ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

UDC 629.7.014-519.05.083:519.713 **doi: 10.32620/reks.2024.1.15**

Valery SALAUYOU

Bialystok University of Technology, Bialystok, Poland

DESCRIPTION STYLES OF FAULT-TOLERANT FINITE STATE MACHINES FOR UNMANNED AERIAL VEHICLES

*The **subject matter** of this article is finite state machines (FSMs), which are used as control devices in unmanned aerial vehicles (UAVs). The **goal** of this study is to develop description styles for fault-tolerant FSMs in hardware description languages (HDLs) that prevent failures in the state register and in the input vector of the FSM. The **tasks** to be solved are as follows: development of description methods for FSM transitions from illegal states in case of failure in the state register, as well as for FSM transitions from each state in case of failure in the input vector; determination of FSM output vector values in case of the above failures; development of description styles for fault-tolerant FSMs; and investigation of the efficiency of the proposed description styles for fault-tolerant FSMs. The **methods** used are: the theory of finite state machines, state encoding methods of FSMs, description styles of FSMs, and Verilog hardware description language. The following **results** were obtained: two styles of describing fault-tolerant FSMs have been developed, safe0 and safe1, which do not increase the area and do not decrease the performance of FSMs, and in some cases allow the area to be reduced (for some examples by a factor of 4.8) and increase the performance (for some examples by a factor of 2.355). In addition, the description styles of fault-tolerant FSMs neutralize design errors when transitions are described in each state but not for all possible values of input variables. **Conclusions**. In this paper, the problem of designing fault-tolerant FSMs when the values of bits in the state register or in the input vector of the FSM change because of the negative external impact is described. Different ways of solving the problem at the level of FSM description in HDL are considered. Two description styles for fault-tolerant FSMs are proposed: safe0 and safe1. The fault tolerance of FSMs is provided in the following manner. When the input vector is not defined in the FSM specification for a specific state, the FSM will remain in the initial transition state, i.e. the FSM will not transit to another state. If the code of the illegal state is set in the state register, the FSM will transition to the start state. For all these faults, the safe0 style provides a zero output vector at the FSM output, whereas the safe1 style preserves the value of the previous output vector. A promising direction for future research seems to be the development of new styles and methods of FSM description, aimed at improving the FSM parameters (an area, a performance and a power consumption), as well as improving the reliability and fault tolerance of FSMs.*

Keywords: *finite state machine (FSM); fault tolerance; hardware description language (HDL); Verilog; field programmable gate array (FPGA); unmanned aerial vehicle (UAV).*

1. Introduction

Fault detection and neutralization is an important task in creating fault-tolerant control devices in space technology, avionics, life-support medical equipment, nuclear reactors, banking systems, and telecommunication servers. Finite state machines (FSMs) are also widely used as control devices in unmanned aerial vehicles (UAVs) or drones. Currently, UAVs are used in many areas of human activity, including military conflicts. One of the ways to combat UAVs is to influence the UAV with an electromagnetic pulse (EMP) or a laser beam, which causes numerous failures in the electrical circuits of the control device.

To build a fault-tolerant FSM, it is important to specify it correctly. The FSM is defined correctly when in each state all possible transitions are defined (i.e. the logical OR of transition conditions from each state are equal to a logical unit), at least one transition from other states leads to each state, each transition condition from some state does not intersect with other transition conditions from this state. When these rules are executed in the FSM specification, invalid transitions between states and transitions to illegal states should not occur. However, often for various reasons the above rules for correct description of FSMs are not enforced. When designing complex FSMs, the last of the above rules is most often violated. For example, from 49 MCNC benchmark examples of FSMs [1], this condition is violated in 22 examples.

The source of FSM failures (for example, because of negative external impacts) can be a fault in the state register. If one or more bits are changed in the state register, the FSM may transit to an illegal state, the transition from which has not been defined in the FSM specification. It is also possible that, because of a negative impact, the code of the legal state will be set in the state register. This corresponds to an invalid transition between legal states. In both cases, normal operation of the FSM will be disrupted.

Let the FSM have M states encoded by a code of length R bits, $R \geq]\log_2 M[$, $]A[$ is the smallest integer greater than or equal to A. The number M_I of illegal states whose codes can be set in the state register is determined by Eq. (1).

$$M_I = 2^R - M. \quad (1)$$

In case of failure in the state register there is a high probability of FSM transition to an illegal state. The problem becomes especially acute when using the one-hot code, which is popular when an FSM is implemented in a field programmable gate array (FPGA).

FSM failures can also be caused by errors in the specification of FSM when transitions from each state are not defined for all possible combinations of input variables. Usually, the FSM specification defines the transition conditions to certain states for specific values of input vectors, while the behavior of the FSM for other values of input variables remains undefined. In addition, for some states it is possible that transition conditions may intersect for transitions to different states, i.e. non-deterministic behavior of the FSM is possible.

2. Related works

2.1. UAV warfare

To combat UAVs, a wide variety of methods are used, which can be divided into two large classes: destructive (machine guns, missiles, ramming by other drones) and non-destructive (all others). The main non-destructive methods of UAV warfare are jamming signals to and from the UAV, noise suppression, powerful EMPs and laser beams.

The problems of UAV warfare by non-destructive methods such as jamming uplink, jamming downlink, GPS jamming, GPS spoofing, and deauthentication packets are considered in [2]. Jamming downlink jamming is also considered in [3]. A swarm of UAVs to track malicious UAVs is used in [4]. The classification of cyber attacks on UAVs is given in [5]. In [6], the issue of jamming UAVs with a concentration of signal power toward the UAV is studied. The UAV countermeasure technology based on partial-band noise jamming is presented in [7]. In [8], the problem of influencing a UAV with a powerful EMP is considered. The use of noise and EMR for jamming UAVs has also been studied [9]. In [10], a software-based Wi-Fi jammer for UAV warfare is proposed. The UAV-controlled interceptor using the parallel approach guidance method is considered in [11]. In [12], self-jamming caused by unintentional electromagnetic noise from multiple electronic devices installed in a UAV is studied.

In this study, EMPs and laser beams are considered as external influences on UAV control devices, which cause failures in the input vector and in the state register of the FSM.

2.2. Fault-tolerant FSMs

FSMs play an important role in digital circuit design because they store the system status and control system functionality. Hence, if errors occur in the FSM, they cause serious problems for the system. Therefore, modern digital systems use fault-tolerant FSMs as control devices.

Failures of FSMs can be caused:

– by radiation or cosmic rays (affecting hardware in nuclear power plants or spacecraft);

– by electromagnetic pulses (EMBs) or laser beams (used in UAV warfare);

– by fault injection attacks (FIA) (used by attackers in cryptographic applications);

– by laser-based fault injection (LFI), when an attacker changes the values of individual flip-flops of the FSM state register, etc.

The problem of designing fault-tolerant FSMs can be solved at different levels:

– at the level of gates (transistors) [13];

– at the register-transfer level (RTL) [14];

– at the logical level by applying special synthesis methods [15, 16], state encoding (state assignment) [17 - 23], and using embedded memory blocks of FPGAs [24, 25];

– at the structural level using special structural models of FSMs [26, 27].

Often, the logical level is linked to the structural level [28]. The above problem can also be solved at the system level [29].

In [13], the structure of a flip-flop for the FSM state register is proposed, which protects the FSM from FIA. In [14], solutions at the RTL level are proposed to ensure the reliability of the FSM in the case of single event upsets (SEUs).

In [15], the FSM state encoding algorithm is considered which, in addition to providing fault tolerance, allows for area and power optimization. In [16], a method to insert hidden state transitions (HSTs) and logic cone modifications into a netlist to enhance the security of the FSM from FIA is presented.

In [17], convolutional codes are used to detect and correct errors in the FSM state register. In [18], secure FSM architectures are proposed based on the idea of randomly selecting one code from the set of codes for each encoding and decoding operation. In [19], SEC-DED (single error correction and double error detection) code is used, and in [20], Hamming 3 code is used to detect and correct SEUs when the FSM is implemented in an

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

FPGA. In [21], fault-tolerant techniques like Triple Modular Redundancy (TMR), Hamming-3 encoding, and safe FSM synthesis are analyzed, which are implemented in live designs targeted at Nuclear Power Plants. In [22], the technique for designing FSMs considering the security requirements for FIAs is discussed. This technique includes secure state encoding and the building of a secure FSM architecture to eliminate vulnerabilities from setup-time violation-based fault attacks. In [23], a transition-based encoding CAD framework (TAMED) is presented that provides FSM robustness to multiple LFI models.

In [24] and in [25], embedded memory blocks of the FPGA are used to design fault-tolerant FSMs in the case of SEUs.

In [26] and in [27], structural models for Moore FSMs and Mealy FSMs are presented, which allow detection and neutralization of faults caused by EMPs and laser beams. The proposed structural models allow to detect and to neutralize the following faults: invalid input and output vectors for the entire FSM and in each state; invalid present and next state code; and invalid transitions between states.

In [28], a method for designing FSMs by selectively applying the fault-tolerant circuit according to the importance of the state is discussed.

In [29], a formal analysis based on symbolic algebra is used to find the FSM vulnerabilities; this allows detecting unauthorized access to the states of the design.

However, all the approaches considered, as a rule, require significant hardware overhead and often lead to a decrease in the performance of the FSMs. Note that researchers pay little attention to methods for optimizing parameters and improving the properties of FSMs, which can be achieved by ways (styles) of describing FSMs in a hardware description language (HDL), as well as by methods implemented in design tools.

3. Aim and tasks of the work

Let us devise the design of a fault-tolerant FSM in the following manner.

Task. Create a FSM that corresponds to the given specification and remains operational when an invalid input vector (which is not defined in the FSM specification) arrives at the FSM input and when the code of an illegal state is set in the status register because of a failure.

Solving this problem is especially important when developing control devices for UAVs. The fact is that the UAV can be affected by EMPs or laser beams, called external negative impacts, which can change the values of bits in the input vector or in the state register of FSMs.

Thus, this study aims to develop description styles for fault-tolerant FSMs in HDLs that prevent failures in the state register and the input vector of the FSM.

To achieve the aim of the research, the following tasks have been developed.

1. To develop ways of describing transitions of the FSM from illegal states in case of failure in the state register.

2. To develop ways of describing the transitions of the FSM from each state in case of failure in the input vector;

3. In the case of the above failures, to develop ways of describing the outputs of the FSM.

4. Develop description styles for building fault-tolerant FSMs.

5. To investigate the effectiveness of the proposed styles of FSM description on the FSM benchmark examples.

4. Design of fault-tolerant FSMs using the description in HDL

The Verilog language has been chosen as the HDL-language, but all the approaches discussed in this paper can also be used in the SystemVerilog and VHDL languages.

4.1. The demonstration example

As an example of an FSM, consider the Mealy FSM, whose state transition graph (STG) is shown in Fig. 1. Note that the methods discussed in this paper can also be used to design fault-tolerant Moore FSMs.

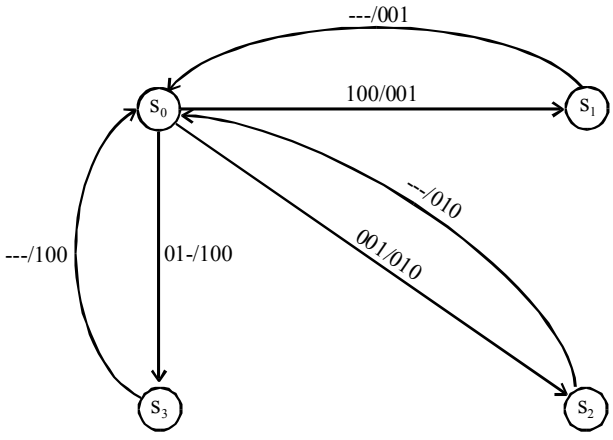


Fig. 1. State transition graph of Mealy FSM

Our FSM has 3 inputs, 3 outputs and 4 states. The vertices of the STG correspond to the states s_0,...,s_3, and the edges of the STG correspond to the transitions of the FSM. The input vector that initiates this transition is written near each edge of the STG, and the output vector that produced during this transition is written with a slash ("/"). Here, the hyphen ("-") can take any bit value: 0 or 1.

Transitions from state s_0 are defined for vectors 100, 001, and 01-, where the vector 01- corresponds to the two vectors 010 and 011. Note that transitions from state s_0 are not defined for vectors 000, 101, 110, and 111. The transitions from states s1, s2 and s3 to state s0 are unconditional, i.e., they are performed for all possible input vectors.

The traditional description of our FSM in the Verilog HDL is as follows:

```
module FSM_Mealy (
        input clk, reset,   // sync and reset signals
        input [2:0] x,              // inputs
        output reg [2:0] y);        // outputs
        reg[2:0] state, next;       // state variables
        localparam [2:0]            // state declaration
        s0 = 0, s1 = 1, s2 = 2, s3 = 3;
        // state register description
        always @(posedge clk, negedge reset)
                if (~reset)         state <= s0;
                else                state <= next;
        always @(*)     // transition description
                case(state)
                        s0: casex(x)
                                3'b100:  next = s1;
                                3'b001:  next = s2;
                                3'b01?:  next = s3;
                            endcase
                        s1: next = s0;
                        s2: next = s0;
                        s3: next = s0;
                 endcase
        always @(*)     // output description
                case(state)
                        s0: casex(x)
                                3'b100:   y = 3'b001;
                                3'b001:   y = 3'b010;
                                3'b01?:   y = 3'b100;
                            endcase
                        s1: y = 3'b001;
                        s2: y = 3'b010;
                        s3: y = 3'b100;
                endcase
endmodule
```

Here we use the style of FSM describing three processes [30], where the first process describes the state register, the second process describes the transitions, and the third process describes the outputs of the FSM. Note that the logic of transitions and outputs of the FSM is described using two levels of **case** statements, where the first level defines the behavior of the FSM depending on the present state (*state*), and the second level defines the behavior of the FSM in each state depending on the input vector (*x*).

When realizing the traditional description of the FSM, the synthesis tools (in our case Quartus system) output warning messages and implement the FSM state register not by flip-flops but by latches. In addition, the latches will be installed on the FSM outputs. Therefore, the number of FPGA logic elements used (an area or an implementation cost) will be quite large. FPGAs do not have latches. The latches in FPGA are implemented using flip-flops and additional logic.

Although the behavior of the FSM will follow the defined specification, the FSM will not operate in illegal states because the traditional description of the FSM does not specify the behavior of the FSM in illegal states.

4.2. FSM transition from illegal states

When a failure occurs in the state register, the FSM can transits to an illegal state and (if the behavior of the FSM in the illegal state is not defined) the FSM fails, i.e. the FSM stops working. Several solutions are possible when the FSM transits to one of the illegal states (Fig. 2):

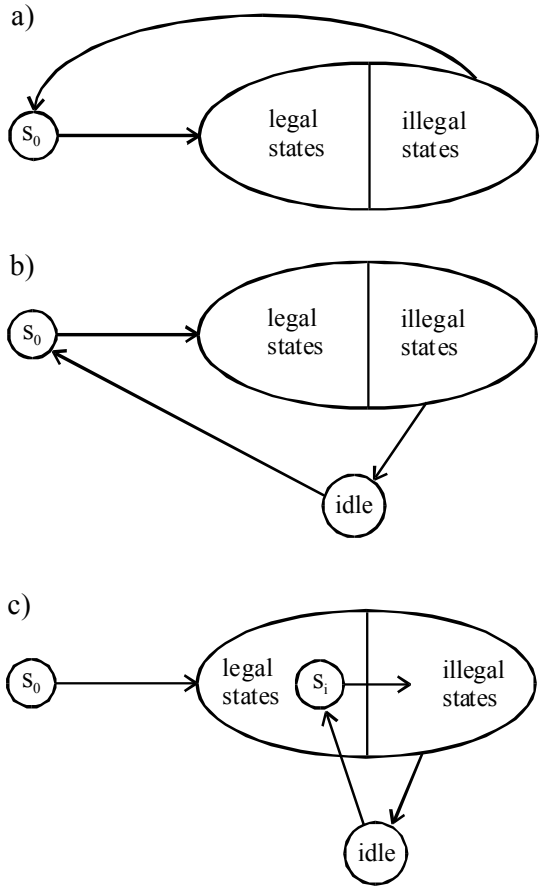


Fig. 2. Variants of FSM transitions from illegal states: a – return to the start state; b – transition to the state idle with return to the start state; c – transition to the state idle with return to the state s_i, from which the transition to the illegal state occurred

a) the FSM transits to the start state s_0;

b) the FSM transits to the additional state *idle*, in which an error flag can be generated or certain actions

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

can be performed to resume the FSM operation, after which the FSM transits to the start state s_0;

c) same as b), but the FSM returns to the state s_i from which the transition to the illegal state occurred.

The state in which the transition begins is called the *initial transition state*, and the state in which the transition ends is called the *final transition state*. In Verilog, it is easier to describe variant a). For this purpose, it is sufficient to add the construction default to the first level case statements with the FSM transition to the start state s0, for example:

```
always @(*)              // transition description
case (state)
    s0: …
    …
    s3: …
    default: next = s0; // transitions from
                        // illegal states
endcase
```

To realize the variant b), the idle state is added to the FSM description. Using the **default** construction in the first-level **case** statement, transitions to the idle state are defined; this corresponds to the FSM transition from the illegal states. In addition, the transition from the *idle* state to the start state is added, for example:

```
localparam [2:0]          // state declaration
    s0 = 0, s1 = 1, s2 = 2, s3 = 3, idle = 4;
…
always @(*)               // transition description
    case (state)
        s0: …
        …
        s3: …
        idle: next = s0;    // from idle to start state
        default: next = idle; // from illegal states
    endcase
```

When implementing variant c), the intermediate variable sr is declared, which stores the code of the state s_i from which the FSM transitioned to the illegal state. For variant c), the description of transitions in our example has the following view:

```
reg [2:0] sr;             // intermediate variable
…
always @(*)               // transition description
    case(state)
        s0: begin sr = s0; next =… end
        s1: begin sr = s1; next =… end
        s2: begin sr = s2; next =… end
        s3: begin sr = s3; next =… end
        idle: next = sr; // to the initial transition state
        default: next = idle; // from illegal states
    endcase
```

Here, the following statement is added to the description of transitions from each state: "sr = si; …", where si is the initial transition state. In case of an erroneous change in the present state code, the sr variable stores the code of the last legal state from which the transition to the illegal state occurred.

When describing outputs, in the state *idle* and in the construct **default**, can optionally be set to the flag illegal_state that indicates the illegal state, for example:

```
output reg illegal_state,   // flag declaration
…
always @(*)               // output description
begin
    illegal_state = 1'b0;
    case (state)
        s0: …
                …
        s3: …
        idle: illegal_state = 1'b1;  // for idle state
        default: illegal_state = 1'b1; // for all illegal
                                       // states
    endcase
end
```

Instead of the flag illegal_state in the state *idle* as well as in the illegal states, it is possible to form a certain value of the output vector (e.g. zero value), which indicates that the FSM is in the illegal state. For example:

```
always @(*)               // output description
    case (state)
        s0: …
        …
        s3: …
        idle: y = 3'b000;  // for idle state
        default: y = 3'b000; // for all illegal states
    endcase
```

4.3. Definition of transitions from each state

In the proposed styles of describing fault-tolerant FSMs, transitions from each state are described using second-level **case** statements. Note that **if-else-if** chains can be used for the same purpose because **case** and **if** statements are interchangeable in this case.

The input vectors of the FSM may contain do not care values, which are denoted in the Verilog description by a question mark ("?"). Because constant elements of **case** statements may contain don't care values, here the **casex** statement is used instead of the **case** statement.

Constant elements of the **case** statement define the transition conditions of the FSM to the legal states. The **default** construct defines the transition of the FSM in the case of mismatch of any constant element with the input vector *x*, i.e., it defines the final transition state for transition conditions that are not defined in the specification

of the FSM. When describing fault-tolerant FSMs, the initial transition state is taken as the final transition state in the **default** construction. For example, in our example, the description of transitions from state s0 has the following view:

```
s0: casex(x)
      3'b100:   next = s1;
      3'b001:   next = s2;
      3'b01?:   next = s3;
      default:  next = s0;
endcase
```

The described behavior of the FSM is shown in Fig. 3.

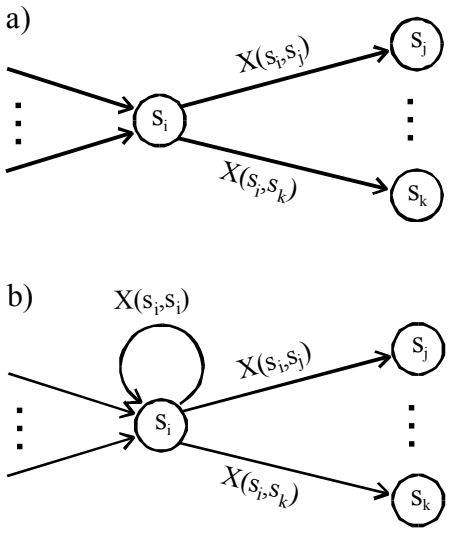


Fig. 3. Transitions of an FSM from state s_i:
a – in the case of traditional description; b – in the case of the description of a fault-tolerant FSM

The transitions from each state s_i of the fault-tolerant FSM are shown in Fig. 3,b, where $X(s_i,s_j)$ is the transition condition (input vector) that initiates the transition from state s_i to state s_j, s_j, $s_i \in S$, S is the set of the FSM states. The transition condition defined in the **default** construction corresponds to expression (2):

$$X(s_i, s_i) = X(s) \backslash \{X(s_i, s_j), \dots, X(s_i, s_k)\}, \quad (2)$$

where X(s) is the set of all input vectors of the FSM.

4.4. Formation of the output vectors of fault-tolerant FSMs

Let $X(s_i)$ be the set of input vectors that initiate transitions from state s_i, $X(s_i) \subset X(s)$. **Question**: what output vector (set of values of output variables) should be formed at the FSM output when any of the vectors of the set $X(s_i)$ do not arrive at the FSM input? These variants are possible:

1) zero output vectors;
2) output vector of values do not care (x…x);
3) output remains the value of the previous output vector;
4) for each state, the value of the output vector is determined by the developer.

In the case of variant 1, there are no active control signals acting on the controlled object, which is not always permissible. The zero output vectors indicates that an invalid input vector arrives at the FSM input. However, if the zero output vector is valid for the FSM, i.e., it is formed in some state (or at some transition) of the FSM, it may indicate a false failure.

The difficulties of using variant 2 lie in the fact that often synthesis tools automatically redefine the values of outputs to optimize the FSM circuit. As a result, do not care values at the output of the FSM will not be formed. The use of variant 2 is allowed only for modeling the behavior of the FSM.

Variant 3 can be used when the sequence of clock cycles of the FSM allows repetition of output vectors; if not, variant 4 should be used.

In the case of variant 4, for transitions from each state, the value of the output vector is determined by the designer and does not result in the negative consequences specified for variants 1 through 3.

The same reasoning holds for the generated values of output vectors in illegal states. For example, in our example, when using variant 1, the description of outputs has the following view:

```
always @(*)                  // output description
    case(state)
        s0: casex(x)
            3'b100:   y = 3'b001;
            3'b001:   y = 3'b010;
            3'b01?:   y = 3'b100;
            default:  y = 3'b000;   // zero output vector
                                    // from the state s0
        endcase
        s1: y = 3'b001;
        s2: y = 3'b010;
        s3: y = 3'b100;
        default:  y = 3'b000;  // zero output vector
                               // from the illegal states
    endcase
```

4.5. Description styles of fault-tolerant FSMs

Two styles are proposed for describing fault-tolerant FSMs in Verilog: safe0 and safe1. In both styles, when describing transitions from each state, the initial transition state is defined in the **default** construct (Fig. 3,b). When the FSM transits to the illegal state, using the variant when the FSM returns to the start state (Fig. 2,a), for example:

```
always @(*)                  // transition description
    case(state)
```

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

```
si: casex(x)
      …
      default: next = si; // return to the initial
                          // transition state
   endcase
…
   default: next = s0; // return to the start state
endcase
```

The styles safe0 and safe1 differ in the way they form the values of output signals. In the safe0 style, the zero output vector is formed in the **default** constructs, which corresponds to variant 1. In the safe1 style, the description of FSM outputs coincides with the traditional description. In this case, when not all possible values of input variables are specified in **the case** statements, the synthesis tool will set latches on the FSM outputs. As a result, the FSM will maintain the value of the previous output vector at the output, which corresponds to variant 3.

Figure 4 shows the functional modeling results of the FSM from our example for the safe0 and safe1 description styles.

From Fig. 4 shows that for the safe0 style, in case of invalid input vectors arriving at the input of the FSM, zero vectors are formed at the FSM outputs, while for the safe1 style, the FSM outputs retain the same values of the outputs.

Note that based on the considered methods for describing fault-tolerant FSMs, other styles of describing fault-tolerant FSMs can be constructed that are better suited for the design of a specific FSM.

5. Experimental Results

The effectiveness of the proposed safe0 and safe1 styles for describing fault-tolerant FSMs has been tested on the FSM benchmarks of the MCNC center [1]. The synthesis was performed using Quartus version 23.1 for the Cyclone 10 LP FPGA family.

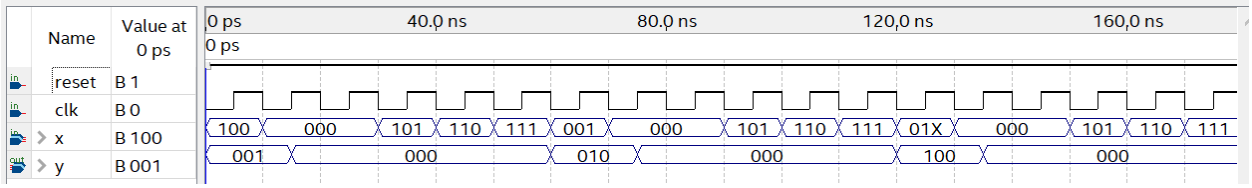
Because the FSM parameters (area and performance) depend heavily on state encoding, all state encoding methods provided by the Quartus system (One-Hot, Gray, Johnson, Minimal Bits and Sequential) were applied for each example. Then, the best results were selected from the obtained results: the minimum area and the maximum performance.

The use of safe0 and safe1 description styles changed the parameters of the FSM for the 20 benchmarks. The experimental results for these examples are shown in Table 1, where i, o, p, and s are the number of inputs, outputs, transitions, and states of the FSM, respectively; L_O, L_{S0}, and L_{S1} are the number of FPGA logic elements used to implement the FSM (i.e. area or cost of implementation) in the case of the traditional description, using the style safe0 and using the style safe1; F_O, F_{S0} and F_{S1} – the same, but with respect to the FSM performance, which is measured in megahertz; L_O/L_{S0}, L_O/L_{S1}, F_{S0}/F_O, and F_{S1}/F_O – relations of the corresponding parameters; Av and Max – arithmetic mean and maximum value of the parameters.

Table 1 shows that using the safe0 style reduces the area for 17 examples. On average, the area is reduced by a factor of 2.148, and the maximum area reduction by a factor of 4.8 is observed for example lion9. Similarly, using safe1 style reduces the area for 19 examples. On average, the area is reduced by a factor of 1.774, and the maximum area reduction by a factor of 4.8 is also observed for the lion9 example.

Using the safe0 style increases the performance for six examples. On average, the performance increases by a factor of 1.192, and the maximum performance increase by a factor of 2.355 is observed for the pma example. Similarly, using safe1 style increases the performance for 9 examples, on average the performance increases by a factor of 1.193, the maximum performance increase by a factor of 2.355 is also observed for the pma example.

a)



b)

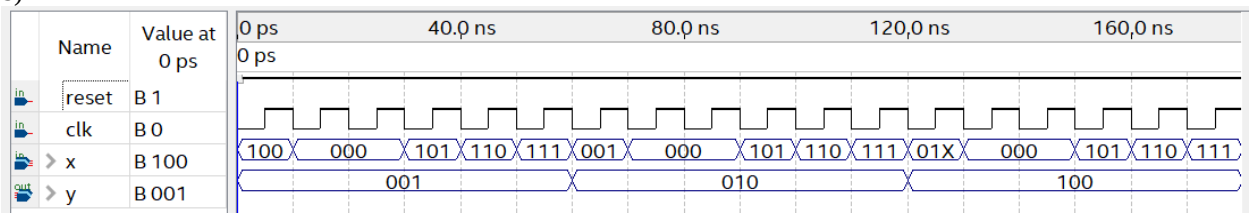


Fig. 4. Results of functional modeling of the Mealy FSM:
a – with description in safe0 style; b – with description in safe1 style

Table 1

Experimental results of description styles of fault-tolerant FSMs (Safe0 and Safe1) compared with the traditional description (Original)

FSM	i	o	p	s	Original		Safe0		Safe1		L_O/L_{S0}	L_O/L_{S1}	F_{S0}/F_O	F_{S1}/F_O
					L_O	F_O	L_{S0}	F_{S0}	L_{S1}	F_{S1}				
bbsse	7	7	56	16	42	253	42	253	39	284	1	1.077	1	1.123
beecount	3	4	28	7	32	0	14	364	15	383	2.286	2.133	1	1
cse	7	7	91	16	100	0	70	268	82	288	1.429	1.220	1	1
ex1	9	19	233	18	119	0	78	247	111	211	1.526	1.072	1	1
ex2	2	2	72	18	60	166	35	254	35	273	1.714	1.714	1.530	1.645
ex3	2	2	36	10	34	0	20	349	23	300	1.7	1.478	1	1
ex4	6	9	21	14	44	0	26	312	33	374	1.692	1.333	1	1
ex5	2	2	32	9	32	298	14	332	16	314	2.286	2	1.114	1.054
ex6	5	8	34	8	57	0	46	256	49	272	1.239	1.163	1	1
ex7	2	2	36	10	32	0	8	346	9	397	4	3.555	1	1
keyb	7	2	170	19	65	217	65	217	65	218	1	1	1	1.005
lion	2	1	11	4	13	0	3	1287	7	364	4.333	1.857	1	1
lion9	2	1	25	9	48	0	10	608	10	608	**4.8**	**4.8**	1	1
pma	8	8	73	24	118	141	93	332	93	332	1.269	1.269	**2.355**	**2.355**
sand	11	9	184	32	208	121	153	241	175	205	1.359	1.189	1.992	1.694
sse	7	7	56	16	42	253	42	253	39	284	1	1.077	1	1.123
styr	9	10	166	30	183	200	130	233	147	237	1.408	1.245	1.165	1.185
tma	7	6	44	20	128	160	59	269	59	269	2.169	2.169	1.681	1.681
train11	2	1	25	11	41	0	17	656	18	380	2.412	2.278	1	1
train4	2	1	14	4	13	0	3	1285	7	414	4.333	1.857	1	1
Av	5.9	6.5	139.6	22.4	69.7	301.3	60.2	438.3	62.5	394.9	**2.148**	**1.774**	**1.192**	**1.193**
Max	19	19	1569	218	387	1269	387	1287	387	1269	**4.8**	**4.8**	**2.355**	**2.355**

6. Discussions

In this paper, the following tasks have been solved to design fault-tolerant FSMs.

1. In case of a failure in the state register, three ways for the FSM transit from the illegal state are proposed: (a) return to the start state; (b) transition to the idle state with return to the start state; (c) transition to the idle state with return to the state s_i, from which the transition to the illegal state occurred (see Fig. 2). For each way, the templates for description in HDL are presented.

2. In case of a failure in the input vector, a method is proposed for describing the FSM transitions from each state (see Fig. 3), while the FSM remains in the initial transition state, the conditions of transitions to which are determined by expression (2).

3. In case of failure in the state register or in the input vector of the FSM, four variants are proposed to determine the values of the output vector: (1) zero vector, (2) vector of don't care values, (3) value of the previous output vector, (4) the output vector specified by the developer. This paper analyzes the advantages and disadvantages of each variant (subsection 4.4).

4. On the basis of the considered descriptions of transitions and outputs of FSMs, two styles of description of fault-tolerant FSMs have been proposed: safe0 and safe1 (subsection 4.5). In both styles, when describing transitions from each state, the construction **default** of the second-level **case** statement defines the initial transition state (see Fig. 3,b). If the FSM transits to an illegal state for both styles, the variant is selected when the FSM returns to the start state (see Fig. 2,a). The styles safe0 and safe1 differ in the way they form the values of output signals. In the safe0 style, the zero output vector is formed in the **default** constructs, which corresponds to variant 1. In the safe1 style, the FSM will save the value of the previous output vector at the output, which corresponds to variant 3.

5. The effectiveness of the proposed styles for describing fault-tolerant FSMs has been studied using FSM benchmarks of the MCNC center in two parameters: area and performance when implementing FSMs in FPGAs (see Table 1).

The experimental results have shown that using the style safe0, compared to the traditional HDL description of FSMs, reduces the area on average by a factor of 2.148

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

(for some examples by a factor of 4.8) and increases the performance on average by a factor of 1.192 (for some examples by a factor of 2.355).

Similarly, using the style safe1, compared to the traditional HDL description of FSMs, reduces the area on average by a factor of 1.774 (for some examples by a factor of 4.8) and increases the performance on average by a factor of 1.193 (for some examples by a factor of 2.355).

Following features provide the advantages of the proposed safe0 and safe1 styles for describing fault-tolerant FSMs:

– in the case of a failure in the input vector or when the transition from some state is not defined in the FSM specification, the next state (the initial transition state) is defined explicitly using the construction **default** of the second-level case statement;

– in the case of a failure in the state register, the FSM transition to the start state is defined explicitly using the construction **default** of the first-level case operator;

– in case of all the above failures, the FSM outputs are defined using the construction **default** of the case statement in safe0 style as a zero vector, and in safe1 style as the value of the previous output vector.

The advantage of this study over the known ones is that the problem of designing fault-tolerant FSMs is solved at the level of FSM description in HDL. This made it possible not only to implement fault-tolerant FSMs but also to reduce the area and increase the performance of FSMs.

Note that known methods for designing fault-tolerant FSMs usually require significant area overhead, which decreases the performance of the original FSM.

EPMs and laser beams that impact the UAV cause failures in the input vector and state register of FSMs that act as control devices for the UAV. Therefore, the proposed description styles for fault-tolerant FSMs are primarily designed for UAV control systems.

Thus, the considered safe0 and safe1 styles of describing fault-tolerant FSMs not only allow improving the fault tolerance but also contribute to the reduction of the area and increase the performance of FSMs, so they can be recommended for practical use.

7. Conclusions

In this paper, the problem of designing fault-tolerant FSMs when the values of bits in the state register or in the input vector of the FSM change because of the negative external impact is described. Different ways of solving the problem at the level of FSM description in HDL are considered. Two styles of describing fault-tolerant FSMs have been proposed, which allow the detection of faults in the state register and in the input vector of the FSM. This prevents transitions of the FSM to illegal states and invalid transitions to legal states.

The fault tolerance of FSM functioning described using the safe0 and safe1 styles is provided as follows. When the input vector is not defined in the FSM specification for a specific state, the FSM will remain in the initial transition state, i.e. the FSM will not transit to another state. If an illegal state code is set in the state register, the FSM will transition to the start state. For all these faults, the safe0 style provides a zero output vector at the FSM output, whereas the safe1 style preserves the value of the previous output vector.

The proposed styles of description of fault-tolerant FSMs do not degrade the parameters of FSMs, which are described by the traditional style, but in some cases allow the reduction of the area (for some examples by a factor of 4.8) and increase of the performance (for some examples by a factor of 2.355). In addition, the description styles of fault-tolerant FSMs allow us to neutralize design errors when transitions in each state are not described for all possible values of input variables.

In addition, the proposed description styles for fault-tolerant FSMs allow neutralizing design errors when transitions in each state are not described for all possible values of input variables.

Thus, **the main contributions** are as follows**:**

- in case of a failure in the FSM state register, three methods have been proposed for the FSM transit from the illegal state, which can be implemented using the HDL;

- the method proposed to describe in HDL the FSM behavior in each state in the case of an invalid input vector or an input vector that is not defined in the FSM specification;

- four variants have been proposed for describing in HDL the FSM output vectors in the case of an invalid input vector or an input vector that is not defined in the FSM specification;

- based on these methods of FSM description, two styles for describing fault-tolerant FSMs are proposed.

The effectiveness of the proposed description styles of fault-tolerant FSMs in terms of area and performance has been investigated on the FSM benchmarks of MCNC center.

Future research development. A promising direction for future research seems to be the development of new styles and methods of FSM description, aimed at improving the FSM parameters (an area, a performance and a power consumption), as well as improving the reliability and fault tolerance of FSMs.

Acknowledgment. The present study was supported by a grant WZ/WI-III/5/2023 from Bialystok University of Technology and was founded from the resources for research by the Ministry of Science and Higher Education.

References

1. Yang, S. Logic Synthesis and optimization benchmarks user guide. Version 3.0. Microelectronics Center of North Caro-lina (MCNC), 1991. 45 p. DOI: 4a86519e41bb8dbaa8d2c9ba434030f48de85ce7.

2. Kratky, M., & Minarik, V. The non-destructive methods of fight against UAVs. *International conference on military technologies (ICMT)*, Brno, Czech Republic, 2017, pp. 690-694. DOI: 10.1109/MILTECHS.2017.7988845.

3. Curpen, R., Bălan, T., Micloş, I. A., & Comănici, I. Assessment of signal jamming efficiency against LTE UAVs. *International Conference on Communications (COMM)*, Bucharest, Romania, 2018, pp. 367-370. DOI: 10.1109/ICComm.2018.8484746.

4. Arnold, C., & Brown, J. Performance evaluation for tracking a malicious UAV using an autonomous UAV swarm. *11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, New York, USA, 2020, pp. 0707-0712. DOI: 10.1109/UEMCON51285.2020.9298062.

5. Kong, P. Y. A survey of cyberattack countermeasures for unmanned aerial vehicles. *IEEE Access*. 2021, no. 9, pp. 148244-148263. DOI: 10.1109/ACCESS.2021.3124996.

6. Jin, W. C., Kim, K., & Choi, J. W. Adaptive jamming considering location information inaccuracy for anti-UAV system. *International Conference on Information Networking (ICOIN)*, Jeju Island, Korea (South), 2021, pp. 480-482. DOI: 10.1109/ICOIN50884.2021.9334027.

7. Lei, Z., Ding, P., Zheng, W., Fei, X., & Fan, H. UAV countermeasure technology based on partial-band noise jamming. *33rd Chinese Control and Decision Conference (CCDC)*, Kunming, China, 2021, pp. 1456-1461. DOI: 10.1109/CCDC52312.2021.9602343

8. Min, S. H., Jung, H., Kwon, O., Sattorov, M., Kim, S., Park, S. H., ... & Park, G. S. Analysis of electromagnetic pulse effects under high-power microwave sources. *IEEE Access*, 2021, no. 9, pp. 136775-136791. DOI: 10.1109/ACCESS.2021.3117395.

9. Šimon, O., Götthans, T., & Popela, M. Commercial UAV jamming possibilities. *32nd International Conference Radioelektronika (RADIOELEKTRONIKA)*, Kosice, Slovakia, 2022, pp. 1-6. DOI: 10.1109/RADIOELEKTRONIKA54537.2022.9764904.

10. Kaushik, K., Negi, R., & Dev, P. An electronic warfare approach for deploying a software-based Wi-Fi jammer. *5th International Conference on Contemporary Computing and Informatics (IC3I)*, Uttar Pradesh, India, 2022, pp. 43-47. DOI: 10.1109/IC3I56241.2022.10073447.

11. Pohasii, S., Korolov, R., Vorobiov, B., Bril, M., Serhiienko, O., & Milevskyi, S. UAVs intercepting possibility substantiation: economic and technical aspects. *IEEE 4th International Conference on Modern Electrical and Energy System (MEES)*, Kremenchuk, Ukraine, 2022, pp. 1-6. DOI: 10.1109/MEES58014.2022.10005710.

12. Watanabe, K., Sakai, R., Tanaka, S., Nagata, M., Osaka, H., Nakamura, A., ... & Gotoh, K. Electromagnetic Interference With the Mobile Communication Devices in Unmanned Aerial Vehicles and Its Countermeasures. *IEEE Access*, 2024, no. 12, pp. 11642-11652. DOI: 10.1109/ACCESS.2024.3351216.

13. Wang, Z., Cui, A., & Qu, G. A low-cost fault injection attack resilient FSM design. *IEEE 33rd International System-on-Chip Conference (SOCC)*, Las Vegas, USA, 2020, pp. 19-24. DOI: 10.1109/SOCC49529.2020.9524779.

14. Cassel, M., & Lima, F. Evaluating one-hot encoding finite state machines for SEU reliability in SRAM-based FPGAs. *IEEE International On-Line Testing Symposium (IOLTS'06)*, Lake Como, Italy, 2006, pp. 6-pp. DOI: 10.1109/IOLTS.2006.32.

15. El-Maleh, A. H. A sequential circuit fault tolerance technique with enhanced area and power. *IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, Abu Dhabi, United Arab Emirates, 2015, pp. 301-304. DOI: 10.1109/ISSPIT.2015.7394348.

16. Juretus, K., & Savidis, I. Synthesis of hidden state transitions for sequential logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020, vol. 40, no. 1, pp. 11-23. DOI: 10.1109/TCAD.2020.2994259.

17. Li, M., Xu, S., Wan, F., Gu, J., Peng, M., & Jiang, J. Non-intrusive design of self-checking FSM based on convolutional codes. *Tsinghua Science & Technology*, 2007, vol. 12, no. S1, pp. 73-77. DOI: 10.1016/S1007-0214(07)70087-1.

18. Wang, Z., & Karpovsky, M. Robust FSMs for cryptographic devices resilient to strong fault injection attacks. *IEEE 16th International On-Line Testing Symposium*. Corfu, Greece, 2010. DOI: 10.1109/IOLTS.2010.5560195.

19. Sooraj, S., Manasy, M., & Bhakthavatchalu, R. Fault tolerant FSM on FPGA using SEC-DED code algorithm. *International Conference on Technological Advancements in Power and Energy (TAP Energy)*, Kollam, India, 2017, pp. 1-6. DOI: 10.1109/TAPENERGY.2017.8397309.

20. Sooraj, S., & Bhakthavatchalu, R. Hamming 3 algorithm for improving the reliability of SRAM based FPGAs. *International Conference on Communication and Signal Processing (ICCSP)*, Chennai, India, 2017, pp. 0938-0942. DOI: 10.1109/ICCSP.2017.8286508.

21. Nidhin, T. S., Bhattacharyya, A., Behera, R. P., Jayanthi, T., & Velusamy, K. Verification of fault tolerant techniques in finite state machines using simulation based fault injection targeted at FPGAs for SEU mitigation. *4th International Conference on Electronics and Communication Systems (ICECS)*, Coimbatore, India, 2017, pp. 153-157. DOI: 10.1109/ECS.2017.8067859.

22. Nahiyan, A., Farahmandi, F., Mishra, P., Forte, D., & Tehranipoor, M. Security-aware FSM design flow

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

for identifying and mitigating vulnerabilities to fault attacks. *IEEE Transactions on Computer-aided design of integrated circuits and systems*, 2018, vol. 38, no. 6, pp. 1003-1016. DOI: 10.1109/TCAD.2018.2834396.

23. Choudhury, M., Gao, M., Tajik, S., & Forte, D. TAMED: transitional approaches for LFI resilient state machine encoding. *IEEE International Test Conference (ITC)*, Anaheim, USA, 2022, pp. 46-55. DOI: 10.1109/ITC50671.2022.00011.

24. Tiwari, A., & Tomko, K. A. Enhanced reliability of finite-state machines in FPGA through efficient fault detection and correction. *IEEE Transactions on Reliability*, 2005, vol. 54, no. 3, pp. 459-467. DOI: 10.1109/TR.2005.853438.

25. Frigerio, L., & Salice, F. RAM-based fault tolerant state machines for FPGAs. *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, Rome, Italy, 2007, pp. 312-320. DOI: 10.1109/DFT.2007.33.

26. Solov'ev V. V. Structural models for failure detection of Moore finite-state machines. Journal of Computer and Systems Sciences International, 2023, vol. 62, no. 6, pp. 977–990. DOI: 10.1134/S1064230723060102.

27. Salauyou V. Structural models of Mealy finite state machines detecting faults in control systems. *Radioelectronic and Computer Systems*, 2023, no. 3, pp. 173-186. DOI: 10.32620/reks.2023.3.14.

28. Choi, S., Park, J., & Yoo, H. Area-efficient fault tolerant design for finite state machines. *International Conference on Electronics, Information, and Communication (ICEIC)*, Barcelona, Spain, 2020, pp. 1-2. DOI: 10.1109/ICEIC49074.2020.9051122.

29. Farahmandi, F., & Mishra, P. FSM anomaly detection using formal analysis. *IEEE International Conference on Computer Design (ICCD)*, Boston, USA, 2017, pp. 313-320. DOI: 10.1109/ICCD.2017.55.

30. Salauyou, V., & Zabrocki, Ł. Coding techniques in Verilog for finite state machine designs in FPGA. *IFIP International Conference on Computer In-formation Systems and Industrial Management (CISIM)*, Belgrade, Serbia, 2019, pp. 493-505. DOI: 10.1007/978-3-030-28957-7_41.

Received 08.01.2024, Accepted 20.02.2024

СТИЛІ ОПИСУ НАДІЙНИХ СКІНЧЕННИХ АВТОМАТІВ ДЛЯ БЕЗПІЛОТНИХ ЛІТАЛЬНИХ АПАРАТІВ

Валерій Соловйов

Об'єктом дослідження є скінченні автомати (ССА), які використовуються як пристрої керування в безпілотних літальних апаратах (БПЛА). Метою роботи є розробка стилів опису робастних (відмовостійких) автоматів на мовах опису апаратури (HDL), які запобігають виникненню збоїв у регістрі станів та вхідному векторі автомата. Задачі, що вирішуються: розробка методів опису переходу автомата з недопустимих станів у випадку збою в регістрі стану, а також переходу автомата з кожного стану у випадку збою у вхідному векторі; визначення вихідних векторів автомата у випадку зазначених вище збоїв; розробка стилів опису надійних (відмовостійких) автоматів; дослідження ефективності запропонованих стилів опису надійних автоматів. Використані методи: теорія скінченних автоматів, методи кодування станів автоматів, методи представлення автоматів, мова опису апаратури Verilog. Отримані наступні результати: розроблено два стилі опису надійних автоматів safe0 та safe1, які не збільшують площу та не зменшують швидкодію автоматів, а в деяких випадках дозволяють зменшити площу (для деяких прикладів у 4.8 рази) та збільшити швидкодію (для деяких прикладів у 2.36 рази). Крім того, стилі опису надійних автоматів нейтралізують помилки проектування, коли переходи описуються в кожному стані не для всіх можливих значень вхідних змінних. Висновки. У статті сформульовано проблему проектування надійних автоматів, коли значення бітів у регістрі стану або у вхідному векторі автомата змінюються внаслідок негативного зовнішнього впливу. Розглянуто різні способи розв'язання проблеми на рівні опису ШПМ мовою HDL. Запропоновано два стилі опису надійних автоматів: safe0 та safe1. Надійність функціонування ШНМ, описаних за допомогою стилів safe0 та safe1, забезпечується наступним чином. Якщо вхідний вектор не визначений у специфікації FSM для певного стану, то FSM залишиться у початковому перехідному стані, тобто FSM не перейде в інший стан. Якщо у регістрі станів задано недопустимий код стану, то FSM перейде у початковий стан. Для всіх цих помилок стиль safe0 забезпечує нульовий вихідний вектор на виході FSM, тоді як стиль safe1 зберігає значення попереднього вихідного вектора. Перспективним напрямком подальших досліджень видається розробка нових стилів та способів опису FSM, спрямованих на покращення параметрів FSM (площі, продуктивності та енергоспоживання), а також на підвищення надійності та відмовостійкості FSM.

Ключові слова: скінченний автомат; надійність; відмовостійкість; мова опису апаратного забезпечення; Verilog; польова програмована логічна матриця; безпілотний літальний апарат (БПЛА).

Соловйов Валерій Васильович – професор, доктор технічних наук, Факультет комп'ютерних наук Білостоцького політехнічного університету, Білосток, Польща

Prof. **Valery Salauyou**, DSc, PhD, Eng, Faculty of Computer Science, Bialystok University of Technology, Bialystok, Poland,
e-mail: v.salauyou@pb.edu.pl, ORCID: 0000-0002-9174-8588, Scopus Author ID: 55699021000