PARITY ERROR DETECTION IN EMBEDDED SYSTEM

M. Stojčev, T. Stanković

Faculty of Electronic Engineering,
Beogradska 14, 18000 Niš, Yugoslavia,
{stojcev, tatjanas}@elfak.ni.ac.yu

Abstract : In this article we describe one suitable approach that enables the designer to insert a boundary-scan and built-in-self-test concepts, as typical design-for-testability techniques in system-on-chip and multichip module embedded system design, for fault-effects detection. For transient error detection implementation of parity error detection into a 36-bit bus transceiver circuit (32-bit data & four parity bits) is given. The bus transceiver can be implemented as custom or semi-custom integrated circuit in submicron technology and low cost FPGA or CPLD circuit, core within a system-on-a-chip, or glue logic (bridge) within the multichip module.

Keywords: embedded systems, parity error detection, bus transceiver

1. Introduction

During last ten years embedded systems has moved toward system-on-a-chip (SOC) and high-level multichip modules solutions. SOC denotes the integration of random logic, processor cores (general purpose and domain specific), SRAMs, ROMs, ASICs, FPGAs, analog components, and sensors/actuators on the same piece of silicon. Multichip modules (MMs) have features that enable smaller, lighter systems, and high speed performance to be obtained by eliminating individual packages and their parasitics i.e. a new approach to interconnecting chips that replaces conventional printed circuit board technology in high performance applications. Integrating various cores or chips on one chip or module hinders external test because individual components are no longer accessible. High transistor counts, aggressive clock frequencies, decreasing voltage levels, and higher bandwidths are the main technical constraints that make sub micron circuits more susceptible to transient faults. Since these faults are random, they go unnoticed during production test, so field engineers must handle them in the system under real time operating conditions [1].

From the other side, direct implementation of some novelty in embedded system requires a radical change in the overall design process. Practical solutions require a full understanding of the hardware and software domains, their relationships in

an embedded environment, and system behavior under fault-free and faulty conditions [2]. Designers usually employ a combination of hardware and software techniques to take advantage of complementary feature such as fault detection and correction through the internal (on-chip) buses during data transfer cycles. Using offline tests, such as boundary scan (BS) IEEE Std 1149.1 (regulates chip testing on single- and multiple-board systems) it is possible to detect permanent fault effects, while with online check or self-test techniques such as built-in-self-test (BIST) the designers can detect transient fault-effects such as those caused by crosstalks, glitches, delays, or oscillations [3]. BS and BIST are typical design-for-testability (DFT) techniques widely used in SOCs and MMs based embedded system design.

Our work aims to develop, implement, and validate techniques for online detection of permanent and transient faults in embedded systems. For permanent faults effects detections, and performing their effective isolation in SOCs and MMs based embedded system, we propose an implementation of BS techniques, while BIST based on parity-error detection scheme we use for online detection of transient faults injected during the bus transfer cycles.. These techniques should come at a reasonable overhead of increased signal delay. Implementation of parity error detection into a bus transceiver circuit used in high-speed reliable ESs, analysis concerning propagation delay, and speed of operation are described.

2. What is an embedded system?

Embedded systems (ESs) are computers incorporated in consumer products or other devices in order to perform application specific functions. ESs can contain a variety of computing devices, such as microcontrollers, application-specific integrated circuits (ASICs), application specific integrated processors (ASIPs), and digital signal processors (DSPs). Unlike in computers, the electronics used in these applications are deeply embedded and must interact with the user and the real word through sensors and actuators. A key requirement is that these computing devices continuously respond to external events in real time [4]. Embedded electronic systems are often highly distributed, and their parts must collaborate to implement a complete application. Because of performance and cost pressures ESs are built using a wide variety of techniques, including software, firmware, ASICs, ASIPs, general purpose and domain-specific processors, FPGA, CPLD, analog circuits, and sensors and actuators. The design of complex ESs is a difficult problem, requiring designers with skills and experience to identify the best solution [5]. Typical applications of ESs include medical electronics (pacemakers), personal communications devices (wireless phones), automobiles (antilock braking systems), aviation (fly-by-wire flight control systems), railroad (high-speed train control), and others.

3. What is SOC design?

A SOC design is defined as a complex IC that integrates the major functional elements of a complete end-product into a single chip or chipset. In general, SOC design incorporates a programmable processor, on-chip memory, and accelerating function units implemented in hardware. It also interfaces to peripheral devices and/or the real world. SOC designs encompass both hardware and software components. Because SOC designs can interface to the real world, they often incorporate analog components, and can in the future, also include opto/microelectronic mechanical system components [2]. Short time to market, large gate counts, and high-performance characterize today's VLSI design environment. SOC technology holds the key to previously mentioned complex applications by enabling high-performance, embedded processing solutions at a low single-chip cost. To quickly create SOC designs with the required complexity, designers must use predesigned intellectual property (IP) blocks, also referred as macros, cores, or virtual components. For SOC designs, this means reusing previously designed cores wherever possible. The more design reuse, the faster the SOC time to market [6]. From system architects point of view quick SOC assembly using cores is not an easy job due to the following reasons: CPU selection, decision which functions will be performed in hardware versus software, integrating cores into SOCs, achieving correct timing, physical design of large systems, testing and system verification, and others [7].

4. What is MM technology

MMs are primarily considered as a suitable packaging solution for advanced VLSI devices for maintaining inherent high performance in digital and mixed integrated circuit. In particular, CMOS microprocessors or digital signal processors with high I/O pin counts, which operate at high clock speed over 100 MHz, need closely placed SRAM, ROM, functional units, cache memory, etc. [8].

5. Target architectures of embedded systems

Usually embedded systems are implemented by processors and application specific hardware in one of the following two ways [9, 10, 11]. CPU and accelerators attached to the host (Fig. 1):

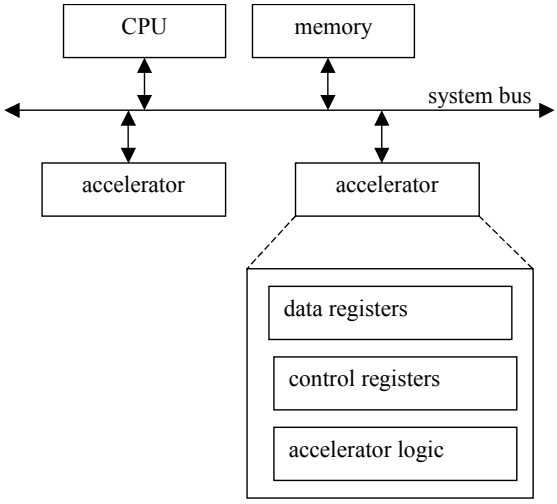

Figure 1: CPU and accelerators in a system; convenient for multichip modules design

The CPU, often called the host, talks to the accelerators through data and control registers in the accelerators. These registers allow the CPU to monitor the accelerator's operation and to give the accelerator commands. An accelerator interacts with the CPU through the programming model interface; it does not execute instructions. Its interface is functionally equivalent to an I/O device, although it usually does not perform input or output [10].

b) application-specific SOC multiprocessors (Fig. 2):

This type of architecture combines custom hardware with embedded software, lending a certain measure of complexity and heterogeneity to the design. The most common architecture in these systems can be characterized as one of coprocessing, i.e. processor working in conjunction with dedicated hardware to deliver a specific application. The particular implementation of the coprocessing architecture varies in the degree of parallelism supported between hardware and software components [11, 12].

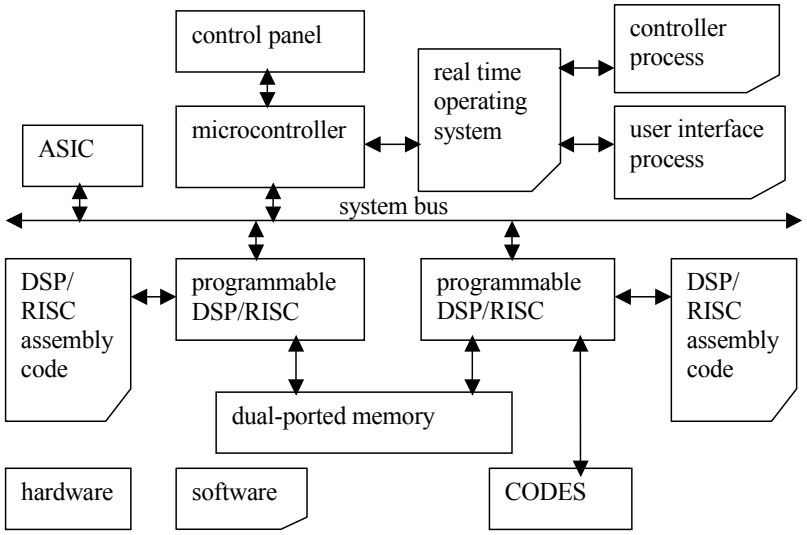

Figure 2: A typical application-specific SOC multiprocessors embedded system; convenient for SOC design

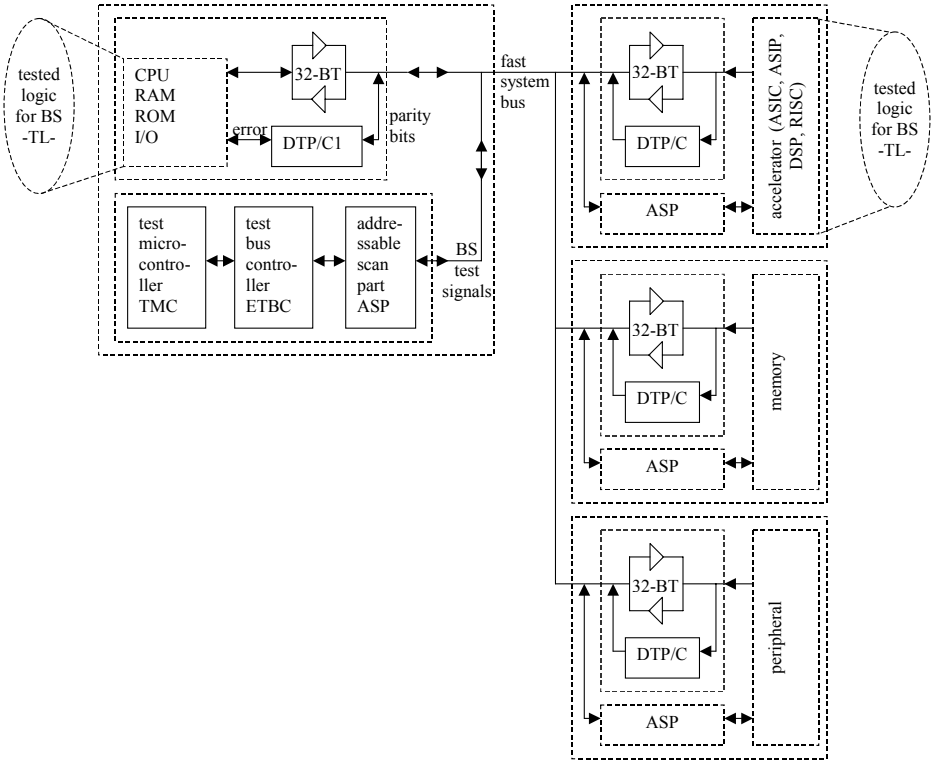

Figure 3: A global structure of embedded system

6. Interconnections as a source of errors

As can be seen from Fig. 1 and Fig. 2 common to both schemes is data transfer through a system bus, i.e. high-speed interconnects. The recent trend in the VLSI industry (SOC and MM designs) and ES architecture toward miniature designs, low power consumption, and increased integration of analog circuits with digital blocks has made the signal integrity analysis a challenging task. The quest for high-speed applications has highlighted the effects of inter connects, such as ringing, overshoots, undershoots, signal delay attenuation, distortion rise-time degradation, reflections, and crosstalk. Interconnects can exist at various levels of design hierarchy such as SOCs, packaging structures, multichip modules, printed circuit boards, and backplanes. In essence the interconnects are responsible for majority of signal degradation in high-speed systems [13]. In order to address these interconnection challenges, ES's engineers need to adopt novel methods for fabrication of VLSI SOCs and multichip ES's architecture, testing, modeling and simulation.

In the case of synchronous systems (such as ESs) dangerous conditions appear when faults that affect signal lines (i.e. bus lines) are sampled (latched) by flip-flops, as constituents of many building blocks (memory, I/O modules), within the embedded system. Therefore, constructing reliable ES represents one of the major design problem because for its correct operation some kind of concurrent error detection, and probably correction, technique should be adopted.

One general approach for concurrent error detection is to encode the outputs of a circuit with an error-detecting code, and to have a checker that monitors the outputs and gives an error indication if a noncode word occurs [14]. In this context we propose a suitable scheme for on-line detection of transient, permanent and delay faults, possibly affecting bus lines of a general 32-bit embedded computer system organized around a single system bus. On-line detection is achieved using of detecting scheme of the kind introduced in [15], here implemented on 32-bit bus transceiver (32-BT) and additionally extended with Boundary scan logic for testing the transceiver circuits to permanent faults (see for example Fig. 3).

7. Types of simple codes for error detection

The process of appending check bits to the information bits is called encoding, the opposite process-extracting the original information bits from a code-word- is known as decoding. In general, any set of objects can be represented by a set of bit strings within which the different bit strings represent the different objects. The set of bit string is called a code, and a particular bit string is called a code word. Any given n-bit code can be regarded as a subset of all possible n-bit strings. String, included in that particular subset are called code words while

strings not included are called noncode words. A code is called an error-detecting code if it has the property that certain types of errors will change a code word into a noncode word. The primary requirements of a code are as follows [16].

(a) it detects all likely errors - within an embedded system an error can be caused by temporal or permanent physical failures and can be defined as the difference between transmitted and received data.

(b) it achieves the desired degree of error detection by using minimum redundancy

(c) the encoding and decoding process is fast and simple, i.e. the corresponding hardware is not complex.

Code can be classified as either separable and nonseparable. A separable code (also called systematic code) is a code in which code words are constructed by appending check bits to the normal output bits. Contrary, in a nonseparable code the information bits are embedded in a code word and can only be extracted by using a decoder. Using the separable code for concurrent error detection has the advantage that no decoding is needed to get the normal output bits. Two types of separable codes that are used for concurrent error detection are Berger codes and parity-checks codes. Parity check-code is a code with the single parity bit equal to the check part of the code word. Because of this, parity checker use a simple structure that is realized as a combinational digital network which computes the sum modulo 2 of the code word bits.

8. Types of error detectors

Many reliable embedded computer systems include error detectors circuits that detect and signal presence of errors. Two main reasons exist for involving error detector in a systems [15]: (a) to prevent the error from reaching the system output; and (b) to locate the position of the error. Error detectors contain checkers, circuits that accept coded information as inputs and determine whether code or noncode words are present at the outputs. The structure of the error detector is determined by the error detecting code.

Fig. 4 shows the general structure of a circuit checked with a separable code. There are three parts [17]: function logic, check symbol generator, and equality checker. The function logic generates the normal outputs, the check symbol generator generates the check bits, and the equality checker determines if they form a codeword.

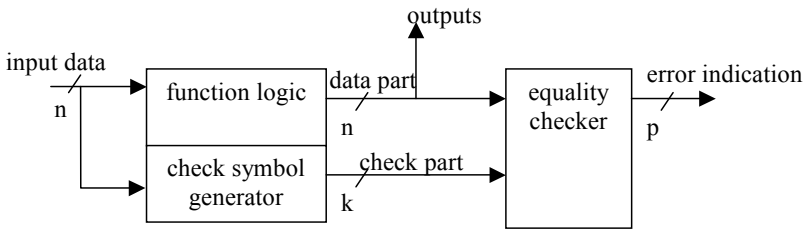

Figure 4: Error detection using a separable code

One widely known separable code is the parity code. It has a very special feature that the check part is a single bit. Because of this, parity checkers use a simpler structure that the general separable code-checker structure of Fig. 4. However a parity check can detect only odd numbers of errors, i.e. even number of errors are undetected.

9. Hardware structure of parity checker/generator

In general, ESs as representatives of computer systems use data in the form of a group of bits (byte, word, double word,...) both for interblocks data transfer and their internal operation. Since there is a possibility that during data transfer and data processing data can be corrupted due to physical defects in the system or intermittent and transient faults, there should be some provisions in the system for detecting and/or correcting erroneous bits in data in order to restore the system to its normal operating mode. This typically requires additional (i.e. redundant) bits to be appended to the data or information bits in encoded data for error detection and/or correction. Thus, the length - the number of bits in encoded data, also known as a code word, is greater than that of the original data [18].

The error detection capability of parity checking may be expanded by including a parity bit for each byte of information. The basic idea is based on partitioning the information bits into several blocks, with each bit appearing in more than one block, and computing the parity for each block. The overlapping of parity bits not only detects more than 1-bit errors, but in the case of a single erroneous bit the location of the bit is also identified [19]. Implementation of this technique requires relatively complex hardware and decreases the systems speed, therefore if is not in the focus of interest for us in this paper.

The structure of parity checker/generator for 9-bit odd-parity code is given in Fig. 5.

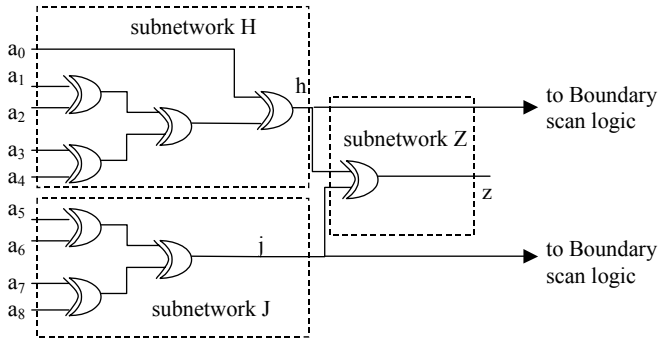

Figure 5: Parity checker/generator for 9-bit odd-parity code

Notice: Input a_8 is connected to logical 1 in the hardware structure of parity generator.

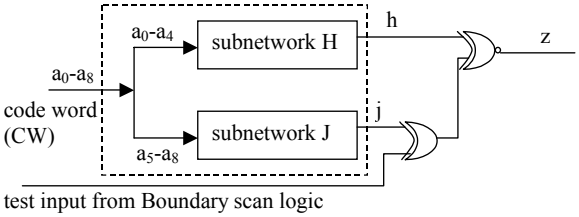

Figure 6: Structure for converting a self-testing checker to a testable single-output checker

When an odd parity code is used, the output *z* equals 1 for all valid code words. The combinational network presented in Fig. 5 can be decomposed into subnetworks H, J and Z each accompanied with a corresponding output, *h*, *j* and *z*, respectively. The self-testing parity checker must have at least two outputs, since a stuck-at-no-error fault on a single output cannot be detected with code word input. The usual practice is to design a self-testing checker with two outputs on which the signal 01 and 10 indicate fault-free operation and the signals 00 and 11 occur in response to a single stuck fault in the checker or a noncode word input. A parity check circuit that has two outputs, each equal to the parity of one of two disjoints subsets of the inputs, is completely self-testing. Most systems with embedded checkers require some indication that an error has been detected by an error checker. From realization point of view some suitable mechanism must be provided to combine the outputs of the individual checkers onto one indicator.

The hardware structure presented in Fig. 6 is used to convert the two-output self-testing checker into a testable single-output checker.

A circuit that combine four pairs of self-testing checker outputs into a single error signal (called 4STC) is sketched in Fig. 7. (This circuit is convenient for error detection in 32-bit computer embedded systems).

The two rail checker from Fig. 7 is a combinational circuits that checks if each pair of inputs has complementary values. It converts the four pairs of signals into one pair of signals that are complements if and only if all of the four input pairs have complementary signals [15].

The logical structure of the two-rail checker design that converts four pairs of input signals to a single pair of output signals is shown in Fig. 8.

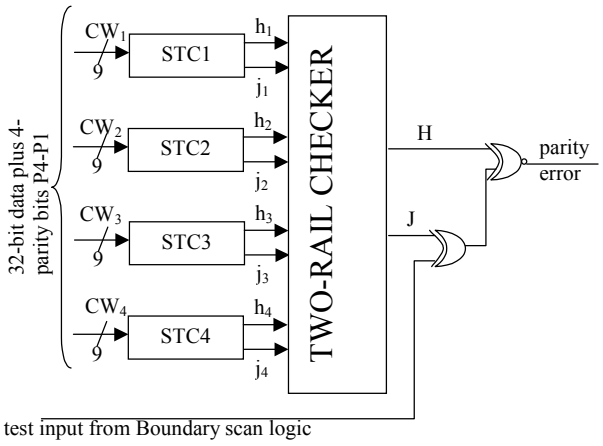

Figure 7: Self-testing checker for 32-bit data plus 4-parity bits, called 4STC

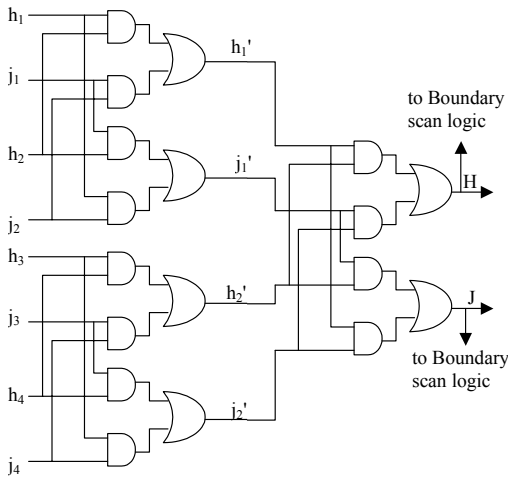

Figure 8: Two-rail checker for four input parts of signals

Notice: Inputs h_i and j_i, i=1...4, drive inputs of Boundary scan logic

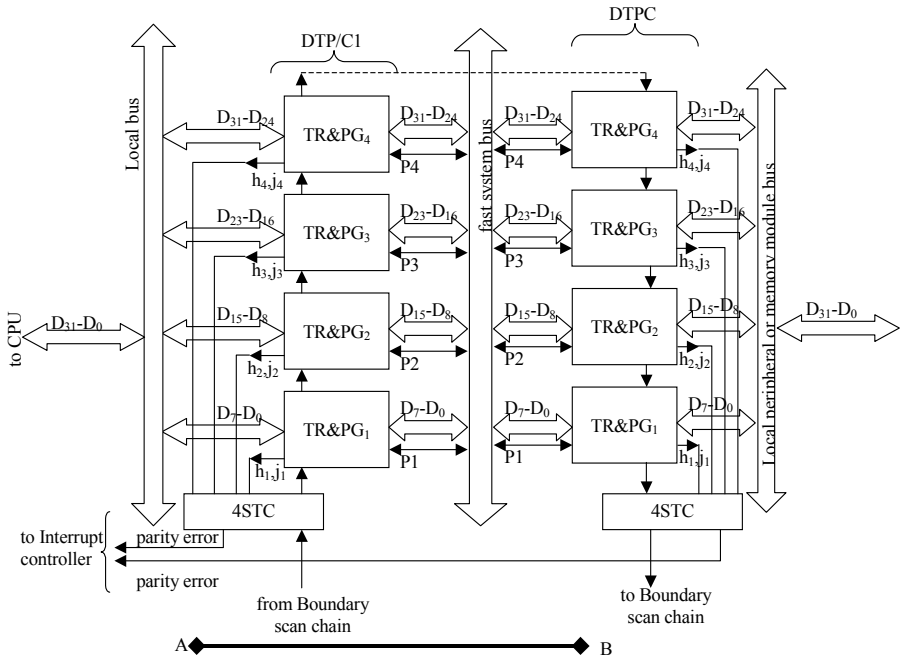

Figure 9: Data transfer through DTP/C1 and DTPC

10. Transceiver and parity generator/checker

As is pictured in Fig. 3 and 9, during data transfer between the host and others constituents (memory, I/O, accelerators) of the system via the system bus, data pass through a pair of transceiver (32-BT) and parity generator/checker circuits PGC.

When the transfer is from A to B the data originated from CPU (local) bus passes through the fast system bus through the DTP/C1 which appends to each byte group a parity check bit *Pi*, *i*=1..4. At the opposite side the DTPC accepts 32-bit data plus four parity bits, transfers data to Local peripheral/memory/accelerator module bus and checks input data for parity error. Odd parity is checked at byte level. If during the transfer some parity error is detected it is signalized by the 4STC. In the opposite direction (data transfer from B to A) transceivers DTP/C1 and DTP/C change roles.

Hardware structures of DTP/C1 and DTP/C are identical. The logic symbol of DTP/C is given in Fig. 10c. Constituents of DTP/C are the following building blocks:

- Direction logic (Fig. 10a): controls direction of data transfer (A to B or B to A).
- 36 bit bus driver stage organized into four groups: each driver stage, see Fig. 10c consists of a pair of three state high current bus drivers D1 and D2, and three multiplexers, MUXAB, MUXBA, and MUXP/C. During normal operation MUXAB and MUXBA transfer data from A_i to C_i and B_i to D_i, respectively (see Fig. 10c), while during test mode they pass through Boundary scan signals BS_{ABi} and BS_{BAi} to C_i and D_i, respectively. Depending on data transfer direction the multiplexer MUXP/C selects a signal from A or B side and directs it to parity generator and checker circuit.
- Parity generator/checker circuits: generates four odd parity check signals. Each of the parity bit P_i, i=1..4, is appended to a corresponding byte group as a a_8 (b_8), a_8' (b_8'), signal.
- 4STC: self-testing checker for 32-bit data and four parity bits.
- Boundary scan logic

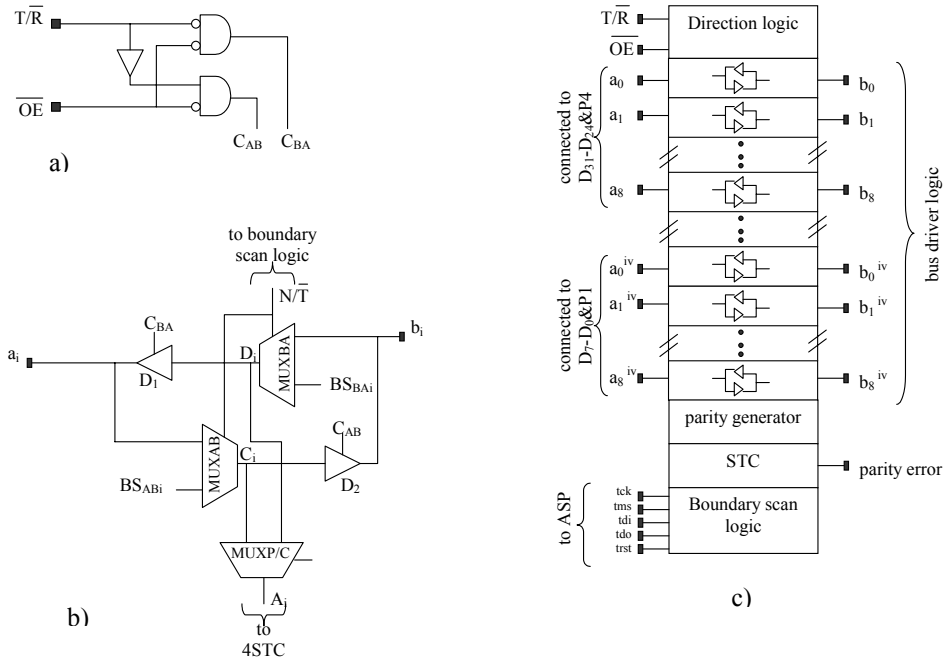

Figure 10: (a) logic for selection of data transfer direction; (b) Bus driver stage; (c) Logic symbol of the transceiver

11. Results

In this section we report our analysis concerning propagation delay, speed of operation, and relative speed decreasing for three different types of bus transceivers implemented as semicustom core part of SOC and glue logic (bridge) within the multichip module into 0,8 μm Double Metal CMOS Standard Cells 5-Volts technology [20].

The first type of bus transceiver, called T1, has identical hardware structure as one described in [21] but with technology defined in [20]. In respect to Fig. 10c) it includes the building blocks three state high current bus driver stages and direction logic.

The hardware structure of the second transceiver, called T2, corresponds to one described in [15]. It has implemented on-line parity error detection logic but without a possibility of static bus transceiver status testing.

The third type of transceiver, called T3, is described in this paper. It has implemented both logic for on-line parity error detection, and logic for boundary scan static circuit testing. The results of our analysis are given in Table I.

Type of transceiver	propagation delay through the transceiver	max. speed of operation	relative speed decr. in respect to T1	high current bus driver stages	on-line parity detection logic	boundary scan logic
T1	8,7 ns (6,1 ns)	115 MHz (163 MHz)	-	+	-	-
T2	13,82 ns (7,2 ns)	72,3 MHz (138 MHz)	59% (18%)	+	+	-
T3	15,15 ns (7,4 ns)	66 MHz (135 MHz)	74% (20%)	+	+	+

Table I: Characteristics of bus transceivers installed in multichip module with capacitive load of C_L=50 pF, and in SOC design with capacitive load of C_L=10 pF (results given in brackets)

By analyzing the results presented in Table I we can conclude the following:

(a) Adding more logic in order to achieve on-line parity error detection (imperative for reliable embedded systems) and possibility for static testing of the bus

transceiver, as a direct consequence, results in increasing of propagation delay through the circuit, i.e. decreasing its speed of operation.

(b) Implementation of boundary scan logic from aspect of increasing the transceiver propagation delay (Type 3 versus Type 2) is not so drastic (especially in SOC design) but the benefits obtained from static circuit testing are more pronounced, i.e. it is easier and faster now to locate the place of possible failure.

12. Conclusion

We have described an approach for detecting parity errors in reliable embedded system. Such error appear as a consequence of transient and permanent faults on bus lines and are on-line detected. The proposed detection scheme of the described 32-bit bus transceiver (36-bit data & four parity check bits) is self-checking with respect to permanent and transient faults. The bus transceiver can be implemented as a custom, semi custom, or FPGA chip.

13. References

1. M. Pflanz, H.T. Vierhaus (2001), "Online check and recovery techniques for depandable embedded processors", *IEEE Micro* Vol. 21, No 5, pp. 24-40
2. H. Chang, et al. (1999), "Surviving the SOC revolution: A guide to platform-based design", *Kluwer Academic Pub.*, Boston
3. B. Murray, J. Hayes (1996), "Testing ICs: Getting to the core of the problem", *IEEE Computer*, Vol. 29, No. 11, pp. 32-38
4. H. Al-Asad, et al. (1998), "Online BIST for embedded systems", *Design&Test of Computers*, Vol. 15 No 4, pp. 17-24
5. R. Leupers (2000), "Code optimization techniques for embedded processors: Methods, algorithms, and tools", *Kluwer Academic Pub.*, Boston
6. M. Birnboun, H. Sachs (1999), "How VSIA answers the SOC dilemma", *IEEE Computer*, Vol. 32, No 6, pp. 42-50
7. R. Bergamaschi, et al. (2001), "Automating the design of SOCs using cores", *IEEE Design&Test of computers*, Vol. 18, No 5, pp. 32-45
8. T. Sudo (1995), "Present and future directions for multichip module technologies", *IEEE Journal of IEEE Solid-state circuits*, Vol. 30, No. 4, pp. 436-442
9. G. DeMicheli, R. Gupta (1997), "Hardware/software co-design", *IEEE Proceedings*, Vol. 85, No 3, pp. 349-365
10. W. Wolf (2000), "Computer as components: Principles of embedded computing system design", *Morgan Kaufmann Pub.*

11. S. Edwards, et. al. (1997), "Design of embedded systems: Formal models, validation, and synthesis", *IEEE Proceedings*, Vol. 85, No 3, pp. 366-390
12. W. Cesario, et. al. (2001), "Colif: A design representation for application-specific multiprocessor SOCs", *IEEE Design&Test of computers*, Vol. 18, No 5, pp. 8-20
13. R. Achar, M. Nakhla (2001), "Simulation of high-speed interconnects", *IEEE Proceedings*, Vol. 89, No 5, pp. 693-728
14. Lala P. (1985), "Fault tolerant and fault testable hardware design", *Prentice Hall*, Englewood Cliffs N.J.
15. McClusky E. J. (1990), "Design techniques for testable embedded error checkers", *IEEE Computer Magazine*, Vol. 23, No. 7, pp. 84-88
16. Rao T. R. N, Fujiwara E. (1989), "Error-control coding for computer systems", *Prentice Hall*
17. Tuba N.A., McClusky E. J. (1997), "Logic synthesis of multilevel circuits with concurrent error detection", *IEEE Transaction of CAD*, Vol. 16, No. 7, pp. 783-789
18. Parag K. Lala (2001), "Self-checking and fault-tolerant digital system design", *Morgan Kaufman Publishers*, San Francisco
19. Sarazin D. B., Malek M. (1984), "Fault-tolerant semiconductor memories", *IEEE Computer*, Vol. 17, No. 8, pp. 49-56
20. Austria Mikro Systeme International (1996), *2.0-Micron, 1.2-Micron, 1.0-Micron and 0.8-micron Standard Cell Databook*
21. Texas Instruments (1998), *ABT Logic - Advanced BiCMOS Technology - A High-Performance Line of 5-V Products Data Book*