burst error *of length b is any pattern of errors which is confined to a b-bit portion of the word.*

Let $X = (x_{31} \ldots x_1 x_0)$ be composed of four bytes, i.e. $X = (B_3 B_2 B_1 B_0)$, where $B_3 = (x_{31} \ldots x_{24})$, $B_2 = (x_{23} \ldots x_{16})$, . $B_1 = (x_{15} \ldots x_8)$, and $B_0 = (x_7 \ldots x_0)$. The examples of three types of u-errors are given in Fig. 2.1

2.2. Basic Properties of EDCs

Let C denote the set of codewords, I — the number of information bits, n—the length of the codeword, and K—the number of check bits ($K = n - I$).

The following characteristics of a code are taken into account while selecting an EDC for a particular application:

1. Class of errors detected.

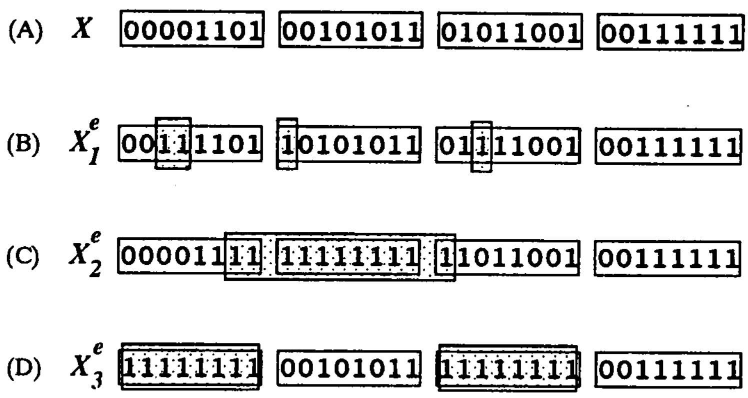

(A) X 00001101 00101011 01011001 00111111
(B) X_1^e 00111101 10101011 01111001 00111111
(C) X_2^e 00001111 11111111 11011001 00111111
(D) X_3^e 11111111 00101011 11111111 00111111

Figure 2.1. Three types of u-errors: Sample codeword (A); 4-unidirectional 0⟶1 error (i.e. t=4) (B); Burst unidirectional 0⟶1 error (i.e. b=11) (C); 2-byte unidirectional 0⟶1 error (in bytes B_1 and B_3) (D)

2. Redundancy of a code, characterized either by the number of redundant bits K, or the *rate* (or *efficiency*) of the code — expressed by the ratio $I/(I+K)$.

3. The *capacity* of the code denoted with $|C|$, i.e. the number of codewords, given a codeword length n.

4. Systematicity and separability of a code: in both a *systematic* and a *separable* code the information bits can be distinguished from the check bits. In a separable code, additionally, the information parts and the check parts of the operands can be processed in parallel by a SC functional circuit. Generally, for a given n and a given class of errors, the capacity of a nonsystematic code is larger than that of a similar systematic code. However, despite this advantage, nonsystematic codes have found limited applications due to significantly more complex encoding and decoding hardware than their systematic counterparts. In particular, decoding of a systematic codeword relies on neglecting its check part, which is not the case in any nonsystematic code. Nevertheless, there are some notable example applications wherein nonsystematic codes can be used efficiently: the state assignments and input/output encodings of sequential machines and many PLA-based control circuits.

5. Speed and complexity of the encoding and checking circuitry.

6. Availability of efficient design methods of SC hardware using an EDC. This

is not a problem as long as non-transforming modules are concerned, such as memory, a bus, or a multiplexer. However, it can be a serious problem when transforming modules are concerned, e.g. an ALU or a control unit.

7. The overall cost of implementing a code in a system which can be measured by:
    (a) The number of extra inputs and outputs of an integrated circuit or a printed circuit board.
    (b) The amount of extra hardware required to implement an encoder, a checker, and a functional circuit modification, which can be measured by: the number of extra gates and/or gate inputs (or transistors), or by the amount of extra chip area.
    (c) The extra delay caused by necessity of using an encoder and a checker, as well as due to the functional circuit modifications.

2.3. Unordered Codes

The partial ordering on the binary n-tuples is defined as

$$X \leq Y \quad iff \quad x_i \leq y_i \text{ for all } i.$$

For example, $(1,0,0,1) \leq (1,1,0,1)$ and $(1,0,0,1) \not\leq (0,1,0,1)$.

Definition 2.8. *Let X and Y be two binary n-tuples. We say that X* covers *Y (written $Y \leq X$) if and only if X has 1s everywhere Y has 1s. If neither $Y \leq X$ nor $X \leq Y$, then we say X and Y are unordered (written $X \not\subset Y$).*

In later discussions the relation $<$ will be used whenever $Y \leq X$ and $Y \neq X$.

Definition 2.9. *A set of binary n-tuples C is called* unordered code *if for every $X, Y \in C$, $X \neq Y$ implies $X \not\subset Y$.*

Thus, an unordered code is one in which no codeword covers some other codeword. In [185] it was shown that a code C is able to detect all u-errors if and only if it is unordered.

2.3.1. Nonsystematic m/n Codes

Definition 2.10. *An m-*out-of-*n* code *(m/n* code, constant-weight code*) is one in which all valid codewords have exactly m 1s and $n - m$ 0s.*

The set of all m/n codewords will be denoted with $C_{m/n}$ and its capacity equals to $|C_{m/n}| = \binom{n}{m}$. The $\lfloor n/2 \rfloor /n$ code is the optimal unordered code [44], in the sense that there is no other unordered code of codeword length n which has more codewords than the $\lfloor n/2 \rfloor /n$ code.

2.3.2. Systematic Unordered Codes

Definition 2.11. *The* optimal systematic unordered code *(OSUC) is one which uses the fewest number of redundant check bits.*

The optimal systematic unordered code was first defined by Berger [9].

Definition 2.12. *The* Berger code $C_{(I,K)}$ *is a systematic code wherein the K check bits P_X are the binary representation of the count of the I information bits J_X which are 0s, where $K = \lceil \log_2(I+1) \rceil$. (An alternative equivalent Berger code has the check part P_X defined as the bit-by-bit complemented number of 1s in the information part J_X.)*

Definition 2.13. *[5] A* Maximal Length Berger *(MLB)* code $C_{MLB(K)}$ *is one for which $K = \log_2(I+1)$, i.e. $I = 2^K - 1$.*

Henceforth we shall use the following concept of concatenated codes introduced in [185], which differs from the concatenated codes known from coding theory [134].

Definition 2.14. *The concatenation of a vertex U with u coordinates and a vertex V with v coordinates is a vertex $W = UV$ with $u+v$ coordinates whose first u coordinates are identical to those in U, and whose last v coordinates are identical to the coordinates in V.*

Definition 2.15. *The concatenation of two codes C_U and C_V of capacity $|C_U|$ and $|C_V|$, respectively, is a code $C_W = C_U \times C_V$ of capacity $|C_W| = |C_U| \cdot |C_V|$ which is formed by concatenating each vertex in C_U with every vertex in C_V.*

Definition 2.16. *A code C of length n is called* equivalent to Berger code $C_{(I,K)}$ *if and only if $n = I + K$, it is systematic, unordered, and has the same number of codewords as $C_{(I,K)}$.*

The codes equivalent to Berger codes were derived by Ashjaee and Reddy [5], [6], and recently by us [151], whereas some other non-optimal unordered systematic and nonsystematic codes can be constructed according to Mak [94] and Smith [186]. The modified Berger codes from [5] are defined for $I = 2^{K-1}$ (only) as the concatenation $C_{MLB(K-1)} \times C_{1/2}$, where two bits of a 1/2 codeword represent x_0 — the least significant bit (LSB) of the information part, and c_{K-1} — the most significant bit (MSB) of the check part. The codes from [6] and [151], which can be constructed for any $I \geq 4$ and for many $I \geq 6$ (but not all), respectively, have the property that all 2^K check parts are used (unlike in Berger codes with $I \neq 2^K - 1$). This property allows to speed up one part of an STC for this code — an output STC for K-pair 2-rail code (see Subsection 3.4). However, the codes from [151] also have the closure property, which is necessary to build a PLA-based STC and also allows one to build an IF STC for this code.

Finally, we should mention one important class of systematic unordered codes — the 2-rail codes.

Definition 2.17. *A K-pair 2-rail code is one which uses K check bits which are the bit-by-bit complements of the $I = K$ information bits.*

A K-pair 2-rail code is nothing else but a special case of an incomplete $K/2K$ code with only 2^K out of $\binom{2K}{K}$ all codewords used. It is the most redundant unordered EDC (100% redundancy), but its main advantage is that the TSC circuits using 2-rail code are generally easy to implement in hardware. Here, 2-rail codes are important, since an STC for various systematic EDCs C can be easily constructed by translating C into a 2-rail code [6], [207]. The most prominent example is the well known *normal checker* (shown here later in Fig. 3.1) which is conceptually the simplest structure that can be used to implement an STC for any systematic code. The topic of designing STCs for specific incomplete UEDCs is not considered in this paper. For more details, an interested reader may refer to [30], [152], [153], [194] — about the STCs for incomplete m/n codes, and to [151] — about the STCs for incomplete OSUCs.

2.4. Other Unidirectional EDCs (UEDCs)

2.4.1. t-UED Codes

The data used in a digital system are in many cases organized in bytes and stored, transmitted or transformed by separate units operating on bytes. Therefore, provided that only single hardware failures can occur and are confined to independent units, instead of u-errors of any multiplicity only up to t u-errors are the most likely to occur (t is the byte length). Also, errors caused by single faults in IF combinational circuits with shared circuitry may cause output u-errors of a limited multiplicity — up to some t,whose value depends on the maximum of outputs which depend on signal on some line of the circuit, e.g. PLAs [19]. As a result, in all the above cases a less redundant (and hence cheaper) t-UEDC can be used instead of an unordered code.

Let n denote the codeword length and t the multiplicity of u-errors detected by a code.

Definition 2.18. *A binary code C of codeword length n is called* t-unidirectional EDC *(t-UEDC) if no set of t u-errors can transform a codeword into another codeword.*

A. Borden t-UEDCs

Definition 2.19. *A* Borden code *$C(n,t)$ is a code which is the union of all m/n codes whose weight is congruent to* $\lfloor n/2 \rfloor$ mod $(t+1)$*, i.e.*

$$C(n,t) = \bigcup_{m=\lfloor n/2 \rfloor \bmod (t+1)} C_{m/n}\ . \tag{2.1}$$

The nonsystematic code $C(n,t)$ was defined by Borden [14] and shown to be the optimal t-UEDC. Its capacity is given by

$$|C(n,t)| = \sum_{m=\lfloor n/2 \rfloor \bmod (t+1)} \binom{n}{m}\ . \tag{2.2}$$

In particular, it was shown [14] that

$$|C(n,2)| = \begin{cases} (2^n+2)/3 & \text{for even } n, \\ (2^n+1)/3 & \text{for odd } n; \end{cases} \tag{2.3}$$

and

$$|C(n,3)| = \begin{cases} 2^{n-2} + 2^{(n-2)/2} & \text{for even } n, \\ 2^{n-2} + 2^{(n-3)/2} & \text{for odd } n. \end{cases} \tag{2.4}$$

The $C(n,t)$ code allows for any t from the interval $1 \le t \le \lceil n/2 \rceil$. Two extreme cases of the $C(n,t)$ code are: (i) for t=1 — the parity code which is systematic, unlike any other Borden code, and (ii) for $n = 2t$ — the constant-weight $t/2t$ code, which are optimal codes capable of detecting all single errors and all u-errors, respectively. (Note that three classes of codes, usually considered separately, can be defined in this simple uniform way.)

B. Systematic t-UEDCs

Systematic t-UEDCs were first proposed by Dong [38], and then the optimal t-UEDCs were given by Bose and Lin [17] and Jha and Vora [71]. In the Bose-Lin codes t is fixed for a given K, whereas in the Jha-Vora codes t also depends on I. As a result, for $K \geq 5$ the Jha-Vora codes are capable of detecting more u-errors for some I. Note however that any t-UEDC has fewer check bits K (i.e. it is less redundant) than a Berger code (which is the optimal AUEDC) for $I \geq 2^K$ only.

Let p_0 (p_1) denote the number of 0s (1s) in the I-bit input vector. Two following codes were given by Bose and Lin.

Bose-Lin Code 1

(i) $K \in \{2, 3\}$: $J = p_0 \bmod 2^K$. The code detects $t = K$ u-errors.

(ii) $K \geq 4$: let $J' = p_0 \bmod 2^{K-1} = (q_{K-2} \dots q_1 q_0)$ be the tentative check part. From J' the final check part $J = (s_{K-1} \dots s_1 s_0)$ is derived, where $s_{K-1} = q_{K-2}$, $s_{K-2} = \bar{q}_{K-2}$, and $s_j = q_j$ for any other j. Now the code detects up to $t = 2^{K-2} + K - 2$ u-errors.

Code 1 is optimal systematic t-UEDC for K = 2, 3, and 6 when it is capable of detecting t = 2, 3, and 6 u-errors, respectively.

Bose-Lin Code 2 (for $K \geq 5$ only)

First, $J' = p_0 \bmod (3 \cdot 2^{K-3}) = \{q_{K-2}, \dots, q_1, q_0\}$ is formed. Then, three MSBs $(q_{K-2} q_{K-3} q_{K-4})$ which are (000), ..., (101) are mapped one-to-one into $(s_{K-1} s_{K-2} s_{K-3}\ s_{K-4})$ as six 2-out-of-4 codewords. The final check part $J = \{s_{K-1} \dots, s_1, s_0\}$ consists of the latter four bits as MSBs and unchanged $K - 4$ LSBs from J'. Code 2 detects up to $t \leq 5 \cdot 2^{K-4} + K - 4$ u-errors, i.e. t is larger than for Code 1.

Jha-Vora Codes

The Jha-Vora codes are capable of detecting more u-errors than the Bose-Lin codes with the same K for $2^K \leq I < k \cdot 2^K$, where $k \leq 1.45$ and it varies with I and K (see Table 2 in [71]). They have a rather sophisticated structure of check parts assignment, which can be done by the following algorithm.

Algorithm 2.1.

1. Let $K = \lfloor \log_2 I \rfloor$. Set $l = I - 2^K + 1$.

2. Derive the set $S(K) = \cup_{j=0}^{\lfloor K/2 \rfloor} C_{(\lfloor K/2 \rfloor - j)/K}$ and arrange the binary K-tuples within $S(K)$ according to their decreasing decimal value. The i-th word of an ordered set $S(K)$ is denoted with $C_i(K)$, $1 \leq i \leq l$.

$x_{I-1} \cdots x_{I-b(K)}$	s_{K-1}	$x_{I-b(K)-1} \cdots x_{I-2b(K)}$	s_{K-2}	$\cdots$	s_p	$x_{b(K)-1} \cdots x_0$	$s_{p-1} \cdots s_0$

Figure 2.2. Codeword format of a Blaum's BUEDC

3. Derive the sets $B_i(K) = A_i(K) \setminus \{A_i(K) \cap \{\cup_{j=i+1}^{l} A_j(K)\}\}$, $1 \le i \le l-1$, and $B_l(K) = A_l(K)$, where $A_i(K) = \{X \mid X \in S(K) \text{ and } X < C_i(K)\}$ and $A_{l+1}(K) = \emptyset$.
4. Derive the set $Y = \{X \mid X \in \{0,1\}^K \text{ and } X \in \cup_{i=1}^{l} A_i(K)\}$, wherein all K-tuples are ordered according to their decreasing decimal value.
5. Derive a sequence Z of check parts Y, $C_1(K)$, $B_1(K)$, $C_2(K)$, $B_2(K), \ldots$, $C_l(K)$, $B_l(K)$ which are to be assigned successively to subsets of information parts of weights $I, I-1, \ldots, 0$.

   Note: If two or more sets have common elements, the sequence includes each instance of the common elements.
6. Find the maximum multiplicity of u-errors detected by this code

   $$t = \min_{i,\, 1 \le i \le l} \{p_2(C_i(K)) - p_1(C_i(K)) - 1\},$$

   where $p_1(C_i(K))$ and $p_2(C_i(K))$ denote the numbers of positions in which $C_i(K)$ occurs. □

For more details on Jha-Vora codes, an interested reader may refer to [71].

2.4.2. Burst-UED Codes (BUEDCs)

Faults in certain semiconductor memory systems tend to produce bursts of u-errors. Hence, codes capable of detecting bursts of a certain length using a minimum number of check bits are important, since further savings in the codeword length are possible (compared to both unordered codes and t-UEDCs).

To date, no nonsystematic burst UEDCs (BUEDCs) have been proposed. The optimal systematic BUEDCs were first proposed by Bose [15] and then improved by Blaum [12]. Given a sufficiently large number of information bits I, let $b(K)$ denote the maximum length of a burst that a systematic BUEDC can detect with K check bits. The Bose BUEDCs have $b(K) = 2^{K-1}$. The Blaum codes were shown more effective than t-UEDCs for $K \ge 3$ and $I \ge 2^K$. Compared to Bose BUEDCs, the Blaum codes are: (i) equivalent for $K \in \{2,3\}$, since they also have $b(K) = 2^{K-1}$, and (ii) more efficient for $K \ge 4$, since they have $b(K) > 2^{K-1}$:

for instance, $b(4) = 9 > 8$, $b(5) = 19 > 16$, and $b(6) = 41 > 32$. Hence we shall concentrate on the more efficient Blaum codes.

Check part assignment for the Blaum codes is easy for a given K. Suppose that check parts are assigned to the groups of information I-tuples with 0, 1, ..., and I 1s. The check parts, which are assigned subsequently and cyclically to these ordered groups, are the following: the all-1s K-tuple, K K-tuples of weight $K-1$ ordered increasingly according to their decimal values, $\binom{K}{K-2}$ K-tuples of weight $K-2$ etc., etc., the all-0s K-tuple which is followed again by the all-1s K-tuple, etc. Obviously, the sequence of check parts eventually repeats for every subsequent sequence of 2^K groups. An example of this encoding is given in the third column of Table 7.5 for $K = 3$ and any $I \geq 8$. The bits in a codeword X of a Blaum BUEDC are arranged as shown in Fig. 2.2, where $J_X = (x_{I-1} \ldots x_1 x_0)$ and $P_X = (s_{K-1} \ldots s_1 s_0)$.

3. SELF-CHECKING CIRCUITS

Any digital system is exposed to internal failures which occur during normal functioning in its elementary modules and thus may produce incorrect results. Any logic circuit which is part of such a system can generate incorrect outputs due to two major causes: its *internal faults* and/or *input errors* generated by some other circuit in a system that feeds a given one. We have already seen that encoding inputs and outputs of a circuit (and internal states—if it is sequential) with some EDCs C_{IN} and C_{OUT} is a necessary means to implement it as SC. In this section we shall present various classes of SC circuits, depending on the degree of protection they offer against the most likely internal faults and input errors.

Prior to analysing the correctness of the output generated by a digital circuit with internal faults or input errors, we introduce the following notation. (This notation and concepts applies to a functional circuit and a checker as well.)

H — a combinational circuit

X — an input space of **H** (the set of all 2^n input binary n-tuples);

Z — an output space of **H** (the set of all 2^s output binary s-tuples);

C_{IN} — an input code space of **H**, $C_{IN} \subset \mathbf{X}$;

C_{OUT} — an output code space of **H**, $C_{OUT} \subset \mathbf{Z}$;

$X = (x_{n-1} \cdots x_1 x_0)$ — input vector of **H**; x_{n-1} and x_0 are the MSB and the LSB, respectively;

$Z = (z_{s-1} \cdots z_1 z_0)$ — output vector of **H**;

X, Z — an input and output vector, respectively;

X^e — an input non-codeword;

F — the set of likely faults in **H**;

f — a fault in **H**;

$Z = H(X)$ — a formal notation for: **H** maps input X to output Z when no fault is present in **H**;

$Z = H(X, f)$ — a formal notation for: **H** maps input X to output Z when a fault f is present in **H**.

3.1. Self-Checking Circuits with Internal Faults

A circuit **H** with an internal fault f can generate three types of output:

1. $H(X, f) = H(X)$—a correct output is generated for X despite a fault f; we say that the *fault f is masked* for an input X.
2. $H(X, f) \neq H(X)$ and $H(X, f) \notin C_{OUT}$—an incorrect detectable output (a non-codeword output) is generated for X as a result of f; we say that the *fault f is detected by X.*
3. $H(X, f) \neq H(X)$ and $H(X, f) \in C_{OUT}$—an incorrect output codeword is generated for X as a result of f; we say that the fault f causes *undetectable output error*, in a sense that an error cannot be detected just by observing whether $H(X, f)$ is in C_{OUT} or not, which is the case in most SC circuits.

Apparently, the circuit **H** achieves the TSC goal when only the first two cases occur, for every fault from some well-defined set of likely faults F. The most often it is assumed that F is a set of all single s/z faults, $z \in \{0, 1\}$. However, there are several special classes of SC circuits which allow for extension of F with various classes of multiple faults (provided that a faulty circuit remains combinational). For instance, F can include: (i) any multiple fault in a single slice of a bit-sliced circuit protected by parity, (ii) any multiple unidirectional fault in an IF circuit using unordered code, (iii) any multiple fault in a circuit that is tested exhaustively during normal functioning. In this monograph, any possibility of extending F to include certain multiple faults will be mentioned, whenever applicable.

The class of circuits that achieve the TSC goal is formally defined in the following way.

Definition 3.1. *[2] A circuit* **H** *is* fault-secure *(FS) for a set of faults F, if for every fault f in F, it never produces an incorrect codeword at the output for a codeword at the input, i.e.*

$$(\forall f \in F)\,(\forall X \in C_{IN})\,(H(X, f) = H(X)) \text{ or } (H(X, f) \notin C_{OUT}).$$

However, the FS property guarantees only that **H** does not generate incorrect outputs for faults from F. Hence, another additional property is needed to guarantee that faults from F do not gather undetected inside a circuit **H**, so that **H** may loose its FS property due to undetected multiple faults (which are not in F anymore).

Definition 3.2. *[20] A circuit* **H** *is* self-testing *(ST) for a set of faults F, if for every fault f in F, it produces a non-code space output for at least one code space input, i.e.*

$$(\forall f \in F)(\exists X \in C_{IN} \mid H(X,f) \notin C_{OUT}).$$

Definition 3.3. *[2] A circuit* **H** *is* totally self-checking *(TSC) for a set of faults F, if it is both FS and ST for F.*

A functional circuit that is TSC for F guarantees its correct operation in the presence of internal faults, provided that each fault f is detected before another fault occurs. Designing TSC functional circuits is desirable, since the TSC property is very strong. Unfortunately, this property can be very difficult to achieve (if at all) in practical circuits and also a TSC realization of a functional circuit may be very costly compared to its non-TSC realization. The survey of design methods of TSC functional circuits using various EDCs can be found e.g. in [145], [159], [207]. To ease the design requirements, the following class of SC functional circuits was introduced, which achieve the TSC goal despite certain undetected internal faults.

Definition 3.4. *[188] A circuit H is* strongly fault-secure *(SFS) for a set of faults F, if for every fault f in F, it is either TSC or it preserves the FS property while f is present.*

The SFS circuits are the largest class of functional circuits with internal faults that meet the TSC goal. However, any SFS circuit can be converted into a TSC circuit by removing redundant gates or lines. Hence, the TSC functional circuits are a class of SC circuits which is most frequently considered in the literature however. Moreover, they are better suited for a formal rigorous verification of their FS and ST properties than all other SC circuits.

The construction of complex TSC systems, built using TSC modules as in Fig. 1.1, was considered in [111], [117], [187], and [207]. These works present some important general theoretical results regarding the *proper* and *optimal* placement of the checkers in a complex SC digital system, which guarantee that the TSC goal is achieved on a system level in the most efficient way. Of particular concern is that all the error signals generated by the STCs are properly handled to monitor the correctness of the system operation [117]. Several implementations of complex SC modules and systems, using a variety of EDCs in each system, have been studied in [25], [37], [111], [121], [132], [169], [184], [207], and [213].

3.2. Self-Checking Circuits with Input Errors

Denote by X^e an input non-codeword resulting from errors in an input codeword X. The circuit **H** with an input error X^e can react in three different ways:

1. $H(X^e) = H(X)$—a correct output is generated for X^e despite input errors; we say that the *circuit* **H** *masks an input error* (or an *input error is corrected by* **H**).
2. $H(X^e) \neq H(X)$ and $H(X^e) \notin C_{OUT}$—an incorrect detectable output (a non-codeword) is generated for X^e; we say that an *input error* X^e *is detected (propagated) by* **H**.
3. $H(X^e) \neq H(X)$ and $H(X^e) \in C_{OUT}$—an incorrect undetectable output (an incorrect output codeword) is generated for X^e (or X^e *causes an undetectable output error*).

The qualification of the correct/incorrect behavior of the circuit **H** with input errors depends on the function performed by **H**. If **H** is a functional circuit, the first two cases are acceptable since no undetected output error is produced in either case. However, if **H** is a *checker*, i.e. **H** is used to signal the errors generated by a functional circuit, then only the case 2) is acceptable, since **H** should signal every occurrence of a non-codeword on its input. Otherwise (case 1), latent permanent faults may accumulate inside the system and deny its SC behavior for some faults from F. When an input non-codeword occurs, then a checker should signal its occurrence by generating a non-codeword output, although it is immaterial which of the error signals is actually generated. Also, when a checker is faulty and a given non-codeword cannot produce a non-codeword output (due to a fault), then it is immaterial what output codeword is produced (i.e. correct or not). This is the reason why the FS property, which is a must in TSC functional circuits, is not required for checkers.

Similarly as for internal faults, depending on how a circuit behaves in the presence of input errors, the following classes of circuits were defined.

Definition 3.5. *[2] A circuit H is* code-disjoint *(CD) if it maps codewords at the inputs to codewords at the outputs and non-codewords at the inputs to non-*

codewords at the outputs, i.e.

$$(\forall X \in C_{IN})(H(X) \in C_{OUT}) \text{ and } (\forall X^e \notin C_{IN})(H(X^e) \notin C_{OUT}) .$$

Let E_{IN} denote the set of non-codewords which can occur on the inputs of **H** as the result of the most likely faults in the circuitry that feeds **H**.

Definition 3.6. *[111] The circuit* **H** *is* error-propagating *(EP) for E_{IN} if it is CD with input non-codewords limited to E_{IN} only.*

The EP property is justified when the occurrence of some input non-codewords is unlikely. It allows for relaxation of some requirements imposed on the circuit, provided that E_{IN} is a proper subset of all input non-codewords; otherwise, the EP circuit is nothing else but a CD circuit. Actually, despite that it was not stated explicitly, many checkers for various codes which have been proposed in the literature and that were called CD, are in fact not strictly CD but EP only (see e.g. [2], [98], [137], [167]), as are some STCs for m/n codes with $n \neq 2m$ presented here in Section 5. For instance, consider an STC for an incomplete m/n code. A strictly CD checker would have to signal an input error whenever any of unused m/n codewords occurs (basically, they should be treated as non-codewords). However, these unused m/n codewords cannot occur as long as the assumed u-error model remains valid, since they might occur as a result of multiple bidirectional errors only, which are assumed unlikely. Similar reasoning applies to any other incomplete unordered code as well.

Definition 3.7. *[111] A circuit* **H** *is* error-secure *(ES) for a set of input errors E_{IN} if: (i)* **H** *maps any input codeword into an output codeword and (ii) either* **H** *maps any input non-codeword into an output non-codeword or* **H** *generates a correct output codeword which would occur if there were no input errors.*

The ES property is useful for functional circuits only, as it relaxes rather demanding requirements stipulated by the CD property, but it allows us to preserve correct operation of a circuit. Obviously, the ES property is inacceptable for a checker, as it would fail to signal some errors.

We also note for completeness, that the strongly code-disjoint (SCD) circuits were announced [118] as the largest class of CD circuits (including checkers) that meet the TSC goal, since they preserve the CD property even in the presence of undetected internal faults from F. However, no SCD circuit considered at the gate level with undetected s/z faults which change the logic function realized by a circuit has been reported yet. In fact, the non-trivial SCD circuits are only known for undetected internal faults considered on the transistor and layout level. That is why we concentrate here on the CD circuits only.

3.3. Self-Testing Checker (STC)

Definition 3.8. *[2] A* self-testing checker *(STC) for a code C and the set of faults F is a circuit H that is CD for C and ST for F.*

Similarly to most STCs considered in the literature, we assume that an STC is a two-output circuit with code outputs $(c_1c_0) \in \{(01),(10)\}$. Therefore the occurrence of a (00) or (11) on the checker output signals detection of an input error or an internal fault in the checker itself. Obviously, a system must initiate further diagnosis to identify the cause of this error signal.

Here we make typical assumptions regarding the fault/error behavior of an SC circuit, which are necessary to allow for a reasonably complex algebraic analysis of faults and errors in circuits being designed.

(A1) Only faults from the set F occur.

(A2) Faults occur one at a time.

(A3) After the occurrence of a fault f in the circuit **H**, a sufficient time elapses during which some codeword from C_{IN} is applied, which is a test for f and which produces a noncode output before a new fault occurs in **H**.

(A4) Errors and faults are not present at the same time.

An STC is designed in such a way that it guarantees that any fault from F (usually single faults) can be detected by producing the (00) or (11) output for some input codeword. However, in reality, even the checker designed as ST may still contain a small hardcore portion — when the assumption regarding F does not hold in real world. For instance, the double fault c_1/z and $c_0/\overline{z}$, $z \in \{0,1\}$, at the checker output could prevent the checker from signalling any input errors thereafter. A subsequent fault in the functional circuit (see Fig. 1.1) and the corresponding error could go unnoticed. Hence, the periodic off-line testing of a small portion of the checking hardware is essential to guarantee reliable operation of these circuits. In the worst case, the hardcore of an STC is limited to its two output lines. Similar considerations apply for the system-level checker that handles error signals provided by separate checkers spread throughout the system.

As is customary, we shall consider the following characteristics of the STCs:

1. The number of gates $Ga(\cdot)$ and the total number of gate inputs $In(\cdot)$—as measures of checker complexity;
2. The number of gate levels $L(\cdot)$—as a measure of checker speed; and
3. The number of tests $|\mathbf{T}(\cdot)|$ to detect all single s/z faults, $z = \{0,1\}$, required —as a measure of checker testability.

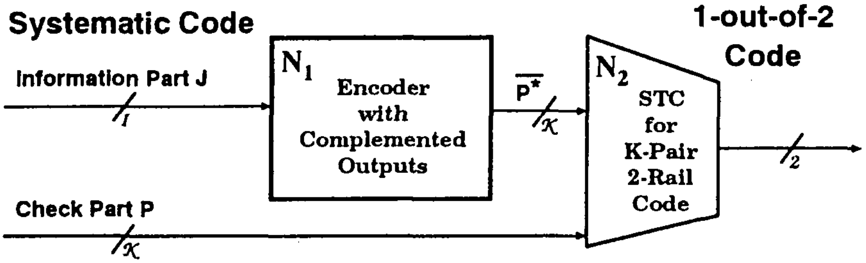


Figure 3.1. Structure of a normal STC for any systematic code

The number of tests needed to test an STC is important for three reasons:

- An error signal generated by the checker signals the occurrence of either an input error or a fault in the checker itself. The fewer test patterns are needed to detect the most likely faults in an STC, the easier is the diagnosis of the cause of error signal.
- It is quite common that only a subset of codewords is actually used when a checker is embedded in a system. Therefore a checker which is testable by a small test set can still be suitable to monitor a circuit that generates only a subset of all codewords, provided that they include all tests necessary for the checker. The more so that the minimal test set for some checkers is not unique.
- The validity of the assumptions (A3) and (A4) can be objectionable in some real world applications, when the fault latency is very large (*fault latency* is the length of time between the occurrence of a fault and the appearance of an error due to that fault). As an example, consider an STC for a systematic code with $I = 32$ information bits, assuming that all 2^{32} inputs are equally likely to occur. Any fault which is tested by exactly 1 out of 2^{32} inputs has an extremely large fault latency. As a result, the checker (which is formally ST) turns out to have internal faults which are virtually untestable during normal functioning. Nevertheless, these assumptions become more realistic if a checker is easily-testable, since fewer tests may fully exercise the checker in shorter periods of time.

The STCs for nonsystematic m/n and Borden codes may have various internal structures, hence they will be detailed in suitable sections. On the other hand, most known checkers for systematic codes (except e.g. those from [5], [141], [142], [151], [166]) have a more uniform structure. A general structure for an STC for *any systematic code*, shown in Fig. 3.1, was suggested in [6] and [207] and will be called a *normal checker.* The circuit N_1 is an encoder with complemented outputs — a combinational circuit that reencodes I information bits J to obtain K complemented check bits $\overline{P^*}$ for J. Under fault-free conditions P^* is equal to P. Then the circuit N_2, an STC for the K-pair two-rail code, checks whether the two input K-tuples (P and $\overline{P^*}$) are bit-by-bit complements of each other. Let's point out that the structure of the normal checker explains why we shall consider in Section 7 the STCs for systematic UEDCs along with encoders.

To be ST, a normal checker must use:

1. An irredundant encoder with complemented outputs N_1; and
2. The comparator N_2 — an STC for the K-pair 2-rail code — which is sufficiently exercised by the 2-rail vectors that occur on its input.

The condition 1 is sufficient, provided that all 2^I codewords are used. If this rather difficult to meet assumption holds, then the fault model assumed for N_1 is not very important here as long as the faulty circuit remains combinational, since N_1 is tested exhaustively during normal functioning. On the other hand, if the circuit N_2 (a 2-rail checker) is implemented as IF, it is ST for any multiple unidirectional s/z fault, $z \in \{0,1\}$ (see Lemma 3 in [185]).

Basically, the circuit N_2 can be designed by using one of the many methods described e.g. in [68], [74], [99], [107], [155], and [207]. The fastest (if one ignores increased delay due to large fan-in and fan-out) two-level circuit N_2 can be used in the STCs only for systematic codes which are *complete*, i.e. use all 2^K check bits combinations. An STC for any *incomplete* code must use a multi-level easily-testable implementation of the circuit N_2. The least amount of hardware requires the K-pair 2-rail STC built of $(K-1)$ 2-pair 2-rail STC modules arranged in a tree form with $\lceil \log_2 K \rceil$ module levels or as an iterative network with $K-1$ levels. The minimal test set for this 2-rail checker consists of only four 2-rail codewords. The 2-pair 2-rail STC, first defined in [20], implements the pair of functions:

$$\begin{aligned} c_1 &= s_0 s_1 + t_0 t_1 \\ c_0 &= s_0 t_1 + s_1 t_0, \end{aligned}$$

where $t_0 = \bar{s}_0$ and $t_1 = \bar{s}_1$ in error-free conditions. It requires all four codewords $(s_1 s_0 t_1 t_0) = \{(0011), (0110), (1001), (1100)\}$ as tests.

The universal STC for an incomplete K-pair 2-rail code, applicable in a normal STC for most systematic EDCs was proposed in [99]. (The only exception is any code using exactly $2^{K-1}+1$ check parts corresponding to subsequent decimals $0 \leq r \leq 2^{K-1}$, such as a Berger code with $I = 2^{K-1}$.) An improved universal STC for an incomplete K-pair 2-rail code with $2^{K-1}+4 \leq |C_{2r}| < 2^K$ we have described recently in [155] ($|C_{2r}|$ denotes the capacity of an incomplete

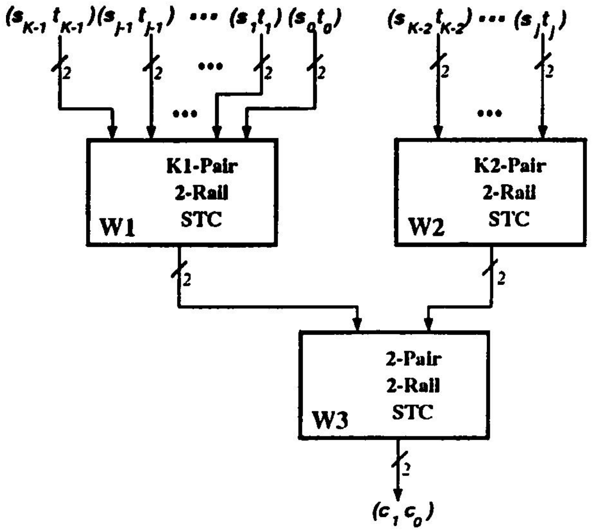

$(s_{K-1}t_{K-1})(s_{j-1}t_{j-1})$ ••• $(s_1t_1)(s_0t_0)$
$(s_{K-2}t_{K-2})$ ••• (s_jt_j)
2
2
2
2
2
2
•••
•••
•••
K1-Pair
2-Rail
STC
W1
K2-Pair
2-Rail
STC
W2
2
2
2-Pair
2-Rail
STC
W3
2
(c_1c_0)

Figure 3.2. Internal structure of the universal STC for an incomplete K-pair 2-rail code with at least 2^{K-1}+2 codewords

2-rail code). It can be implemented as shown in Fig. 3.2. The principal difference compared to [99] is that the block **W1** may have $K1 > 2$ pairs of inputs, which may reduce the number of gate levels in the whole checker. The actual value of $K1$ depends directly on $|C_{2r}|$ according to the inequalities:

$$2^{K-1} + a \leq |C_{2r}| < 2^K, \quad \text{where} \quad 2^j \leq a < 2^{K-1} - 1 \text{ and } j \geq 1. \qquad (3.1)$$

Thus, the main problem in designing an STC for many systematic UEDC is how to construct an encoder with complemented outputs N_1. This topic will be elaborated in general terms in the next section and in more details in Section 7.

4. COUNTERS OF 1s

As it was pointed out in the Introduction, checking whether a binary n-tuple X is a codeword of some UEDC involves generating the weight of X in one form or another. Checking can be as simple as generating a shear weight of the check part — as for Berger codes, but for all other UEDCs it involves finding a nontrivial function of weight. It is therefore natural that the STCs (and encoders in case of systematic codes) employ counters of 1s. In this section we shall study the design principles of counters of 1s, realized by employing two essentially different design concepts. One is a circuit generally built on the basis of carry-save adders (CSAs) consisting of FA and HA cells, that will be called henceforth a *parallel counter.* The other is a counter of 1s built using a multi-output threshold circuit T^n, that will be referred to a *T^n-based counter of 1s.* On the basis of these two design concepts, several variations of the STCs (and encoders — in case of systematic codes) needed by specific codes will be derived in sections that follow.

4.1. Parallel Counters

4.1.1. Design and Complexity

Parallel counters were introduced by Wallace [208] and Dadda [26] for applications in fast multipliers. In fact, all important results on parallel counters appeared in the literature on computer arithmetic (in particular on multipliers and multioperand adders) and, quite surprisingly, none of them have been noticed by the researchers working on FT hardware, cf. [13], [99], etc.

Definition 4.1. *A* parallel $(n;q)$ counter *is a combinational network with q outputs and $n \leq 2^q - 1$ inputs, where the binary number represented by the q outputs is the number of 1s present at the inputs; obviously, $q = \lceil \log_2(n+1) \rceil$.*

Under the name *carry showers* Foster and Stockton [43] described a method for designing large parallel counters built of FAs, i.e. (3;2) counters. Then this method was generalized by Swartzlander [192] and Kobayashi and Ohara [76] to build large parallel counters from smaller ones. Despite all these results, all works on FT hardware and, in particular, on designing STCs for UEDCs — [15], [17], [67], [68], and d-EC/UEDCs — [13], [124], use slightly less efficient parallel counters only from [99] (an easily-testable version of those from [3]).

Now we shall concentrate on the parallel counters from [76] built of FAs, which are not only the fastest but are also optimal in terms of hardware. The parallel $(n;q)$ counter is built of $FA(n;q) = n - q$ FAs and $0 \leq HA(n;q) < q$ HAs, where $HA(n;q) = 0$ if and only if a counter is *saturated,* i.e., for $n = 2^q - 1$. Let $St_P(n;q)$ be the total delay time of an n-input parallel counter built as a network of FAs and HAs, measured in Δ, where Δ $(\frac{1}{2}\Delta)$ is the delay introduced by an FA (HA). The lower-bound [76] and the upper-bound [192] on $St_P(n;q)$ are given by

$$\lceil \log_3(n) \rceil + \lceil \log_2(n+1) \rceil - 2 \leq St_P(n;q) \leq 2\lfloor \log_2(n) \rfloor - 1. \qquad (4.1)$$

To make possible the exact computation of delay, Table 4.1 provides the exact number of stages $St_P(n;q)$ on a FA tree that processes n input bits, that we have

Table 4.1. The exact number of stages $St_P(n;q)$ on a FA tree that processes n input bits

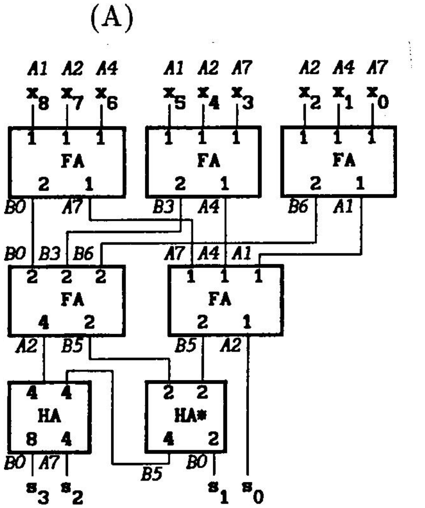
n	Stages [Δ]	
	Optimal	[99]
$2 \div 3$	1	1
$4 \div 7$	3	3
$\boxed{8 \div 9}$	4	5
$10 \div 15$	5	5
$\boxed{16 \div 27}$	6	7
$28 \div 31$	7	7
$\boxed{32 \div 55}$	8	9
$56 \div 63$	9	9
$\boxed{64 \div 127}$	10	11
$\boxed{128 \div 135}$	11	13
$\boxed{136 \div 255}$	12	13

derived for most n of practical value. Either version of a parallel counter uses $n - q$ FAs, but those from [99] usually use a few more HAs. The boxes indicate n for which the optimal counters are faster than those from [99].

Further details on implementing large parallel counters using smaller ones (such as (7;3) and (15;4)) can be found in [76]. For instance, the parallel (127;7) counter can be built in seven stages of a total of 31 parallel (7;3) counters. The latter can be realized in three ways: (i) with 26 FAs; (ii) with a single 128 × 3 ROM; and (iii) as a gate level circuit from [104], which was shown slightly faster and less complex than that using FAs.

Example 4.1. *The logic scheme of the optimal parallel (9;4) counter is given in Fig. 4.1(A). The weights of all signals entering and leaving the cells are marked. To save space, a shorthand notation, such as shown in Fig. 4.1(B), will be used for other parallel counters as well.* □

In a general case, the n-input parallel counter of 1s can be described by using the table (such as in Fig. 4.1(B)) with $q = \lceil \log_2(n+1) \rceil$ columns marked with G_j, $j \in \{0, 1, \ldots, q-1\}$. The entry in the column G_j denotes either how many bits of a given weight 2^j are present at a given stage of computation, or what modules operate on the bits of weight 2^j.



(B)

G_3	G_2	G_1	G_0
—	—	—	9
—	—	—	3 FAs
—	—	3	3
—	—	FA	FA
—	1	2	1
—	← HA	← HA	—
1	1	1	1
s_3	s_2	s_1	s_0

Figure 4.1. Optimal parallel (9;4) counter: Interconnection scheme (A); Shorthand notation (B)

4.1.2. Testing of the Parallel Counter

The test set sufficient to detect all multiple faults affecting a single cell (FA or HA) of a parallel $(n; q)$ counter built as a Wallace tree can be derived by using the heuristic procedure proposed in [21]. Its size equals to 8 for $n = 2^q - 1$ (saturated parallel counters) and is upper-bounded by $8(q-1)$ for other n. However, as the examples below show, the minimal test set can be significantly smaller than $8(q-1)$. The only systematic procedure of generating the minimal test set has been proposed for both saturated and nonsaturated parallel counters designed for easy testability according to the method proposed in [99]. One limitation of the design method from [99] is that it generates a unique realization of a parallel counter for a given n which is not optimal. In contrast, the FAs and HAs in optimal parallel counters can be interconnected in many ways and therefore the minimal test set must be derived for a particular scheme. Derivation of interconnections between FAs and HAs of the optimal parallel counter which requires the fewest number of tests, as far as we know, remains an open problem. In [67] it was shown that the differential cascode voltage switch (DCVS) logic implementations of the schemes from [99] are testable for many classes of faults typical for DCVS by four and $5\lceil\log_2(n+1)\rceil$ tests for $n = 2^q - 1$ and other n, respectively.

Now we will present some basic ideas about generating a quasi-minimal test set for the optimal parallel counter. The complete (and in some cases minimal) test set for the optimal parallel counter will be derived similarly to [21]. We begin by defining the basic 3-bit test sequences as listed in Table 4.2. They have

Table 4.2. Basic test sequences for a FA

$A1$	$A2$	$A4$	$A7$	$B0$	$B3$	$B5$	$B6$
0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1
1	0	0	1	0	1	1	0

the following properties: (i) the sequences Ai (Bj) are of even (odd) weight; (ii) the index i (j) is the decimal value of the sequence with the bottommost bit being the LSB; and (iii) a sequence Ai is the bit-by-bit complement of Bj with $j = 7 - i$. If an FA receives three different sequences Ai (Bj) on its inputs, it is equivalent to applying all three tests of weight one (two). Then, its Sum output produces the fourth (with respect to three input sequences) unused sequence Ai (Bj) whereas its $Carry$ produces a bit-by-bit complement of Ai (Bj). Therefore applying to the FA inputs three different sequences Ai and the all-1s vector (111) followed by their complements exercises exhaustively an FA. Now we will use this observation to generate a complete test set for a parallel counter. First, some sequences Ai or Bj are assigned to the primary inputs of the parallel counter. Then, the sequences occurring on the inputs of all cells are calculated. If each FA (HA) cell receives three (two) different sequences, then the whole parallel counter is tested by applying eight tests only: the all-1s vector, three tests implied by the primary inputs sequence assignment, and the bit-by-bit complements of these four tests. Any cell that is not tested exhaustively by a given test set can be easily identified, since it receives at least two identical sequences Ai or Bj on its inputs. This information either can prompt to search for another input sequence assignment or it can be used to identify all modules with some missing tests for which special extra tests will be generated.

Example 4.1 (Cont'd). *The parallel (9;4) counter from Fig. 4.1(A) is tested for most faults by applying the following tests: the all-0s vector, three tests corresponding to the sequences Ai marked on the scheme, and four tests being the bit-by-bit complements of these four tests. Only an HA marked with an asterisk * does not receive all tests: it needs two extra tests that provide it with missing (01) and (10) tests, e.g. (001 001 000) and (000 000 110), respectively. Thus, this counter needs only 10 tests compared to estimated 24.*

For comparison, we include an alternative scheme of the 9-input parallel counter designed according to the method from [99]. This scheme, shown in Fig. 4.2, has the same complexity as the one given above, but has one more FA stage (its delay is 4Δ). For this scheme we have derived the minimal test set which consists of only 8 tests. The test patterns can be easily derived from the notation

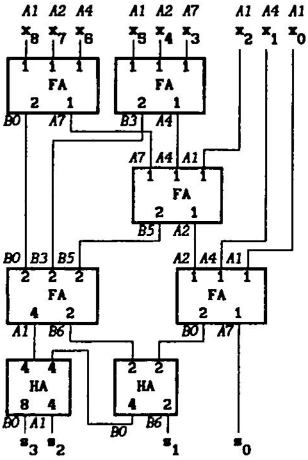


Figure 4.2. Parallel (9;4) counter by Marouf and Friedman [99]

shown on the scheme.

Now compare the characteristics of the above scheme against two schemes considered in the literature. First, note that according to [99], the scheme from Fig. 4.2 requires 24 tests, but we have shown that 8 tests is enough. Secondly, a scheme from [21] designed for testability is tested by 9 tests but it uses: six FAs, one HA, and a 2-input AND gate, and it introduces the delay of 3.5Δ. □

These examples prove that the parallel counters designed according to [76], which use the least amount of hardware, not only offer the best speed, but are also easily-testable. It can be proved, by using similar argument as in [21], that they require no more than $8(q-1)$ tests. The above example and our analysis of several other examples show that this bound is quite loose, since the actual number of tests is usually very close to 8. Finally, it should be emphasized that speed is a significantly more important parameter than the size of the minimal test set, at least when encoders, decoders, and STCs for error codes are concerned. Especially, when only two or four extra tests are needed and no extra hardware cost is involved.

4.2. Multi-Output Threshold Circuits T^n

4.2.1. Survey of Implementations of a Threshold Circuit T^n

Let $S = \{x_1, x_2, \ldots, x_n\}$ denote a set of n input variables and m denote a threshold.

Table 4.3. Parameter estimations of various threshold circuits T^n

Type of Circuit T^n	Gates	Gate Levels	Tests
Two-Level AND-OR	$2^n + n - 2$	2	2^n
Cellular Arrays [62], [169]	$n^2 - n$	$2n - 3$	$2n$
Completely Bifurcated	$\frac{1}{2}n^2 + n\log n - \frac{3}{2}n + 1$	$2\log n - 1$	$\frac{1}{4}n^2 + n + 1 - c$
Sorting Network [8], [149]	$\leq \frac{1}{2}n\left(\log^2 n - \log n + 4\right) - 2$	$\frac{1}{2}\log^2 n + \frac{1}{2}\log n$	$n + \lceil n/2 \rceil$

[1] $c = 0$ for n even and $c = \frac{1}{4}$ for n odd

[2] $\log n$ denotes $\lceil \log_2 n \rceil$

Definition 4.2. *A* threshold function T_m^n *is a switching function of n variables from the set S, which takes the value 1 if and only if at least m out of n input variables from S are 1s,* $1 \leq m \leq n$.

Definition 4.3. *A* multi-output threshold circuit T^n *is a circuit that implements all n T_m^n threshold functions of n variables,* $1 \leq m \leq n$.

Several implementations of the threshold circuit T^n have been proposed in the literature under various names. Hurst [62] and Edwards [42] proposed various implementations of the circuit T^n under the name of a *digital summation threshold logic* (DSTL) *gate.* The cellular realization by Reddy and Wilson [168] was called a T^n *array.* In the literature on STCs for m/n codes, where threshold circuits were used as basic building elements, the function T_m^n was called a *majority function* [2], [49], [54], [98], or a *threshold function* [136], [137]. Finally, Lamagna [84] briefly mentioned that the threshold functions of n variables and the *sorting function* (i.e. the set of functions implemented by T^n) are equivalent. Until recently [143], [148], the latter notice as well as voluminous literature on SNs have been completely overlooked by the researchers working in the field of logic design. For instance, in spite of the existence of the less complex circuits T^n described as SNs in the sixties [8] and later [75], [39], [205], only the circuits proposed under the name of threshold or majority circuits were used in the STCs for m/n codes considered in [2], [49], [54], [98], [136], [137], [167], [177], [178]. Similarly, the DSTL gates from [62] (1973) and [42] (1978) were proposed many years later than Batcher's SNs [8] (1968), which were significantly less complex implementations of T^n already available at that time.

Throughout this work we will use the term 'threshold circuit' generically and the term 'sorting network' only to this particular class of implementations of T^n.

The parameter estimations of various implementations of the circuit T^n are summarized in Table 4.3. The parameters of the Batcher's SNs using odd-even merging are given, despite that there exist slightly simpler and faster SNs for some n. The number of gate levels in an SN is equal to the maximum number of comparators in contact with any *path* through the network [75]. The size of the minimal test set for each version of a threshold circuit is derived for all single s/z faults. From Table 4.3 we conclude the following:

1. The SNs are the least complex of all circuits T^n considered and require the smallest number of tests.
2. Only the two-level (for $n \geq 3$) and the so called *completely bifurcated ciruits* (for $n \geq 7$) seem superior to the SNs with respect to the number of gate levels. However, it occurs at the cost of significant amount of hardware and assuming no restrictions on fan-in and fan-out (which may be questionable for most technologies).

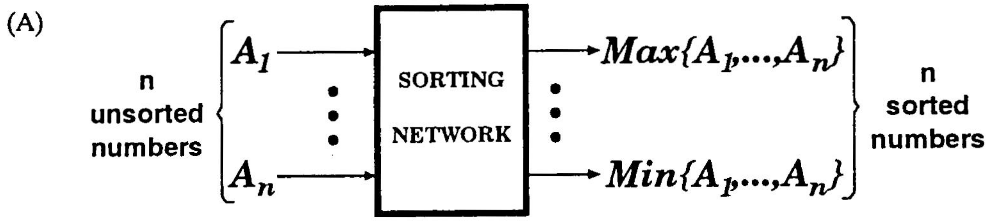
Now we shall concentrate on designing circuits T^n as SNs.

4.2.2. T^n Implemented as a Sorting Network

An n-input *sorting network* (SN) is a switching network with n outputs that for any combination of inputs $\{A1, A2, \ldots, An\}$ generates the outputs which are a sorted permutation of inputs, e.g. in nonincreasing order (see Fig. 4.3(A)). A *binary* SN is entirely built of simple identical *comparator cells* with two inputs and two outputs such as one shown in Fig. 4.3(B). For brevity, we will call such a cell simply a *comparator* (the symbol of a comparator given in Fig. 4.3(B) is traditionally used in the literature on sorting). A comparator executes a *pass* or *interchange* operation, depending on the inputs. As explained in Fig. 4.3(B), here the upper output $M = MAX(A1, A2)$ and the lower output $m = MIN(A1, A2)$ carry the maximum and the minimum of the two inputs $A1$ and $A2$. An SN built using this element sorts in nonincreasing order. However, here we are interested in the simplest implementation of a comparator cell — such as shown in Fig. 4.4(B), which implies that in our case an SN sorts 0s and 1s — see Fig. 4.4(A).

The problem of designing SNs has been studied extensively in the literature for many years. The milestone result is the construction given by Batcher [8] in 1968. In 1974 Van Voorhis [205] presented constructions of SNs which provide the best upper bound for most practical values of n. An excellent survey of many SNs designs was provided by Knuth [75]. In particular, he listed some *ad hoc* constructions of the SNs for $n \leq 16$ invented by many authors, which are either more efficient or faster than those from [8] and [205].

Batcher [8] proposed two schemes of SNs constructed from: (1) odd-even merging networks (MNs) and (2) bitonic sorters using the sorting-by-merging scheme. The two schemes have the same number of levels but only the first one, which is built of fewer comparators, is of interest here. Batcher's SNs use the optimal number of comparators for $n \leq 8$. The least complex schemes of



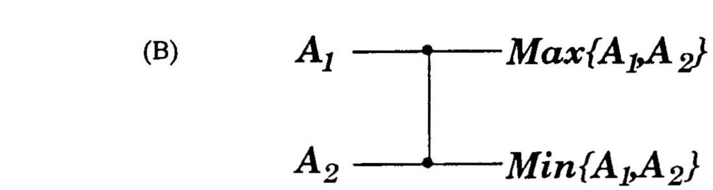
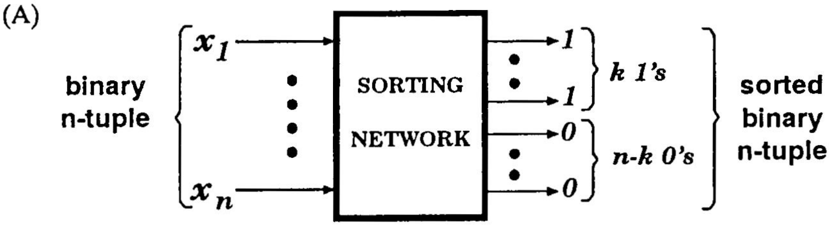


Figure 4.3. General structure of an SN (A); A comparator cell (B)



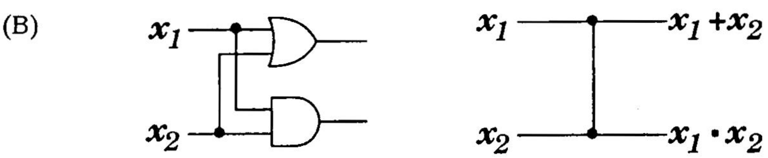


Figure 4.4. The structure of an SN considered here (A); Logic gate implementation of a comparator cell (B)

SNs for $n > 8$ can be found for $9 \leq n \leq 16$ in [75] (Fig. 49, p. 228). The fastest schemes of the SNs for $n = 6, 9, 10, 12$, and 16 can be found in [75] (Fig. 51, p. 231). The two least complex constructions of SNs currently known as $n \to \infty$, found independently by Drysdale [39] and Van Voorhis [205], use the most efficient 16-input SN by Green (refer to Fig. 49 on p. 228 in [75]). Van Voorhis' scheme is slightly better: e.g. for $n = 256$ it requires 3651 comparators

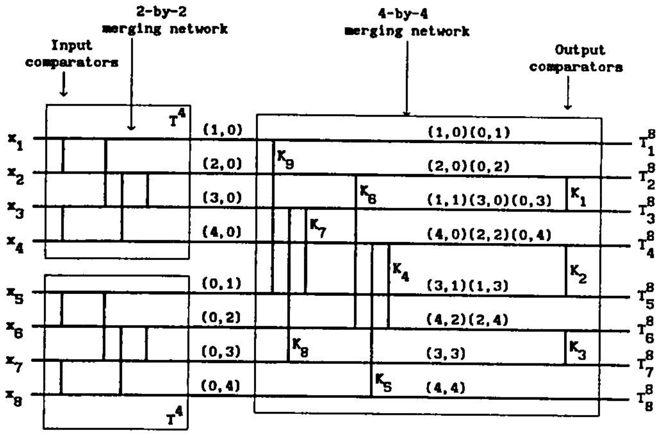


Figure 4.5. Batcher's 8-input sorting network T^8 using odd-even merging

while Green's requires 3657. For most n of practical value for logic designer (i.e. for $n \leq 64$) the hardware savings offered by the schemes from [39] and [205] are not significant: for instance, Van Voorhis' SNs with $n = 16, 32$, and 64 use 60, 180, and 514 comparators compared with Batcher's 63, 191, and 543. Moreover, the irregular structures of the SNs from [75], [39], [205] make them difficult to analyze. Therefore, with a few exceptions, we concentrate on Batcher's SNs.

For completeness, we include the strongest lower bound (Lamagna [84]) and upper bound (Van Voorhis [205]) for the number of comparators $S(n)$ in the SN with n inputs (the number of two-input gates equals $2S(n)$):

$$S(n) \geq n(\log_2 n) + 0.5n\,(\log_2(\log_2 n)) + O(n) \tag{4.2}$$

and

$$S(n) \leq 0.25n(\log_2 n)^2 - 0.395n(\log_2 n) + 1.4306n - 1.5. \tag{4.3}$$

The parameters of the Batcher's SNs for $3 \leq n \leq 16$ are included in Table 4.4. More references on threshold circuits and other details about SNs can be found in [143], [148].

Table 4.4. Parameters of threshold circuits T^n realized as optimal SNs for $3 \leq n \leq 16$

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Gates	6	10	18	24	32	38	50	62	74	78	96	106	118	126
Levels	3	3	5	5	6	6	8	9	10	9	10	10	10	10
Tests	6	6	8	9	11	12	12	15	17	16	20	21	23	24

The 2^k-input odd-even merging SN can be built using the following iterative rule. It consists of 2^{k-1} comparators followed by 2^{k-2} 2-by-2 MNs followed by

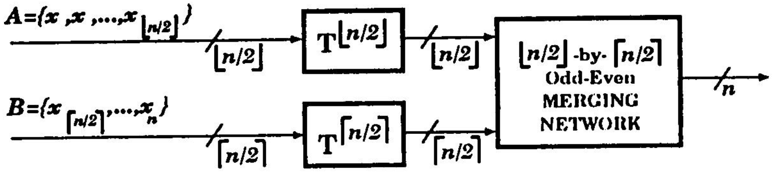

$A=\{x,x,\ldots,x_{\lfloor n/2\rfloor}\}$
$B=\{x_{\lceil n/2\rceil},\ldots,x_n\}$
$\lfloor n/2\rfloor$
$\lceil n/2\rceil$
$T^{\lfloor n/2\rfloor}$
$T^{\lceil n/2\rceil}$
$\lfloor n/2\rfloor$
$\lceil n/2\rceil$
$\lfloor n/2\rfloor$-by-$\lceil n/2\rceil$ Odd-Even MERGING NETWORK
n

Figure 4.6. General structure of the n-input SN T^n using odd-even merging

2^{k-3} 4-by-4 MNs, *etc.* Figure 4.5 shows the scheme of the 8-input SN (and the 4-input SN inside). It is seen that the 4-by-4 MN from Fig. 4.5 is built of two 2-by-2 MNs and three output comparators. In general case, the 2^k-by-2^k MN is built of two 2^{k-1}-by-2^{k-1} MNs and $2^k - 1$ output comparators. A more straightforward approach to the design of an SN with any n follows from the generalized version of Batcher's construction given in [75]. The idea is to sort n elements by sorting the first $\lfloor n/2 \rfloor$ and the last $\lceil n/2 \rceil$ elements independently by a pair of smaller SNs, then applying an $\lfloor n/2 \rfloor$-by-$\lceil n/2 \rceil$ MN. The general structure of thus constructed SN is shown in Fig. 4.6. In this way an SN can be designed iteratively for any n, provided that the SNs with $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ inputs are available and we know how to build a $\lfloor n/2 \rfloor$-by-$\lceil n/2 \rceil$ MN. It is important to note that the circuits $T^{\lfloor n/2 \rfloor}$ and $T^{\lceil n/2 \rceil}$ can be realized using any method.

This set of basic building blocks — two 4-input SNs connected to the 4-by-4 MN — is distinguished in the 8-input SN from Fig. 4.5, which also explains the principle of recursive construction of the MN. The 4-by-4 MN is built of a pair of 2-by-2 MNs and three output comparators. The first (odd) MN formed by three comparators K_7, K_8, and K_9 is applied to the odd-weighted outputs of the two circuits T^4. The second (even) MN formed by three comparators K_4, K_5, and K_6 is applied to the even-weighted outputs of the two circuits T^4. Then the first line of the even MN and the second line of the odd MN are connected to the first output comparator K_1, and so on. Similarly, two input comparators in the 4-input SN, which are distinguished in Fig. 4.5, can be seen in this context as two circuits T^2.

Basically, any realization of the circuit T^n can be used as a basic building block in any encoder and checker presented here. However, we have shown in [143], [148], and in the previous subsection that the circuit T^n realized as a special implementation of an n-input SN offers the best performance in terms of complexity and the number of tests. Therefore in all parameter estimations given hereafter, we assume those of the most efficient circuit T^n implemented as an SN, unless stated otherwise. The complexity characteristics of the T^n circuit realized as a Batcher's odd-even MN are the following:

$$Ga(T^n) = \frac{1}{2}n(\lceil \log_2^2 n \rceil - \lceil \log_2 n \rceil + 4) - 2 \quad \text{and} \qquad (4.4)$$

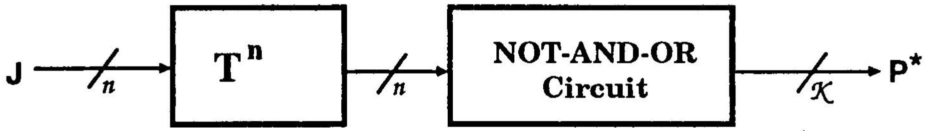


Figure 4.7. General structure of various T^n-based circuitry supporting the use of systematic UEDCs

$$L(T^n) = \frac{1}{2}\lceil \log_2 n \rceil (\lceil \log_2 n \rceil + 1). \tag{4.5}$$

The size of the minimal complete test set (complete means that it contains tests of all $n+1$ weights) that detects all single s/z faults is

$$|\mathrm{T}(T^n)| = n + \lceil n/2 \rceil . \tag{4.6}$$

The procedure of generating the minimal test set for any Batcher's SN and some other special SN designs can be found in [148].

Finally, it is worth to point out that, besides superb performance, the SNs have the following structural advantage, important for VLSI implementation:

- The SNs have a uniform regular structure using only one type of a simple cell;
- Any gate in the SN has both fan-in and fan-out equal two; and
- The SNs can be easily pipelined on a low level.

4.2.3. T^n-Based Counter of 1s

A. Design of the T^n-Based Counter of 1s

The general structure of the T^n-based counter of 1s as well as any encoder with simple or complemented outputs is shown in Fig. 4.7.

The following two special classes of threshold functions will be used to generate the functions of the circuit proposed in Fig. 4.7.

(F1) $\overline{T_i^n}$ — it is 1 if and only if less than i out of n inputs are 1s. In particular, $\overline{T_1^n}$ is 1 when all n inputs are 0s.

(F2) $T_i^n \cdot \overline{T_j^n}$ $(i < j)$ — it is 1 if and only if at least i but less than j out of n inputs are 1s.

Example 4.2. *Design the T^n-based 9-input counter of 1s. The basic block is the circuit T^9 that can be realized by using any known method. Let the outputs of the*

NOT-AND-OR circuit be denoted $\{\overline{s_3}, \overline{s_2}, \overline{s_1}, \overline{s_0}\}$. *The first and the third columns of Table 4.5 define the functions of the NOT-AND-OR circuit in terms of* T_j^9. *Observe the following:*

- $\overline{s_3} = 1$ *for any input* X *of weight at least eight;*
- $\overline{s_2} = 1$ *for any input* X *of weight at least four but less than eight;*
- $\overline{s_1} = 1$ *for any input* X *of weight either at least two but less than four or at least six but less than eight;*
- $\overline{s_0} = 1$ *for any input* X *of weight that is an odd number.*

The above description easily translates to the following functions expressed in terms of the special threshold functions (F1) and (F2):

$$\begin{aligned}
\overline{s_3} &= T_8^9 \\
\overline{s_2} &= T_4^9 \cdot \overline{T}_8^9 \\
\overline{s_1} &= T_2^9 \cdot \overline{T}_4^9 + T_6^9 \cdot \overline{T}_8^9 \\
\overline{s_0} &= T_1^9 \cdot \overline{T}_2^9 + T_3^9 \cdot \overline{T}_4^9 + T_5^9 \cdot \overline{T}_6^9 + T_7^9 \cdot \overline{T}_8^9 + T_9^9
\end{aligned}$$

Only the threshold functions T_i^9 *with even* i *occur both in the complemented and uncomplemented form. The threshold circuit* T^9 *realized as a 9-input SN requires 50 2-input gates, has 8 gate levels, and its complete test set which detects all single s/z faults consists of 14 tests (it will be shown below that these tests also detect all single s/z faults in the NOT-AND-OR circuit). Thus the whole* T^9*-based counter of 1s is built of 63 gates with a total of 125 inputs and has 11 levels. Below we shall show that it needs only 14 tests. This compares favorably with the parallel (9;4) counter whose measures are: 68 gates, a total of 121 inputs, 10 levels, and 10 tests (here it is assumed that an FA is built of 12 gates in three levels and an HA is built of 4 gates in two levels).* □

The NOT-AND-OR network in the T^n-based counter of 1s is built of up to: $\lfloor n/2 \rfloor$ inverters, $2^q - q - 1$ 2-input AND gates, and up to q OR gates with a total of up to $2^q - 2$ inputs. In a more general case of any T^n-based circuit from Fig. 4.7, the complexity of the NOT-AND-OR circuit that is fed by a circuit T^n can be found by inspection of the columns s_j of the encoding table in the following way:

(i) The total of AND gates producing terms $T_j^n \cdot \overline{T}_{j+i}^n$ equals the number of strings of 1s preceded and followed by a 0; and

(ii) The total of inputs T_j^n (or $\overline{T}_{j+i}^n$) to the above OR gates equals the number of strings of 1s (0s) not followed (preceded) by a 0 (1).

Table 4.5. Encoding table of the Berger code $C_{(9;4)}$

Weight of the Information Part	Check Part s_3 s_2 s_1 s_0	Complemented Check Part $\bar{s}_3$ $\bar{s}_2$ $\bar{s}_1$ $\bar{s}_0$
0	1 1 1 1	0 0 0 0
1	1 1 1 0	0 0 0 1
2	1 1 0 1	0 0 1 0
3	1 1 0 0	0 0 1 1
4	1 0 1 1	0 1 0 0
5	1 0 1 0	0 1 0 1
6	1 0 0 1	0 1 1 0
7	1 0 0 0	0 1 1 1
8	0 1 1 1	1 0 0 0
9	0 1 1 0	1 0 0 1

B. Testing of the T^n-Based Counter of 1s

First of all, note that there are only $n+1$ output patterns $(T^n_1 T^n_2 \ldots T^n_n)$ that can be generated by a fault-free circuit T^n, each containing a string of 1s on the leftmost bits: (00...0), (10...0), (110...0), ..., (11...10), (11...1). Naturally, these patterns are generated by applying inputs from the minimal complete test set for the circuit T^n. These patterns must suffice to detect any single s/z fault in the NOT-AND-OR circuit. This precludes its PLA or ROM implementation for use in an STC, since, generally, either must be tested exhaustively for most likely faults, unless it is modified for better testability.

Theorem 4.1. *The $n+1$ patterns generated by the circuit T^n during normal functioning detect all single s/z faults in a gate level irredundant realization of the NOT-AND-OR circuit.*

Proof. Consider a circuit that realizes the function

$$\overline{s_k} = \ldots + T^n_a \cdot \overline{T^n_{a+b}} + T^n_i \cdot \overline{T^n_{i+j}} + T^n_u \cdot \overline{T^n_{u+v}} + \ldots,$$

where $\ldots < a < a+b < i < i+j < u < u+v < \ldots$. Consider the faults affecting the AND gate $T^n_i \cdot \overline{T^n_{i+j}}$ and the OR gate $\overline{s_k}$.

Any fault of the type: (a) input of an inverter s/1 or output of an inverter s/0; (b) any s/0 fault of the AND gate (including its output); and (c) the output s/0 fault of the OR gate, are detected by any input X of weight w such that $i \leq w \leq i+j$.

Any fault of the type: (a) input of an inverter s/0 or output of an inverter s/1; (b) the s/1 fault of the input $\overline{T}^n_{i+j}$ of the AND gate; (c) the s/1 fault on the input line $T^n_i \cdot \overline{T}^n_{i+j}$ to the OR gate; and (d) the output s/1 fault of the OR gate, are detected by any input X of weight $w = i + j$.

Finally, the s/1 fault of the input line T^n_i to the AND gate is detected by any input X of weight $a \leq w < a + b$. Similar argument applies in trivial cases when the OR gate is fed directly by some T^n_i or $\overline{T}^n_i$ signal. □

The above theorem implies that the T^n-based counter of 1s is tested for all single s/z faults by the complete test set for the circuit T^n.

In the forthcoming sections we shall show that the recognition of the equivalence between special realizations of SNs and multi-output threshold circuits revealed by us in [143], [148], has made the counter of 1s implemented with the circuit T^n attractive as a basic building block for the encoders and the STCs for Berger and other UEDCs, in particular, for small n. The more so, that the functions of the encoders are easily generated for an arbitrarily complex check bits assignment, which is not the case for the circuits using a parallel counter.

5. STCs FOR m/n CODES

In this section the STCs for the following classes of m/n codes will be subsequently presented: 1) $m/2m$, $m \geq 2$; 2) $m/(2m+1)$, $m \geq 3$; 3) $2/n$, $n \geq 5$; and 4) $1/n$, $n \geq 7$. The selection of m/n codes for which the STCs will be presented here resulted from their high potential for practical applications. The $m/2m$ and $m/(2m+1)$ codes are optimal unordered codes — they offer maximum of unordered codewords for a given codeword length n. The $2/n$ codes are particularly well suited for state assignment of sequential circuits, since they minimize a total of gate inputs in the 2-level AND-OR excitation circuitry, due to small fan-in of all AND gates. Also, the STCs for $2/n$ codes are essential building blocks for the least complex implementations of STCs for $1/n$ codes, also given here. The $1/n$ codes occur very frequently, not only in FT systems, e.g. at the outputs of decoders, control circuits, etc.

A critical survey of the STCs for each of the above classes of m/n codes will be given in the appropriate subsection. The most efficient STCs for most m/n codes with $m \geq 3$ and $2m + 1 \leq n \leq 4m$ we have given in [136], [137], and for other m/n codes with $m \geq 3$ and $n > 4m$ in [154]. The design of PLA-based STCs for m/n codes have been proposed by many authors [30], [130], [163], [194], and [209]. The most efficient designs were presented by us in [149], [153], [154]. However, the PLA-based STCs are not considered here, as they pose essentially different more difficult design problems, since it is necessary to consider not only stuck-at faults but also faults which are typical for PLAs, such as crosspoint faults and bridging of adjacent lines. Yet another extended fault model is needed for the transistor-level STCs (for MOS technologies) which have been proposed in [58], [69], [70], [86], and [133]. The survey of various STCs for the m/n codes and a more complete bibliography can be found e.g. in [136], [179].

5.1. STCs for $m/2m$ Codes

The most important random logic designs of the STCs for $m/2m$ codes were given in: [2], [49], [133], [143], [147], [167], and [185]. The least complex currently known $m/2m$ code checker for most m of practical value can be realized according to the general functions derived by Anderson and Metze in 1973 [2] as follows.

Algorithm 5.1.

1. Partition the set of $n = 2m$ input bits into two disjoint subsets A_1 and A_2 of size $n_{a_1} = n_{a_2} = m$.
2. Derive the functions $T_i^{n_{a_1}}$ and $T_i^{n_{a_2}}$, $1 \leq i \leq m$, of the threshold circuits $T^{n_{a_1}}$ and $T^{n_{a_2}}$.
3. Generate the functions h_1 and h_2 of the STC for the $m/2m$ code:

$$h_1 = \sum_{i=0}^{m} T_i^{n_{a_1}} \cdot T_{m-i}^{n_{a_2}}, \quad \text{i odd} \tag{5.1}$$

$$h_2 = \sum_{i=0}^{m} T_i^{n_{a_1}} \cdot T_{m-i}^{n_{a_2}}, \quad \text{i even} \tag{5.2}$$

An STC given by (5.1) and (5.2) has the following parameters:

$$Ga(m/2m) = 2 \cdot Ga(T^m) + m + 1; \tag{5.3}$$

$$In(m/2m) = 2 \cdot In(T^m) + 2m - 1; \tag{5.4}$$

$$L(m/2m) = L(T^m) + 2; \tag{5.5}$$

$$|\mathbf{T}(m/2m)| \geq |\mathbf{T}(T^m)|, \tag{5.6}$$

where $Ga(T^m)$ etc. are the parameters of the threshold circuits used.

In 1977, Smith [185] proposed another design which offerred the best performance for any $m \geq 3$. Its major part consists of $2m(m-1)$ cells (the same as used in SNs, shown in Fig. 4.4(B)) organized in $m-1$ levels, whereas the output part consists of a pair of m-input AND and OR gates. It was shown tested by only $2m$ codewords.

Clearly, the complexity of the checker that realizes (5.1) and (5.2) strongly depends on the complexity of the circuits $T^{n_{a_1}}$ and $T^{n_{a_2}}$ (i.e. T^m). Only recently (1990) our recognition of the optimality of the circuits T^n realized as SNs [143], [148] has made possible to outperform the checker proposed by Smith. The numerical values (for $m \leq 16$) of the parameters of a threshold circuit T^m can be taken from Table 4.3, whereas for $m > 16$ they can be derived from the formulae (4.4)–(4.6). The minimal test set $\mathbf{T}(m/2m)$ can be constructed by concatenating the elements of the sets $\mathbf{T}(T^{n_{a_1}})$ and $\mathbf{T}(T^{n_{a_2}})$ to form the vectors of the form $X = (X'X'')$, where $X \in \mathbf{T}(m/2m)$, in such a way that: 1) $X \in \mathbf{T}(T^{n_{a_1}})$ and $X'' \in \mathbf{T}(T^{n_{a_2}})$; 2) $w(X') + w(X'') = m$; and 3) all the vectors from $\mathbf{T}(T^{n_{a_1}})$ and from $\mathbf{T}(T^{n_{a_2}})$ appear exactly once in some vector from $\mathbf{T}(m/2m)$ ($w(X)$ denotes the weight of a vector X). It can be shown that $|\mathbf{T}(m/2m)| = \frac{3}{2}m$ for m even and $|\mathbf{T}(m/2m)| = 2m$ for m odd. The larger number of tests for odd m results from unfavorable weight distribution of the tests which form the complete minimal test set for Batcher's SNs T^m with m odd.

In [147] we have shown that partitioning of the set of input bits into $k \geq 3$ blocks of approximately the same size may lead to further savings for small m. Let $IP(m,n) = \{n_{a_1}, n_{a_2}, \ldots, n_{a_k}\}$ denote the input partition of the set A of n input bits into k blocks A_j, $1 \leq j \leq k$, such that each block A_j contains n_{a_j} bits ($1 \leq n_{a_j} \leq m$) and $\sum_{i=1}^{k} n_{a_j} = n$. Without loss in generality, assume that $n_{a_1} \leq n_{a_2} \leq \ldots \leq n_{a_k}$ and $A_1 = \{x_1, x_2, \ldots, x_{n_{a_1}}\}$, $A_2 = \{x_{n_{a_1}+1}, x_{n_{a_1}+2}, \ldots, x_{n_{a_1}+n_{a_2}}\}$, etc. Given the input partition $IP(m,n) = \{n_{a_1}, n_{a_2}, \ldots, n_{a_k}\}$, the set $C(m/n)$ of all m/n codewords can be partitioned into some π disjoint blocks, where each block, denoted by $(j_1 j_2 \ldots j_k)$, contains all m/n codewords that have exactly j_i 1s on the input bits from A_i, $1 \leq i \leq m$, i.e.

$$C(m/n) = \bigcup_{\{j_1+j_2+\cdots+j_k=m\}} (j_1 j_2 \ldots j_k). \tag{5.7}$$

Provided that the set $C(m/n)$ given by (5.7) is properly partitioned into two proper subsets C_1 and C_2 (to preserve both the ST and CD properties), the checker functions may have the following general form:

$$h_r = \sum_{(j_1 j_2 \ldots j_k) \in C_r} T_{j_1}^{n_{a_1}} T_{j_2}^{n_{a_2}} \ldots T_{j_k}^{n_{a_k}}, \qquad r \in \{1,2\}. \tag{5.8}$$

Obviously, the functions $T_{j_l}^{n_{a_l}}$ ($1 \leq l \leq k$) can be realized by any version of circuit T^n. If n_{a_l} is small, the 2-level circuits T^{n_l} allow us to build fast 4-level STCs for some $m/2m$ codes. The partitions C_1, C_2 of $m/2m$ codes, some of which result in simpler checkers, which we have derived in [147], are included in Table 5.1. The don't care products listed in the last column can be included in an arbitrarily

Table 5.1. Optimal product partitions for the fast STCs for the m/2m and m/(2m+1) codes

Code	Input Partition	C_1	C_2	Don't Care Products
3/6	{2,2,2}	(210) (102) (021)	(201) (120) (012)	(111)
4/8	{2,3,3}	(220) (202) (112) (031) (013)	(211) (130) (103) (022)	(121)
5/10	{3,3,4}	(311) (230) (203) (122) (113) (032) (014)	(320) (302) (221) (212) (131) (104)	none
6/12	{4,4,4}	(420) (402) (321) (312) (240) (204) (132) (123) (042) (024)	(411) (330) (303) (231) (222) (213) (141) (114) (033)	none
7/14	{4,5,5}	(430) (412) (331) (322) (304) (250) (223) (142) (115) (052) (034)	(421) (403) (340) (313) (232) (205) (151) (124) (043) (025)	(241) (214) (133)
3/7	{1,2,2,2}	(1200) (1020) (1002) (0201) (0120) (0111) (0012)	(1110) (1101) (1011) (0210) (0102) (0021)	none
4/9	{2,2,2,3}	(2200) (2020) (2002) (1210) (1111) (1102) (1012) (0220) (0202) (0121) (0013)	(2110) (2101) (2011) (1201) (1120) (1021) (1003) (0211) (0112) (0103) (0022)	none
5/11	{2,3,3,3}	(2300) (2120) (2111) (2102) (2012) (1301) (1220) (1121) (1103) (1031) (1013) (0320) (0302) (0212) (0131) (0023)	(2210) (2201) (2030) (2021) (2003) (1310) (1211) (1202) (1130) (1112) (1022) (0311) (0230) (0221) (0203) (0113) (0032)	(0122)
6/13	{3,3,3,4}	(3201) (3120) (3111) (3021) (3003) (2310) (2301) (2220) (2103) (2031) (2013) (1311) (1230) (1212) (1203) (1122) (1032) (1014) (0321) (0303) (0231) (0213) (0114) (0033)	(3300) (3210) (3102) (3030) (3012) (2211) (2202) (2130) (2121) (2112) (2022) (2004) (1320) (1302) (1221) (1131) (1113) (1104) (1023) (0330) (0312) (0222) (0204) (0132) (0123) (0024)	none

selected set C_i. It was shown that these checkers use the fewest number of gates for $3 \leq m \leq 6$ and are faster than any other checkers. For $m > 6$, the STCs given by (5.1) and (5.2) and built using SNs offer better performance. The minimal test set for all checkers specified in Table 5.1 can be generated as described in [147].

Finally, it is worth to mention the STCs for $m/2m$ codes from [133], which are built of a pair of m-input modified parallel counters followed by an STC for the K-pair 2-rail code using m out of 2^K combinations, where $K = \lceil \log_2 m \rceil$. The latter STC is built of about $2(m-K)$ FAs and $K-1$ 2-pair 2-rail STCs with about $St_P(m) \cdot t_{FA} + 2K$ gate levels (for $St_P(m)$ see Table 4.1). Asymptotically, it is the least complex and the fastest STC for an $m/2m$ code: it is built of $O(m)$ gates with $O(\log m)$ gate levels which is less than $O(m \log^2 m)$ gates and $O(\log^2 m)$ gate levels in the best available T^n-based checker. However, the complexity parameters of the checkers from [133] involve very large constants which make them inferior for most m of practical value (e.g. for any $m \leq 32$).

5.2. STCs for $m/(2m+1)$ Codes

The most important random logic designs of the STCs for $m/(2m+1)$ codes were given in: [2], [49], [54], [98], [136], [137], [147], [178], and [204]. The fewest number of gates amongst them use our checkers from [147] — for the 3/7 and 4/9 codes, and from [136], [137] — for $m \geq 5$. It is worth to point out that all checkers from [147] can be implemented with only 3 or 4 levels, unless fan-in limitations are taken into account.

The fast STCs from [147] assume the same general form (5.8) as already given above for the $m/2m$ codes. The partitions of practical importance, which were derived in [147], are included in Table 5.1.

A set of simplified functions for the STCs for $m/(2m+1)$ codes can be derived from those in [136], [137] as follows. The set of $2m+1$ input bits $\{x_1, \ldots, x_m, x_{m+1}, \ldots, \ x_{2m+1}\}$ is partitioned into two disjoint subsets A_1 and A_2 with $n_{a_1} = m$ and $n_{a_2} = m+1$ bits, respectively. Additionally, the set A_2 is partitioned into two disjoint subsets $A_{2,1}$ and $A_{2,2}$ with $n_{a_{2,1}} = \lfloor (m+1)/2 \rfloor$ and $n_{a_{2,2}} = \lceil (m+1)/2 \rceil$ bits, respectively. Now an STC consists of the ST/CD translator of the $m/(2m+1)$ code into the incomplete 2/4 code (the codeword $(h_1^1 h_2^1 h_3^1 h_4^1) = (0011)$ is not used) and the well known STC for the 2/4 code with $(h_1^1 h_2^1 h_3^1 h_4^1)$ as inputs (assume $m = 2$ in (5.1) and (5.2)). The translator realizes the following functions:

$$h_1^1 = T^m_{\lfloor m/2 \rfloor} + T^{n_{a_{2,1}}}_{n_{a_{2,1}}} T^{n_{a_{2,2}}}_{n_{a_{2,2}}-1} \tag{5.9}$$

$$h_2 = T^{m+1}_{\lceil m/2 \rceil} + T^{n_{a_{2,1}}}_{n_{a_{2,1}}-1} T^{n_{a_{2,2}}}_{n_{a_{2,2}}} \tag{5.10}$$

$$h_3^1 = \sum_{i=1,\, i\, odd}^{m-1} T_i^m T_{m-i}^{m+1} + \sum_{j=1,\, i\, odd}^{m-1} T_{m-j}^m T_j^{m+1} \tag{5.11}$$

$$h_4^1 = \sum_{i=1,\, i\, even}^{m-1} T_i^m T_{m-i}^{m+1} + \sum_{j=1,\, i\, even}^{m-1} T_{m-j}^m T_j^{m+1} \tag{5.12}$$

The realizations which are even less complex than those that we have reported in [136], [137] (which use completely bifurcated circuits T^n) result when the circuits T^m and T^{m+1} are realized as SNs (in the circuit T^{m+1}, unused cell that realizes the functions T^{m+1}_m and T^{m+1}_{m+1} can be neglected). It can be easily shown that a checker given by (5.9)–(5.12) has the following parameters now:

$$Ga(m/(2m+1)) = Ga(T^m) + Ga(T^{m+1}) + 8; \tag{5.13}$$

$$In(m/(2m+1)) = 2 \cdot In(T^m) + 2m + 11; \tag{5.14}$$

$$L(m/(2m+1)) = L(T^{m+1}) + 4; \tag{5.15}$$

$$|\mathbf{T}(m/(2m+1))| \;\geq\; \max\{2m+1, |\mathbf{T}(T^{m+1})| - 1\}. \tag{5.16}$$

The minimal test set $\mathbf{T}(m/(2m+1))$ can be derived similarly as for an STC for the $m/2m$ codes; see also [136], [137].

5.3. STCs for $2/n$ Codes

The STCs for $2/n$ codes ($n \geq 5$) can be designed by using the methods from [2], [98], [136], [138], and [204]. The least complex checker, proposed by us in [136], [138], consists of t ($t \geq 2$) blocks which perform the following translations:

$$2/n \xmapsto{\mathbf{H^1}} 2/n_1 \xmapsto{\mathbf{H^2}} 2/n_2 \xmapsto{\mathbf{H^3}} \ldots 2/n_{t-2} \xmapsto{\mathbf{H^{t-1}}} 2/4 \xmapsto{\mathbf{H^t}} 1/2,$$

where $n > n_1 > n_2 > \ldots > n_{t-2} > 4$. The following algorithm can be used to generate the functions of the STC for any $2/n$ code, $n \geq 5$.

Algorithm 5.2.

Initially, $A = \{x_1, x_2, \ldots, x_n\}$, and $n_0 = n$, and $i = 1$.

1. Find s_i such that the following conditions are fulfilled: $W_1) \div W_4)$ — if $i = 1$, and $W_1) \div W_5)$ — if $i > 1$, where:
   $W_1)$ $2 \leq s_i \leq n_{i-1} - 2$;
   $W_2)$ $r_i = \lceil \log_2(n_{i-1}/s_i) \rceil \leq \lceil n_i/2 \rceil$;
   $W_3)$ $s_i + r_i = n_i < n_{i-1}$;
   $W_4)$ $b_i = \lfloor (n_{i-1}/s_i) \rfloor \neq 2$ or $c_i = \lceil (n_{i-1}/s_i) \rceil \neq 3$;
   $W_5)$ if $r_i = 1$ and $b_i = 1$ and $r_{i-1} > 2$ then $r_{i-1} \leq v_i + 1$,
   where: $v_i = n_i - s_i \cdot b_i$ — if $b_i \neq c_i$, or $v_i = s_i$ — if $b_i = c_i$.
2. Partition A into s_i disjoint subsets $A^{(j)}$ of size: b_i — for $1 \leq j \leq s_i - v_i$, and c_i — for $s_i - v_i + 1 \leq j \leq s_i$, such that the ordering of input bits and of subsets $A^{(j)}$ is preserved according to the increasing indices.
3. Derive the functions $h^i_j = \sum_{h^{i-1}_u \in A^{(j)}} h^{i-1}_u$, $1 \leq j \leq s_i$, in the form which allows for a tree realization using 2-input OR gates. Note that if $i = 1$, then $h^{i-1}_u = x_u$, for $1 \leq u \leq n$.
4. Derive the functions $h^i_{s_i+p}$, $1 \leq p \leq r_i$, in the form:

$$h^i_{s_i+p} = \sum_{j=1} \sum_{\substack{k=2l+1 \\ 0 \leq l \leq 2^{p-1}}} \left(\sum_{h^{i-1}_u \in A^{(j)}_{p,k}} h^{i-1}_u \right) \cdot \left(\sum_{h^{i-1}_v \in A^{(j)}_{p,k+1}} h^{i-1}_v \right), \tag{5.17}$$

where $A^{(j)}_{p,k}$, $A^{(j)}_{p,k+1}$ are disjoint subsets having respectively $\lfloor|A^{(j)}_{p-1,(k+1)/2}|/2\rfloor$ and $\lceil|A^{(j)}_{p-1,(k+1)/2}|/2\rceil$ elements of the set $A^{(j)}_{p-1,(k+1)/2}$, which are obtained as a result of the p-th partition of the set $A^{(j)}$.

5. If $b_i = 2^l$ and $c_i = 2^l + 1$ for $l = 2, 3, 4, \ldots$ then any product of bits from $A^{(j)}$ transfer from $h^i_{n_{i-1}}$ to $h^i_{n_i}$ for each j, $1 \leq j \leq s_i - v_i$.
6. If $i > 1$ and $r_{i-1} > 1$, modify the functions h^i_l, $1 \leq l \leq n_i$, as follows: swap the indices in each of $\lfloor r_{i-1}/2\rfloor$ pairs of input bits $h^{i-1}_{s_{i-1}-2l}$ and $h^{i-1}_{n_{i-1}-2l}$, where $j = 2l+1$ ($j = 2l$) and $0 \leq l \leq \lfloor r_{i-1}/2\rfloor - 1$ ($1 \leq l \leq \lfloor r_{i-1}/2\rfloor$) if r_{i-1} even (odd).
7. Implement the circuit $\mathbf{H^1}$ in such a way that if $r_i > 1$, then the subcircuits realizing the functions h^i_j, $1 \leq j \leq n_i - 1$, use shared OR gates wherever possible.
8. If $n_i > 4$, assume a new set of input variables $A = \{h^i_j \ : \ 1 \leq j \leq n_i\}$, increment parameter i by 1, and return to Step 1.
9. Implement the circuit $\mathbf{H^t}$ ($t = i+1$) — the STC for the 2/4 code — given by the functions:

$$\begin{aligned} h^t_1 &= \left(h^i_1 + h^i_3\right)\left(h^i_2 + h^i_4\right), \\ h^t_2 &= h^i_1 h^i_3 + h^i_2 h^i_4. \end{aligned}$$

□

In the above algorithm, each of $t-1$ executions of Steps 1–7 generates the logic functions of a block $\mathbf{H^i}$, $1 \leq i \leq t-1$. During Steps 2–4, the set $A^{(i-1)}$ of input bits of the block H^i is partitioned on the disjoint subsets and the functions h^i_j are generated in such a manner that shared OR gates can be used systematically. If the conditions W_2), W_4), and W_5) hold in Step 1, then the modifications specified in Steps 5 and 6 are feasible in any case. These modifications guarantee that the codewords that appear on the output of the block $\mathbf{H^{i-1}}$ in normal operation (i.e., when no fault is present in the checker and a $2/n$ codeword is applied on the primary inputs) are sufficient for detection of any single s/z fault, $z \in \{0,1\}$, in subsequent blocks $\mathbf{H_j}$, $1 \leq j \leq t$.

The parameters of an STC for the $2/n$ code designed by Algorithm 5.2 are as follows:

$$Ga(2/n) = 2n + 3\left(\sum_{i=1}^{t-1}\lceil\log_2(n_{i-1}/s_i)\rceil\right) - 2, \tag{5.18}$$

$$In(2/n) = 5n + 5\left(\sum_{i=1}^{t-1}\lceil\log_2(n_{i-1}/s_i)\rceil\right) - 8, \tag{5.19}$$

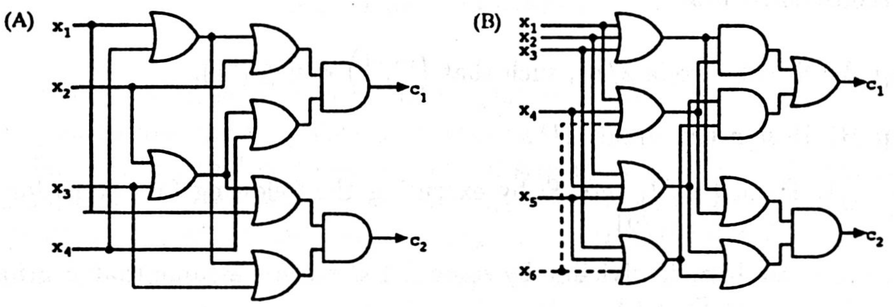


Figure 5.1. The STCs for: 1/4 code (A); 1/5 and 1/6 codes (B)

$$L(2/n) = t + 1 + \left(\sum_{i=1}^{t-1} \lceil \log_2(n_{i-1}/s_i) \rceil \right). \tag{5.20}$$

The test set sufficient for the detection of any s/z fault, $z \in \{0,1\}$, in the circuit generated by Algorithm 5.2 consists of n circular shifts of two 1s followed by $n-2$ 0s. Depending on the choice of a pair of parameters (s_i, r_i), Algorithm 5.2 may provide a number of different versions of the circuit for any $n \geq 6$ [58].

5.4. STCs for $1/n$ Codes

Several design methods for the STCs for $1/n$ codes have been proposed: [2], [77], [103], [113], [136], [138], [161], [162], and [167]. The least complex STCs for $1/n$ codes with $4 \leq n \leq 6$ were proposed by Rabara [161] ($n = 4$) and by Anderson and Metze [2] ($n = 5$ and 6) — they are shown in Fig. 5.1. Several fast 3-level and 4-level STCs for many $1/n$ codes can be designed by using the methods from [161], [162], and [77]. However, the least complex checkers — in terms of the number of gate inputs — for most $n \geq 7$ can be designed by using our algorithm from [136], [138]. The checker has the general cascade structure:

$$1/n \overset{\mathbf{H^1}}{\longmapsto} 2/n_1 \overset{\mathbf{H^2}}{\longmapsto} 1/2.$$

$\mathbf{H^1}$ is an ST/CD translator which converts a $1/n$ code into a set H^1, H^1 being a subset of the $2/n_1$ code, and such that $|H^1| = n$. It consists of a single level of n_1 OR gates with a total of $2n$ inputs. $\mathbf{H^2}$ is an STC for the $2/n_1$ code which can be designed by using Alg. 5.2. The design algorithm for the $1/n$ code is as follows.

Algorithm 5.3.

Step A: Select a code $2/n_1$, such that $\binom{n_1-1}{2} < n \leq \binom{n_1}{2}$.

Step B: Design the circuit $\mathbf{H}^1$.

1. Form $\lfloor n_1/2 \rfloor$ sets S_i by executing the following four steps for any i, $1 \leq i \leq \lfloor n_1/2 \rfloor$.
   (a) If n_1 is divisible by some i, $1 < i < n_1$, assume that $p = i$ and go to Step (d).
   (b) If n_1 and i have some $q > 1$ as the smallest common divider, assume $p = q$ and go to Step (d).
   (c) Assume $p = 1$.
   (d) Form an ordered set S_i of pairs (u, v) such that $S_i = \bigcup_{k=1}^{p} S_{i,k}$, where $S_{i,k} = ([k + l \cdot i] \bmod n_1,\ [k + (l+1) \cdot i] \bmod n_1)$ with $l = 0$ — if $i = \lfloor n_1/2 \rfloor$, and $0 \leq l \leq \lfloor n_1/p \rfloor - 1$ — in other cases.
2. Allocate any input variable x_k, $1 \leq k \leq n$, to a pair of sets A_u and A_v as follows: if $(i-1) \cdot n_1 < j \leq i \cdot n_1$ then $x_j \in A_u$ and $x_j \in A_v$, where (u, v) is a w-th pair in the set S_i, $w = j - (i-1) \cdot n_1$.
3. Formulate the functions realized by the circuit $\mathbf{H}^1$ in the form:

$$h_u^1 = \sum_{x_j \in A_u} x_j, \quad 1 \leq u \leq n_1.$$

Step C: Design the circuit $\mathbf{H}^2$ by using Algorithm 5.2. □

The checker obtained from Alg. 5.3 has the following parameters:

$$Ga(1/n) = n_1 + Ga(2/n_1), \tag{5.21}$$

$$In(1/n) = 2n + In(2/n_1). \tag{5.22}$$

No other checker has the number of gate inputs that grows linearly as $O(2n)$. For instance, the other cost-efficient STCs for $1/n$ codes from [161], [162], [77] have at least $O(4n)$ gate inputs. The test set for any STC for the $1/n$ code consists of all n codewords. Note also that the least complex PLA-based STCs for $1/n$ codes given in [130] and [152] also use the translation of the $1/n$ code into the concatenated code $C_{1/n_1} \times C_{1/n_2}$ which is a special case of the $2/(n_1 + n_2)$ code.

5.5. Final Remarks

An encoder for most m/n codes is, in general, a complex circuit [78]. Nevertheless, there are many applications wherein systematic encoding of data is immaterial, e.g. internal state assignment of sequential circuits. The design of SC combinational circuits using m/n codes has been considered in [32], [35], [79], [139], [140], and [149]. Several implementations of fail-safe or SC synchronous sequential circuits and self-checking microprogrammed control units have been protected by the m/n codes: [24], [35], [37], [48], [56], [78], [95], [132], [191], [196], [199], [207], and [211]. The fail-safe and SC asynchronous sequential circuits have been proposed in [112], [126], [176]. The m/n codes have been found useful to protect highly-structured arrays, such as PLAs and ROMs: [94], [140], [149], [163], and [188]. The applications of the m/n codes in commercial FT computers include the Bell Laboratories' ESS processor [24], [78], [191], [199], and Harris computers [63].

6. STCs FOR BORDEN CODES

STCs for Borden code $C(n,t)$ were proposed only in recent years in: [47], [59], [66], [125], and [146]. The STCs from [66] and [125], which are built using a number of STCs for m/n codes, are excessively complex and have highly irregular internal structure. Additionally, the STC from [66] has a limitation that the all-0s and all-1s vectors are excluded from Borden codes which include them, since an STC does not exist for a $0/n$ code or n/n code. The STCs from [47] were claimed significantly less complex than the checkers from [66], but the complexity of these checkers grows quickly with t. For instance, for $t \geq 4$, they use modules composed of multi-input gates which are also difficult to test and to implement in MOS technologies. The most recent checkers from [59] which are built using $2k$ parallel counters, are very efficient, but they can be built for a very narrow range of n and t limited to $t = 2^m - 2$ $(m \geq 2)$ and $n = 2k(t+1)$ $(k \geq 1)$, which include: $n = 6, 12, 18, 24, \ldots$ — for $t = 2$, and $n = 14, 28, 42, 56, \ldots$ — for $t = 6$. Here we shall present the extension of our earlier results from [146] on designing STCs for all Borden codes by using circuits T^n. These new checkers not only have no limitations reported above but also have significantly better performance than all known designs (except some checkers from [59]. Also, they are the easiest to design, as we have managed here to derive their logic functions for any Borden code.

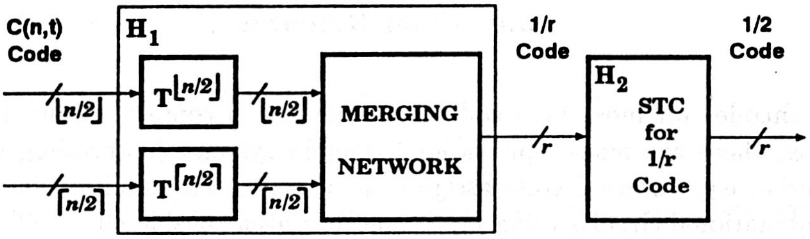


Figure 6.1. The new STC for Borden code C(n,t)

6.1. General Structure of an STC

An STC considered here (Fig. 6.1) is built using two basic blocks: $\mathbf{H_1}$, an ST/CD translator of the $C(n,t)$ code into the $1/r$ code, $r \geq 2$, and — if $r > 2$ — $\mathbf{H_2}$, an STC for the $1/r$ code. If the input to $\mathbf{H_1}$ is not a Borden codeword, then the output generated by $\mathbf{H_1}$ is not a $1/r$ codeword and $\mathbf{H_2}$ signals the input error conditions by generating a non-1/2 code output. The direct translation into the 1/2 code will be shown feasible only for some special cases of n and t (Table 6.2). Although the translation of any other Borden code into the 1/3 code will be shown possible, it should rather be avoided, because an STC for the 1/3 code is difficult to implement [51]. To reduce the amount of hardware, r should be kept as small as possible and therefore $r = 4$ is the best choice. The logic scheme of the least complex STC for the 1/4 code from [161] was shown in Fig. 5.1(A). Clearly, the proper realization of the circuit $\mathbf{H_1}$ is the only design problem.

First, we shall show how to partition the input space of the checker into regular subsets which allows for a straightforward construction of $\mathbf{H_1}$ by using two threshold circuits $T^{\lfloor n/2 \rfloor}$ and $T^{\lceil n/2 \rceil}$. Then the partitioning of the Borden code into subsets and the basic functions of the translator are derived. Most part of this section is devoted to the testability analysis of the merging network (MN). Finally, the general procedure of designing an ST/CD $\mathbf{H_1}$ is given and its logic functions are derived. The theory presented here is illustrated with an example of designing an STC for the $C(8,2)$ code.

The checker from Fig. 6.1 is constructed on the basis of a special partitioning of the input space $\mathbf{X}$ of the checker (i.e. the set of all 2^n input n-tuples) into regular subsets defined as follows.

First, the set of input variables $I = \{x_1, x_2, \ldots, x_n\}$ is partitioned into a pair of subsets $A = \{x_1, x_2, \ldots, x_{n_a}\}$ and $B = \{x_{n_a+1}, \ldots, x_n\}$ of the size $n_a = \lfloor n/2 \rfloor$ and $n_b = \lceil n/2 \rceil$. Now the set $\mathbf{X}$ can be seen as the union of $(\lfloor n/2 \rfloor + 1)(\lceil n/2 \rceil + 1)$ pairwise disjoint subsets (j, k)

$$\mathbf{X} = \bigcup_{0 \leq j \leq n_a} \bigcup_{0 \leq k \leq n_b} (j,k), \tag{6.1}$$

where (j,k) denotes the subset of $\binom{n_a}{j} \cdot \binom{n_b}{k}$ input n-tuples with exactly j 1s on the bits from A and exactly k 1s on the bits from B.

The set $C_{m/n}$ can now be represented as

$$C_{m/n} = \bigcup_{i=\underline{i}}^{\bar{i}} (i, m-i), \tag{6.2}$$

where $\underline{i} = \max\{0, i - n_b\}$ and $\bar{i} = \min\{m, n_a\}$.

Let $W = \{w : 0 \leq w \leq n \text{ and } w \equiv \lfloor n/2 \rfloor \mod (t+1)\}$ be the set of weights that occur in $C(n,t)$. The $C(n,t)$ code can be partitioned into q subsets

$$C(n,t) = \bigcup_{w \in W} \bigcup_{i=\underline{i}}^{\bar{i}} (i, w-i), \tag{6.3}$$

where

$$\begin{aligned} q &= \sum_{w \in W,\ w \leq n_a} (w+1) + \sum_{w \in W,\ w > n_a} (n - w + 1) \\ &= n + \sum_{w \in W,\ w \leq n_a} w + \sum_{w \in W,\ w > n_a} (n - w). \end{aligned} \tag{6.4}$$

Example 6.1. *Design an STC for the $C(8,2)$ code which has the structure from Fig. 6.1. The set of inputs $I = \{x_1, x_2, \ldots, x_8\}$ is first partitioned into two disjoint subsets $A = \{x_1, x_2, x_3, x_4\}$ and $B = \{x_5, x_6, x_7, x_8\}$. With this partition of I, the input space $\mathbf{X}$ of the checker (i.e. the set of all 2^8 input 8-tuples) can be represented as $\mathbf{X} = \cup_{j=0}^{4} \cup_{k=0}^{4} (j,k)$, the union of 25 disjoint subsets (j,k). Now the $C(8,2)$ code, which is $C(8,2) = C_{1/8} \cup C_{4/8} \cup C_{7/8}$, can be represented as in Eqn. (6.3) as the union of $q = 9$ disjoint subsets (j,k)*

$$C(8,2) = \{(0,1) \cup (1,0)\} \cup \{(0,4) \cup (1,3) \cup (2,2) \cup (3,1) \cup (4,0)\}$$
$$\cup \{(3,4) \cup (4,3)\} .$$

The diagram introduced in Fig. 6.2 is a graphical illustration of this partitioning. The row and column numbers correspond to j and k, respectively. The entries representing subsets (j,k) are marked with 1s and the empty entries represent the subsets (j^e, k^e) of non-codewords. □

A logic function which is 1 for any n-tuple from (j,k) and 0 for any other input can be expressed as $p(j,k) = T_j^{n_a} \cdot \overline{T_{j+1}^{n_a}} \cdot T_k^{n_b} \cdot \overline{T_{k+1}^{n_b}}$. Thus, any logic function

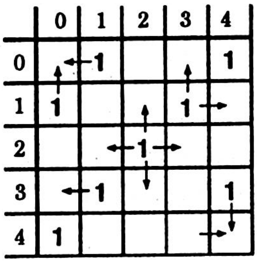


Figure 6.2. Partitioning diagram of the Borden code C(8,2) and identification of untestable s/1 faults

which is 1 for a collection of subsets (j,k) can be easily realized with the help of a pair of circuits T^{n_a} and T^{n_b}. Now suppose that the set $C(n,t) = \bigcup_{j+k\in W}(j,k)$ is partitioned into r nonempty disjoint subsets $\mathbf{Y_i}$, $1 \le i \le r$, where r can assume some value from the interval $1 \le r \le q$. The basic functions of $\mathbf{H_1}$ are

$$y_i = \sum_{(j,k)\in\mathbf{Y_i}} T_j^{n_a} \cdot \overline{T_{j+1}^{n_a}} \cdot T_k^{n_b} \cdot \overline{T_{k+1}^{n_b}}, \quad 1 \le i \le r, \qquad (6.5)$$

where $\sum$ denotes logic OR and, to preserve the generality of the formulae, it is assumed that: (i) $T_j^{n_v} = 0$ if $j > n_v$; (ii) $T_j^{n_v} = 1$ if $j \le 0$; (iii) $\overline{T_j^{n_v}} = 0$ if $j \le 0$; and (iv) $\overline{T_j^{n_v}} = 1$ if $j > n_v$.

A non-ST checker may have $r = 1$, whereas an STC — which is of our concern — must have $r \ge 2$. To reduce the size of $\mathbf{H_2}$, as many products $p(j,k)$ as possible should be OR-ed together. It will be shown that $r \in \{2,3\}$ is feasible for any Borden code.

Example 6.1 (Cont'd). *The partitioning of $C(8,2)$ translates easily to the structure of $\mathbf{H_1}$ built of two threshold circuits T^{4a} and T^{4b} which feed the MN. Table 6.1 lists $q = 9$ basic products $p(j,k)$ which correspond to the legal subsets (j,k).* □

6.2. Code-Disjoint Property

Suppose that $\mathbf{X} = \cup_{j=0}^{n_a} \cup_{k=0}^{n_b} (j,k)$ is partitioned into three subsets:

- $C(n,t)$ — the set of Borden codewords;
- $\mathbf{X_{nc0}}$ — the set of subsets (j^e, k^e) for which all products $p(j,k)$ are 0s and therefore the MN produces the all-0s output; and

Table 6.1. Basic and modified products used in the translator H_1 for the C(8,2) code

(j,k)	$p(j,k)$	$p'(j,k)$
(0,1)	$\overline{T}_1^{4a} \cdot \overset{*}{T_1^{4b}} \cdot \overline{T}_2^{4b}$	$\overline{T}_1^{4a} \cdot \overline{T}_2^{4b}$
(0,4)	$\overline{T}_1^{4a} \cdot T_4^{4b}$	T_4^{4b}
(3,4)	$T_3^{4a} \cdot \overset{*}{\overline{T}_4^{4a}} \cdot T_4^{4b}$	
(1,0)	$\overset{*}{T_1^{4a}} \cdot \overline{T}_2^{4a} \cdot \overline{T}_1^{4b}$	$\overline{T}_1^{4b}$
(4,0)	$T_4^{4a} \cdot \overline{T}_0^{4b}$	
(1,3)	$\overset{*}{T_1^{4a}} \cdot \overline{T}_2^{4a} \cdot T_3^{4b} \cdot \overset{*}{\overline{T}_4^{4b}}$	$T_1^{4a} \cdot \overline{T}_3^{4a} \cdot T_3^{4b}$
(2,2)	$\overset{*}{T_2^{4a}} \cdot \overset{*}{\overline{T}_3^{4a}} \cdot \overset{*}{T_2^{4b}} \cdot \overset{*}{\overline{T}_3^{4b}}$	$T_2^{4a} \cdot \overline{T}_3^{4a}$
(3,1)	$T_3^{4a} \cdot \overset{*}{\overline{T}_4^{4a}} \cdot \overset{*}{T_1^{4b}} \cdot \overline{T}_2^{4b}$	$T_2^{4a} \cdot \overline{T}_4^{4a} \cdot \overline{T}_2^{4b}$
(4,3)	$T_4^{4a} \cdot T_3^{4b} \cdot \overset{*}{\overline{T}_4^{4b}}$	$T_4^{4a} \cdot T_3^{4b}$

An asterisk '*' denotes an input with untestable s/1 fault.

- $\mathbf{X_{nc1}}$ — the set of subsets (j^e, k^e) for which at least two products $p(j,k)$ are 1s, and therefore the MN produces some i/r $(i > 1)$ output.

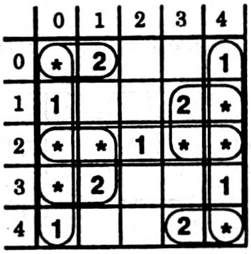
The translator $\mathbf{H_1}$ defined by (6.5) is CD because it generates a $1/r$ codeword for any input from $C(n,t)$ and the all-0s output otherwise, i.e. we have $\mathbf{X_{nc0}} = \{\mathbf{X} \setminus C(n,t)\}$ and $\mathbf{X_{nc1}} = \emptyset$. However, it will be seen in the next subsection that such a circuit is not ST for some s/1 faults on the inputs to certain AND gates. To ensure that $\mathbf{H_1}$ is ST for any single s/z fault, some products $p(j,k)$ in (6.5) will have to be modified in such a way that the set $\mathbf{X_{nc1}}$ becomes nonempty. The modified translator $\mathbf{H_1}$ preserves the CD property, provided that it generates some i/r codeword output, $i > 1$, for any $X^e \in (j^e, k^e)$ and for any $(j^e, k^e) \in \mathbf{X_{nc1}}$. The latter condition is satisfied if

$$\begin{aligned} &[\forall (j^e, k^e) \in \mathbf{X_{nc1}}] \ \left[\exists p'(j_1, k_1) \in y_{i_1},\ p'(j_2, k_2) \in y_{i_2} \text{ with } i_1 \neq i_2\right] \\ &\text{such that } \ [\forall X^e \in (j^e, k^e)] \ \left[p'(j_1, k_1)(X^e) = p'(j_2, k_2)(X^e) = 1\right], \end{aligned} \tag{6.6}$$

where $p'(j_1, k_1)$, $p'(j_2, k_2)$ denote some modified products $p(j_1, k_1)$, $p(j_2, k_2)$.

Example 6.1 (Cont'd). *Examples of modifications that allow us to meet the above requirements are shown in Fig. 6.3, where:*

- *the entries with 1s and 2s correspond to $(j,k) \in C(n,t)$;*
- *the empty entries correspond to $(j^e, k^e) \in \mathbf{X_{nc0}}$; and*



	0	1	2	3	4
0	*	2			1
1	1			2	*
2	*	*	1	*	*
3	*	2			1
4	1			2	*

Figure 6.3. Modification map of the AND gates p(j,k) in the STC for the Borden code C(8,2)

- *the entries with an asterisk '*' correspond to* $(j^e, k^e) \in \mathbf{X_{ncl}}$.

Any square cluster corresponds to some modified product $p'(j,k)$. *For instance, the cluster containing the entries (0,0) and (0,1) corresponds to the product* $p'(0,1) = \overline{T}_1^{4a} \cdot \overline{T}_2^{4b}$ *obtained from* $p(0,1) = \overline{T}_1^{4a} \cdot T_1^{4b} \cdot \overline{T}_2^{4b}$. *Graphically, fulfilling the condition (6.6) means that every entry* (j^e, k^e) *appears in at least two clusters and the products corresponding to these clusters are in at least two different functions* y_{i_1} *and* y_{i_2}. *The latter information is indicated by marking the entries* $(j,k) \in C(n,t)$ *with* $i \in \{1,2\}$ — *the indices of the functions* y_i *containing* $p(j,k)$. □

6.3. Self-Testing Property

Now we shall set the conditions under which the translator $\mathbf{H_1}$ that realizes the basic functions (6.5) is ST for all single s/z faults. Two factors may affect the ST property of $\mathbf{H_1}$:

1. Some faults in $\mathbf{H_1}$ cannot be tested by inputs from $C(n,t)$ only; and
2. Some faults in $\mathbf{H_1}$ which could be tested by inputs from $C(n,t)$ are not detected due to improper partitioning of $C(n,t)$ into subsets $\mathbf{Y_i}$.

Consider the 4-input AND gate $p(j,k) = T_j^{n_a} \cdot \overline{T}_{j+1}^{n_a} \cdot T_k^{n_b} \cdot \overline{T}_{k+1}^{n_b}$. Any s/0 fault in $p(j,k)$ is tested by any codeword from (j,k). Since the s/1 fault on the output of $p(j,k)$ is tested by any test for any input line s/1 fault, only the input line s/1 faults remain of our concern. We begin with the s/1 fault on the input line $T_j^{n_a}$, for brevity denoted as $T_j^{n_a}/1$. All tests for this fault are in any subset (j',k') such that for any $X \in (j',k')$ $T_j^{n_a}(X) = 0$ and $\overline{T}_{j+1}^{n_a}(X) = T_k^{n_b}(X) = \overline{T}_{k+1}^{n_b}(X) = 1$, i.e. if and only if $k' = k$ and $j' < j$. Similar reasoning and notation applies to three other s/1 faults as well as for any other AND gate, possibly with fewer than four inputs.

All input lines of the AND gates $p(j,k)$ whose s/1 faults are not tested by applying Borden codewords can be formally identified in the following way: (1) $T_j^{n_a}/1$ (or $T_k^{n_b}/1$) is untestable if $j > 0$ (or $k > 0$) and (j,k) has the smallest j (or k) amongst subsets (j,k) in $C(n,t)$ for a given k (or j); and (2) $\overline{T}_{j+1}^{n_a}/1$ (or $\overline{T}_{k+1}^{n_b}/1$) is untestable if $j < n_a$ (or $k < n_b$) and (j,k) has the largest j (or k) amongst subsets (j,k) in $C(n,t)$ for a given k (or j). Alternatively, the same can be done by inspection of the partitioning diagram (Fig. 6.2) in the following way: (1) Any product $p(j,k)$ with $j > 0$ $(k > 0)$ which corresponds to the uppermost (leftmost) set (j,k) in the k-th column (j-th row) has untestable $T_j^{n_a}/1$ $(T_k^{n_b}/1)$ fault; and (2) Any product $p(j,k)$ with $j < n_a$ $(k < n_b)$ which corresponds to the bottommost (rightmost) set (j,k) in the k-th column (j-th row) has untestable $\overline{T}_{j+1}^{n_a}/1$ fault $(\overline{T}_{k+1}^{n_b}/1)$.

Example 6.1 (Cont'd). *Any s/1 fault of the product $p(j,k)$, untestable by $C(8,2)$ codewords, can be easily found with the help of the partitioning diagram from Fig. 6.2: it corresponds to any arrow leaving the (j,k) entry (marked with 1 or 2) which points towards missing tests (codewords). The results of such an analysis are summarized in the second column of Table 6.1, wherein all inputs with untestable s/1 faults are marked with an asterisk.* □

To eliminate the s/1 faults untestable by Borden codewords, we suggest to replace any product $p(j,k)$ containing such faults with some 'equivalent' product $p'(j,k)$, which is 1 not only for all codewords from (j,k) but also for non-codewords X^e from some subsets (j^e,k^e). Three types of modifications may occur:

(M1) $p'(j,k)$ has the same number of inputs as $p(j,k)$ but $p(j,k) \neq p'(j,k)$;

(M2) $p'(j,k)$ is obtained from $p(j,k)$ by removing one or two inputs in $p(j,k)$; and

(M3) $p'(j,k)$ replaces $u \geq 2$ products $p(j_1,k_1),\ldots,p(j_u,k_u)$.

The modification M3 may involve other sets $(j_i,k_i) \in C(n,t)$ having some untestable inputs as well as those having all input faults tested. Recall, however, that in all three cases the following condition (6.6) necessary to preserve the CD property of $\mathbf{H_1}$ must be satisfied:

for each $(j^e,k^e) \in \mathbf{X_{nc1}}$ there are at least two modified products $p'(j_1,k_1)$, $p'(j_2,k_2)$ which are 1s for any $X^e \in (j^e,k^e)$ (or, graphically, each entry (j^e,k^e) marked with $*$ appears in at least two clusters marked with different indices).

Note: The modifications M2 and M3 decrease a total of gate inputs, whereas the modification M3 also decreases the number of AND gates.

Example 6.1 (Cont'd). *Examples of modifications that eliminate untestable s/1 faults and preserve the CD property are shown in Fig. 6.3. The modified products obtained on the basis of Fig. 6.3 are in the third column of Table 6.1. Indexing of the subsets (j,k) from $C(8,2)$ with 1s and 2s implies the partitioning of $C(8,2)$ into $r = 2$ sets $\mathbf{Y_i}$ which guarantees the ST property and preserves the CD property of $\mathbf{H_1}$. The final functions of the translator $\mathbf{H_1}$, which itself is an STC for the $C(8,2)$ code, are:*

$$\begin{aligned} y_1 &= T_4^{4b} + \overline{T}_1^{4b} + T_2^{4a} \cdot \overline{T}_3^{4a} \\ y_2 &= \overline{T}_1^{4a} \cdot \overline{T}_2^{4b} + T_1^{4a} \cdot \overline{T}_3^{4a} \cdot T_3^{4b} + T_2^{4a} \cdot \overline{T}_4^{4a} \cdot \overline{T}_2^{4b} + T_4^{4a} \cdot T_3^{4b} \qquad \square \end{aligned}$$

6.4. General Design Procedure of the Circuit $\mathbf{H_1}$

Procedure 6.1.

1. Partition the set I of n input bits into two subsets A and B with $n_a = \lfloor n/2 \rfloor$ and $n_b = \lceil n/2 \rceil$ bits, respectively.
2. Partition the $C(n,t)$ code into subsets (j,k).
3. Generate the basic products $p(j,k)$ of the translator $\mathbf{H_1}$.
4. Identify all untestable input s/1 faults of the AND gates $p(j,k)$.
5. Modify all AND gates $p(j,k)$ with untestable input s/1 faults to meet the requirements of the ST property and the condition (6.6) necessary to preserve the CD property.
6. Assign each member of $\mathbf{Y}$, the set of modified products $p'(j,k)$, to exactly one of r functions y_i, $r \geq 2$, in such a way that $\mathbf{H_1}$ meets the conditions for the CD and ST properties. □

Theorem 6.1. *The circuit $\mathbf{H_1}$ designed by Procedure 1 is CD.*

Proof. The circuit $\mathbf{H_1}$ designed by Procedure 1 is CD as it meets the conditions:

C1) For any $X \in C(n,t)$ exactly one of the functions y_i is 1 for X;

C2a) For any $X^e \in \mathbf{X_{nc1}}$ at least two functions y_i and y_j are 1s for X^e; and

C2b) For any $X^e \in \mathbf{X_{nc0}}$ all functions y_i are 0s for X. □

Now we shall prove that the circuit $\mathbf{H_1}$ built of two irredundant circuits T^{n_a} and T^{n_b} (realized by using any method) is ST. First, observe that for any input n_v-tuple of weight w_v, $v \in \{a, b\}$, a non-faulty circuit T^{n_v} produces an output $(T_1^{n_v} T_2^{n_v} \ldots T_{n_v}^{n_v})$ of the same weight with w_v 1s followed by $n_v - w_v$ 0s, and hence it can produce only $n_v + 1$ output patterns. The concatenation of such patterns, generated by the circuits T^{n_a} and T^{n_b}, should be mapped properly by the MN (which is CD) into either a $1/r$ or a non-$1/r$ code output, depending whether the input X entering T^{n_a} and T^{n_b} is a Borden codeword. These restricted subset of output patterns of T^{n_v} should suffice to ensure the ST property of the MN. Despite single s/z faults of a circuit T^{n_v} may result in producing a large variety of incorrect output patterns, we shall take advantage of the following properties.

Property 6.1. *For any single s/z fault ($z \in \{0,1\}$) in an irredundant circuit T^{n_v}, there is at least one input of weight w_v such that $(T_1^{n_v} T_2^{n_v} \ldots T_{n_v}^{n_v})$ generated by the faulty circuit T^{n_v} is of the form $(\underbrace{11\ldots1}_{w_v-1}00\ldots0)$ and $(\underbrace{11\ldots1}_{w_v+1}00\ldots0)$ for $z = 0$ and $z = 1$, respectively.*

Property 6.2. *If $n_a = \lfloor n/2 \rfloor$ and $n_b = \lceil n/2 \rceil$, each circuit T^{n_v} receives all 2^{n_v} inputs, i.e. it is tested exhaustively, even when only Borden codewords are applied.*

Theorem 6.2. *The circuit* $\mathbf{H_1}$ *designed by Procedure 1 is ST for any single s/z fault.*

Proof. We shall show that for any single s/z fault in the circuit $\mathbf{H_1}$ there exists a Borden codeword which produces a non-$1/r$ code output.

First, note that the circuits T^{n_a} and T^{n_b} convert any input Borden codeword of weight $w = w_a + w_b$ into an output Borden codeword also of weight $w = w_a + w_b$ but with bits sorted as follows: w_a 1s, $n_a - w_a$ 0s, w_b 1s, $n_b - w_b$ 0s.

Suppose that a circuit T^{n_v} has a s/z fault. Due to Properties 6.1 and 6.2, there is some X in $C(n, t)$ such that the output of the faulty circuit T^{n_v} produces (instead of w_v 1s followed by $n_v - w_v$ 0s) the regular pattern of: (i) $w_v - 1$ 1s followed by $n_v - w_v + 1$ 0s — for $z = 0$; and (ii) $w_v + 1$ 1s followed by $n_v - w_v - 1$ 0s — for $z = 1$. This guarantees that the weight of an output produced by T^{n_a} and T^{n_b} differs by one from a correct one, and hence it is not a Borden codeword. The MN, which is CD, maps it into some non-$1/r$ codeword, which is then mapped by $\mathbf{H_2}$ into (00) or (11) output. Thus, only the faults of the MN need to be considered now. We shall consider the most demanding case of the MN realized as a three-level NOT-AND-OR network. Naturally, if this network is ST, then its multi-level implementation is also ST.

Case 1: A s/z fault on the line $T_i^{n_v}$ prior to fan-out into $T_i^{n_v}$ and $\overline{T}_i^{n_v}$ may cause a bidirectional error $T_i^{n_v} \rightarrow \overline{T}_i^{n_v}$ and $\overline{T}_i^{n_v} \rightarrow T_i^{n_v}$ beyond the fan-out point. However, such a fault is tested by any Borden codeword X with exactly i $(i-1)$ 1s on the bits from V if $z = 0$ $(z = 1)$. This is because for any such X, the fault produces a single error which is propagated to the checker output as though there was a single error on the checker input.

Case 2: Consider a product $p'(j,k) = T_{j_1}^{n_a}\overline{T}_{j_2}^{n_a}T_{k_1}^{n_b}\overline{T}_{k_2}^{n_b}$. Any fault of the type: (i) the input s/1 (output s/0) of an inverter; (ii) any s/0 fault of an AND gate; and (iii) the output s/0 fault of an OR gate fed by $p'(j,k)$, are all detected by any input X with $j_1 \leq w_a < j_2$ and $k_1 \leq w_b < k_2$, e.g. any $X \in (j,k)$. Any fault of the type: (i) the s/1 fault of the input line $T_{j_1}^{n_a}$ to an AND gate; and (ii) the s/1 fault on the output of an AND gate (input line to an OR gate) and the output s/1 fault of an OR gate, is detected by any input X with $w_a < j_1$ and $k_1 \leq w_b < k_2$. Such a Borden codeword X is guaranteed to exist for any such fault as a result of modifying all basic products $p(j,k)$ with untestable input s/1 faults and the assignment rules of the products $p'(j,k)$ to the functions y_i. Similar reasoning applies to the faults $\overline{T}_{j_2}^{n_a}/1$, $T_{k_1}^{n_b}/1$, and $\overline{T}_{k_2}^{n_b}/1$, as well as to the input s/0 (output s/1) faults of an inverter. □

From Theorems 6.1 and 6.2 and the assumption that $\mathbf{H_2}$ is an STC for the $1/r$ code, we conclude that the circuit from Fig. 6.1 designed by Procedure 1 is an STC for Borden code.

6.5. General Functions of the Circuit $\mathbf{H_1}$

Here we shall give the functions of the ST/CD translator $\mathbf{H_1}$ of a $C(n,t)$ code into a $1/z$ code with the minimum possible $z = 2$ or 3. They have been derived by analyzing regular patterns seen in the modification maps, such as given in Figures 6.3 and 6.4–6.6. (Actually, a modification map is the most convenient tool for designing an STC for a Borden code $C(n,t)$ with many n of practical importance.) The ST/CD translator into the 1/4 rather than 1/3 code can be obtained by splitting *any* (but exactly one) of three functions into a pair of subfunctions.

Let $A1, B1, C1_{par}$ denote some elementary functions, where $par \in \{odd, even\}$. The basic functions of an STC for the $C(n,t)$ code for t even assume the preliminary forms:

$$y_1 = A1 + B1 + C1_{even} \tag{6.7}$$

$$y_2 = C1_{odd} \tag{6.8}$$

$$y_3 = 0 \tag{6.9}$$

Table 6.2. Classification of some C(n,t) codes according to three cases of n considered

t	Case 1	Case 2	Case 3
1	$\boxed{\text{Any } n}$	—	—
2	$\boxed{\text{4, 5}}$	6	$\boxed{\text{7, 8}}$
	$\boxed{\text{9–11}}$	12–14	
	$\boxed{\text{15–17}}$	18–20	
	$\boxed{\text{21–23}}$	24–26	
	$\boxed{\text{27–29}}$	30–32	
3	$\boxed{\text{6, 7}}$	$\boxed{\text{8–10}}$	11, 12
	$\boxed{\text{13–15}}$	16–18	19, 20
	$\boxed{\text{21–23}}$	24–26	27, 28
	$\boxed{\text{29–31}}$	32–34	35, 36
4	$\boxed{\text{8, 9}}$	10–12	13, 14, $\boxed{\text{15, 16}}$
	$\boxed{\text{17–19}}$	20–22	23–26
	$\boxed{\text{27–29}}$	30–32	33–36
5	$\boxed{\text{10, 11}}$	$\boxed{\text{12–14}}$	15–20
	$\boxed{\text{21–23}}$	24–26	27–32
	$\boxed{\text{33–35}}$	36–38	39–44
6	$\boxed{\text{12, 13}}$	14–16	17–22, $\boxed{\text{23, 24}}$
	$\boxed{\text{25–27}}$	28–30	31–38
	$\boxed{\text{39–41}}$	42–44	45–52
7	$\boxed{\text{14, 15}}$	$\boxed{\text{16–18}}$	19–28
	$\boxed{\text{29–31}}$	32–34	35–44
	$\boxed{\text{45–47}}$	48–50	51–60
8	$\boxed{\text{16, 17}}$	18–20	21–30, $\boxed{\text{31, 32}}$
	$\boxed{\text{33–35}}$	36–38	39–50
	$\boxed{\text{51–53}}$	54–56	57–68

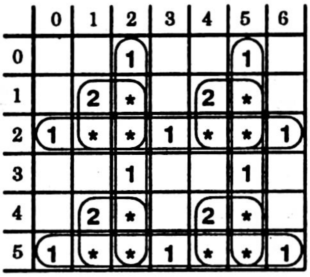
but for t odd — $B1$ should be moved from y_1 to y_2.

The functions (6.7)–(6.9) will be detailed and modified (when necessary) for three separate cases as exemplified in Table 6.2. The values of n in boxes indicate that a particular code can be translated directly to the 1/2 code.

Case 1: $n = 2k(t+1) - c$, where: $c \in \{1,2\}$ if $k = 1$ and $c \in \{1,2,3\}$ if $k > 1$ The elementary functions are:

$$A1 = \sum_{j=1}^{k} T^{n_a}_{j(t+1)-\lceil c/2\rceil} \cdot \overline{T}^{n_a}_{j(t+1)-\lceil c/2\rceil+1} \tag{6.10}$$

$$B1 = \sum_{p=1}^{k} T^{n_b}_{p(t+1)-1} \cdot \overline{T}^{n_b}_{p(t+1)} \tag{6.11}$$



	0	1	2	3	4	5	6
0			1			1	
1		2	*		2	*	
2	1	*	*	1	*	*	1
3			1			1	
4		2	*		2	*	
5	1	*	*	1	*	*	1

Figure 6.4. Modification map of the AND gates p(j,k) in the STC for the Borden codes C(n,2), n=∈{9,10,11}

$$C1_{par} = \sum_{\substack{1 \leq i < t \\ i \text{ par}}} \left(\left(\sum_{j=1}^{k} T^{n_a}_{j(t+1)-\lceil c/2 \rceil - i} \cdot \overline{T}^{n_a}_{j(t+1)-\lceil c/2 \rceil + 1} \right) \right. \tag{6.12}$$

$$\left. \cdot \left(\sum_{p=1}^{k} T^{n_b}_{(p-1)(t+1)+i} \cdot \overline{T}^{n_b}_{p(t+1)} \right) \right) \tag{6.13}$$

Note: Two special cases are included: (i) an STC for the parity code ($t = 1$ and any k) with $C1_{par} = 0$; and (ii) an STC for the $t/2t$ code ($k = 1$, $c = 2$, and any t) — as defined in [2].

Example 6.2. *The modification map of an STC for the $C(11,2)$ code is shown in Fig. 6.4. The modification map for the $C(10,2)$ code can be obtained by neglecting the rightmost column, and for the $C(9,2)$ code — by neglecting both the rightmost column and the uppermost row and decrementing the row indices by one. Similar regular pattern of clusters repeats for any other n covered by Case 1. For clarity, any entry corresponding to (j,k) is denoted with an integer $i \in \{1,2,3\}$, where i is an index of a function y_i containing the product $p'(j,k)$ corresponding to a cluster containing i. In this way, the CD and ST properties of the MN can be verified by inspection and the translator functions can be readily derived from the modification maps.* □

Case 2: $t \geq 2$, $n = 6$ and $n = 2k(t+1) + c$, where $k \geq 1$ and $c \in \{0,1,2\}$ The elementary functions are:

$$A1 = \sum_{j=1}^{k} T^{n_a}_{j(t+1)+\lfloor c/2 \rfloor - 1} \cdot \overline{T}^{n_a}_{j(t+1)+\lfloor c/2 \rfloor} \tag{6.14}$$

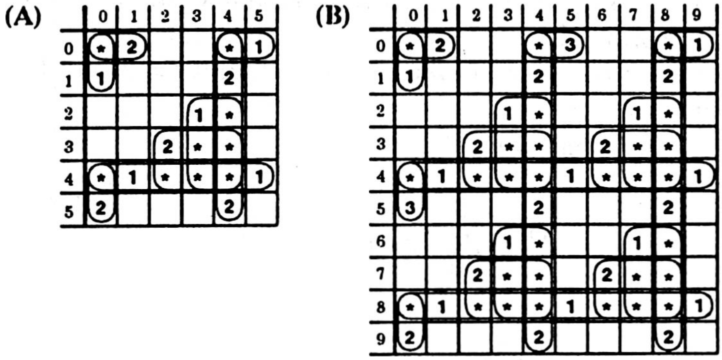


Figure 6.5. Modification map of the AND gates p(j,k) in the STCs for the Borden codes: C(n,3), n∈{8,9,10} (A); C(18,3) (B)

$$B1 = \sum_{p=1}^{k} T^{n_b}_{p(t+1)} \cdot \overline{T}^{n_b}_{p(t+1)+1} \tag{6.15}$$

$$C1_{par} = \sum_{\substack{1 \le i < t \\ i \text{ par}}} \left(\left(\sum_{j=1}^{k} T^{n_a}_{j(t+1)+\lfloor c/2 \rfloor -1-i} \cdot \overline{T}^{n_a}_{j(t+1)+\lfloor c/2 \rfloor} \right) \right. \tag{6.16}$$

$$\left. \cdot \left(\sum_{p=1}^{k} T^{n_b}_{p(t+1)-t+i} \cdot \overline{T}^{n_b}_{p(t+1)+1} \right) \right) \tag{6.17}$$

The functions (6.7)–(6.9) must be extended as follows.

1. Include $T^{n_a}_{n_a-1} \cdot \overline{T}^{n_b}_{1}$ in y_2.
2. Include $\overline{T}^{n_a}_{1+\lfloor c/2 \rfloor} \cdot \overline{T}^{n_b}_{1}$: (a) in y_3 — for $C(6,2)$; or (b) in y_2 (y_1) — for $n > 6$ if t even (odd).
3. If $c = 2$ and t even (odd) — include $\overline{T}^{n_a}_{1} \cdot \overline{T}^{n_b}_{2}$ in y_1 (y_2) and $\overline{T}^{n_a}_{1} \cdot T^{n_b}_{n_b-1}$ in y_2 (y_1).
4. If $k > 1$ — include $\left(\sum_{j=1}^{k-1} T^{n_a}_{j(t+1)+\lfloor c/2 \rfloor -1} \cdot \overline{T}^{n_a}_{j(t+1)+\lfloor c/2 \rfloor +1}\right) \overline{T}^{n_b}_{1}$ in y_3.
5. If $k > 1$ and $c = 2$ — include $\left(\sum_{p=1}^{k-1} T^{n_b}_{p(t+1)} \cdot \overline{T}^{n_b}_{p(t+1)+2}\right) \overline{T}^{n_a}_{1}$ in y_3. □

Figures 6.5 (A) and (B) show the modification maps applicable for the translators: $C(n,3) \rightarrow 1/2$ — for $n \in \{8,9,10\}$, and $C(n,3) \rightarrow 1/3$ — for $n \in \{16,17,18\}$, respectively.

Case 3: $n = 2k(t+1)+2+c$, where: $c \in \{-1,0\}$ if $t = 2$, and $1 \leq c \leq 2(t-2)$ if $t \geq 3$

The elementary functions are:

$$A1 = \sum_{j=1}^{k} T^{n_a}_{j(t+1)-d-1} \cdot \overline{T}^{n_a}_{j(t+1)-d} \tag{6.18}$$

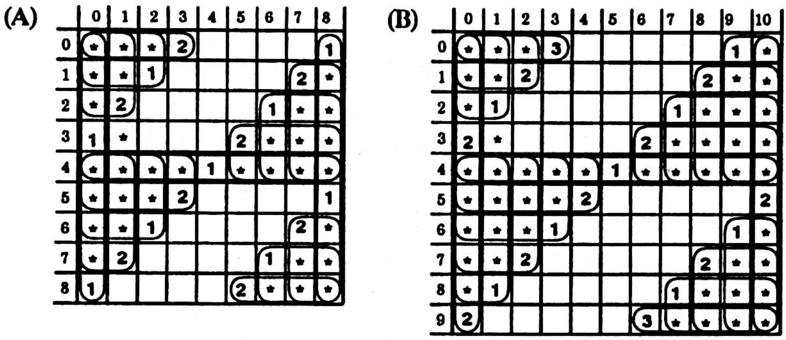
$$B1 = \sum_{p=1}^{k} T^{n_b}_{p(t+1)+\lceil c/2\rceil+1} \cdot \overline{T}^{n_b}_{p(t+1)+\lceil c/2\rceil+2} \tag{6.19}$$

$$\begin{aligned} C1_{par} \ = \ & \sum_{\substack{1\leq i<t \\ i \text{ par}}} \left(\left(\sum_{j=1}^{k+1} T^{n_a}_{(j-1)(t+1)-d+i} \cdot \overline{T}^{n_a}_{j(t+1)-d} \right) \right. \\ & \left. \cdot \left(\sum_{p=1}^{k} T^{n_b}_{p(t+1)+\lceil c/2\rceil+1-i} \cdot \overline{T}^{n_b}_{p(t+1)+\lceil c/2\rceil+2} \right) \right) \end{aligned} \tag{6.20}$$

$$C2_{par} = \sum_{\substack{1\leq i\leq 1+\lceil c/2\rceil \\ i \text{ par}}} \left(\left(\sum_{j=0}^{k} T^{n_a}_{j(t+1)-d-1} \cdot \overline{T}^{n_a}_{j(t+1)+2+\lfloor c/2\rfloor-i} \right) \overline{T}^{n_b}_{i+1} \right), \tag{6.21}$$

where $d = 0$ (1) if n even (odd). The functions (6.7)–(6.9) must be extended with $C2_{par}$ as follows.

1. Include one of the functions $C2_{even}$ and $C2_{odd}$ that contains the term $\left(T^{n_a}_{k(t+1)-d-1} \cdot \overline{T}^{n_a}_{k(t+1)-d+1} \overline{T}^{n_b}_{2+\lceil c/2\rceil}\right)$ in y_2 and the other in y_1.
2. Move $\overline{T}^{n_a}_{1} \cdot \overline{T}^{n_b}_{2+\lfloor c/2\rfloor}$ to y_3, except when t even and $k = 1$ and $c \in \{2t-5, 2t-4\}$.
3. If $c \in \{2t-5, 2t-4\}$: (a) replace $\left(\sum_{j=0}^{k} T^{n_a}_{j(t+1)-d-1} \cdot \overline{T}^{n_a}_{j(t+1)+2+\lfloor c/2\rfloor}\right) \cdot \overline{T}^{n_b}_{1}$ with $\overline{T}^{n_b}_{1}$; and (b) move $T^{n_a}_{n_a} T^{n_b}_{t+1}$ to y_3 — if t odd and $k = 1$.
4. If $1 \leq c \leq 2t-6$, move to y_3: $T^{n_a}_{t-d} \cdot \overline{T}^{n_b}_{1}$ — if $k = 1$, or $\left(\sum_{j=1}^{k-1} T^{n_a}_{j(t+1)-d-1} \cdot \overline{T}^{n_a}_{j(t+1)+2+\lfloor c/2\rfloor}\right) \cdot \overline{T}^{n_b}_{1}$ — if $k > 1$.



(A)

	0	1	2	3	4	5	6	7	8
0	★	★	★	2					1
1	★	★	1					2	★
2	★	2					1	★	★
3	1	★				2	★	★	★
4	★	★	★	★	1	★	★	★	★
5	★	★	★	2					1
6	★	★	1					2	★
7	★	2					1	★	★
8	1					2	★	★	★

(B)

	0	1	2	3	4	5	6	7	8	9	10
0	★	★	★	3						1	★
1	★	★	2						2	★	★
2	★	1						1	★	★	★
3	2	★					2	★	★	★	★
4	★	★	★	★	★	1	★	★	★	★	★
5	★	★	★	★	2						2
6	★	★	★	1						1	★
7	★	★	2						2	★	★
8	★	1						1	★	★	★
9	2						3	★	★	★	★

Figure 6.6. Modification map of the AND gates p(j,k) in the STCs for the Borden codes: C(16,4) (A); C(19,5) (B)

5. If $k > 1$, move $\left(\sum_{p=1}^{k-1} T^{n_b}_{p(t+1)} \cdot \overline{T}^{n_b}_{p(t+1)+t}\right) \cdot T^{n_a}_{n_a}$ to y_3. □

Figures 6.6(A) and 6.6(B) show the modification maps of the translators: $C(16,4) \rightarrow 1/2$ and $C(19,5) \rightarrow 1/3$, respectively.

The ease of designing a multi-level MN in a systematic way, which may be necessary for large n, is evident from the regular form of the functions of the MN for the STC for the $C(32,4)$ code:

$$\begin{aligned} y_1 = {} & T^{16a}_5 \cdot \overline{T}^{16a}_6 + T^{16a}_{10} \cdot \overline{T}^{16a}_{11} + T^{16a}_{15} \cdot \overline{T}^{16a}_{16} + T^{16b}_5 \cdot \overline{T}^{16b}_6 + T^{16b}_{10} \cdot \overline{T}^{16b}_{11} + T^{16b}_{15} \cdot \overline{T}^{16b}_{16} \\ & + \left(T^{16a}_3 \cdot \overline{T}^{16a}_6 + T^{16a}_8 \cdot \overline{T}^{16a}_{11} + T^{16a}_{13} \cdot \overline{T}^{16a}_{16}\right)\left(T^{16b}_3 \cdot \overline{T}^{16b}_6 + T^{16b}_8 \cdot \overline{T}^{16b}_{11} + T^{16b}_{13} \cdot \overline{T}^{16b}_{16}\right) \end{aligned}$$

$$\begin{aligned} y_2 = {} & \left(T^{16a}_2 \cdot \overline{T}^{16a}_6 + T^{16a}_7 \cdot \overline{T}^{16a}_{11} + T^{16a}_{12} \cdot \overline{T}^{16a}_{16}\right)\left(T^{16b}_2 \cdot \overline{T}^{16b}_6 + T^{16b}_7 \cdot \overline{T}^{16b}_{11} + T^{16b}_{12} \cdot \overline{T}^{16b}_{16}\right) \\ & + \left(T^{16a}_4 \cdot \overline{T}^{16a}_6 + T^{16a}_9 \cdot \overline{T}^{16a}_{11} + T^{16a}_{14} \cdot \overline{T}^{16a}_{16}\right)\left(T^{16b}_4 \cdot \overline{T}^{16b}_6 + T^{16b}_9 \cdot \overline{T}^{16b}_{11} + T^{16b}_{14} \cdot \overline{T}^{16b}_{16}\right) \\ & + \overline{T}^{16a}_1 \cdot \overline{T}^{16b}_2 + \overline{T}^{16a}_1 \cdot T^{16b}_{15} + T^{16a}_{15} \cdot \overline{T}^{16b}_1 \end{aligned}$$

$$y_3 = \overline{T}^{16a}_1 \cdot T^{16b}_5 \cdot \overline{T}^{16b}_7 + \overline{T}^{16a}_1 \cdot T^{16b}_{10} \cdot \overline{T}^{16b}_{12} + T^{16a}_5 \cdot \overline{T}^{16a}_7 \cdot \overline{T}^{16b}_1 + T^{16a}_{10} \cdot \overline{T}^{16a}_{12} \cdot \overline{T}^{16b}_1$$

6.6. Parameter Estimation and Comparison

The parameters of the STC for the $C(n,t)$ code proposed here can be estimated by using the following formulae:

$$Ga(n,t) < Ga(T^{\lfloor n/2 \rfloor}) + Ga(T^{\lceil n/2 \rceil}) + (q + r) + Ga(1/r); \qquad (6.22)$$

$$In(n,t) < In(T^{\lfloor n/2 \rfloor}) + In(T^{\lceil n/2 \rceil}) + 5q + In(1/r); \qquad (6.23)$$

$$L(n,t) = L(T^{\lceil n/2 \rceil}) + 2 + L(1/r); \qquad (6.24)$$

Table 6.3. Parameters of various STCs for some Borden codes C(n,t)

Code	Gates		Gate Inputs		Levels	
$C(n,t)$	New	[47]	New	[47]	New	[47]
$C(8,2)$	27	54	59	114	5	8
$C(16,2)$	87	107	178	318	9	10
$C(16,4)$	91	324	199	> 2000	8	10
$C(24,2)$	182	168	370	524	15	10
$C(24,4)$	196	547	439	> 3000	14	10
$C(32,2)$	297	228	616	598	16	12
$C(32,4)$	303	> 707	630	> 4000	17	10
$C(32,8)$	283	?	608	?	12	?

where: $Ga(T^{\lfloor n/2 \rfloor})$, $Ga(T^{\lceil n/2 \rceil})$, etc., are the parameters of the threshold circuits $T^{\lfloor n/2 \rfloor}$, $T^{\lceil n/2 \rceil}$; and $Ga(1/r)$, etc., are the parameters of the STC for the $1/r$ code. The latter are all equal to 0 for $r = 2$, whereas for $r = 4$ they are: $Ga(1/4) = 8$, $In(1/4) = 16$, and $L(1/4) = 3$. In (6.22) and (6.23), the complexity of the MN is loosely upper-bounded by q four-input AND gates and r OR gates with a total of q inputs (similarly to [47], we do not count inverters). However, its actual complexity is significantly lower, due to modifications of the basic products suggested to eliminate untestable s/1 faults, which reduce a total of inputs to the AND gates and the number of AND gates as well. For large n, further hardware reduction is possible by implementing the MN in a multi-level form, which is fairly easy due to regular structure of the MN functions (as it was shown for the $C(32,4)$ code).

The asymptotic complexity estimations of a cost-efficient STC built by using the circuits T^{n_a} and T^{n_b} implemented as SNs are: $Ga(n,t) = O(n \log^2 n)$, $In(n,t) = O(n \log^2 n)$, and $L(n,t) = O(\log^2 n)$ (for exact parameters see Table 4.3 and formulae (4.4)–(4.6)).

Now we shall compare the new STCs for Borden codes with existing designs [66], [47]. The least complex were claimed those from [47], but there are two reasons that this seems not be true for codes with $t > 4$, for which no comparisons were done. First, the complexity estimations of the checkers from [66] provided in [47] are not accurate, since significantly less complex STCs for the m/n codes from [137] were not taken into account. Secondly, the complexity of the checkers from [47] grows quickly with t. As there are no general formulae available to estimate the parameters of the checkers from [66], [47], we shall compare them for sample codes only.

The new STCs are superior to those from [66] due to the following argument. The threshold circuits $T^{\lfloor n/2 \rfloor}$ and $T^{\lceil n/2 \rceil}$ contribute from 74% (for $C(8,2)$) to 89% (for $C(32,8)$) to the complexity of our checkers listed in Table 6.3. These circuits must also be used to implement an STC for the $\lfloor n/2 \rfloor / n$ code, which is an integral part of any checker from [66]. However, the checkers from [66] also use STCs for many other m/n codes and some output circuitry which, as a whole, are significantly more complex than our MN and the STC for the 1/4 code. For instance, the least complex STC for the $C(8,2)$ code from [66] can be built by using the optimal checkers: for the 4/8 code — from [143], and the STCs for the 1/8 and 7/8 codes using multi-level 3-pair 2-rail STCs. The $Ga/In/L(8,2)$ count for this checker is 71/141/9 *vs.* 27/59/5 (plus five inverters) in our design.

Compared to the checkers from [47], the design of our checkers is conceptually simpler: the major part (i.e. the circuits T^{n_v}) is built using a simple cell, whereas the remaining part is naturally amenable for multi-level realization (the MN) and inherently simple (the STC for the 1/4 code — Fig. 5.1). On the other hand, the major part of the checkers from [47] is built using two types of complex modules designed to ensure that they are tested exhaustively during normal functioning. These are: (i) the weight subcounters built of AND gates with up to m inputs and OR gates with up to 2^{m-1} inputs; and (ii) the adders mod $(t+1)$ built of AND gates with up to $2m$ inputs and OR gates with up to 2^{2m-1} inputs, where $m = \lceil \log_2(t+1) \rceil$. These modules are relatively simple when realized using random logic for $t = 2$ and 3 only, as they require many gates with six and more inputs for $t \geq 4$ (when $m \geq 3$). The above criticism is confirmed by the figures given in Table 6.3, which indicate that the checkers from [47] could be less complex than our checkers for codes with large n ($n > 30$) and $t = 2$ and 3 only, due to the $O(n \log^2 n)$ growth of the complexity of the circuits $T^{\lfloor n/2 \rfloor}$ and $T^{\lceil n/2 \rceil}$. Nevertheless, the gate input count in the checkers from [47] becomes excessive already for $t = 4$. Recall also that the modules used in the latter checkers use many multi-input gates with large fan-out which can be difficult to implement efficiently in some technologies, and that all input variables to any module are used both complemented and uncomplemented. Contrary, most part of our checker is built of 2-input gates of fan-out two, only a few OR gates in the MN may have fan-in problems, and only a few threshold variables $T_i^{n_v}$ are used both complemented and uncomplemented.

In summary, the STC for Borden code $C(n,t)$ presented in this section is general, as it covers the parity codes ($t = 1$), the unordered $t/2t$ codes ($n = 2t$), and non-trivial Borden codes ($1 < t < \lceil n/2 \rceil$). Despite hardware efficiency, the other advantage of these checkers, which makes them particularly attractive for VLSI implementation, is a highly regular modular structure, whose major part (from 74% to 89%) consists of identical cells composed of two-input gates with fan-out of two. It is worth to note that the complexity of the new STCs for any Borden code $C(n,t)$ with a given n is only slightly higher than the complexity of the SN-based STC for the $t/2t$ code presented in Section 5 (e.g. 13% more gates in case of the $C(32,4)$ and 16/32 codes). This is an encouraging argument for using a Borden code instead of a $t/2t$ code of the same codeword length, when larger capacity of a unidirectional error code is needed. Also, the design concepts presented here can be extended to other, not necessarily SC, circuits having a similar structure composed of two or more threshold circuits. One example is an efficient residue modulo 3 generator proposed recently by the author [155].

7. ENCODERS AND STCs FOR SYSTEMATIC UEDCs

In this section both encoders and STCs for the following classes of systematic UEDCs will be subsequently presented: 1) optimal systematic unordered codes — Berger codes and modified Berger codes; 2) t-UEDCs — Bose-Lin codes and Jha-Vora codes; and 3) BUEDCs by Blaum. Since an STC for any systematic UEDC considered here can be realized as a normal checker by using an encoder for that code (see Fig. 3.1), for readers' convenience, not only STCs but also the encoders will be presented. Every circuit considered here can be designed on the basis of the counters of 1s presented in Section 4. Therefore two versions of encoders and STCs will be given for most codes: one derived from a parallel counter and the other using the T^n circuit. A critical survey of the STCs (and encoders) for each of the above classes of systematic UEDCs will be given in the appropriate subsection.

7.1. STCs for Optimal Systematic Unordered Codes (OSUCs)

7.1.1. Encoders for Berger Codes

Any $(I;K)$ counter of 1s with complemented outputs can be used as an encoder for a Berger code $C_{(I,K)}$. Thus, either counter of 1s discussed in Section 4 applies: (i) the parallel $(I;K)$ counter followed by a bank of K inverters; and (ii) the T^n-based counter of 1s with a NOT-AND-OR circuit modified, in such a way that the inverters are not needed (as shown by the following example).

Example 7.1. *Design of an encoder for the $C_{(9,4)}$ code.*
One version is a parallel counter of 1s from Fig. 4.1. The other is designed similarly as the T^9-based counter of 1s, also given in Section 4. The encoder specification is provided by the first two columns of Table 4.5 and its functions are:

$$s_3 \;=\; \overline{T}_8^9$$

$$\begin{aligned} s_2 &= \overline{T^9_4} + T^9_8 \\ s_1 &= \overline{T^9_2} + T^9_4 \cdot \overline{T^9_6} + T^9_8 \\ s_0 &= \overline{T^9_1} + T^9_2 \cdot \overline{T^9_3} + T^9_4 \cdot \overline{T^9_5} + T^9_6 \cdot \overline{T^9_7} + T^9_8 \cdot \overline{T^9_9} \qquad \square \end{aligned}$$

Clearly, the functions of the encoder for a Berger code are dual to the functions of the counter of 1s (here it is used as an encoder with complemented outputs) given previously: they can be easily derived by: (i) swapping the OR and AND operators, and (ii) swapping inverters by changing T^9_i to $\overline{T^9_i}$ and $\overline{T^9_i}$ to T^9_i, in the functions of the counter of 1s. As a result, any T^n-based encoder with complemented outputs does not need to use explicitly the bank of K output inverters. Note that also in this case, only the threshold functions T^9_i with even i occur both in the complemented and uncomplemented form.

7.1.2. STCs for Berger Codes with $I \neq 2^{K-1}$

The STCs for Berger and equivalent codes (for which the generic term *optimal systematic unordered codes (OSUCs)* will be used throughout this section) were proposed in [5], [6], [90], [99], [141], [143], [151], and [166]. The schemes from [6], [90], [99], and [143] are realized as normal checkers. On the other hand, the other checkers apply to some I only, namely: [5] — $I = 2^K - 1$, [141] — $I \in \{2^K - 2, 2^K - 1\}$, and [151] — most I, except those I which are only slightly larger than 2^{K-1}. The latter schemes do not distinguish between information and check bits and realize some functions defined on all input bits, so that all the circuitry can be implemented as IF. The checkers from [6] and [99] employ some non-optimal counters of 1s built of FAs and HAs. In [90] a special design of a counter of 1s is proposed. It has $I + 1$ inputs, since it operates on I information bits extended by the most significant check bit, and is built of a number of cellular symmetric function circuits followed by an MN. An STC for Berger codes proposed by us in [143] is the first STC for any systematic UEDC that is built using a T^n circuit. Here, we shall concentrate on the two most efficient designs: T^n-based and those using parallel counters.

Having available various counters of 1s, an STC for any Berger code with $I \neq 2^{K-1}$ can be easily realized as a normal checker from Fig. 3.1. This is not the case of an STC for a Berger code with $I = 2^{K-1}$ that requires special consideration. Although an STC for this code was recently proposed in [166], a circuit is significantly more complex than the one described below for an equivalent code.

7.1.3. STCs for Modified Berger Codes with $I = 2^{K-1}$

The codes that have an information part length I being a power of 2 are of primary importance in practical applications. The STC for a Berger code

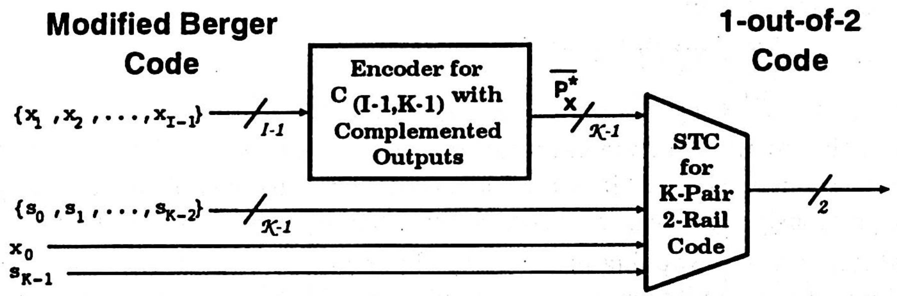


Figure 7.1. STC for a modified Berger code $C_{(I,K)}$ with I=2^{K-1} bits

with $I = 2^{K-1}$ cannot be implemented as a 2-output normal checker. This is because the combinational K-pair 2-rail STC (the block N_2), which is tested for all s/z faults and has only one codeword for which the bit s_{K-1} is 0, is not known. However, an STC can be designed for a *modified Berger code* $C^*_{(I,K)}$ with $I = 2^{K-1}$ defined in [5], as we have shown in [141] and [143].

The modified Berger code can be represented as

$$C^*_{(I,K)} = C_{(I',K')} \times C_{1/2}, \tag{7.1}$$

where $I' = I - 1 = 2^K - 1$ and $K' = K - 1$. The $C^*_{(I,K)}$ code is formed by concatenating the MLB code $C_{(I',K')}$ with the 1/2 code. The 1/2 codewords are formed by the least significant information bit x_0 and the most significant check bit s_{K-1}. The remaining $I+K-2$ bits $\{x_{I-1}, \ldots, x_2, x_1, s_{K-2}, \ldots, s_0\}$ constitute the words of the MLB code $C_{(I',K')}$. An encoder for the $C^*_{(I,K)}$ code consists of any encoder for the MLB $C_{(I-1,K-1)}$ code and an inverter to generate the most significant check bit $s_{K-1} = \bar{x}_0$. Thus, the complexity of the encoder is virtually the same as for the MLB code, since not I but only $I-1$ bits contribute to its complexity. Its version using a parallel counter is extremely efficient, since any MLB code employs a saturated parallel counter of 1s. (Recall that a parallel counter is known to be at its best with respect to all parameters when it is saturated.)

The structure of the STC for the $C^*_{(I,K)}$ code is given in Fig. 7.1. Obviously, either version of the encoder with complemented outputs described above can be used. Since the K-pair 2-rail STC receives all 2^K 2-rail codewords, it can be implemented in any version (including 2-level).

The principal advantage of the above modified Berger code is that it allows us to design an STC which is extremely cost-efficient—an STC for the $C^*_{(I,K)}$ code is only slightly more complex than an STC for the MLB $C_{(I-1,K-1)}$ code, as it requires just one extra module of the 2-pair 2-rail STC. As for the encoder for the $C^*_{(I,K)}$ code—it consists of any counter of 1s on the bits $\{x_{I-2}, \ldots, x_1, x_0\}$ and an inverter that produces $s_{K-1} = \bar{x}_0$. More details about this checker can be found in [141].

7.1.4. Minimal Test Set for an STC for Berger Codes

The minimal test set $\mathbf{T}(C_{(I,K)})$ for an STC for the Berger code with $I \neq 2^K$, formed by selecting a subset of codewords of $C_{(I,K)}$, is generated in the following way.

1. Derive $\mathbf{T}_{min}(I; K)$, the minimal test set for the I-input counter of 1s, which must be complete for the T^n-based counter of 1s.
2. Derive $\mathbf{T}(C_{(I,K)})$ on the basis of $\mathbf{T}_{min}(I; K)$ by concatenating each vector J_X in $\mathbf{T}_{min}(I; K)$ with the suitable check part P_X of the Berger code $C_{(I,K)}$.

Obviously, $|\,\mathbf{T}(C_{(I,K)})| \geq \max\{|\mathbf{T}_{min}(I; K)|, |\mathbf{T}_{min}(C_{2r}|)\}$, i.e. the testability of any STC for the Berger code directly depends on the testability of both a counter of 1s used and an STC for an incomplete K-pair 2-rail code C_{2r}. The above procedure of generating $\mathbf{T}_{min}(I; K)$ applies to an STC using any implementation of the counter of 1s. Assuming temporarily that $|\mathbf{T}_{min}(I; K)| \geq |\,|\mathbf{T}_{min}(C_{2r})|$, which is justified, since many easily-testable STCs for any incomplete 2-rail code used here are known (see Subs. 3.3), we can concentrate on the size of $\mathbf{T}_{min}(I; K)$. In particular, if an STC employs the optimal parallel counter, then $|\mathbf{T}(C_{(I,K)})| \leq 8K$, but it is a very loose upper-bound since in reality $|\mathbf{T}(C_{(I,K)})|$ is close to 8 (see Section 4). If an STC employs an encoder with complemented outputs using T^n realized as an SN, then $|\mathbf{T}(C_{(I,K)})| = n + \lceil n/2 \rceil$. However, it should not be forgotten that the outputs produced by the circuit N_1 for vectors from $\mathbf{T}(C_{(I,K)})$ should sufficiently exercise the circuit N_2 — the checker for the K-pair 2-rail code. We have shown in Subs. 3.3 that the largest 2-rail module used for N_2 must not have more than K^* pairs of inputs, where $2K^* \leq |\mathbf{T}_{min}(I; K)|$.

The minimal test set $\mathbf{T}(C_{(I,K)})$ for an STC for the modified Berger code with $I = 2^K$ is formed in a similar way, except that it is generated on the basis of the $\mathbf{T}_{min}(I-1; K-1)$, and hence its capacity depends on $|\mathbf{T}_{min}(I-1; K-1)|$.

7.1.5. Parameter Estimation and Comparison

Here we shall compare various realizations of the STCs for some Berger codes with $I \neq 2^{K-1}$ and modified Berger codes with $I = 2^{K-1}$. The new checkers employing two versions of counters of 1s described in Section 4 will be compared against other existing designs. The figures collected in Table 7.1 characterize the

Table 7.1. Paramaters of various STCs for Berger codes

Code	Gates			Levels		
I	New	[99]	[90]	New	[99] ‡	[90]
6	45	52	36	12	13	9
7	53	58	48	11	11	9
8	59	72	52	11	19(11)	11
14	144	142	155	17	19	11
15	156	150	160	15	17	11
16	162	166	164	15	25(17)	13
31	435	346	574	20	23	13
32	441	356	585	20	31(29)	13
64	1185	834	$\gg$1600	26	37(29)	17
	† 954			26		

† This version of the STC uses the optimal 63-input SN by Van Voorhis [205]
‡ In parentheses we give the number of levels in the STC using the optimal parallel counter, provided that it is smaller than in [99]

following implementations of the STCs. Every circuit is a normal checker which employs the fastest 2-rail checker that is allowed: (i) 2-level — in the checkers from [99] and presented here for $I \in \{2^K - 1, 2^{K-1}\}$, and in the checkers from [90] for $I = 2^K - 1$ only; and (ii) 4-level from Fig. 3.2 — in all other cases. The checkers from [99] are built of an easily-testable parallel counter (also proposed in [99]) composed by $I - K$ FAs in $2K - 3$ stages (note that delay estimations given for this checker in [90] are incorrect). The checkers from [99], besides being the slowest, have also the disadvantage of having four outputs for $I = 2^{K-1}$, unlike any other checker. As for the checkers presented here, one version which employs the optimal parallel counter, differs from a similar circuit from [99] in the number of FA stages for some codes (see delay figures given in parentheses). The other T^n-based version is built of the optimal SN followed by I inverters, I 2-input AND gates, and K OR gates with a total of about $I + K$ inputs. The complexity figures of all checkers using parallel counters were derived assuming that an FA is built of 12 gates in three levels and by ignoring a small number of HAs.

The checkers using the optimal parallel counters suggested here are asymptotically the least complex — $O(I)$ gates and gate inputs, and the fastest — $O(\log I)$ gate levels, but these advantages materialize for very large I. The T^n-based checker can be the best choice when high speed achieved at a reasonable cost is of primary importance and I is not too large (say, $I \leq 64$). The other advantages enjoyed by both the T^n-based encoder and the STC for Berger code, important for VLSI implementation, are: (i) the design and the minimal test generation are

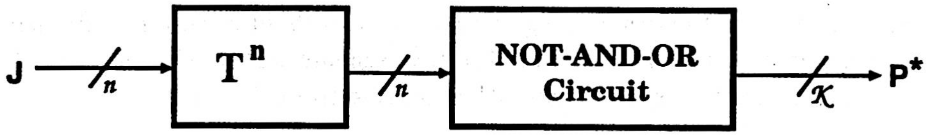


Figure 7.2. General structure of various T^n-based circuitry supporting the use of systematic UEDCs

straightforward (it will be seen later that this is true for any systematic UEDC); (ii) regular structure — the circuit T^n is entirely built using simple cells of one type and only some of K OR gates have more than two inputs; (iii) speed; and (iv) high testability.

7.2. STCs for Systematic t-UEDCs

Here we shall consider the encoders and STCs for the best known t-UEDCs which are the least redundant—two classes of Bose-Lin codes [17], and those which for some K are capable of detecting u-errors of the largest multiplicity t—the Jha-Vora codes [71]. (For more details on these codes, refer to Subsection 2.4.1.) This and remaining subsections present extentions of our earlier results from [144].

7.2.1. Encoders for Bose-Lin Codes

An encoder for Code 1 is built of: (i) a mod 2^K counter of 0s — when $K \in \{2,3\}$; and (ii) a mod 2^{K-1} counter of 0s and one inverter — when $K \geq 4$.

An encoder for Code 2 is conceptually more complex than for Code 1, since it requires a mod $3 \cdot 2^{K-3}$ counter of 0s and a 3-input 4-output translator of the three MSBs from the counter. No particular hardware implementation of the encoder for Code 2 has been given either in [17] or [68]. Naturally, the design of an STC for Code 2 has also not been given in the literature yet.

The Bose-Lin codes use mod A counters of 0s as basic modules, $A \in \{2^K, 2^{K-1}, 3 \cdot 2^{K-3}\}$. Any counter of 0s can be implemented with a counter of 1s whose inputs are complemented, and therefore we shall explicitly consider only the mod A counters of 1s. On the other hand, as we have already pointed out in Subsection 4.2, for the T^n-based realizations, by their nature, the form of inputs is immaterial, i.e. they employ the same structure from Fig. 7.2, no matter whether 1s or 0s are counted.

Here we shall propose two schemes of encoding circuitry, conceptually similar to those proposed for the Berger codes: one using a modified optimal parallel

$(I;q)$ counter, and the other using the I-input threshold circuit T^n. By using the same argument as given in Section 4, either scheme will be shown more efficient both in hardware and speed than either scheme from [17] or [68]

A. Mod 2^K Parallel Counters of 1s

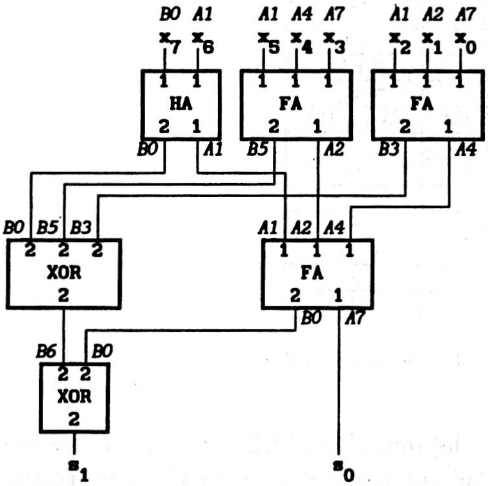
The first realization of the I-input mod 2^K parallel counter of 1s was given along with the construction of the Bose-Lin t-UEDCs [17]. It is designed as a tree-type circuit which contains i-bit ripple-carry adders at level i, where $i \in \{1, 2, \ldots, K-1\}$, and K-bit ripple-carry adders with ignored carry of weight 2^K in the remaining v levels. A scheme is therefore built using three types of modules: FAs, HAs, and 2-input XOR gates, which total for more than I modules. It introduces the delay equal $\left[1 + (2 - \frac{1}{2}) + (3 - \frac{1}{2}) + \cdots + (K - 1 - \frac{1}{2}) + (K - \frac{1}{2})v\right] \cdot \Delta$ which simplifies to $\left[\frac{1}{2}K^2 - K + (K - \frac{1}{2})v\right] \cdot \Delta$. (Here we assume that the maximal delay Δ—introduced by two XOR gates of an FA—is equivalent to four gate levels; consequently, the delay of an HA or a 2-input XOR gate equals $\frac{1}{2}\Delta$.) An improved design was given recently by Jha [68], who modified the parallel counter of 1s from [99]. He showed that it requires $8K$ tests for $K \in \{2, 3\}$ and $8(K-1)$ tests for $K \geq 4$.

A new mod 2^K parallel counter of 1s can be easily derived from the parallel $(n;q)$ counter given in [76]. It is also built using FAs and HAs, as well as some XOR gates, i.e. it uses the same modules as the mod 2^K parallel counters from [17] and [68]. The design is explained through the following example.

Example 7.2. *The scheme of the new 8-input mod 4 parallel counter of 1s is given in Fig. 7.3. The same shorthand notation that we have used for ordinary parallel counters is employed. The part of the mod 2^K parallel counter that operates on the inputs of weight $w < 2^{K-1}$ (i.e. from the columns G_j, $0 \leq j < K-1$), uses only FAs and HAs. The only modification is that the inputs of weight 2^{K-1} (i.e. from the column G_{K-1}) are handled by XOR gates only. This is because the carries that could be generated by FAs and HAs with inputs of weight 2^{K-1} can be ignored in the mod 2^K counter.*

The above circuit is built of three FAs, one HA, and three 2-input XOR gates and introduces the delay of 2.5Δ. All the necessary data on the minimal test set that we derived for this counter are included in Fig. 7.3(A). (Note also that for this test set, all four combinations occur at the output of the encoder, and hence, when it is used to build an STC, the 2-rail checker is fully tested as well.) Table 7.2 which compares two existing schemes of the 8-input mod 2^K parallel counter with the one given above, clearly shows the superiority of the latter, primarily with respect to the delay and the test set size. □

(A)



(B)

G_1	G_0
—	8
—	HA FA FA
3	3
XOR XOR	FA
2	1
XOR	—
1	1

Figure 7.3. New 8-input mod 4 parallel counter of 1s:
Logic scheme (A); Shorthand notation (B)

Table 7.2. Paramaters of various 8-input mod 4 parallel counters

Version	FAs	HAs	XORs	Stages [Δ]	Tests
[17]	—	7	6	3.5	?
[68]	3	1	3	3.5	16
New	3	1	3	2.5	8

Table 7.3. Paramaters of various 24-input mod 8 parallel counters

Version	FAs	HAs	XORs	Stages [Δ]	Tests
[17]	15	7	6	7.5	?
[68]	15	4	5	6	24
New	16	2	5	4.5	10

Table 7.3 shows similar improvements of the new design for the 24-input mod 8 counter of 1s (shown in Fig. 7.4). (The minimal test set for the mod 2^K counter of 1s was given neither in [17] nor in [15].)

A general systematic method of generating the minimal test set for a mod 2^K parallel counter of 1s is a complex open problem because, as it was pointed out, the parallel counters using CSAs are not unique. Nevertheless, it can be

G_2	G_1	G_0
—	—	24
—	—	8 FAs
—	8	8
—	FA FA HA	FA FA HA
3	6	3
XOR XOR	FA FA	FA
3	3	1
XOR XOR	FA	—
3	1	1
XOR	—	—
1	1	1

Figure 7.4. New 24-input mod 8 parallel counter of 1s

shown, by using the argument similar to the one given in Section 4 for ordinary parallel counters, that the upper-bound on the test set size for the new counter is the same as derived by Jha [68] for his design, i.e. $\leq 8K$ — for $K \in \{2,3\}$, and $8(K-1)$ — for $K \geq 4$.

The overall number of modules (FAs, HAs, and 2-input XOR gates) used in the new I-input mod 2^K parallel counter of 1s equals $I-1$: the reduction from I to K bits is done with a total of $I-K$ FAs and 2-input XORs, whereas at most $K-1$ HAs can be needed to improve the distribution of bits of various weight, at a given stage of computation. This figure is useful while evaluating hardware savings in encoders (and STCs) for Bose-Lin codes compared to the same circuits for Berger codes with the same I.

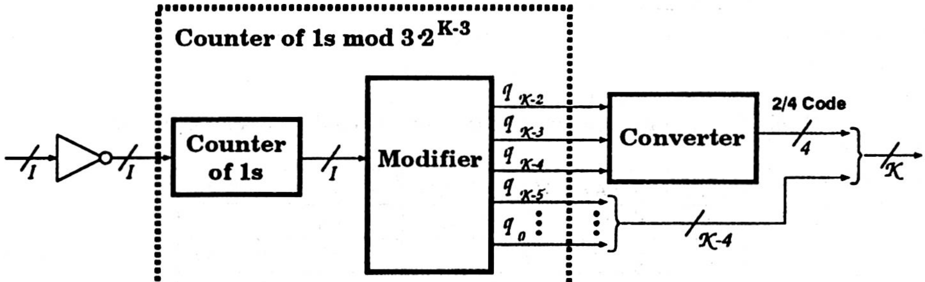


Figure 7.5. Internal structure of the mod $3{\cdot}2^{K-3}$ parallel counter of 1s

B. Parallel mod $3 \cdot 2^{K-3}$ Counter of 1s

To date, no particular realization of a mod $3 \cdot 2^{K-3}$ counter of 1s has been proposed. The general scheme of a mod $3 \cdot 2^{K-3}$ parallel counter of 1s is shown in Fig. 7.5. The parallel counter of 1s has $K = \lceil \log_2(I+1) \rceil$ outputs q_i, $0 \leq i \leq$

Table 7.4. Encoding table of the 8-bit 2-UED Bose-Lin code

WEIGHT	CHECK PART	
	s_1	s_0
0	0	0
1	1	1
2	1	0
3	0	1
4	0	0
5	1	1
6	1	0
7	0	1
8	0	0

$K-1$. No general functions have been found for a modifier that generates the mod $3 \cdot 2^{K-3}$ output, except for $q_i = z_i$ for $0 \leq i \leq K-5$. The following minimized general functions of the converter to the 2/4 code were derived assuming that (000) is encoded as (0011), (001) as (0101), etc., in the increasing order:

$$\begin{aligned} s_{K-1} &= q_{K-2} + q_{K-3}q_{K-4} \\ s_{K-2} &= q_{K-3} \oplus q_{K-4} \\ s_{K-3} &= \overline{q_{K-3}} \\ s_{K-4} &= \overline{q_{K-3} \oplus q_{K-4}} \end{aligned}$$

The parallel counter of 1s is tested exhaustively during normal functioning. The converter to the 2/4 code is tested for all s/z faults by six combinations used, provided that (110) and (111) are treated as don't cares. We conjecture that the minimized modifier can be tested for all s/z faults, except the case when $I = 2^p$ (the most important for practical applications). This is because for $I = 2^p$ there is only one combination — (100...0) — which has 1 on the MSB of the output produced by the parallel counter of 1s. As a result, this encoder cannot be used to build an STC for any Bose-Lin Code 2 with $I = 2^p$.

C. T^n-Based Encoder for Any Bose-Lin Code

A T^n-based encoder and a complemented check bits generator can be easily built for either of two Bose-Lin codes. Both circuits are conceptually similar and use the same structure as for Berger codes (see Fig. 7.2).

Example 7.3. *Design the encoder for the 2-UED Bose-Lin code, i.e. the 8-input mod 4 counter of 1s, by using the circuit T^8. The encoding is given in Table 7.4. Assume that the 8-input threshold circuit T^8 is available (see the logic scheme in*

Table 7.5. Encodings of the t-UED Bose-Lin Code 2 with K=5 (Encoding 1) and burst 3-UED Blaum code for K=3 (Encoding 2)

WEIGHT	Encoding 1					Encoding 2		
	s_4	s_3	s_2	s_1	s_0	s_2	s_1	s_0
0	1	1	0	0	1	1	1	1
1	1	1	0	0	0	0	1	1
2	1	0	1	0	1	1	0	1
3	1	0	1	0	0	1	1	0
4	1	0	0	1	1	0	0	1
5	1	0	0	1	0	0	1	0
6	0	1	1	0	1	1	0	0
7	0	1	1	0	0	0	0	0
8	0	1	0	1	1	1	1	1
9	0	1	0	1	0	0	1	1
10	0	0	1	1	1	1	0	1
11	0	0	1	1	0	1	1	0
12	1	1	0	0	1	0	0	1
...		.	.	.		.	.	.

Fig. 4.5). The functions of the NOT-AND-OR circuit are the following:

$$s_1 = T_2^8 \cdot \overline{T}_4^8 + T_6^8 \cdot \overline{T}_8^8$$
$$s_0 = T_1^8 \cdot \overline{T}_2^8 + T_3^8 \cdot \overline{T}_4^8 + T_5^8 \cdot \overline{T}_6^8 + T_7^8 \cdot \overline{T}_8^8.$$

The circuit is built of 50 gates with a total of 98 inputs and requires 12 tests. It compares favorably to three alternative implementations whose parameters are listed in Table 7.2: it is faster (9 gate levels) than the circuit from [68] (14 gate levels), and slightly faster than our mod 4 parallel counter of 1s (10 gate levels). It is slightly more complex than our parallel counter: it uses 50 vs. *43 gates.* □

Example 7.4. *Design the T^n-based encoder for the Bose-Lin Code 2 for some $I \geq 16$ and $K = 5$.*

Encoding 1 in Table 7.5 shows the check part encoding of the t-UED Bose-Lin Code 2 with $K = 5$ for the inputs of weight $0 \leq w \leq 12$ only, as for $w = 12$ the sequence of 12 different check parts repeats. The functions of the T^n-based encoder can be easily generated by inspection of Table 7.5. For instance, the functions s_4 and s_3 are:

$$s_4 = \overline{T}_6^I + T_{12}^I \cdot \; \ldots$$
$$s_3 = \overline{T}_2^I + T_6^I \cdot \overline{T}_{10}^I + T_{12}^I \cdot \; \ldots . \qquad \square$$

Note: A T^n-based encoder seems suitable for smaller I, but its competitiveness with a parallel counter version (in terms of complexity) increases with K (a Berger code is an extreme case). This is because for larger I ($I \geq 16$) the complexity

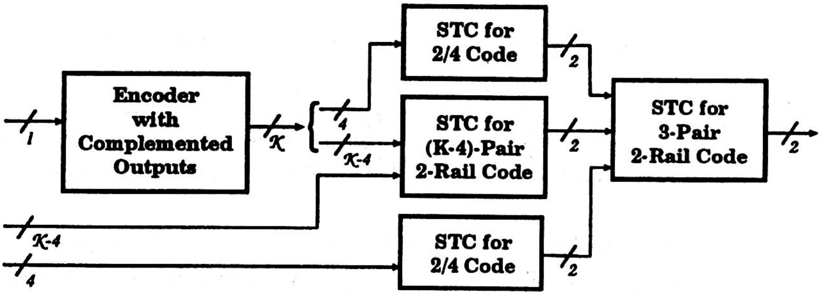

Encoder with Complemented Outputs
I
K
4
K-4
STC for 2/4 Code
STC for (K-4)-Pair 2-Rail Code
STC for 2/4 Code
2
2
2
STC for 3-Pair 2-Rail Code
2
K-4
4

Figure 7.6. Special STC for Bose-Lin t-UED Code 2

of T^n circuits grows faster than of the mod A parallel counter wherein all FAs operating on the MSBs are replaced with XOR gates.

7.2.2. STCs for Bose-Lin Codes

The design of an STC for Bose-Lin Code 1 has been considered only by Jha [68] and by us [144]. In [68], it was the normal checker as in Fig. 3.1, wherein an encoder with complemented outputs consists of a mod 2^K parallel counter preceded and followed by I and K inverters, respectively. A mod 2^K parallel counter was built by modifying the parallel counter from [99] and it was shown that the resulting checker requires at most $8K - 4$ codeword tests (we gave some details in Section 4). However, no particular STC has been given in the literature for the Bose-Lin Code 2 [17].

With the new encoders described above, two efficient realizations of an STC for the Bose-Lin Code 1 can be designed fairly easy now. They only differ in the circuitry used to implement the encoder with complemented outputs:

(i) An improved mod 2^K parallel counter preceded and followed by I and K inverters, respectively; and

(ii) Direct T^n-based implementation, as in Fig. 7.2, which does not use inverters neither on its inputs nor on its outputs.

The minimal test set for an STC realized using any of the two circuits is derived on the basis of the minimal test set for the encoder with complemented outputs in the same way as for an STC for the Berger code.

Basically, for any code a normal checker from Fig. 3.1 can be designed. In particular, a complemented check bits generator can be designed as an encoder followed by a bank of K inverters, whereas with the T^n circuit a direct implementation is feasible. An encoder of the structure from Fig. 4.1 or Fig. 7.3, designed by using any method proposed here, can be used. However, for Bose-Lin Code 2

Table 7.6. Encoding table of the Jha-Vora code with I=18

Weight	0	1	2	3	4	5	6	7	8	9
Check Part	1111	1110	1101	1100	1011	1010	1001	0111	0110	0101
Weight	10	11	12	13	14	15	16	17	18	
Check Part	0011	1100	0100	1010	0010	1001	1000	0001	0000	

which employs a very special encoding scheme and for which no particular STC has been given in the existing literature, we propose a special STC, shown in Fig. 7.6. The use of two STCs for the 2/4 code results in slightly better performance than that of the normal checker. Code-disjointness of the above circuit follows from the fact that no u-error can transform one 2/4 codeword into another 2/4 codeword.

7.2.3. Encoder and STC for Jha-Vora Codes

An encoder for the Jha-Vora code proposed in [71] consists of the counter of 0s followed by a modifier circuit implemented with a PLA or a ROM. An important disadvantage of any Jha-Vora code is that its check part, in general, does not use all 2^K combinations. As a result, in most cases, a checker using such a circuit is not ST for all faults, since a PLA as well as a ROM must be tested exhaustively. And, indeed, no STC for the Jha-Vora code has been given in the literature yet.

Here we will consider the Jha-Vora code with $I = 18$ to show that the design of an encoder and an STC for this class of codes is straightforward as the T^n-based circuit.

Example 7.5. *Design an encoder for the Jha-Vora code with $I = 18$.*
The check bits assignment is shown in Table 7.6. The encoder implements the functions:

$$\begin{aligned}
s_3 &= \overline{T}_7^{18} + T_{11}^{18} \cdot \overline{T}_{12}^{18} + T_{13}^{18} \cdot \overline{T}_{14}^{18} + T_{15}^{18} \cdot \overline{T}_{17}^{18} \\
s_2 &= \overline{T}_4^{18} + T_7^{18} \cdot \overline{T}_{10}^{18} + T_{11}^{18} \cdot \overline{T}_{13}^{18} \\
s_1 &= \overline{T}_2^{18} + T_4^{18} \cdot \overline{T}_6^{18} + T_7^{18} \cdot \overline{T}_9^{18} + T_{10}^{18} \cdot \overline{T}_{11}^{18} + T_{13}^{18} \cdot \overline{T}_{15}^{18} \\
s_0 &= \overline{T}_1^{18} + T_2^{18} \cdot \overline{T}_3^{18} + T_4^{18} \cdot \overline{T}_5^{18} + T_6^{18} \cdot \overline{T}_8^{18} + T_9^{18} \cdot \overline{T}_{11}^{18} + T_{15}^{18} \cdot \overline{T}_{16}^{18} + T_{17}^{18} \cdot \overline{T}_{18}^{18}
\end{aligned}$$

The functions of the complemented check bits generator needed to implement the STC are dual to the functions of the encoder (such as those given in Ex. 7.5) and can be derived from them by using the method described in Section 4. The proof that a normal checker for the Jha-Vora code built by using the above circuit is ST is similar to the argument given earlier in Subs. 7.1. It can be shown that the STC for Jha-Vora code with $I = 18$, designed similarly as in Ex. 7.5, is completely tested by applying 27 selected codewords.

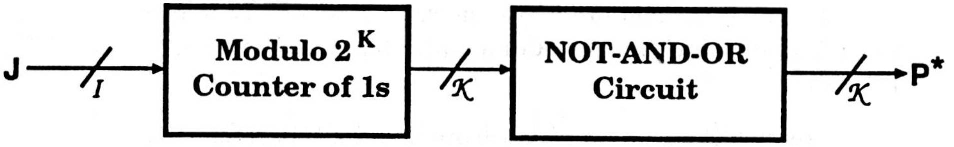


Figure 7.7. General structure of an encoder for the BUEDCs, using a parallel counter

7.3. STC for BUED Codes

The STCs for BUED codes by Bose [15] and Blaum [12] have been given in [15], [68], and [144]. All of them are conceptually similar to the STCs for systematic t-UEDCs already presented in Subs. 7.2. Here we shall briefly present the least complex and the fastest designs proposed by us in [144].

For the Bose and Blaum BUEDCs (see Subs. 2.4.2) the two approaches can be used to build encoders and STCs. One employs the scheme from Fig. 7.7, i.e. the encoder is built of a mod 2^K parallel counter of 1s (the same as used for the Bose-Lin codes) and the NOT-AND-OR circuit, which is the same in Blaum's and our design. The latter circuit, in general, requires all 2^K (up-to-K)-input AND gates which feed K OR gates. Since the logic minimization allows us to decrease the total number of inputs to the AND gates only, its PLA or ROM implementation is feasible. However, the PLA- or ROM-based encoder cannot be used to build an STC, since it may not be ST for faults specific for PLAs and ROMs.

The T^n-based encoder for the Blaum code uses the structure from Fig. 7.2 and is designed in the same way as for other UEDCs already presented in this section. The complexity of the AND-OR part of the encoder for the Blaum BUEDC with a given K can be found in the following way. The exact number of AND gates, $AND(K)$, derived from Tables I–IV in [12] for $I = 2^K$ is: $AND(3) = 4$, $AND(4) = 14$, $AND(5) = 34$, and $AND(6) = 75$. Clearly, the number of AND gates grows almost linearly with I for any K of practical importance. The total number of inputs to K OR gates equals $AND(K)+K$. An STC for any BUEDC can be built as a normal checker by using any scheme of an encoder.

8. CONCLUSIONS

Two principal problems in the design of fault-tolerant computers can be solved at a reasonable cost by using error detecting codes (EDCs) and implementing circuits as self-checking:

1. How to protect a computer system against undetected errors caused by internal faults which have predominantly temporary character (over 90% of all faults)?
2. How to reduce the amount of hardcore in a digital system to the minimum?

Since the assumption of single error occurrence may not be valid for certain types of digital circuits (e.g. using shared logic) or/and internal failures (e.g. in a power supply line), the means for detection of multiple errors may be needed. Unidirectional errors represent a well defined class of multiple errors which have been observed as the result of many failures of VLSI circuits. Reliable operation of a checker, i.e. a circuit that monitors whether the data produced by a functional circuit represent codewords of some EDC, can be ensured even in the presence of internal failures in it, by implementing it as self-testing.

There exists a large body of literature presenting several design methods for self-testing checkers (STCs) for various classes of unidirectional EDCs (UEDCs). Despite many similarities shared by different UEDCs, the state-of-the-art in the field of designing the STCs for these codes can be seen as a collection of ad hoc methods rather than a coherent uniform theory. We have noticed that the performance of the STCs designed by using some of these methods could have been improved for many codes, provided that the best available basic building blocks were known to the designers. Moreover, there have been some classes of UEDCs for which the STCs have not been known to date.

This monograph is an attempt to present the systematic approach to designing cost-effective and fast STCs for the most important known classes of UEDCs. We have concentrated on the design methods which generate circuits which are suitable for VLSI implementation.

In Section 2 we have given the systematic presentation of the basic properties of EDCs with the emphasis on the concepts and applications of UEDCs. In particular, we have observed that we can take advantage of the fact that, for all UEDCs considered, checking the weight of a binary vector (i.e. the number of 1s in it) is the starting point for determining whether it is a codeword of some UEDC.

In Section 3 we have provided the theoretical background for designing a wide spectrum of self-checking circuits, including self-checking functional circuits. However, the emphasis is on the properties of STCs which are the main topic of this monograph. To allow for quantitative evaluation and comparison of alternative realizations of an STC for the same code, the STCs will be characterized by the numbers of gates, gate inputs, levels, and tests.

In Section 4 we have surveyed existing realizations of circuits that can be used to determine the weight of a binary vector in the most efficient way. Two conceptually different classes of circuits have been distinguished:

1. *Parallel counters*, which can be entirely built of full- and half-adders and operate in predominantly carry-save mode; and
2. *Multi-output threshold circuits* T^n with all inputs of weight 1, which can be implemented in the most efficient way as a special case of sorting networks (SNs) using simple cells composed of a pair of 2-input AND and OR gates.

Although parallel counters and circuits T^n have been studied extensively for many years, the most efficient designs have been unknown to the researchers working on fault-tolerant hardware. On one hand, most results on parallel counters appeared in the works on arithmetic circuits, in particular, multipliers. We have shown that some parallel counters are not only faster (at no cost) than commonly used circuits from [99], but also that they require only slightly more than eight tests. On the other hand, the most efficient results on multi-output threshold circuits T^n appeared in the literature on SNs, with applications in computer algorithms rather than logic design. Only recently the author observed the above and proved that the circuits T^n implemented as SNs, besides uniform cellular structure and superb hardware and speed performance, also require the minimal number of tests which is about $3n/2$.

In Section 5 we have presented the most efficient schemes of STCs for those m/n codes which have the best potential for practical applications. These include the most frequently used optimal unordered codes, i.e., the $m/2m$ and $m/(2m+1)$ codes, as well as the $2/n$ and $1/n$ codes. The STCs for the $m/2m$ and $m/(2m+1)$ codes are built using circuits T^n only. This is because the only known STC for $m/2m$ codes built using parallel counters proved more complex and slower than its T^n-based counterpart for any m of practical value. In fact, both the cost-effective and the high-speed version of an STC for the $m/2m$ and $m/(2m+1)$ codes proposed here is the least complex and requires the fewest number of tests compared to any other similar design. The STCs for $2/n$ codes given here are designed by proper partitioning of the elementary threshold funtion T_2^n, whereas the STCs for $1/n$ codes are built using the translator of the $1/n$ code to some $2/n'$ code and the STC for the $2/n'$ code. The number of gate inputs in the new STC for the $1/n$ codes grows linearly as $O(2n)$ compared to at least $O(4n)$ in all other designs.

In Section 6 we have proposed the design method of the STCs for Borden codes, which are the optimal t-UEDCs. We have shown that a checker for any Borden code can be built using a pair of circuits T^n, based on logic functions derived here. This is unlike most existing Borden code checkers which are not only conceptually complex, but also significantly more costly in terms of the amount of hardware and delay. Similarly to most STCs for nonsystematic m/n codes, an STC using parallel counters is available for only a few Borden codes, mainly due to excessive complexity and design difficulties.

In Section 7 we have presented the new design methods of both encoders and STCs for various systematic UEDCs. Since any UEDC uses a check part generated on the basis of the weight of an information part, an efficient counter of 1s (or 0s) is the central element used to build an encoder or an STC for any of these codes. For all codes considered; i.e. unordered Berger and equivalent codes (which are all-UEDCs), t-UEDCs, and burst-UEDCs, two versions of both an encoder and an STC have been presented. These two versions — one employing a parallel counter and the other a T^n-based circuit — are in a sense complementary, as for smaller n the T^n-based circuits are generally more efficient than those built of parallel counters. It is worth to point out that the design of any T^n-based encoder and an STC, suggested here, is extremely easy for *any* UEDC. In particular, we have managed to design the STCs for t-UED codes: Code 2 by Bose and Lin [17] and all the codes by Jha and Vora [71], for which no implementation of an STC has been known before. All new checkers proved to be faster, less complex, and easier to test than similar circuitry which has been built on the basis of a commonly used easily-testable parallel counter composed of a number of ripple-carry adders.

The results presented in this monograph show that very cost-effective and fast STCs for all UEDCs considered and encoders for systematic UEDCs can be built. The author hopes that they will help in using UEDCs in digital systems more often, as the amount of hardware overhead required for concurrent error detection can be significantly reduced if the circuitry presented here is used. Also, the design methods presented here prove that realization of STCs and encoders for UEDCs is not such a difficult design task anymore. The practical importance of these results is also linked to the growing awareness that using self-checking circuits is becoming mandatory even in general-purpose computers.

All circuits presented here have many advantages which make their VLSI implementation feasible, such as: (1) a highly regular structure, (2) a relatively small delay compared to other similar circuits; and (3) easy testability with a minimal test set which is easy to find. Not less important advantage of the approaches presented here is straightforward design, no matter how complex the algorithm of generating the check bits is, provided that the check bits are derived on the basis of weight of the remaining part of a codeword (which is the case of all known UEDCs). Obviously, the design concepts presented here can be extended to design an encoding/decoding circuitry for any existing or future UEDC and d-EC/UEDC in which the check bits are a function of the weight of the information part.

The major problems requiring future research and related to this monograph, that still remain open, include the analysis of the suitability of the presented alternative designs for particular VLSI technologies, such as CMOS, by taking into account stringent fan-in and fan-out limitations, the feasibility of using complex gates, and an extended, perhaps more accurate, fault model which includes e.g. delay faults. Another issue is the design of encoding/decoding circuitry for numerous classes of codes which combine unidirectional or asymmetric error detection with correction of symmetric errors. This is because the efficiency of these codes strongly depends on the speed and complexity of encoders and decoders and because encoding/decoding algorithms described by simple formulas and procedures do not necessarily transfer into equally efficient hardware. Not less important are more specific topics related to extending our earlier work [148] on finding the minimal test sets for the optimal but highly irregular SNs by Van Voorhis [205], as well as finding minimal test sets for non-saturated optimal parallel counters and their derivatives. Finally, the methods for systematic design of totally self-checking functional circuits, protected by various UEDCs other than already used unordered codes (which are potentially cheaper to implement) are needed. Their future developments should be supported by a thorough analysis of their suitability and potential hardware cost efficiency compared to similar circuits protected by using unordered and other EDCs. Obviously, the ultimate goal of this and future research on the topics presented here is how to design a totally self-checking yet cost-efficient computer system. Its feasibility will strongly depend on future development of automatic design, optimization, and verification tools to support self-checking design of complex digital modules.

REFERENCES

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital System Testing and Testable Design*, New York, Computer Science Press, 1990.

[2] D. A. Anderson and G. Metze, "Design of totally self-checking check circuits for m-out-of-n codes," *IEEE Trans. Comput.*, vol. C-22, pp. 263–269, Mar. 1973.

[3] T. O. Anderson, "Modular switching network for generating the weight of a binary vector," *Comput. Design*, vol. 2, pp. 106–110, Apr. 1972.

[4] D. B. Armstrong, A. D. Friedman, and P. R. Menon, "Design of asynchronous circuits assuming unbounded gate delays," *IEEE Trans. Comput.*, vol. C-18, pp. 1110–1120, Dec. 1969.

[5] M. J. Ashjaee and S. M. Reddy, "Totally self-checking checkers for a class of separable codes," in *Proc. 12th Annu. Allerton Conf. Circuit and System Theory*, Monticello, IL, Oct. 1974, pp. 238–242.

[6] M. J. Ashjaee and S. M. Reddy, "On totally self-checking checkers for separable codes," *IEEE Trans. Comput.*, vol. C-26, pp. 737–744, Aug. 1977.

[7] A. Avižienis, H. Kopetz, and J. C. Laprie, Eds., *The Evolution of Fault-Tolerant Computing*, Springer-Verlag, Wien and New York, 1987.

[8] K. E. Batcher, "Sorting networks and their applications," in *Proc. 1968 SJCC*, AFIPS, vol. 32, 1968, pp. 307–314.

[9] J. M. Berger, "A note on error detection codes for asymmetric binary channels," *Inform. Contr.*, vol. 4, pp. 68–73, Mar. 1961.

[10] H. J. Beuscher and W. N. Toy, "Check schemes for integrated microprogrammed control and data transfer circuitry," *IEEE Trans. Comput.*, vol. C-19, pp. 1153–1159, Dec. 1970.

[11] R. E. Blahut, *Theory and Implementation of Error Correcting Codes*, Addison-Wesley, Reading, MA, 1983.

[12] M. Blaum, "Systematic unidirectional burst detecting codes," *IEEE Trans. Comput.*, vol. C-37, pp. 453–457, Apr. 1988.

[13] M. Blaum (Ed.), *Unidirectional Error Control Codes*, IEEE Press, 1992.

[14] J. M. Borden, "Optimal asymmetric error detecting codes," *Inform. Contr.*, vol. 53, pp. 66–73, Apr. 1982.

[15] B. Bose, "Burst unidirectional error-detecting codes," *IEEE Trans. Comput.*, vol. C-35, pp. 350–353, Apr. 1986.

[16] B. Bose and D. J. Lin, "PLA implementation of k-out-of-n code TSC checker," *IEEE Trans. Comput.*, vol. C-33, pp. 583–588, June 1984.

[17] B. Bose and D. J. Lin, "Systematic unidirectional error detecting codes," *IEEE Trans. Comput.*, vol. C-34, pp. 1026–1032, Nov. 1985.

[18] B. Bose, "Byte unidirectional error correcting codes," in *Dig. Pap. 19th Int. FTC Symp.*, Chicago, IL, July, 1989, pp. 222–228.

[19] M. Boudjit and M. Nicolaidis, "A tool for computing output code spaces and verifying the self-checking properties in complex self-checking systems," *IEICE Trans. Inf. & Syst.*, vol. E75-D, pp. 824–834, Nov. 1992.

[20] W. C. Carter and P. R. Schneider, "Design of dynamically checked computers," in *Proc. IFIP Conf.*, Edinburgh, Scotland, Aug. 1968, pp. 878–883.

[21] A. Chatterjee and J. A. Abraham, "C-testability for generalized tree structures with applications to Wallace trees and other circuits," in *Proc. Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 11–13, 1986, pp. 288–291.

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

[23] J. B. Clary and R. A. Sacane, "Self-testing computers," *Computer*, vol. 12, pp. 49–59, Oct. 1979.

[24] R. W. Cook, W. H. Sisson, T. F. Storey, and W. N. Toy, ”Design of a self-checking microprogram control,” *IEEE Trans. Comput.*, vol. C-22, pp. 255–262, Mar. 1973.

[25] Y. Crouzet and C. Landrault, ”Design of self-checking MOS-LSI circuits: Application to a four-bit microprocessor,” *IEEE Trans. Comput.*, vol. C-29, pp. 532–537, June 1980.

[26] L. Dadda, ”Some schemes for parallel multipliers,” *Alta Freq.*, vol. 34, pp. 349–356, May 1965.

[27] R. David and P. Thevenod-Fosse, “Design of totally self-checking asynchronous modular circuits,” *J. Des. Autom. Fault-Tolerant Comput.*, vol. 2, pp. 271–278, Oct. 1978.

[28] I. David, R. Ginosar, and M. Yoeli, ”An efficient implementation of Boolean functions as self-timed circuits,” *IEEE Trans. Comput.*, vol. 41, pp. 2–11, Jan. 1992.

[29] A. Dell'Acqua *et al.*, “System level policies for fault tolerance issues in the Fermi Project,” in *Proc. Int. Workshop on Defect and Fault Tolerance in VLSI Systems*, Venice, Italy, Oct. 27–29, 1993, pp. 1–8.

[30] L. V. Derbunovich and V. V. Neshveev, ”Design of totally self-checking checkers using PLAs,” *Autom. Remote Control*, vol. 47, pp. 149–156, No. 4, 1986.

[31] J. Devaney and R. Thomas, ”Faults & failures: $\Delta V=0.7V=85000$ irate travelers,” *IEEE Spectrum*, p. 54, Feb. 1992.

[32] M. Diaz, ”Design of totally self-checking and fail-safe sequential machines,” in *Dig. Pap. 4th Int. FTC Symp.*, Urbana, IL, 1974, pp. 3–19 – 3–24.

[33] M. Diaz, J. C. Geffroy, and M. Courvoisier, ”On-set realization of fail-safe sequential machines,” *IEEE Trans. Comput.*, vol C-23, pp. 133–138, Feb. 1974.

[34] M. Diaz and J. M. deSouza, ”Design of self-checking microprogrammed controls,” in *Dig. Pap. 5th Int. FTC Symp.*, Paris, France, June 1975, pp. 137–142.

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

[36] V. V. Dimakopoulos *et al.*, ”On TSC checkers for m-out-of-n codes,” *IEEE Trans. Comput.*, vol. 44, pp. 1055–1059, Aug. 1995.

[37] C. P. Disparte, ”A design approach for an electronic engine controller self-checking microprocessor,” in *Proc. EUROMICRO Conf.*, 1981, North-Holland, pp. 243–247.

[38] H. Dong, ”Modified Berger codes for detection of unidirectional errors,” *IEEE Trans. Comput.*, vol. C-33, pp. 572–575, June 1984.

[39] R. L. (Scot) Drysdale and F. H. Young, ”Improved divide/sort/merge sorting networks,” *SIAM J. Comput.*, vol. 4, pp. 264–270, Sept. 1975.

[40] L. A. Dunning, G. Dial, and M. Varanasi, ”Unidirectional 9-bit byte error detecting codes for computer memory systems,” in *Dig. Pap. 19th Int. FTC Symp.*, Chicago, IL, July, 1989, pp. 216–221.

[41] L. A. Dunning and M. Varanasi, "Unidirectional byte error detecting codes for computer memory systems," *IEEE Trans. Comput.*, vol. C-39, pp. 592–595, Apr. 1990.

[42] C. R. Edwards, "Some improved designs for the digital summation threshold logic (DSTL) gate," *Computer J.*, vol. 21, pp. 73–78, No. 1, 1978.

[43] C. C. Foster and F. D. Stockton, "Counting responders in an associative memory," *IEEE Trans. Comput.*, vol. C-20, pp. 1580–1583, Dec. 1971.

[44] C. V. Freiman, "Optimal error detection codes for completely asymmetric binary channels," *Inform. Contr.*, vol. 5, pp. 64–71, Mar. 1962.

[45] W. K. Fuchs, Ch.-Y. Chien, and J. A. Abraham, "Concurrent error detection in highly structured logic arrays," *IEEE J. Solid-State Circuits*, vol. SC-22, pp. 583–594, Aug. 1987.

[46] E. Fujiwara and D. K. Pradhan, "Error-control coding in computers," *Computer*, vol. 23, pp. 63–72, July 1990.

[47] E. Fujiwara and M. Yoshikawa, "A design method for cost-effective self-testing checker for optimal *d*-unidirectional error detecting codes," *IEICE Trans. Inf. & Syst.*, vol. E75-D, pp. 771–777, Nov. 1992.

[48] N. Gaitanis, C. Halatsis, and M. Sigala, "Fail-safe counters," *Digit. Process.*, vol. 5, pp. 43–57, No. 1/2, 1979.

[49] N. Gaitanis and C. Halatsis, "A new design method for *m*-out-of-*n* checkers," *IEEE Trans. Comput.*, vol. C-32, pp. 273–283, Mar. 1983.

[50] J. C. Geffroy, M. Courvoisier, and M. Diaz, "Realisation de circuits séquentiels asynchrones autotestables," *RAIRO*, vol. 7, pp. 75–92, Nov. 1973 – J-3 (in French).

[51] P. Golan, "Design of totally self-checking checker for 1-out-of-3 code," *IEEE Trans. Comput.*, vol. C-33, p. 285, Mar. 1984.

[52] M. Goto, "Rates of unidirectional 2-column errors detectable by arithmetic codes," in *Dig. Pap. 10th Int. FTC Symp.*, Kyoto, Japan, Oct. 1980, pp. 21-25.

[53] J. Gray and D. P. Siewiorek, "High-availability computer systems," *Computer*, vol. 24, pp. 39–48, Sept. 1991.

[54] C. Halatsis, N. Gaitanis, and M. Sigala, "Fast and efficient totally self-checking checkers for *m*-out-of-($2m\pm1$) codes," *IEEE Trans. Comput.*, vol. C-32, pp. 507–511, May 1983.

[55] C. Halatsis and N. Gaitanis, "Positive fail-safe realization of synchronous sequential machines," *IEEE Trans. Comput.*, vol C-28, pp. 167–172, Feb. 1979.

[56] M. P. Halbert and S. M. Bose, "Design approach for a VLSI self-checking MIL-STD-1750A microprocessor," in *Dig. Pap. 14th Int. FTC Symp.*, Kissimmee, Florida, June 20–22, 1984, pp. 254–259.

[57] M. P. Halbert, "Self-checking computer module based on the Viper microprocessor," *Microprocessors and Microsystems*, vol. 12, pp. 264–270, 1988.

[58] T. N. Haniotakis, A. M. Paschalis, and D. Nikolos, "Fast and low-cost TSC checkers for 1-out-of-n and $(n-1)$-out-of-n codes in MOS transistor implementation," *Int. J. Electronics*, vol. 71, pp. 781–791, Nov. 1991.

[59] T. Haniotakis *et al.*, "Totally self-checking checkers for Borden codes," *Int. J. Electronics*, vol. 76, pp. 57–64, Jan. 1994.

[60] S. Hatakenaka and T. Nanya, "A design method of SFS and SCD combinational circuits," *IEICE Trans. Inf. & Syst.*, vol. E75-D, pp. 819–823, Nov. 1992.

[61] R. Horst, D. Jewett, and D. Lenoski, "The risk of data corruption in microprocessor-based systems," in *Dig. Pap. 23rd Int. FTC Symp.*, Toulouse, France, June 22–24, 1993, pp. 576–585.

[62] S. L. Hürst, "Digital-summation threshold-logic gates: a new circuit element," *Proc. IEE*, vol. 120, pp. 1301–1307, No. 11, 1973.

[63] M. J. Iacoponi and D. K. Vail, "The fault tolerance approach of the advanced architecture on-board processor," in *Dig. Pap. 19th Int. FTC Symp.*, Chicago, IL, July, 1989, pp. 6–12.

[64] H. Imai, Ed., *Essentials of Error-Correcting Coding Techniques*, Academic Press, 1990.

[65] Y. Iwadare, E. Fujiwara and K. Iwasaki, "Coding theory applications in fault tolerant computing," *IEICE Trans.*, vol. E74, pp. 244–258, Feb. 1991.

[66] N. K. Jha, "A totally self-checking checker for Borden's code," *IEEE Trans. Comp.-Aided Des.*, vol. CAD-8, pp. 731–736, July 1989.

[67] N. K. Jha, "Testing of differential cascode voltage switch one-count generators," *IEEE J. Solid-State Circuits*, vol. SC-25, No. 1, pp. 246–253, Feb. 1991.

[68] N. K. Jha, "Totally self-checking checker designs for Bose-Lin, Bose, and Blaum codes," *IEEE Trans. Comp.-Aided Des.*, vol. CAD-10, pp. 136–143, Jan. 1991.

[69] N. K. Jha and J. A. Abraham, "Techniques for efficiently implementing totally self-checking checkers in MOS technology," *Int. J. of Comput. & Math. with Applic.*, vol. 13, pp. 555–566, No. 5–6, 1987.

[70] N. K. Jha and S. Kundu, *Testing and Reliable Design of CMOS Circuits*, Kluwer Academic Publishers, Norwell, MA, 1990.

[71] N. K. Jha and M. B. Vora, "A systematic code for detecting t-unidirectional errors," in *Dig. Pap. 17th Int. FTC Symp.*, Pittsburgh, PA, July 6–8, 1987, pp. 96–101.

[72] N. K. Jha and S.-J. Wang, "Design and synthesis of self-checking VLSI circuits," *IEEE Trans. Comp.-Aided Des.*, vol. 12, pp. 878–887, June 1993.

[73] B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley, Reading, MA, 1989.

[74] J. Khakbaz, "Self-testing embedded parity trees," in *Dig. Pap. 12th Int. FTC Symp.*, Santa Monica, CA, June 1982, pp. 109–116.

[75] D. E. Knuth, "*The Ajrt of Computer Programming, Vol. III, Sorting and Searching,*" 2nd Ed., Reading, MA, Addison-Wesley, 1973, ch. 5.

[76] H. Kobayashi and H. Ohara, "A synthesizing method for large parallel counters with a network of smaller ones," *IEEE Trans. Comput.*, vol. C-27, pp. 753–757, Aug. 1978.

[77] M. Kotočova, "Design of self-checking check circuits for some 1-out-of-n codes," in *Proc. FTSD-4 — 4th Int. Conf. on Fault-Tolerant Systems and Diagnostics*, Sept. 28–30, 1981, Brno, Czechoslovakia, pp. 241–247.

[78] G. D. Kraft and W. N. Toy, *Microprogrammed Control and Reliable Design of Small Computers*, Prentice-Hall, Englewood Cliffs, N.J., 1981.

[79] A. Kraśniewski, "Efficient design methods of self-checking AND-OR circuits," *Referaty* — Zeszyt 112, Inst. Telekom. Polit. Warsz., Warszawa, 1983 (in Polish).

[80] A. Kraśniewski, *Designing Self-Testable VLSI Circuits*, Prace Naukowe Politechniki Warszawskiej, Elektronika, Z. 83, Wydawnictwa Politechniki Warszawskiej, Warszawa 1989 (in Polish).

[81] S. Kundu and S. M. Reddy, "On symmetric error correcting and all unidirectional error detecting codes," *IEEE Trans. Comput.*, vol. C-39, pp. 752–761, June 1990.

[82] C.-S. Lai and C.-L. Wey, "An efficient output function partitioning algorithm reducing hardware overhead in self-checking circuits and systems," in *Proc. 35th Midwest Symp. Circuits Systems*, 1992, pp. 1538–1541.

[83] C.-S. Lai and C.-L. Wey, "*SOLiT*: an automated system for synthesising reliable sequential circuits with multilevel logic implementation," *IEE Proc.-Comput. Digit. Tech.*, vol. 142, pp. 49–54, Jan. 1995.

[84] E. A. Lamagna, "The complexity of monotone networks for certain bilinear forms, routing problems, sorting, and merging," *IEEE Trans. Comput.*, vol. C-28, pp. 773–782, Oct. 1979.

[85] J.-C. Laprie, "Dependability: Basic concepts and associated terminology," *PDCS Techn. Rep. Series*, No. 31, May 1990.

[86] G. B. Laskaris, T. N. Haniotakis, A. M. Paschalis, and D. Nikolos, "Efficient design of TSC checkers for 1-out-of-n codes in MOS transistor implementation," *Int. J. Electronics*, vol. 69, pp. 805–817, Dec. 1990.

[87] E. L. Leiss, "Data integrity in digital optical disks," *IEEE Trans. Comput.*, vol. C-33, pp. 818–827, Sep. 1984.

[88] É. Leser, "La grande panne du réseau des cartes bancaires (The great outage of the banking card network)," *Le Monde*, Mardi — 29 Juin 1993, p. 18 (in French).

[89] J.-C. Lo, "Reliable floating-point arithmetic algorithms for error-coded operands," *IEEE Trans. Comput.*, vol. C-43, ss. 400–412, Apr. 1994.

[90] J.-Ch. Lo and S. Thanawastien, "The design of fast totally self-checking Berger code checkers," in *Dig. Pap. 18th Int. FTC Symp.*, Tokyo, Japan, June 1988, pp. 226–231.

[91] J.-Ch. Lo, S. Thanawastien, T. R. N. Rao, and M. Nicolaidis, "An SFS Berger check prediction ALU and its application to self-checking processor designs," *IEEE Trans. Comp.-Aided Des.*, vol. 11, pp. 525–540, Apr. 1992.

[92] J.-Ch. Lo, S. Thanawastien, and T. R. N. Rao, "Berger check prediction for array multipliers and array dividers," *IEEE Trans. Comput.*, vol. 42, pp. 892–896, July 1993.

[93] G. Mago, "Monotone functions in sequential circuits," *IEEE Trans. Comput.*, vol. C-22, pp. 928–933, Oct. 1973.

[94] G. P. Mak, J. A. Abraham, and E. S. Davidson, "The design of PLAs with concurrent error detection," in *Dig. Pap. 12th Int. FTC Symp.*, Santa Monica, CA, June 1982, pp. 303–310.

[95] G. K. Maki, "A self-checking microprocessor design," *J. Des. Autom. Fault-Tolerant Comput.*, vol. 3, pp. 15–27, Jan. 1978.

[96] M. Malek and K. H. Jau, "Cost-effective error detection codes in multicomputer networks," *Microproc. and Microprogr.*, vol. 20, pp. 331–343, 1987.

[97] J. Marin, C. André, and F. Boeri, "Conception de systèmes séquentiels totalement autotestables á partir des réseaux de Petri," *RAIRO Automatique*, vol. 10, pp. 5–22, Nov. 1976 – J-3 (in French).

[98] M. A. Marouf and A. D. Friedman, "Efficient design of self-checking checker for any m-out-of-n code," *IEEE Trans. Comput.*, vol. C-27, pp. 482–490, June 1978.

[99] M. A. Marouf and A. D. Friedman, "Design of self-checking checkers for Berger codes," in *Dig. Pap. 8th Int. FTC Symp.*, Toulouse, France, June 1978, pp. 179–184.

[100] H. Masuyama and N. Yoshida, "Design of fail-safe asynchronous sequential machines," *Trans. IECE of Japan*, vol E60, pp. 527–532, Oct. 1977.

[101] K. Matsuzawa and E. Fujiwara, "Masking asymmetric line faults using semi-distance codes," *Trans. IEICE*, vol. E73, pp. 1278–1286, Aug. 1990.

[102] V. I. Maznev, "Synthesis of totally self-checking sequential circuits," *Autom. Remote Control*, vol. 38, pp. 913–920, No. 6, 1977.

[103] V. I. Maznev, "Design of self-checking $1/p$ checkers," *Autom. Remote Contr.*, vol. 39, pt. 2, pp. 1380–1383, Sept. 1978.

[104] M. Mehta, V. Parmar, and E. E. Swartzlander, Jr., "High-speed multiplier design using multi-input counter and compressor circuits," in *Proc. 10th Symp. Comput. Arithm.*, Grenoble, France, June 26–28, 1991, pp. 43–50.

[105] P. Mellor, "CAD: Computer-aided disaster," *High Integrity Systems*, vol. 1, pp. 101–156, No. 2., 1994.

[106] M. N. Meyers, "The AT&T telephone network outage of January 15th, 1990," Special presentation at the *20th Int. FTC Symp.*, Newcastle upon Tyne, UK, June 26–28, 1990.

[107] Y. Min and J. Li, "Strongly fault secure PLA's and totally self-checking checkers," *IEEE Trans. Comput.*, vol. C-37, pp. 863–867, July 1988.

[108] E. F. Moore and C. E. Shannon, "Reliable circuits using less reliable relays, Part I," *J. Franklin Inst.*, vol. 262, pp. 191–208, Sept. 1956.

[109] E. F. Moore and C. E. Shannon, "Reliable circuits using less reliable relays, Part II," *J. Franklin Inst.*, vol. 262, pp. 281–297, Oct. 1956.

[110] T. Nanya and T. Kawamura, "A note on strongly fault secure sequential circuits," *IEEE Trans. Comput.*, vol. C-36, pp. 1121–1123, Sept. 1987.

[111] T. Nanya and T. Kawamura, "Error secure/propagating concept and its application to the design of strongly fault secure processors," *IEEE Trans. Comput.*, vol. C-37, pp. 14–24, Jan. 1988.

[112] T. Nanya and Y. Tohma, "Design of self-checking asynchronous sequential circuits," in *Dig. Pap. 10th Int. FTC Symp.*, Kyoto, Japan, Oct. 1980, pp. 278–281.

[113] T. Nanya and Y. Tohma, "A 3-level realization of totally self-checking checkers for m-out-of-n codes," in *Dig. Pap. 13th Int. Symp. on Fault-Tolerant Comp.*, June 28–30, 1983, Milan, Italy, pp. 173–176.

[114] J. von Neumann, "Probabilistic logic and the synthesis of reliable organisms from unreliable components," in *Automata Studies*, C. E. Shannon and J. McCarthy, Eds., Princeton University Press, Princeton, 1956, pp. 43–98.

[115] M. Nicolaidis, "Evaluation of a self-checking version of the MC 68000 microprocessor," *Microproc. and Microprogr.*, vol. 20, pp. 235–247, 1987.

[116] M. Nicolaidis, "Shorts in self-checking circuits," *J. Electr. Testing: Theory and Appls*, vol. 1, pp. 257–273, 1991.

[117] M. Nicolaidis, "Fault secure property versus strongly code disjoint checkers," *IEEE Trans. Comp.-Aided Des.*, vol. 13, pp. 651–658, May 1994.

[118] M. Nicolaidis and B. Courtois, "Strongly code-disjoint checkers," *IEEE Trans. Comput.*, vol. C-37, pp. 751–756, June 1988.

[119] M. Nicolaidis and M. Boudjit, "New implementations, tools, and experiments for decreasing self-checking PLAs overhead," in *Proc. ICCD'91*, pp. 275–281.

[120] M. Nicolaidis and M. Boudjit, "Verification of self-checking properties by means of output code space computation," in *Proc. EDAC'92*, Brussels, Belgium, Mar. 16–19, 1992, pp. 169–175.

[121] M. Nicolaidis and B. Courtois, "Layout rules for the design of self-checking circuits," in *VLSI Conf.*, Tokyo, Japan, Aug. 1985, pp. 261–272.

[122] M. Nicolaidis and B. Courtois, "Design of NMOS strongly fault secure circuits using unidirectional error detecting codes," in *Dig. Pap. 16th Int. FTC Symp.*, Vienna, Austria, July 1986, pp. 22–27.

[123] M. Nicolaidis and B. Courtois, "Self-checking logic arrays," *Microproc. Microsyst.*, vol. 13, pp. 281–290, May 1989.

[124] D. Nikolos and A. Krokos, "Theory and design of t-error correcting, k-error detecting and d-unidirectional error detecting codes with $d > k > t$," *IEEE Trans. Comput.*, vol. 41, pp. 411–419, Apr. 1992.

[125] D. Nikolos *et al.*, "Totally self-checking checkers for optimal t-unidirectional error detecting codes," in *Proc. 13th Int. Conf. Fault-Tolerant Syst. Diagn.*, Varna, Bulgaria, June 20–22, 1990, pp. 326–331.

[126] F. Özgüner, "Design of totally self-checking asynchronous and synchronous sequential machines," in *Dig. Pap. 7th Int. FTC Symp.*, June 1977, pp. 124–129.

[127] E. W. Page and E. W. Marinos, "Programmable array realizations of synchronous sequential machines," *IEEE Trans. Comput.*, vol C-26, pp. 811–818, Aug. 1977.

[128] B. Parhami and A. Avizienis, "Detection of storage errors in mass memories using low-cost arithmetic error codes," *IEEE Trans. Comput.*, vol. C-27, pp. 302-308, Apr. 1978.

[129] A. Park and R. Maeder, "Codes to reduce switching transients across VLSI I/O pins," *ACM Comp. Arch. News*, vol. 20, pp. 17–21, 1992.

[130] A. Paschalis, "Efficient PLA design of TSC 1-out-of-n code checkers," *Int. J. Electronics*, vol. 73, pp. 471–484, Mar. 1992.

[131] A. Paschalis, "Efficient structured design of totally self-checking M-out-of-N code checkers with $N > 2M$ and $M = 2^K - 1$," *Int. J. Electronics*, vol. 77, pp. 251–257, Aug. 1994.

[132] A.M. Paschalis, C. Halatsis, and G. Philokyprou, "Concurrently totally self-checking microprogram control unit with duplication of microprogram sequencer," *Microproc. and Microprogr.*, vol. 20, pp. 271–281, 1987.

[133] A. M. Paschalis, D. Nikolos, and C. Halatsis, "Efficient modular design of TSC checkers for m-out-of-n codes," *IEEE Trans. Comput.*, vol. C-37, pp. 301–309, Mar. 1988.

[134] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2nd Ed., MIT Press, Cambridge, MA, 1972.

[135] W. H. Pierce, *Failure-Tolerant Computer Design*, Academic Press, New York and London, 1965.

[136] S. J. Piestrak, "Design methods of self-testing checkers for constant-weight codes," Ph. D. dissert., Inst. of Eng. Cybern., Techn. Univ. of Wrocław, 1982 (in Polish).

[137] S. J. Piestrak, "Design method of totally self-checking checkers for m-out-of-n codes," in *Dig. Pap. 13th FTC Int. Symp.*, June 28–30, 1983, Milan, Italy, pp. 162–168.

[138] S. J. Piestrak, "Design method of self-testing checkers for 1-out-of-n codes," in *Proc. FTSD'6 — 6th Int. Conf. on Fault- Tolerant Systems and Diagnostics*, Sept. 5–7, 1983, Brno, Czechoslovakia, pp. 57–63.

[139] S. J. Piestrak, "Improved design rules of two-level totally self-checking combinational circuits using m-out-of-n codes," in *Proc. FTSD'7 — 7th Int. Conf. on Fault-Tolerant Systems and Diagnostics*, Oct. 4–6, 1984, Sofia, Bulgaria, pp. 186–193.

[140] S. J. Piestrak, "PLA implementation of totally self-checking circuits using m-out-of-n codes," in *Proc. ICCD'85, Int. Conf. on Computer Design: VLSI in Computers*, Port Chester, N.Y., Oct. 1–3, 1985, pp. 777–781.

[141] S. J. Piestrak, "Design of fast self-testing checkers for Berger codes," *IEEE Trans. Comput.*, vol. C-36, pp. 629–634, May 1987.

[142] S. J. Piestrak, "Design of high-speed and cost-effective self-testing checkers for low-cost arithmetic codes," *IEEE Trans. Comput.*, vol. C-39, pp. 360–374, Mar. 1990.

[143] S. J. Piestrak, "The minimal test set for sorting networks and the use of sorting networks in self-testing checkers for unordered codes," in *Dig. Pap. 20th Int. FTC Symp.*, Newcastle upon Tyne, UK, June 26–28, 1990, pp. 457–464.

[144] S. J. Piestrak, "Efficient encoding/decoding circuitry for systematic unidirectional error-detecting codes," in *Proc. 5th Int. Conf. Fault-Tolerant Computing Systems*, Nürnberg, FRG, Sept. 25–27, 1991, Springer-Verlag, Berlin, pp. 181–193.

[145] S. J. Piestrak, "Error detecting codes for microprocessor systems," in *Proc. Microcomputer'91 Conf.*, Sept. 1991, Szklarska Poręba, Poland, Prace Naukowe ICT Polit. Wrocł., Nr 89, Ser. Konferencje Nr 39, Wydawnictwa Politechniki Wrocławskiej, pp. 37–52.

[146] S. J. Piestrak, "Design of a self-testing checker for Borden code," in *Proc. ICCD'91, Int. Conf. on Computer Design: VLSI in Computers*, Cambridge, MA, Oct. 14–16, 1991, pp. 582–585.

[147] S. J. Piestrak, "Design of fast self-testing checkers for m-out-of-$2m$ and m-out-of-$(2m \pm 1)$ codes," *Int. J. Electronics*, vol. 74, pp. 177–199, Feb. 1993.

[148] S. J. Piestrak, "The minimal test set for multi-output threshold circuits implemented as sorting networks," *IEEE Trans. Comput.*, vol. 42, pp. 700–712, June 1993.

[149] S. J. Piestrak, "General design principles of self-testing code-disjoint PLAs," *Proc. ATS'93 — 2nd Asian Test Symp.*, Beijing, China, Nov. 18–19, 1993, pp. 287–292, IEEE Press.

[150] S. J. Piestrak, "Design of residue generators and multioperand adders modulo-3 built of multi-output threshold circuits," *IEE Proc.-Comput. Digit. Tech.*, vol. 141, pp. 129–134, March 1994.

[151] S. J. Piestrak, "Design of TSC code-disjoint inverter-free PLA's for separable unordered codes," in *Proc. ICCD'94, Int. Conf. on Computer Design: VLSI in Computers & Processors*, Cambridge, MA, Oct. 10–12, 1994, pp. 128–131, IEEE Computer Society Press.

[152] S. J. Piestrak, "General design of area-efficient PLA self-testing checkers for 1-out-of-n codes," *Int. J. Electronics*, vol. 77, pp. 1077–1090, Dec. 1994.

[153] S. J. Piestrak, "Design of minimal-level PLA self-testing checkers for m-out-of-n codes," submitted to *IEEE Trans. VLSI.*

[154] S. J. Piestrak, "Design of self-testing checkers for m-out-of-n codes with $m \geq 3$ and $n > 4m$," *Tech. Rep.* PRE-61/94, Inst. of Eng. Cybern., Techn. Univ. of Wrocław, 1994, 38 pp.

[155] S. J. Piestrak, "Self-testing checkers for 2-rail codes," *Tech. Rep.* PRE-62/94, Inst. of Eng. Cybern., Techn. Univ. of Wrocław, 1994, 20 pp.

[156] S. J. Piestrak and T. Nanya, "Towards totally self-checking delay-insensitive systems," in *Dig. Pap. 25th Int. Symp. on Fault-Tolerant Computing*, Pasadena, CA, June 27–30, 1995, IEEE Computer Society Press.

[157] S. J. Piestrak, "Self-checking design in Eastern Europe," to appear in *IEEE Design and Test of Computers*, Sp. Issue on Design and Test in Eastern Europe, Spring 1996.

[158] D. K. Pradhan, "Asynchronous state assignments with unateness properties and fault-secure design," *IEEE Trans. Comput.*, vol. C-27, pp. 396–404, May 1978.

[159] D. K. Pradhan, Ed., *Fault Tolerant Computing: Theory and Techniques, Vol. I and II*, Prentice-Hall, Englewood Cliffs, N. J., 1986.

[160] D. K. Pradhan and J. J. Stiffler, "Error-correcting codes and self-checking circuits," *Computer*, vol. 13, pp. 27–37, Mar. 1980.

[161] V. Rabara, "Design of self-checking checker for 1-out-of-n code ($n > 3$)," in *Proc. FTSD'4 — 4th Int. Conf. on Fault-Tolerant Systems and Diagnostics*, Sept. 28–30, 1981, Brno, Czechoslovakia, pp. 234–240.

[162] V. Rabara and A. Rabarova, "Three-level TSC checkers for 1-out-of-n code," in *FTSD'8 — Proc. 8th Int. Conf. on Fault-Tolerant Syst. Diagnostics*, Katowice, Poland, 1985, pp. 223–229.

[163] J. Rajski and V. K. Agarwal, "Testing and applications of inverter-free PLAs," *IEEE Design and Test of Computers*, pp. 30–40, Dec. 1987.

[164] T. R. N. Rao, *Error Coding for Arithmetic Processors*, Academic Press, New York, 1974.

[165] T. R. N. Rao and E. Fujiwara, *Error-Control Coding for Computer Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1989.

[166] T. R. N. Rao, G. L. Feng, M. S. Kolluru, and J.-Ch. Lo, S. Thanawastien, "Novel totally self-checking Berger checker designs based on generalized Berger code partitioning," *IEEE Trans. Comput.*, vol. 42, pp. 1020–1024, Aug. 1993.

[167] S. M. Reddy, "A note on self-checking checkers," *IEEE Trans. Comput.*, vol. C-22, pp. 1100–1102, Oct. 1974.

[168] S. M. Reddy and J. R. Wilson, "Easily testable cellular realizations for the (exactly p)-out-of-n and (p or more)-out-of-n logic functions," *IEEE Trans. Comput.*, vol. C-23, pp. 98–100, Jan. 1974.

[169] D. A. Rennels, "Architectures for fault-tolerant spacecraft computers," *Proc. IEEE*, vol. 66, pp. 1255–1268, Oct. 1978.

[170] G. Russell and I. L. Sayers, *Advanced Simulation and Test Methodologies for VLSI Design*, Van Nostrand Reinhold Int., London, UK, 1989.

[171] Yu. L. Sagalovitsh, *State Encoding and Reliability of Automata*, Sviaz, Moskva, SU, 1975 (in Russian).

[172] R. M. Sahni, "Reliability of integrated circuits," in *Proc. Int. IEEE Computer Group Conf.*, Washington, D. C., 1970, pp. 213–219.

[173] K. Sapiecha, *Testing and Diagnosis of Digital Systems*, PWN, Warszawa, 1987 (in Polish).

[174] K. Sapiecha, A. Hławiczka, A. Kraśniewski, H. Krawczyk, S. J. Piestrak, and J. Sosnowski, "Wiarygodność — pożądana czy nieodłączna cecha współczesnych systemów cyfrowych," *Mat. Konf. Informatyka na Wyższych Uczelniach dla Gospodarki Narodowej*, Tom I, Gdańsk, Poland, 17–19 Nov. 1994, pp. 125–129 (in Polish).

[175] V. V. Sapozhnikov, Vl. V. Sapozhnikov, and V. G. Trokhov, "Design of self-checking sequential networks," *Avtom. Vychisl. Tekh.*, vol. 11, pp. 6–11, No. 3, 1977.

[176] V. V. Sapozhnikov and Vl. V. Sapozhnikov, "Synthesis of totally self-checking asynchronous automata," *Automat. Remote Contr.*, vol. 40, pp. 124–134, Feb. 1979.

[177] V. V. Sapozhnikov and Vl. V. Sapozhnikov, *Discrete Automata with Error Detection*, Energoatomizdat, Leningrad, SU, 1984 (in Russian).

[178] V. V. Sapozhnikov and Vl. V. Sapozhnikov, "Universal algorithm for synthesizing self-checking testers for constant-weight codes," *Probl. Inf. Transm.*, vol. 20, pp. 128–137, No. 2, 1984.

[179] V. V. Sapozhnikov and Vl. V. Sapozhnikov, "Self-checking checkers for balanced codes," *Autom. Remote Control*, vol. 53, pp. 321–348, Mar. 1992.

[180] R. M. Sedmak and H. L. Liebergot, "Fault tolerance of a general purpose computer implemented by very large scale integration," *IEEE Trans. Comput.*, vol. C-29, pp. 492–500, June 1980.

[181] F. F. Sellers, Jr., M.-Y. Hsiao, and L. W. Bearnson, *Error Detecting Logic for Digital Computers*, McGraw-Hill, New York, 1968.

[182] D. P. Siewiorek and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*, 2nd Ed., Digital Press, Bedford, MA, 1992.

[183] A. D. Singh and S. Murugesan, "Fault-tolerant systems," *Computer*, vol. 23, pp. 13–17, July 1990.

[184] T. J. Slegel and R. J. Veracca, "Design and performance of the IBM Enterprise System/9000 Type 9121 Vector Facility," *IBM J. Res. Develop.*, vol. 35, pp. 367–381, May 1991.

[185] J. E. Smith, "The design of totally self-checking check circuits for a class of unordered codes," *J. Des. Autom. Fault-Tolerant Comput.*, vol. 2, pp. 321–342, Oct. 1977.

[186] J. E. Smith, "On separable unordered codes," *IEEE Trans. Comput.*, vol. C-33, pp. 741–743, Aug. 1984.

[187] J. E. Smith and P. Lam, "A theory of totally self-checking system design," *IEEE Trans. Comput.*, vol. C-32, pp. 831–844, Sep. 1983.

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

[189] Ye. S. Sogomonyan and Ye. V. Slabakov, *Self-Checking Circuits and Fault-Tolerant Systems*, Radio i Svjaz, 1989 (in Russian).

[190] J. Sosnowski, *Transient Fault Analysis and Tolerance in Digital Systems*, Prace Naukowe Politechniki Warszawskiej, Elektronika, Z. 103, Wydawnictwa Politechniki Warszawskiej, Warszawa 1993 (in Polish).

[191] T. F. Storey, "Design of a microprogram control for a processor in an electronic switching system," *BSTJ*, vol. 56, pp. 183–232, Feb. 1976.

[192] E. E. Swartzlander, Jr., "Parallel counters," *IEEE Trans. Comput.*, vol. C-22, pp. 1021–1024, Nov. 1973.

[193] L. Tallini, L. Merani, and B. Bose, "Balanced codes for noise reduction in VLSI systems," in *Dig. Pap. 24th Int. FTC Symp.*, Austin, TX, June 15–17, 1994, pp. 212–218.

[194] D. L. Tao, C. R. P. Hartmann, and P. K. Lala, "A general technique for designing totally self-checking checker for 1-out-of-*n* code with minimum gate delay," *IEEE Trans. Comput.*, vol. C-41, p. 881–886, July 1992.

[195] R. Thomas and S. Kundu, "Synthesis of fully testable sequential machines," in *Proc. EDAC'91*, 1991, pp. 283–288.

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

[197] Y. Tohma, "Coding Techniques in Fault-Tolerant, Self-Checking, and Fail-Safe Circuits," ch. 5 in *Fault Tolerant Computing: Theory and Techniques, Vol. I*, D. K. Pradhan, Ed., Prentice-Hall, Englewood Cliffs, N. J., 1986.

[198] N. Tokura, T. Kasami, and A. Hashimoto, "Fail-safe logic nets," *IEEE Trans. Comput.*, vol C-20, pp. 323–330, March 1971.

[199] W. N. Toy, Fault-tolerant design of local ESS processors," *Proc. IEEE*, vol. 66, pp. 1126–1145, Oct. 1978.

[200] V. I. Varshavsky *et al.*, "Possibility of implementing an asynchronous interface using a self-synchronizing code with identifier," *Autom. Contr. Comput. Sci.*, vol. 15, pp. 77–81, No. 5, 1981.

[201] V. I. Varshavsky, L. Ya. Rosenblum, and A. R. Taubin, "Totally self-checking asynchronous combinational circuits and the indicatability property," *Automat. Remote Contr.*, vol. 43, pp. 689–696, No. 5, 1982.

[202] V. I. Varshavsky (Ed.), *Self-Timed Control of Concurrent Processes*, Kluwer Acad. Publ., Norwell, MA, 1990.

[203] T. Verhoeff, "Delay-insensitive codes—An overview," *Distr. Computing*, vol. 3, pp. 1–8, No. 1; 1988.

[204] I. S. Vizirev, "Design of totally self-checking checkers for constant-weight codes," *Autom. Contr. Comput. Sci.*, vol. 16, pp. 34–40, No. 1, 1982.

[205] D. C. Van Voorhis, "An economical construction for sorting networks," in *Proc. AFIPS NCC*, 1974, pp. 921–927.

[206] J. F. Wakerly, "Detection of unidirectional multiple errors using low-cost arithmetic codes," *IEEE Trans. Comput.*, vol. C-24, pp. 210–212, Feb. 1975.

[207] J. F. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications*, North-Holland, New York, 1978.

[208] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 14–17, Feb. 1964.

[209] S. L. Wang and A. Avižienis, "The design of totally self-checking circuits using programmable logic arrays," in *Dig. Pap. 9th Int. FTC Symp.*, Madison, WI, 20–22 June 1979, pp. 173–180.

[210] R. H. Wilcox and W. C. Mann, Eds., *Redundancy Techniques for Computing Systems*, Spartan Press, Washington, D.C., 1962.

[211] I. Williamson, "Design of self-checking and fault-tolerant microprogrammed controllers," *Radio and Electronic Engineer*, vol. 47, pp. 449–454, Oct. 1977.

[212] C. Y. Wong, W. K. Fuchs, J. A. Abraham, and E. S. Davidson, "The design of a microprogram control unit with concurrent error detection," in *Dig. Pap. 13th Int. FTC Symp.*, Milan, Italy, June 28–30, 1983, pp. 476–483.

[213] M. M. Yen, W. K. Fuchs, and J. A. Abraham, "Designing for concurrent error detection in VLSI: Application to a microprogram control unit," *IEEE J. Solid-State Circuits*, vol. SC-22, pp. 595–605, Aug. 1987.

SYNTEZA SAMOTESTOWALNYCH UKŁADÓW KONTROLNYCH DLA KODÓW WYKRYWAJĄCYCH BŁĘDY JEDNOKIERUNKOWE

Cyfrowe układy samosprawdzalne, w których wejścia, wyjścia i stany wewnętrzne są kodowane kodami wykrywającymi błędy, pozwalają na detekcję w czasie rzeczywistym uszkodzeń wewnętrznych trwałych i przemijających. Od wielu lat obserwowano, że wiele klas uszkodzeń występujących w cyfrowych układach scalonych wielkiej skali integracji (VLSI), systemach pamięci półprzewodnikowej RAM i na dyskach optycznych, powoduje występowanie błędów jednokierunkowych. W monografii przedstawiono ogólne podejście do syntezy układów cyfrowych umożliwiających użycie kodów wykrywających błędy jednokierunkowe (UEDC) w systemach tolerujących uszkodzenia. Podano metody syntezy samotestowalnych układów kontrolnych (STC) dla niesystematycznych UEDC, a dla systematycznych UEDC, zarówno koderów jak i STC, charakteryzujących się małą złożonością i dużą szybkością działania. Rozpatrywane klasy kodów obejmują: 1) kody niesystematyczne — kody stałowagowe (m z n) i kody Bordena; oraz 2) kody systematyczne — kody Bergera i kody im równoważne, kody wykrywające t błędów jednokierunkowych i kody wykrywające wiązki błędów jednokierunkowych. Pokazano, że kodery i STC dla dowolnego z wymienionych kodów można zbudować stosując wielowyjściowy układ progowy zrealizowany jako szczególny przypadek sieci sortującej. Alternatywna metoda, polegająca na wykorzystaniu tzw. licznika równoległego, składającego się wyłącznie z sumatorów pełnych, półsumatorów i bramek XOR, nadaje się do syntezy STC dla większości z tych kodów. Wszystkie prezentowane STC mają zdolność bieżącej detekcji (tj. w czasie normalnej pracy układu) wszystkich pojedynczych uszkodzeń wewnętrznych typu sklejenie oraz dużej liczby uszkodzeń wielokrotnych. Prezentowane układy charakteryzują się małą złożonością, dużą szybkością działania i są łatwo testowalne. Ponadto mają one regularną strukturę złożoną z bramek o małej liczbie wejść, co pozwala na ich łatwą implementację w powszechnie stosowanych technologiach realizacji układów scalonych VLSI.

CONTENTS

SPIS TREŚCI

PRACE NAUKOWE INSTYTUTU CYBERNETYKI TECHNICZNEJ
(wydane w latach 1991–1995)

Nr 85, Monografie nr 18, W. Jacak, *Roboty inteligentne. Metody planowania działań i ruchów*, Wrocław 1991

Nr 86, Monografie nr 19, J. Biernat, *Właściwości strukturalne i metody oceny miar niezawodności systemów progowych*, Wrocław 1991

Nr 87, Monografie nr 20, A. Janiak, *Dokładne i przybliżone algorytmy szeregowania zadań i rozdziału zasobów w dyskretnych procesach przemysłowych*, Wrocław 1991

Nr 88, Monografie nr 21, J. Halawa, *Metody wyznaczania transmisji uproszczonych i ich zastosowanie w automatyce i elektroenergetycė*, Wrocław 1991

Nr 89, Konferencje nr 39, *Microcomputer '91 – design, practice, education*, Wrocław 1991

Nr 90, Monografie nr 22, Z. Hasiewicz, *Identyfikacja sterowanych systemów o złożonej strukturze*, Wrocław 1993

Nr 91, Monografie nr 23, K. Styczeń, *Optymalne sterowanie cykliczne nieliniowych systemów dynamicznych*, Wrocław 1993

Nr 93, Konferencje nr 40, *IV Krajowa konferencja robotyki.* Tom 1, Wrocław 1993

Nr 94, Konferencje nr 41, *IV Krajowa konferencja robotyki.* Tom 2, Wrocław 1993

Wydawnictwa Politechniki Wrocławskiej
są do nabycia w następujących księgarniach:
„Politechnika”
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław
budynek A-1 PWr., tel. (0-71) 20 25 34,
„Tech”
plac Grunwaldzki 13, 50-377 Wrocław
budynek D-1 PWr., tel. (0-71) 20 32 52
Prowadzimy sprzedaż wysyłkową

ISSN 0324-9786