Novel Gate Design Method for Short-Duration Test

Jan Bělohoubek

Dept. of Digital Design, Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic

jan.belohoubek@fit.cvut.cz

Abstract. *In this paper, a novel logic gate design method will be presented. This method allows to test combinational parts of the circuit using a short-duration offline test. Short-duration offline tests are usable when fault-recovery in duplex-based systems is required and downtime should be minimized at the same time. The presented method adopts some principles from dual-rail logic and asynchronous circuits design.*

Keywords

Offline-test, C-element, dual-rail logic, preset, testability, controlability, observability.

1. Introduction

As digital systems are becoming more complex, technology scales down and new applications in challenging environments are continuously developed, pressure to digital systems dependability still rises. In the past years, there were developed many techniques to build dependable systems while saving resources.

A dependable system is able to correct physical fault consequences to prevent system failure.

In the literature, different fault types are described. These are *permanent faults*, (long and short duration) *transient faults* and *intermittent faults* [1].

More models can be used for fault modeling in digital circuits such as *open and short faults* or *bridging faults*. In this paper, a *stuck-at-fault model* for *permanent faults* is used.

Methods for construction of dependable systems are based on redundancy – *area redundancy* (hardware duplication), *time redundancy* (recomputation, software redundancy), or *information redundancy* (coding). Short-duration transient faults can be well detected (and corrected) using time redundancy, while area redundancy is used for long-duration transient faults and permanent faults. For intermittent faults it depends on their behaviour.

In the area redundancy, *duplex systems* may be used for error detection. Duplexes are created by duplicating functional modules and adding a *checker*, which signalizes data validity using a dedicated signal (this signal is in *ERROR* or *OK* state). *TMR systems* based on module triplication and *voters* are used for error correction [1].

The goal of this work is to develop a method which will allow to use a duplex scheme to correct detected errors. In the past, this was done in two different ways.

The first possibility is using of *self-checking modules* [2], [3]. If self-checking approach is used, at least one of the modules in error-correcting duplex must be self-checking. The second possibility is *offline testing*.

The main disadvantage of self-checking modules is the area overhead. On the other hand, offline tests are usually time consuming or/and have not reasonable fault-coverage. The disadvantages are following: the downtime[1] is too long and fault-recovery may be impossible. In this paper a new method to design short-duration offline tests with 100% gate-level stuck-at-fault coverage will be presented.

2. Stuck-at-fault observability and controlability

In this section, fault observability and controlability will be discussed. Stuck-at-fault model will be considered.

To design short-duration tests with reasonable fault-coverage, testing of all faults by only few test vectors is required. Very simple test vectors `all-zero` or `all-one` are used. When applying these input vectors, observing similarly simple output vectors is the advantage (ideally the same vectors in case of *fault-free* circuit). Detection of all possible faults is required at the same time.

When using `AND-OR` based logic and simple vectors described above, `OR` gates tend to mask `s@0` faults and `AND` gates to mask `s@1` faults. Meeting both requirements is possible when symmetric elements are used. Note that the requirement for simple output values can be disturbed by usage of inverters – the circuit must be *monotonic*.

Thanks to the reconfiguration ability, similarly short tests are feasible on FPGAs [4] configuring FPGA LUTs as

[1] Time when the circuit is not operational because the offline-test is in progress.

XORs and NXORs. Time-consuming full circuit reconfiguration is required to perform these tests.

In common logic circuits, both controlability and fault-observability are very difficult tasks. Testing all possible faults implies large tests with a huge amount of test-vectors. Moreover, some faults can be untestable because of the *reconvergence* [5]. The mentioned XOR gates are very sensitive to reconvergence.

This work exploits fault-observability and fault-controlability in another type of circuits, where all gate-level faults are observable independently of the reconvergence. The next advantage is that all gate-level multiple faults are also detected. These circuits can be tested using very simple test vectors as denoted above. The proposed circuits are completely based on *C-elements*, which are extensively used by designers of asynchronous systems [6]. C-elements are symmetric and *state-holding*. As will be described in this paper, state-holding C-elements could realize *monotonic* logic gates. Inverter problem solution will be described in section 3.

2.1. Fault-observability and C-element

Probably the most commonly used implementation of C-element is a *semi-static C-element* (see Figure 1). Another interesting implementation is a *dynamic C-element*. The structure of the dynamic C-element differs from semi-static C-element only in the absence of the *weak inverter*.

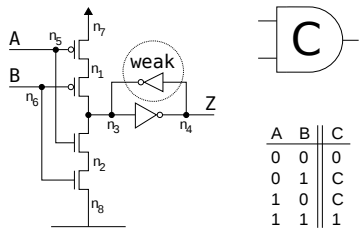

Fig. 1. Semi-static C-element implementation, symbol and truth table.

C-element is symmetric and state-holding. As explained in further text, this property in fact allows to test C-element nets for both: s@0 and s@1 faults.

Assume a circuit composed of interconnected C-elements only. At first, a test at the gate level will be described. When all inputs of this C-element-based circuit are set to 0, after all transitions are finished, outputs of all C-elements are 0 (including circuit outputs). At this moment, when all inputs of such a circuit will be changed to 1, outputs of all C-elements should switch to 1. But if there is a s@0 fault present, then the respective C-element output will hold 0 and this value is propagated along the whole path to the circuit outputs – the fault is observable.

Similarly, s@1 faults can be observed. Fault presence prohibits C-element transitions on the path from the fault to the circuit outputs. This implies that all stuck-at-faults are testable in the C-element-based circuit. The gate-level test has 100% fault-coverage.

Now we will shortly describe faults at the transistor level [8]. In Figure 1, 8 signals are denoted (n_1 – n_8). For simplicity we will assume only those faults that take effect in all forks of the signal. Both – s@0 and s@1 – at denoted signals are testable in the C-element. All faults except of s@1 at signal n_1 and s@0 at signal n_2 are covered by gate-level test described above. Testing of the uncovered transistor-level faults will be discussed later.

3. Proposed gates

As described in Section 2, a circuit containing the C-elements only has good fault-observability and fault-controlability properties. It will be described in this section, how regular logic gates can be constructed using C-elements only. To achieve this, *dual-rail logic* will be used.

3.1. Dual-rail logic

In general, dual-rail logic uses two complement signals to transfer one logic value. *Dual-rail quasi-delay insensitive* (*QDI*) is a design style widely used in the asynchronous design area.

Traditional AND and OR gates are used in QDI. The states of the signals in QDI are interpreted as follows:

00 – spacer

10 – logic one

01 – logic zero

11 – erroneous state

Before the computation, all inputs are preset to 00 in QDI. This causes that values on all gate outputs in the circuit are set to 0. The transitions from 00 to 01 or to 10 are used for *completion detection* (11 can be used for error detection [2]).

Two phases are also regularly switching: the *preset phase* (preset to 00) and the *computational phase* (transition to 01 or 10). Note that dual-rail logic with complementary signals allows to implement NOT function as a *wire-swap* only.

3.2. Gate levels

In the following text we will use terms *gate level* and *circuit depth*. The level is the maximal distance from the circuit inputs. For gates connected to circuit inputs we define their level to be 1. The level of the n-input gate G is:

$$l_G = max\{l_0, l_1, \ldots, l_{n-1}\} + 1 \quad (1)$$

where l_i is the level of the gate connected to i-th input of the gate G. The circuit depth of a k-output circuit C is:

$$d_C = max\{l_0, l_1, \ldots, l_{k-1}\} \quad (2)$$

where l_i is the level of the gate connected to the i-th circuit output.

3.3. Generalized C-element

In Figure 2 you can see the structure of a generalized C-element. A generalized C-element is similar to the C-element presented before [6]. In generalized C-element, there are two driving *MOS nets* i.e. *N-MOS* and *P-MOS* denoted `N` and `P` respectively. Both may implement any meaningful function created by using *P-type* or *N-type* transistors respectively [7].

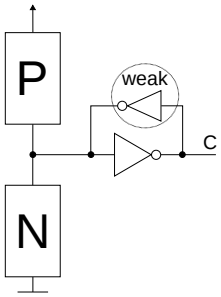

Fig. 2. Generalized C-element. The memory element is driven by N-MOS and P-MOS nets connected to VCC and GND respectively.

3.4. Generalized C-element based dual-rail gates

Combining properties of previously described approaches, C-element-based regular logic gates can be created. Next lines will describe how to construct dual-rail `AND/NAND` gates. Dual-rail gates are dual structures. Both functions (`AND` and `NAND`) are implemented using the same `AND/NAND` structure with inverted *output signal meaning*. Dual-rail logic with complementary signals will be used because *monotonic* behaviour is required. Dual-rail `OR/NOR` gates can be also created similarly using the proposed transistor structures. Implementation of `XOR` gates is not possible because *XOR* is not monotonic and thus it cannot be implemented using the state holding C-element.

In QDI, preset phase is driven by a special – spacer (`00`) – value on circuit inputs. To implement dual-rail gates using C-elements only, we need to preset some C-elements to `1` and the other to `0`. Therefore we need a special *preset* signal for each C-element.

As illustrated in Figure 3, `AND/NAND` gate can be created by using two C-elements with a preset signal. The first C-element is in *preset phase* (using signal `P`) preset to `0` and holds this value while at least one of the inputs stays `0`. Input signal transitions are performed during the *preset phase*. If both inputs are `1`, output will change to `1` in the computational phase. This behaviour corresponds to the `AND` gate. For the second C-element, there are similar assumptions: it is preset to `1` and this value persists on the output while at least one input is `1` – this is obviously a `NAND` gate when assuming that inputs of the C-element are inverted.

Combinational circuits can be constructed using these gates. Every gate in these circuits works in two phases. Both phases are also illustrated in Figure 3. When the signal `P` is asserted, the *preset* phase is running – C-elements are preset to the specified value. All transitions on input signals `A` and `B` should finish during this phase. In the second phase inputs will be evaluated and the outputs of both C-elements will stabilise in correct states corresponding to the dual-rail `AND/NAND` behaviour.

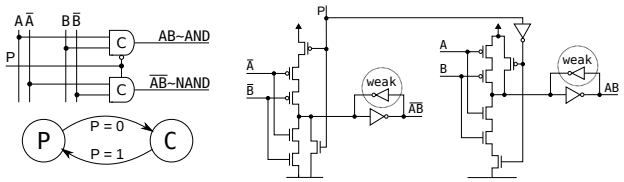

Fig. 3. Dual-rail AND/NAND gate implemented using two generalized C-elements (with preset) and state transition diagram for gate-operating phases (P $\sim$ preset; C $\sim$ computation).

The CMOS implementation in Figure 3 demonstrates the behaviour of the gate implemented using C-elements with preset. A gate implemented this way has not a good controlability and it is not possible to test all mentioned faults. In Figure 4, the complete proposed `AND/NAND` gate is shown. Additional signals (compared to Figure 3) bring symmetry into the gate design and increase the gate controlability. *Test signals* $\mathtt{T}_1$ and $\mathtt{T}_2$ and *input-enable signals* $\mathtt{E}_1$ and $\mathtt{E}_2$ will be described in the following lines. During normal operation (preset and computational phases): $\mathtt{P}_i = \mathtt{E}_i$ and $\mathtt{T}_1 = 1$ and $\mathtt{T}_2 = 0$.

In preset phase:
$\mathtt{P}_1 = \mathtt{E}_1 = \mathtt{T}_1 = 1$ and $\mathtt{P}_2 = \mathtt{E}_2 = \mathtt{T}_2 = 0$.

In computational phase:
$\mathtt{P}_1 = \mathtt{E}_1 = 0$, $\mathtt{T}_1 = 1$, $\mathtt{P}_2 = \mathtt{E}_2 = 1$ and $\mathtt{T}_2 = 0$.

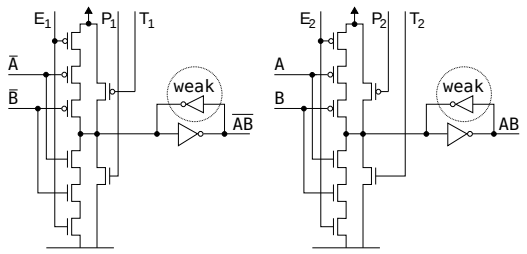

Fig. 4. Dual-rail AND/NAND gate implementation with preset. Signals E_1 and E_2 are input enable signals and signals T_1, T_2 are only for test reasons.

3.5. Testability of proposed gates

Additional signals (P_i, T_i and E_i) improve symmetry and controlability compared to the C-element described in 2.1.

Stuck-at-faults on inputs A, B, $\overline{A}$ and $\overline{B}$ and on enable signals E_1 and E_2 can be tested the same way as described for the semi-static C-element in 2.1 conditioned that preset signals are *inactive* (i.e. $P_1 = 0$, $P_2 = 1$, $T_1 = 1$ and $T_2 = 0$). To test all faults, the process of the test must be following: at first 0 should be set to all circuit inputs and enable signals, then all inputs and enable signals should be set to 1, and finally back to 0. The delay between value changes should be equal to the circuit propagation time. Values 0, 1 and 0 will be observed on all circuit outputs if the circuit is *fault-free*. When a fault is detected during this test, it will be distinguished as an inverted value on the affected outputs. Denote this test *test No. 1*.

But additionally, there is a need to test at least the preset function (signals P_1 and P_2) of every gate. Complementary signals T_1 and T_2 are necessary because they bring symmetry into complementary gates and allow full-testability.

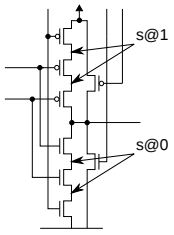

Fig. 5. Faults difficult to test in proposed C-element.

Additionally faults shown in Figure 5 can be observed only if input signals and input enable signals differ.

All remaining s@0 and s@1 faults (not tested in test No. 1) can be observed during the *topological wave test*. The wave test progress for the whole circuit is as follows: in the first phase – all preset (and test) signals are set to *inactive state* and all circuit inputs and enable signals are set to 0 – this value is then propagated to the circuit outputs (note, that this phase is identical to the last phase of the test No. 1).

After the first phase is finished, for gates in level 1 enable signals E_1 and E_2 stay 0. For the other gates, E_1 and E_2 are set to 1. Then all circuit inputs are set to 1. This will allow value 1 propagation in the whole circuit. If s@0 is present, the path from this fault to the affected outputs will hold the value 0. Gates in level 1 stay intact because they are set to be sensitive only for zeroes on the inputs.

At this moment (borderline between t_4 and t_5 in Figure 6), the output vector of the circuit should be compared with the expected response – zeroes should be only on the outputs connected to gates in level 1. This will allow to detect some faults from Figure 5 depending on the circuit structure[2]. Naturally, this moment should be extended to a time-interval of the reasonable length.

When *propagation paths* are prepared, gates in level 1 are preset to 1 using preset signals P_1 and T_2. After circuit-propagation delay, value 1 should appear on every circuit output. If the value 0 preserves, a fault has been detected.

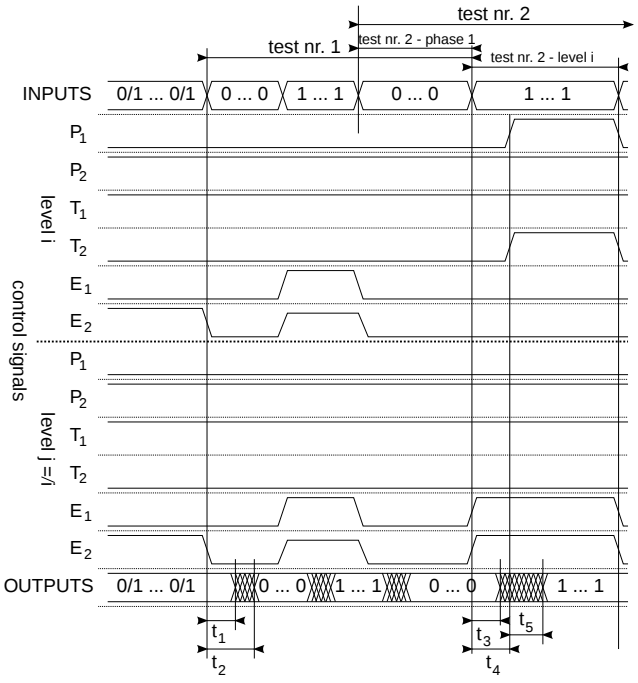

Fig. 6. Test progress for *test No. 1* followed by first phase of *test No. 2* and i-th level-test (first phase of *test No. 2* and i-th level-test alternate until all levels are tested). Delay denoted t_1 is the *minimal circuit propagation time*, t_2 is *maximal circuit propagation time*, t_3 is the minimal propagation time from circuit inputs to circuit outputs when gates in level i are inactive, t_4 is the maximal propagation time from circuit inputs to gate inputs in level i and t_5 is the maximal propagation time from gates on level i.

[2]To detect all mentioned faults, gate levels must be divided to *sub-levels*. Each circuit output is connected to max. 1 gate from the given sub-level. The test is then not level-based but also sub-level-based – replace any "level" term in the text by the "sub-level" term.

After the test of all gates in the first level is done, gates in another levels are tested in topological order in the same way:

At first all $\mathtt{E}_i$ signals in all gates and circuit inputs are set to `0`. Then $\mathtt{E}_i$ signals for all gates except of gates on the tested level and circuit inputs will be set to `1` and finally, signals $\mathtt{P}_1$ and $\mathtt{T}_2$ in gates on the tested level will be set to `1`. After the propagation delay a fault or the correct value can be observed on every circuit output.

The test will proceed in this order until gates in all levels are tested. On the circuit outputs, values `0` and `1` should switch regularly in well defined time intervals and with delays based on gate depths. This test will be denoted as *test No. 2*.

After the whole test No. 2 is finished (all circuit levels were tested), another test should be done for signals $\mathtt{P}_2$ and $\mathtt{T}_1$ analogously. Setting circuit inputs to `1` and presetting to the value `0` will be performed during this test. The last test will be denoted as *test No. 3*.

By concatenating all the tests – test No. 1, test No. 2 and test No. 3 – the *complete test* is designed. Our complete test is able to detect all faults described above.

As it can be deduced from previous lines, the test has the initial part, which is the same for every circuit, but additionally the test duration depends primarily on the circuit depth (the number of gate levels). Thus smaller circuit depths are advantageous. The test length is approximately:

$$(2 + (2 \cdot 2k)) \cdot t_p, \tag{3}$$

where k is the circuit depth and t_p is the circuit propagation time.

4. Conclusions

New method for logic gate design was presented. The proposed method allows short-duration offline testing of circuits composed of these gates. Fault controlability and observability properties were discussed and complete test progress was presented.

Acknowledgements

This research has been in part supported by CTU grant SGS15/119/OHK3/1T/18. This work was supervised by Prof. P. Fišer, FIT CTU in Prague and Dr. J. Schmidt, FIT CTU in Prague.

References

[1] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[2] I. David, R. Ginosar, and M. Yoeli, "Self-timed is self-checking," *Journal of Electronic Testing*, vol. 6, no. 2, pp. 219–228, 1995. [Online]. Available: http://dx.doi.org/10.1007/BF00993088

[3] J. Ruan, Z. Wang, K. Dai, and Y. Li, "Design and test of self-checking asynchronous control circuit," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, ser. Lecture Notes in Computer Science, N. Azémard and L. Svensson, Eds. Springer Berlin Heidelberg, 2007, vol. 4644, pp. 320–329.

[4] M. Renovell, J. Portal, J. Figuras, and Y. Zorian, "Minimizing the number of test configurations for different fpga families," in *Test Symposium, 1999. (ATS '99) Proceedings. Eighth Asian*, 1999, pp. 363–368.

[5] L. Biwei, C. Shuming, and H. Xiao, "Analysis of glitch reconvergence in combinational logic ser estimation," in *Modeling Simulation, 2008. AICMS 08. Second Asia International Conference on*, May 2008, pp. 1015–1020.

[6] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective*, 1st ed. Kluwer Academic Publishers, Boston, 2001.

[7] D. Thompson, "Improved c-element and logic reduction and completion detection circuits," Jan. 12 2006, wO Patent App. PCT/GB2005/002,412. [Online]. Available: http://www.google.com/patents/WO2006003368A2?cl=en

[8] J. Brzozowski and K. Raahemifar, "Testing c-elements is not elementary," in *Asynchronous Design Methodologies, 1995. Proceedings., Second Working Conference on*, May 1995, pp. 150–159.

About Authors...

Jan BĚLOHOUBEK was born in 1990 in Pilsen. He received the bachelor's degree in Computer Engineering at University of West Bohemia in Pilsen. The master's degree he received in Digital System Design at Czech Technical University in Prague. Today he is in the 1st year of Ph.D. programme at Faculty of Information Technology at the same university.