Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 22 (2014) 132 – 137

www.elsevier.com/locate/procedia

3rd International Conference on Through-life Engineering Services

Self-healing fuel pump controller mapped into memory based finite state machine

Philipp Schiefer*, Richard McWilliam, Alan Purvis

Durham University, South Road, Durham, DH1 3LE, United Kingdom

* Corresponding Philipp Schiefer. Tel.: +44 191 334 2418; fax: +44 191 334 2408. *E-mail address:* philipp.schiefer@durham.ac.uk

Abstract

This paper describes our on-going research into the design of finite state machines (FSMs) that exhibit self-healing characteristics. The approach adopted here is based on conversion of the traditionally adopted logic hardware design into generic look-up table (LUT) format. Instead of relying upon bespoke hardware mitigation strategies such as triple modular redundancy, our approach relies upon well-established data error detection and correction (EDC) codes that are ideally suited to protecting LUTs. This 'memory-mapping' of logic brings self-healing capabilities that can be applied to a wide variety of FSMs. We illustrate our method by mapping a generic automotive used fuel pump controller (FPC) design to LUT format. Built-in repair is and fault monitoring are both considered to be extremely important embedded control applications and we therefore discuss significant benefits that can be brought by incorporating self-healing capability to the underlying hardware. We demonstrate the design principles of our approach verify the state-based behavior of the resulting FSM. We further discuss the how content addressable memory (CAM) can be used to achieve efficient address mapping. In order to protect against address errors occurring at the input, a two-stage LUT implementation is used that removes errors occurring in the input data stream as well as protection of the state mapping itself.

Peer-review under responsibility of the Programme Chair of the 3rd InternationalThrough-life Engineering Conference

Keywords: Finite State machine; Memory Look-Up Tables; Fuel Pump Controller; Content Access Memory

Technical glossary

Abbr	Definition
CAM	Content-addressable memory
ECU	Engine control unit
EDC	Error detection and correction
FPC	Fuel pump controller
FPGA	Field-programmable gate array
FPS	Fuel pressure system
FSM	Finite-state machine
LUT	Look-up table
MUX	Multiplexer
PLD	Programmable logic device
SEU	Single upset event
STB	State transition block

1. Introduction

In our world today every fossil fuel burning engine in different applications requires a type of fuel storage and a means of getting this fuel to the engine at the right pressure and amount required. This is done in all the cases with the help of an electric pump which can be controlled through a simple relay or a fuel pump controller (FPC) based on a PLD. For controlling the pressure for the simple relay solution a mechanical pressure stabilizing system is part of this fuel supply system. Mechanical system can cause problems in unstable pressure , which is why most of fossil fuel driven engines are equipped with pressure controlled FPC. FPCs are based on PLDs programed in standard programming logic

Peer-review under responsibility of the Programme Chair of the 3rd InternationalThrough-life Engineering Conference
doi:10.1016/j.procir.2014.07.151

according to a given specification that describes the system behavior in regards of performance and total car system interfacing. The core of a PLD system contain in most cases a type of microcontroller based system and in some cases a field programming gate array (FPGA) surrounded by sensor and output interfacing hardware. Today`s FPC are a part of complex engine systems controlled by several electrical control units (ECU) that require inter-system communication for maintaining optimal system behavior for performance in the event of sensor degradation and to handle fault conditions. This research work focuses on the main task of the FPC providing fuel at the right amount and pressure to the engine. The adaptation is achieved by converting and coding the state transition diagram into a memory based system which only uses the memory data to perform the FSM task. For accurate pressure control feedback or feed-forward control can be utilized in the application. What particular controller hardware gets chosen depends on the design and the preference of the designer. The research work described in this paper is based on a P (proportional) controller type which can be implemented by switching on the power to fuel pump at a given voltage (V) level to produce the required minimal fuel pressure to start the engine. The required fuel pressure is achieved by incrementing or decrementing the voltage level in a stepwise fashion. A LUT based FPC simulation is presented in Figure 1 which is based on a MatLab adaptation of our approach. Other than the P type controller, a PI (proportional integer) or PID (proportional integer derivative) type controller behavior can been used.

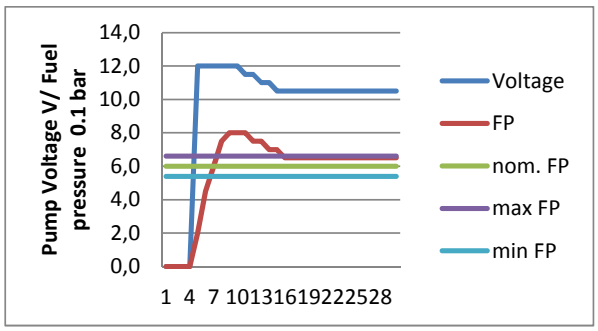


Fig. 1 Simulation data of memory mapped FPC

The FPC is then mapped into a self-protecting FSM in the form of a Moore state machine, wherein the memory is CAM based in order to produce a space-efficient mapping. CAM based memory offers the advantage over linear addressable memory, that is uses pattern matching for data access. Through this feature the memory structure is compactor and memory gaps can be avoided. The task of the self-protecting of the FSM is been done through dynamic input filtering with the goal of limiting the possibility of FSM upsets. The task of self-controlling and healing are part of the goal of maintain a functional LUT based FSM in case of memory data corruption. This three features are seen as self-* qualities for creating a fault tolerant system. For the FSM coding the approach of self-configuring FSM for LUT out of [1] gets used. The complete fuel pressure system (FPS), comprising FPC and fuel pump will be limited in this simulation to the FPC only and simulated. A basic FPS as a block diagram gets presented in Figure 2.

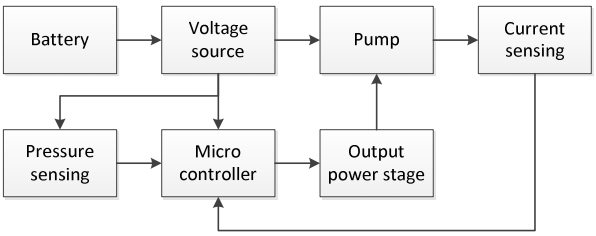


Fig. 2: Basic block diagram of a FPS

2. LUT-based architectures for implementing FSMs

FSMs are used in many applications where a fixed mapping between input and output logic patterns is needed. In its simplest form, the FSM stores a direct one-to-one mapping between input logic data patterns and the corresponding output data patterns. This is normally achieved by storing all of the possible mappings in the form a LUT and using the current input pattern to look-up the relevant output pattern. For simple input/output mappings, the contents of the LUT can be determined directly by analyzing the system's state transition diagram.

More complex input/output mappings are possible, in which case a formal approach can be used. The logic design of an FSM with a given number of inputs I, outputs O and states S can be defined as a 5-tuple (I,O,S,δ,ω). The state transition function is $\delta : I \times S \rightarrow S$ and the output function is $\omega : I \times S \rightarrow O$ [2-4]. An FSM alters its current state following the defined state transition defined by the input specified at this state and is defined as either a Moore or a Mealy state machine [5-7] as illustrated in Figure 3.

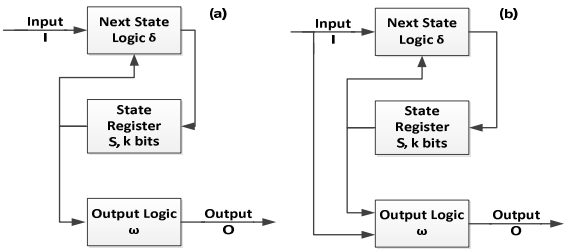


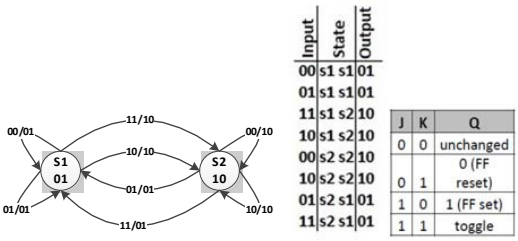
Fig. 3 Moore (a) and Mealy (b) FSMs [7]

The difference between Moore and Mealy is defined by the output response in regards to the input stimulus. In a Moore based FSM the output functionality is controlled by the state only (see Figure 3a) and defined as $\omega : S \rightarrow O$. The output function of a Mealy state machine depends on the state and input and is defined as $\omega : I \times S \rightarrow O$ (see Figure 3b) [2-4]. Between the two designs there is an important dissimilarity: - Mealy require less state transition than a Moore. In this regards a Mealy FSM has more compact state transition

structure. In order to translate a Mealy into a Moore FSM state splitting has to be done and this increases significantly the number of states and state transitions [8].

The creation of a FSM starts with defining the state transition diagram which can then be converted into a state transition table. Figure 4a presents a state diagram of a JK-flip-flop as an example and Figure 4b shows the associated state transition table. Figure 4c shows the trust table and the FSM challenge of creating the toggle function. The table (Figure 4b) contains the state transition in conjunction with the applied inputs (I) per table row, which also includes the required output (O) information. The structure of the state (S) definition in this table is configured in the following way: the state information on the left represents the current state and on the right it represents the state the FSM switches to after associated input sequence has been applied. In the special case of "don`t care" conditions, where O is independent of the previous state, a "-" symbol is used and the FSM is incompletely specified. The logic definition of the FSM changes into a 6-tuple (I,O,S,δ,ω,λ), where λ represents the don`t care conditions [8].

The state table in Figure 4b shows within the input information (in this case both inputs J and K are shown as one value) all possible combination of J and K and the output information of Q and $\overline{Q}$. The trust table of the JK-flip-flop (presented in Figure 4c) shows the logical definition.



Input	State	Output
00	s1 s1	01
01	s1 s1	01
11	s1 s2	10
10	s1 s2	10
00	s2 s2	10
10	s2 s2	10
01	s2 s1	01
11	s2 s1	01

J	K	Q
0	0	unchanged
0	1	0 (FF reset)
1	0	1 (FF set)
1	1	toggle

(a) (b) (c)

Fig. 4 JK-flip-flop FSM state representation

3. Mapping FSM state representation into memory

The structure of the resulting FSM state transition table (See Figure 4b) can be used for transferring it into memory LUT. The basic principal of transferring FSM into a LUT memory structure is to allocate one LUT row per state transition whereby each row is accessed by unique LUT address and data has a fixed information structure [9]. In the case the minimum number of LUT row entries is defined by the number of state transition entries. An increase of the number of entries can occur due to "don't care" entries within the input sequences at one FSM state definition. The LUT can then be controlled by inputs and memory address feedback to create the necessary outputs and state transition. By introducing a binary state coding of the individual states (see Figure 5a) placing them within the table accordingly (see Figure 5b). The memory address is the combination of input (Figure 5b red bits) and current state (Figure 5b green bits) combined in binary format. The data stored at this memory location is the output bits (Figure 5b yellow bits), which in this case also acts as the necessary feedback information to create the current state in the address pointer of the FSM.

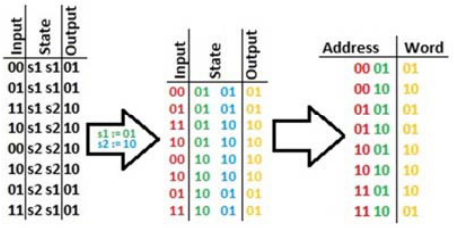


Fig. 5: JK-flip-flop state transition table transformation into memory

By assuming that the number of unique addresses appearing in the resulting table is N_addr. The JK-flip-flop memory mapping example presented in Figure 4b indicates that the addressable memory size is defined by $2^{\mathrm{N\_addr}}$. Hence memory size will grow exponentially by the number of addressable locations, which determined by is the total number of input and state encoded bits[10]. The structuring of the addressed states gets arranged through the input structure which is fixed and which creates gaps throughout the information stored in the LUT. For example the above JK-flip-flop needs $2^4 = 16$ addressable word locations, from which only 9 are used. This is further broken down into 8 addresses associated with JK-flip-flop working conditions and 1 address reserved for reset or start up memory location. Hence 7 locations are unused and cannot be allocated for the state machine functionality. These undesirable memory locations can still be accessed within the FSM in the event of an address fault and hence they require a present condition to prevent misbehavior of the FSM. If this precaution is not taken the FSM may force an undefined state transition including incorrect output information and a non-recoverable state.

4. Self-protecting FSM against invalid input sequences

The block diagram of the functionality for self-protecting a Moore FSM is presented in Figure 6. Protecting the FSM against execution of invalid states, input filtering and blocking is required for this task. Our research is driven by the idea to filter undesired input stimulus in an effective way without creating a massive memory overhead. For this task we altered the FSM design. Two primary blocks of the FSM can be identified labelled as the input block and state transition block (STB). All the data needed required by the state mapped-design is stored in CAM because of the two specific advantages: firstly CAM continues data storage and retrieving of data within a single clock cycle. Secondly, in the event that data matching is unsuccessful the CAM indicates this condition with a flag which can be used as a fault indicator of the system if required.

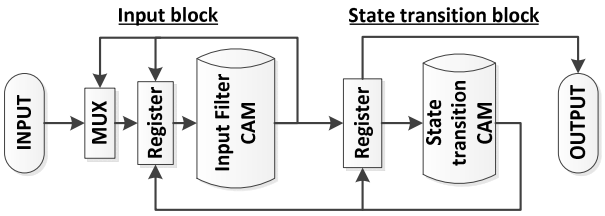


Fig. 6: Block diagram of the self-protecting FSM against input upsets showing input and state transition blocks

Our approach targets the self-protection of the input circuit of the FSM against input sequences which could interfere with the designed FSM state transition flow. Erroneous input sequences can lead to unspecified state transition of the FSM (see Section II, point λ) or unspecified memory entries within the state transition memory and can be caused by faulty sensor or SEU events on the input side of the FSM. For achieving the self-protecting of influence of erroneous input stimulus on the FSM state transition the state dependent input information must to be excluded from the main FSM state transition information stored in the state memory data. This disassociation between the data requires a cascaded memory structure performed through the two blocks of our design (see Figure 6). The information link for change of state transition between input and state transition block is performed with the help of coded data. Coded data used for the FSM state transition prevents unclear specified input state transition sequences and reduces the data structure of the FSM memory data. This research work does not work with unspecified memory entries to eliminate possible undesirable state transition but instead this is been done through content filtering.

The detailed operation of the input block and STB is now described.

a) Input block functionality

The input block presented on the left in Figure 6 filters all the input sequences stimulating accordingly to the present state transition of the FSM and which is supplied to the input block by the STB. The input block is made up out of three individual sub functional blocks, which are an incoming register, MUX and input filter CAM. The register and MUX are getting their control data from word data out of the input filter CAM of the pre-set stage selection done by the STB. The register contains D-flip-flops where the reset inputs are used to suppress the associated input data supplied at their inputs defined by the control data of the input filter CAM. With this all unrequired input information is getting blocked from taking any effect on the behaviour of the FSM. A subsequent MUX is used to reduce the remaining input information into a compact data package without unnecessary data gaps and stored in the register. This MUX is controlled by the same control data from the input filter CAM. Within the register the data from the MUX, input filter CAM data and current state of the FSM is combined for creating a unique content search pattern for the input filter CAM. Accessing the input filter CAM with this data will produce a pre-defined state transition data trigger, which is different from the input data but linked to a specific state transition. This bit of data is transferred into the register of the STB.

b) STB functionality

The STB is a traditional design of a FSM with the difference that a CAM is used instead of a standard type memory. The advantages of using a CAM is the compact data structure which is achieved by using contents for addressing the data. With this a linear addressing of a memory is not required and only the needed data can be stored and makes the addressing of the memory data unique for the application. The FSM can be implemented from the information of state transition table and can be converted into the appropriated memory based FSM data structure. The address register is enabled by the search and match signal of the input block CAM. With this a link between both blocks are established.

5. Adaptation of FPC into memory logic based on state diagram

The Moore style FSM behavior gets used in this research work running on CAM hardware. A block diagram of the proposed FPC structure that we are using for our research is shown in Figure 7. For the CAM FSM block of this diagram the self-protecting FSM described in point 4 will be used. .

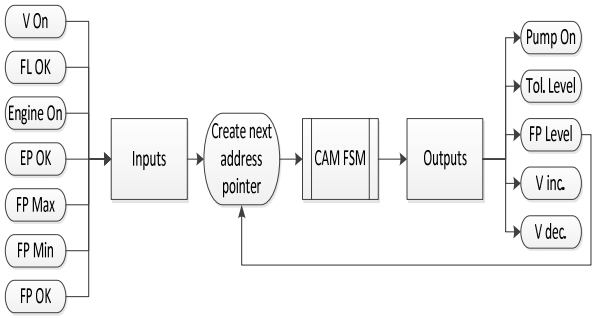


Fig. 7: Block diagram of a FPC including input and output signals

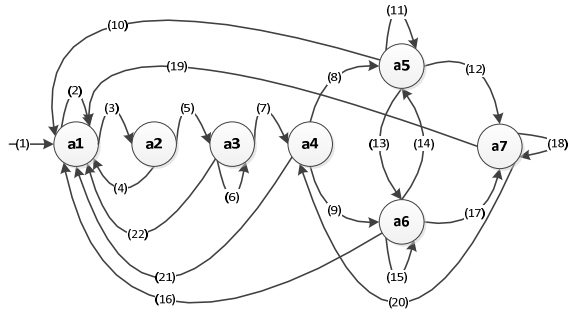
The following input signals are getting used by the memory based FSM (MB-FSM) on the input side:

V on : (VO) voltage on
FL OK : (FLO) fuel level in tank OK
Engine ON : (EO) engine is switched on
EP OK : (EPO) engine fuel pressure
FP Max : (FPA) fuel pressure above specification
FP Min : (FPI) fuel pressure below specification
FP OK : fuel pressure OK

These signals are getting used on the output side of the MB-FSM: Pump On:

Pump on : (PO) switch fuel pump on
Tol Level : (TL) current tolerance band
FP Level : (FPL) fuel pressure value
V inc : (VI) increase fuel pump voltage
V dec : (VD) decrease fuel pump voltage

The state diagram for the simulated FPC can be seen in Figure 8a.



(a)

(1) reset	(9) FPI=1; EPO=0; FPO=0	(17) FPO=1
(2) VO=0	(10) VO=0; EO=0; FLO=0	(18) FPO=1
(3) VO=1	(11) FPA=1; EPO=0; FPO=0	(19) VO=0; EO=0; FLO=0
(4) FLO=0	(12) FPO=1	(20) FPO=0
(5) FLO=1	(13) FPI=1; EPO=0; FPO=0	(21) VO=0; EO=0; FLO=0
(6) EO=0	(14) FPA=1;EPO=0;FPO=0	(22) VO=0; EO=0; FLO=0
(7) EO=1	(15) FPI=1; EPO=0; FPO=0	
(8) FPA=1; EPO=0; FPO=0	(16) VO=0; EO=0; FLO=0	

(b)

(1) -	(9) PO=1; VD=0; VI=1	(17) PO=1; VD=0; VI=0
(2) -	(10) PO=0	(18) PO=1
(3) -	(11) PO=1; VD=1; VI=0	(19) PO=0
(4) -	(12) PO=1; VD=0; VI=0	(20) PO=1; FPL=xx; TL=xx
(5) -	(13) PO=1; VD=0; VI=1	(21) PO=0
(6) -	(14) PO=1; VD=1; VI=0	(22) -
(7) PO=1; FPL=xx; TL=xx	(15) PO=1; VD=0; VI=1	
(8) PO=1; VD=1; VI=0	(16) PO=0; VD=0; VI=0	

(c)

Fig. 8: Design of FPC, (a) State diagram of the simulated FPC of this research paper, (b) state transitions triggered by input signals, (c) output related state transition.

The definition of the different main state transition requirements (STR) are presented in Figure 8b and the main state transition trigger. This state diagram (Figure 8a) is only the broad representation of the FPC FSM state description. The example for the control behaviour of the MB-FSM based FPC gets presented in Figure 1. The nominal FP for this simulation is an estimated and simulated pressure value with a ±10% tolerance band. This simulation has been done to show the basic control performance of the FPC done on a MB-FSM. All external input and outputs where part of the overall simulation and shows the capability of memory based controller.

The memory requirement of a traditional FSM approach can be calculated with $M = 2^{i+s} * (o + s)$ where i is number of inputs, s is number of states and o is number of outputs. This comes to a memory size of 40960 bits. The memory requirement of the CAM based FSM can be calculated with $M = s * (2 * i + o + 2 * c + 3 * s)$ where i is number of inputs, s is number of states, o is number of outputs and c is unique link number between CAM blocks. For the MB-FSM FPC a memory size of 220 bits would be required.

6. Self-healing in case of memory data corruption

Today modern integrated circuits are driven by continues effort of shrinking transistor size which make them more susceptible to bit upsets caused by alpha particle or neutron hits. This effect can be measured in the soft-error rate (SER) and effect is also referred to as the single-event upsets (SEU). Due to their uniform structure and the nature of storing binary information makes them susceptible for SEU effects. In paper [11] the protection of CAM against 1 bit alteration caused by SEU per data word with the help of error-correcting-match scheme got described. This scheme works on the bases of hamming code and added several parity bits for each CAM data word. Comparing the circuit of a standard memory against the CAM reveals that the main difference is in the addressing of the data. Within the standard memory a centralized addressing logic creates the required individual data location. This structure is centralized and clustered away from the individual memory cell. Comparing this to the structure of a CAM the main difference is within the addressing and its logic design. CAM is based on a searching instead of a direct addressing. Each memory cell has its own data matching circuit and it is not centralized [12]. Because of this the hardware overhead compared to a standard memory module is not bigger. But a CAM is going to be more susceptible to SEUs due to the nature of contain searching. If a contain search does not generate a match a close match will be taken and which has be avoided in any case of a FSM application.

The self-healing approach we are proposing in our research work for handling single bit alteration in a CAM data word is based on the use of single parity checking. The principal of this approach works on the duplicate data storage within the CAM distinguished only by a single bit information within the contain search pattern required for finding and addressing of the CAM data word. This single bit or switching bit of the contain search pattern is governed by the parity bit check of the original CAM data word. In the case of a parity fault in the original data word the switch bit gets set and the back-up data gets read out. This is been performed in the way that the address pointer still containing the current state transition pointer which is getting altered that the parity fault bit gets set. Due to the parity fault the execution of state transition gets retained for one clock cycle and the altered pointer gets executed. This task of altering the state pointer instead of correcting the faulty memory data showed the best performance within our simulation and required system logic requirements. Every type of data reconstruction based on error correction approaches requires a certain logic or controller. The goal in our research was to limit the controlling logic to a minimum and utilize memory for the majority of the task performance. In the case of both data words within the CAM are been corrupted the FSM gets halted and an insuperable fault within the system has occur.

7. Summary and Conclusion

Any system which can cope with filtering away undesirable input stimulus and handles data alteration including recovery requires an overhead in processing and hardware if based on a PLD type controller. Every off this tasks will increase the workload on the PLD and reflexes into variable response time on a given input stimulus. Our research shows that with using a LUT based controller offers the following benefits: contain response time, fault tolerance to

single memory data corruption and input stimulus filtering. The response time on an input stimulus for the CAM based system is a fixed time of two clock cycles. Only in the case of a single memory data corruption error within the stored data within one or both of the CAMs the maximum number of clock cycle will be four. The current limitation of memory data corruption on a single error is limiting the fault tolerance of our work for the current simulation based application and was the scope of this phase of our in progress research work. Adaptation of altered parity check concepts are targeted for multiple data bit upsets and data reconstruction will be achievable. In our system based on simulation a stable response time of the memory based controller can be achieved and adapted application on this system will benefit from this. The use of fault tolerance based on parity bit evaluation has been done on using duplicate data entries which can be accessed through content variation for exclusion of hardware overhead. Input filtering reduces the impact of system malfunction by only evaluation of the required signals at the current application step.

Acknowledgements

This work is supported by the EPSRC Centre for Innovated Manufacturing in Through-life Engineering Services EP/I033246/1.

References

[1] P. Schiefer, R. McWilliam, and A. Purvis, "Creating a Self-configuring Finite State Machine out of Memory Look-up Tables," *Procedia CIRP,* vol. 11, pp. 363-366, // 2013.

[2] R. Senhadji-Navarro, I. Garcia-Vargas, G. Jimenez-Moreno, and A. Civit-Ballcels, "ROM-based FSM implementation using input multiplexing in FPGA devices," *Electronics Letters,* vol. 40, pp. 1249-1251, 2004.

[3] L. Frigerio and F. Salice, "RAM-based fault tolerant state machines for FPGAs," in *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT '07. 22nd IEEE International Symposium on*, 2007, pp. 312-320.

[4] R. Senhadji-Navarro, I. Garcia-Vargas, and J. L. Guisado, "Performance evaluation of RAM-based implementation of Finite State Machines in FPGAs," in *Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on*, 2012, pp. 225-228.

[5] E. F. Moore, "Gedanken-experiments on sequential machines," *Automata studies,* vol. 34, pp. 129-153, 1956.

[6] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell System Technical Journal,* vol. 34, pp. 1045-1079, 1955.

[7] R. Leveugle, R. Rochet, G. Saucier, L. Martinez, and C. Pitot, "A synthesis tool for fault-tolerant finite state machines," in *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*, 1993, pp. 502-511.

[8] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines-a survey," *Proceedings of the IEEE,* vol. 84, pp. 1090-1123, 1996.

[9] X. Wendling, R. Rochet, and R. Leveugle, "ROM-based synthesis of fault-tolerant controllers," in *Defect and Fault Tolerance in VLSI Systems, 1996. Proceedings., 1996 IEEE International Symposium on*, 1996, pp. 304-308.

[10] I. Garcia-Vargas, R. Senhadji-Navarro, G. Jimenez-Moreno, A. Civit-Balcells, and P. Guerra-Gutierrez, "ROM-Based Finite State Machine Implementation in Low Cost FPGAs," in *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*, 2007, pp. 2342-2347.

[11] K. Pagiamtzis, N. Azizi, and F. N. Najm, "A Soft-Error Tolerant Content-Addressable Memory (CAM) Using An Error-Correcting-Match Scheme," in *Custom Integrated Circuits Conference, 2006. CICC '06. IEEE*, 2006, pp. 301-304.

[12] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: a tutorial and survey," *Solid-State Circuits, IEEE Journal of,* vol. 41, pp. 712-727, 2006.