Shubinsky I.B., Schäbe H.

A SYSTEMATIC APPROACH TO PROTECTION AGAINST GLITCHES

The paper describes the essence of glitches, their emergence and influence on the results of a system's operation. Methods of identification of transient failures are considered. The paper describes methods of protection against glitches and offers an integrated approach to ensuring the resilience of a system to glitches. The principle put forth in the paper shows what key property a system has to own to be resilient to glitches. The paper further provides an example and a conclusion on the study results.

Keywords: *glitches, transient failures, data corruption, distortion of a program sequence.*

1. Introduction

Usually the theory of reliability and safety pays much attention to failures, in particular hazardous ones. Besides them, however, there are other types of events that at first glance seem to be less important. These are glitches [1,2].

Glitches are defined the following way. Functional glitches are events non-recurrently distorting information processed or stored in IT systems caused by external or internal destabilizing factors (disturbances) [1].

The issue has become more critical for two reasons. First, the miniaturization of computer systems leads to an increasing sensitivity of hardware. Second, the weakening of the Earth magnetic field leads to a growing number of charged cosmic particles arriving at the Earth's surface. As a result, the rate of glitches is three or more times greater than the failure rate of hardware [1].

This paper presents a systematic method for developing a system that is resilient to glitches.

The second section of the paper describes what glitches are, how they occur and what follows. The third section discusses detecting methods. The further section describes methods of protection against glitches and provides a systematic approach. The principle presented below shows what main feature a system should have to be resilient to glitches. At the end of the paper an example is provided and a conclusion is made.

2. Glitches

In information systems, glitches that belong to intermittent failures have no significant effect on the reliability of an information system. This is explained by the following reasons:

• Many information systems are classified as mission-critical systems. Therefore, when designing an information system special measures to stabilize the system are taken regarding acceptable limits of temperature and humidity, required conditioning of rooms and racks, careful design of electrical schemes, excluding events as races and competion of signals, etc.

• Many information systems work in real time and perform well-defined informational tasks according to a given schedule. For these systems, the results of operation are primarily influenced by functional glitches, i.e. such hardware failures that, under certain circumstances, can lead to management errors and have serious consequences for the entire system, and even for the environment.

• Glitches are characterized by the fact that they exist only for a very short time. They occur in hardware and disappear by themselves, i.e. hardware returns to its original physical state, but not necessarily to its original functional state.

Hardware glitches can be caused by:

Disturbances of inputs and power supply

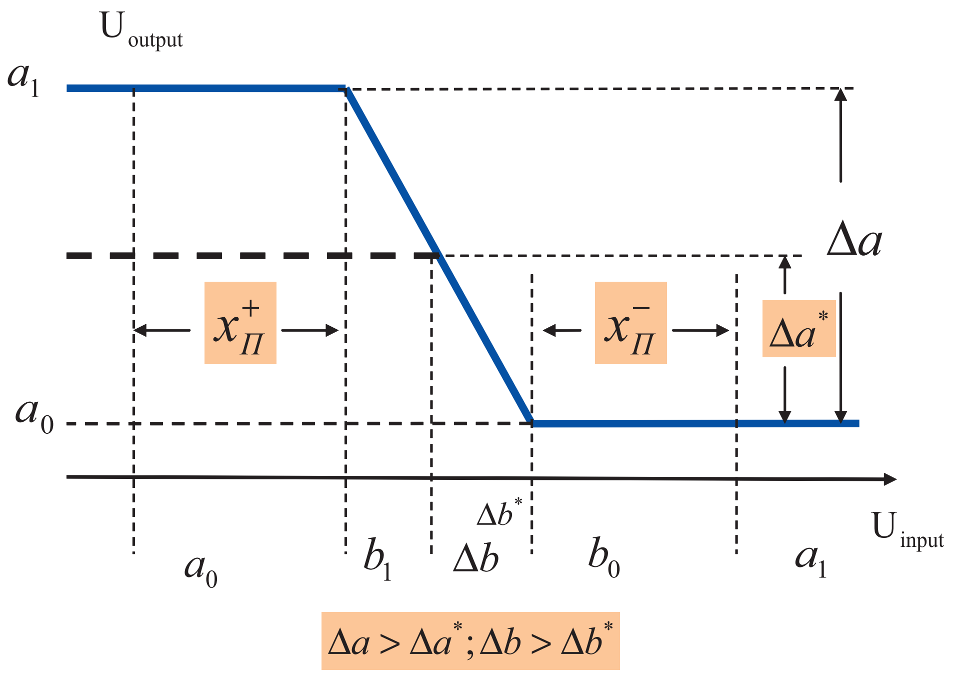

Fig. 1. Typical transfer characteristics of a digital integrated circuit

The actual transfer characteristic of a circuit can be approximated by a piecewise linear function. Fig. 1 provides an example of transfer characteristics with their parameters: a_i (i = *1.0*) is the output voltage level of a log book or zero; b_i is the threshold value of input signal for switches like 1 → 0 (i = *1*) or 0 → I (i = *0*); $\bar{b} = \frac{b_1 + b_0}{2}$ – the possible band of switching $\Delta b = /b_1 - b_0/$. For a typical transfer characteristic, we single out three areas corresponding to different states of a circuit. Area I is closed state of a circuit, area III is open state of a circuit, and intermediate area II is switching state. Under the influence of trigger disturbance, the circuit can move from area I to area III, i.e. producing an incorrect output code "I" instead of the "0" code (glitch of I → 0 type) . Under the influence of blocking disturbance, there can be a glitch of 0 → I type.

From fig.1 it can be seen that for a given value of logic swing Δa, a technological variance between threshold voltages b_1 and b_0 leads to smaller admissible amplitudes of input disturbances x_D^+ and x_D^-, and consequently to reduction of circuit noise immunity. The similar result occurs in case of reduction of logic swing Δa, which is made to increase a circuit speed of response. The smaller is the value of logic swing $\Delta a^* < \Delta a$, the lesser is the time of circuit switching $\Delta b^* < \Delta b$ and thus the better is its speed of response.

Experimental data on glitches of integrated circuits are not available. To assess the characteristics of glitches, they use experimental and computational methods of prediction based on experimental data about on input signals and disturbances, transfer and amplitude frequency characteristics of circuits. These data are source data for calculating a range of possible probabilities for transient failures of a digital logic element in an integrated circuit, which is often called as a gate similar to a transistor in the second generation element base. Based on the calculated values of gate transient failures, the known structure and functions implemented, glitch rate and probabilities are calculated for digital integrated circuits (IC).

Electromagnetic interference

If disturbances can affect the system through inputs, power supply and weight, then the system can also be exposed to electromagnetic interference. Consequently, it can induce a disturbing current. This disturbance influences the same way on circuit contacts as a disturbance coming directly through contacts and wires and as described above.

Passing of radioactive particles

The third mechanism is the impact of radioactive particles. Usually, at the first step gamma rays pass close to IC. This is likely to be radiation produced due to radioactive decay. Gamma radiation has the property of passing through material easily, and it is difficult to protect against it. A gamma quantum can yank out an alpha particle that as a charged particle can come to inputs or directly inside of IC on registers and memory. Due to this impact, at least one bit might be distorted.

The consequences of glitches can be as follows:

• *Distortion of written data.*

In this case, the system functions on the basis of incorrect information. In this case, even correctly executed functions provide a false result. And this process continues further if an error in the data has not been corrected. An error can get into the output results of IC functioning, making a false command and causing a hazardous failure.

• *Changing a program sequence.*

This means that the system will not perform a function at all, or will perform another function instead. There can also be events of too early, too late or inappropriate performing a function, which as a result can lead to a hazardous failure.

3. Detection of transient failures and protection against their hazardous consequences

There are a number of well-known methods of protection against the effects of transient failures [1]. First of all, methods for detecting transient failures are required. The following methods of detection are known:

• *Multi-channeling.* In this approach, the results of two or more channels are compared. If their results differ, a failure or a glitch is detected in a system.

• *Checksums.* Besides the data themselves, a checksum is saved. Changing the data will cause an event when a checksum received and a checksum newly obtained on the basis of the received data will not match. Using this method, we can detect many faults, but not all of them.

• *Algorithms*. Developers use algorithms that are tolerant to transient failures. However, for some tasks one cannot find or construct such algorithms.

• *Multi-version programming*. Based on the same algorithm, two or more versions of the software are developed. The results are compared. Since the impact of a transient failure is supposed to be different for both of the software versions, glitches have to be detected.

These well-known, classical methods of protection are suitable for a wide class of faults and failures.

After having detected transient failures, one has to eliminate them in a system using HW/SW means. However, additional HW/SW means used for detection and elimination are also susceptible to transient failures. Therefore, they should also be protected against transient failures.

4. A systematic approach

In this section we construct a systematic method for protection against transient failures.

A system failure caused by a glitch represents itself through violation of logic conditions of system functioning as well as through decreased accuracy or capacity of a system. These distorted intermediate or output results are registered by a system directly or indirectly. And implicit saving is more dangerous, as it is more difficult to detect.

An example of direct saving is the distortion of data such as system parameters. Indirect saving occurs, for example, if a system performs an incorrect operation that will lead to distortion of data, which are then saved to the memory.

Software can be imagined as finite-state automaton. The state of automaton is defined by the data recorded in the memory that either came from the outside (sensor data, the state of external devices etc.) or were derived as a result of the software processes themselves (calculation results, commands from software to external devices etc.). We should keep in mind that all information is discrete and depends only on the entire history of the system, but not on its future.

Now the computing system can be described as follows. Its state at a given moment of time is described as a vector in a space of a system's states that contain all necessary information for the unambiguous description of a system: all data received, SW code, register values etc. Though of a big size, a state vector is finite. Then software and a digital system can be described by means of a Markov chain model. This is explained by the fact that in a Markov chain model as in a computing system, the behavior of a system in the future depends on the present and does not depend on when and how the system came into that state.

Therefore, the task of protection against glitches is bolted down two subtasks:

1. Protection of a fixed state against transient failures;
2. Protection of the transition into next state against distortion by transient faults.

In order to solve both of these subtasks, for each of them the following measures can be applied:

1. *Checksums, redundant recording.* With these methods it is possible to detect distortion of data about the system state, and if the method of information storage includes redundant information, it is even possible to restore distorted information. This may be possible if one uses an error-correcting code, as applied on CD, or information is saved twice with separate checksums for each version.

In order to have a safe working system, one can implement a finite-state machine, which reflects all important aspects of the information processing process, especially the distinction between hazardous and safe states. It is important to find or construct a state machine, which is as small as possible.

The standard EN 50128 (2011) in Annex D.24 mentions a method of Finite State Machines [3]. It is this measure that was motivated by protection against transient failures.

2. *Multi-version programming*. If software is programmed in various versions carrying out calculations using different methods, transient failures cannot distort the data processing in both versions of the software in such a manner that the results will be the same. It is also possible to apply the same code on different computers where it is translated in a different way in different executable software. If a finite-state machine is used, before each state transition one can set checking that uses different algorithms.

5. Example

This section gives an example of a system that uses a systematic approach for protection against transient failures presented above. It is assumed that a safe computer is required for calculations. In addition, these computations have to be carried out cyclically. For example, the braking distance is computed every 100 milliseconds in order to verify that the train will be able to stop before the next signal at danger.

As the first step, we should define those data, which clearly describes the state of a system. In this example, these are a position, speed, deceleration and a braking curve, which describes the braking parameters of a train. Further, the time or cycle number is required. Obviously, these data should be written into memory together with a checksum. Then it is possible to detect data corruption caused by glitches. If, however, it is important not only to detect but also restore the correct data, i.e. restoring the correct state, then it is necessary either to store data redundantly or to use error-correcting codes. In the first case, the data is stored twice with a hash code. Then it is easy to determine which version of the data has been damaged. In the second case, one needs to pay attention to the fact that with a large number of transient failures a false recovery, i.e. recovery of wrong state would become probable.

In any case we need to prove by calculations that the measures implemented against transient failures detect errors with a sufficiently high probability, so that the probability of undetected distortions is sufficiently low.

As the second step, the system needs to be protected against the consequences of transient failures when transition from one state to another occurs. The easiest option is to determine the next state several times. Calculations are carried out twice, so that possible errors can be detected. If in this case the calculation is repeated for the third time, then an error can be corrected by switching to that state, which has been determined by the majority of calculations.

In this example, calculations are repeated on the same processor and with the same software version, as in the first calculation. This is possible because transient failures will not lead to a permanent failure. It should be noted that the described measures do not protect against permanent or systematic failures.

In connection with the transition, however, one needs to consider the software e that carries out calculations to determine the subsequent state. If a transient failure distorts the software, then the software remains in a distorted form in the memory, although the memory would store the software correctly at the next occasion of software uploading to the system. Therefore, the software must be written into memory together with a checksum. Moreover, for each application of this software, i.e. also when re-computing the transition to the next system state, it is necessary to check the integrity of the software (or portions thereof) with the checksum.

6. Conclusion

In this paper, we have studied transient failures. We have described the causes of transient failures and methods of protection against them. Hence, we have derived a systematic approach, which consists in two steps. At the first step, the system is represented as a Markov automaton, the state of which is memorized using protection methods, for example using a checksum. At the second step, a reliable method of transition between states is constructed, e.g. using redundant calculations. The approach is illustrated by an example.

References

1. **Shubinsky I.B.** Functional reliability of information systems. Moscow, 2012.

2. **Haiyu Qi, Sanka Ganesan, Michael Pecht.** No-fault-found and intermittent failures in electronic products. Microelectronics Reliability 48 (2008) 663-674.

3. EN 50128 Railway applications – Communication, signalling and processing systems – Software for railway control and safety systems, 2011.