On-line Self-Checking of Microprogram Control Units[1]

Ilya Levin* and Mark Karpovsky**

*Tel Aviv University
School of Education,
Ramat-Aviv 69978, Israel
Center for Technological Education
E-mail: ilia1@post.tau.ac.il

**Boston University
Department of Electrical,
Computer and System Engineering
College of Engineering, Boston, MA 02215
E-mail: MR@enga.bu.edu

Abstract

The paper introduces a new approach for designing of self-checking Microprogram Control Units (MCU). Using a Finite State Machine (FSM) as a form of MCU representation leads to its PLA-implementation having a system of product terms which are orthogonal and complete. An additional important property of MCU is related to a limited total number of possible code-words. These features can be used for construction of self-checking PLAs with relatively small overheads.

Keywords: Self-checking, on-line checking, microprogram control unit.

1. Output vectors of fault-free devices are always code-words of a specific code, and error detection is implemented by checking whether the present output belongs to the code.

Major difficulties in design of self-checking devices are related to complexity of decoding (which is verification that a given output is a code-word).

An objective of the paper is to derive a new method of synthesis of self-checking MCUs, which is highly technological and provides essential reduction of overheads.

Comparison of the proposed method with known two approaches, (i.e. the concurrent error detection by series of checkers [3], and the concurrent testable PLA using modified Berger code [2, 4]) shows that the proposed approach provides less expensive and more technological

[1] This research was supported in part by NSF under grant No. MIP9630096.

checking circuitry. It becomes possible because of the following properties of MCU: completeness, orthogonality and the fact that a number of code-words in an arbitrary control unit is significantly smaller than a total number of its product terms.

These properties enable to design the checker by implemen

form of output functions of the MCU. Each product term in this form corresponds to a certain output code vector. Generally, this approach was considered unacceptable, because a total number of code-words of an arbitrary device may be very large. However, in the case of MCUs this approach is very perspective because usually, a number of

-words is significantly smaller then a number of its product terms. Based on this important assumption we propose an architecture of a self-checking MCU.

A self-checking MCU consists of its own circuit to be checked and a checker, which checks its outputs to see if an error has occurred. The checker has an ability to expose its own faults as well.

Concurrent checking of PLA is only possible when one product term (i.e. only one row in the AND array of the PLA) is activated at a time by any input vector. Obviously, not every logical system, which is implemented within PLA, satisfies such a condition. However, there is a large class of systems, which do satisfy this condition. It is the class of MCUs [1].

2. The main idea of on-line self-checking is to detect non-code outputs. To prevent possible fault masking while taking into account unidirectional nature of faults, we will code code-words (OR-array code vectors) by the well-known modified Berger code [2].

The main idea of our approach is to implement a self-checking checker as a sum of products of Boolean function z(y_1 $_N$) of code-word variables y_1 $_N$, which function z is equal 1 if the code-word is code and equal 0, if the code-word is not.

The method will be described with the reference to Fig. 1 and Fig. 2, where Fig. 1 is a direct checking scheme and Fig. 2 is a scheme based on compressing of micro-instructions.

2.1 Fig.1 shows a suggested checking scheme of a self-checking MCU implemented as a PLA. The PLA to be checked consists of two arrays: AND array (M1) and OR array (M2). The OR-array M2 consists of two parts. The first part comprises columns generating output functions y_1 $_N$ and the second part comprises columns generating the next state functions.

Input lines to this PLA scheme are x_1 $_L$, output lines are y_1 $_N$. Next state lines D_1 $_R$ connect the OR-array to the memory. Present state lines t_1 $_R$ connect the memory to the AND-array.

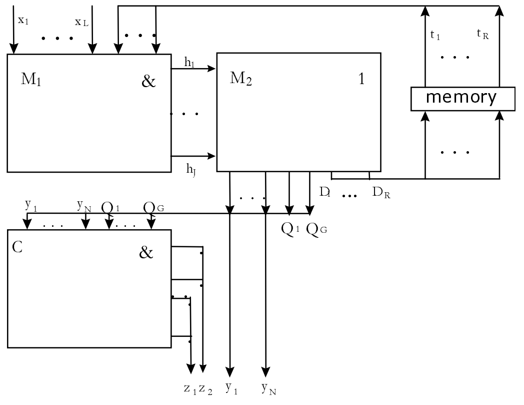

Figure 1. Architecture of Self-Checking MCU

We suggest to introduce additional circuitry which is composed of the following two portions.

Firstly, there is an additional PLA marked C, which will be a checker for output functions of the original PLA. The AND array of the checker C is programmed to have non-repeating codes of the OR array of the original PLA. These non-repeating codes form the list of Berger coded code-words of the MCU to be implemented. Thus, the number of rows in AND array of the checker C is equal to the number of possible code-words (T). OR array of the checker consists of only two vertical output lines z_1 and z_2. Each of the lines is connected to half of product lines of the AND array of the checker.

Secondly, there are several additional (coding) columns in the AND array M1 in the PLA to be checked. Correspondingly, there are additional output lines, which represent redundant bits for Berger code with dual-rail outputs.

The scheme works as follows. Suppose a fault in the AND-array or in the first part of the OR-array leads to generation of a non-code output vector. This fact will be indicated by z_1=0, z_2=0 which means the presence of a fault. (For the case when the output vector is equal to a code vector of the AND array of the checker, causing output z_1=1, z_2=0 or z_1=0, z_2=1).

Checking of the second (next state part) of the OR-array will be discussed below.

Let us describe a synthesis procedure for designing the above scheme.

The AND array (M1) has the same program as the AND array of the initial MCU. OR-array includes additional output columns corresponding to redundant bits of Berger code for code-words. Each row of the OR array is appended by vector

in vector of the corresponding output row.

-

repeating rows of the OR array of the PLA to be

two columns z_1 and z_2 for indication of a presence of a fault.

The designed checker C is a totally self-checking checker. It is explained by the fact that single errors occurring within the checker C do not lead to its malfunctioning, namely:

- cross-

urn both z1 and z2 to 0;

- stack-at-1 and stack-at-0 of input lines and product terms will be detected in the same manner;
- cross-

functioning;

- stack-at-product faults and stack-at faults of output lines will be detected by z_1=z_2.

2.2. The OR array of the PLA to be checked is a sparse matrix, since the number of outputs written in each row of the table of MCU is much smaller than a number of possible subsets of the set Y={y_1 $_N$} of the MCU. For minimization of the area of the OR array, we will encode each micro-instruction by a binary code.

The scheme of a self-checking MCU with micro-instructions compressing is shown in Fig. 2.

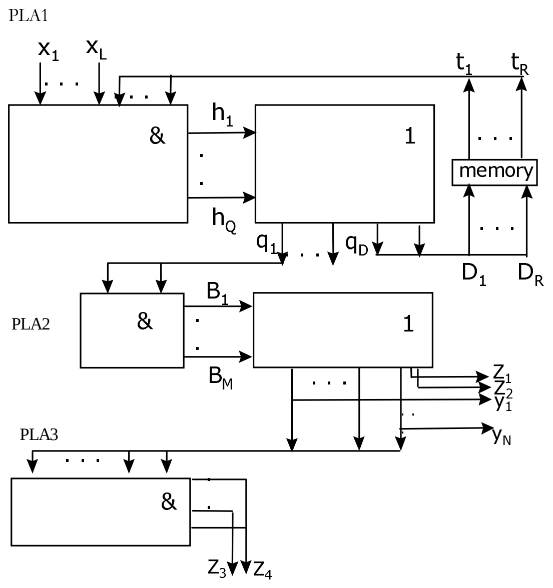

Figure 2. Micro-instruction compressing scheme for self-checking MCU

For the proposed scheme comprises:

1. The OR array of the upper PLA (PLA1) implements encoding of micro-instructions.
2. PLA2 performs decoding of the set of micro-
   the modified Berger code.
3. PLA3 is a checker for the OR-array of PLA2.

The OR array of PLA2 produces two fault detection signals z_1 and z_2 since AND array of the PLA2 is a checker for PLA1. Fault signals z_1 and z_2 detect all faults of the scheme except for cross-points and stack-at-product errors in OR-array of PLA2. To detect these faults we will use PLA3 (Checker).

The presence of two couples (z_1, z_2) and (z_3, z_4) of fault indicating signals renders the scheme more reliable. Particularly, it can indicate not just the fact, but also the location of faults. It also enables to detect some multiple faults.

2.3. In the above-suggested schemes the proposed checkers checks just output functions. Therefore, these checker are unable to detect faults, which occur in the next state portion of the OR array.

We suggest a new approach to solve this problem with minimum redundancy by using the same approach for checking which was introduced above, but without any additional checker. That becomes possible owing to the fact that state variables are transferred to the AND array of the original PLA which includes all product terms corresponding to output vectors of the next state variables. If MCU states are encoded by any code detecting unidirectional errors, a fault occurring in the next state portion of the OR array will lead to a non-code vector of the state variables. Consequently, this output non-code vector will not activate any product term of the AND array of the original PLA, which provides for fault detection. In other words, the AND array will play a part of a checker for the next state part of the OR array of the PLA.

We note, that an error may be detected a bit later then it is necessary: i.e. at the next clock after the fault appearance. But this disadvantage is compensated by the minimal overhead. Actually, the additional circuitry comprises few columns to be introduced according to a special state

can be used for the state assignment.

3. Let us make a cost estimation for implementing our design using the following notations.

L number of input lines,

N number of output lines,

Q number of product terms,

R number of states of FSM,

T number of code-words.

W(R)

for the set of FSM states,

B(Q)

set of product terms.

The estimated cost So of the area of an original PLA can be calculated as follows:

$So = 2LQ + 2(\text{intlog}_2 R)Q + NQ + (\text{intlog}_2 R)Q = Q(2L + 3(\text{intlog}_2 R) + N)$.

The estimated cost S_1 of the first proposed self-checking structure (Fig. 1) can be calculated as follows:

$S_1 = Q(2L + 3W(R) + N + B(Q)) + 2T(N + B(Q) + 1)$.

Cost S_2 of the second proposed self-checking scheme (Fig. 2) can be estimated as follows:

$S_M = 2QL + Q*(W(T) + W(R)) + 2T*W(T) + T(N + B(T)) + 2T(N+B(T))$

The percentage of area overheads for each of the proposed schemes is computed as shown below:

$\Omega_1=(S_1-S_o)/S_o$;

$\Omega_2=(S_2-S_o)/S_o$.

Overhead values, calculated for 47 different FSM benchmarks in accordance with the above two equations, are presented in Table 1.

Name	L	N	R	Q	T	So	S1	Ω1	S2	Ω2
acdl	16	27	22	214	23	15194	20518	0.35	12282	-0.19
amtz	23	52	85	261	103	30276	47705	0.58	43182	0.426
araf	25	65	44	193	61	25090	38506	0.53	31048	0.237
ass1-3	5	25	19	60	19	2820	5040	0.79	3910	0.387
basecamp	18	39	14	75	31	6300	10665	0.69	9304	0.477
bech	18	39	14	75	31	6300	10665	0.69	9304	0.477
berg	21	51	35	173	52	18684	28245	0.51	22739	0.217
big	18	28	17	185	17	14060	18512	0.32	11664	-0.17
bs	19	13	17	185	17	11655	15008	0.29	11014	-0.05
bull	44	13	24	281	24	31753	36784	0.16	30630	-0.04
cat	11	22	15	45	17	2385	4248	0.78	3626	0.52
cow	49	24	24	261	25	34974	41122	0.18	32382	-0.07
cpu	14	29	16	45	23	3105	5479	0.76	5248	0.69
cyr	20	75	42	198	59	25740	40168	0.56	31169	0.211
exam161	16	17	11	50	14	2900	4416	0.52	3424	0.181
examp1	12	13	24	168	24	8232	11568	0.41	8352	0.015
examp10	10	13	19	76	19	3420	5206	0.52	4142	0.211
examp15	13	20	18	129	17	7482	10654	0.42	7030	-0.06
examp16	13	18	18	132	17	7392	10550	0.43	7014	-0.05
examp17	8	17	11	50	15	2100	3660	0.74	2715	0.293
examp18	10	12	15	52	15	2132	3548	0.66	2811	0.318
examp2	43	18	18	127	18	14732	17846	0.21	14536	-0.01
examp4	31	24	24	137	25	13426	17342	0.29	13562	0.01
examp6	11	20	11	56	18	2856	4764	0.67	3848	0.347
examp7	12	20	17	110	18	6160	9040	0.47	6160	0
go1	18	64	58	228	97	26220	45466	0.73	41218	0.572
knot2	7	9	9	32	14	1024	1868	0.82	1626	0.588
kobz	19	53	29	231	57	23793	35370	0.49	26322	0.106
lcu	15	24	22	113	23	7458	10826	0.45	7870	0.055
lift	14	30	20	62	27	4340	7346	0.69	6654	0.533
lift2	14	16	13	50	17	2650	4264	0.61	3564	0.345
max	26	41	26	157	46	16485	24106	0.46	19673	0.193
md	22	53	59	338	84	37856	54866	0.45	42292	0.117
ort	61	48	56	214	94	39590	54236	0.37	51956	0.312
oshr	19	72	55	213	69	26625	42639	0.6	34698	0.303
pr	24	17	18	158	17	12166	15758	0.3	11462	-0.06
rafi	18	70	52	248	78	30008	47972	0.6	38508	0.283
ratm	19	57	73	234	60	26442	38916	0.47	29358	0.11
raz	23	72	40	148	70	19684	34296	0.74	32572	0.655
rm	14	15	15	50	16	2600	4108	0.58	3540	0.362
roiz	17	53	35	251	48	25602	36537	0.43	24753	-0.03
sara	19	44	36	112	45	10864	17716	0.63	15745	0.449
sasi	19	54	38	185	65	19795	31480	0.59	27130	0.371
vl_10	15	18	18	264	18	15840	21420	0.35	13452	-0.15
vl_6	14	18	17	169	17	9802	13626	0.39	8832	-0.1
vl_20	14	29	18	367	18	25323	33153	0.31	18042	-0.29

Table 1. Results for FSM Benchmarks

Average overhead Ω_1 of the first scheme (≈40%) can be estimated as a good result in light of the two main advantages of the proposed structure, i.e. its technological simplicity and total self-checking ability. Furthermore, the proposed structure allows checking of both the logical portion of MCU and its next state portion, which feature is not provided in any known scheme.

Average overhead Ω_2 (≈10%) of the second structure looks even more promising.

Summary

We have proposed an approach in the area of synthesis of self-checking Microprogram Control Units (MCU). In spite of tremendous strides made in the theory of self-checking design, an efficient synthesis procedure for design of self-checking MCUs has not been developed. We have tried to fill this vacuum by proposing a structure of self-checking MCU. Owing to use of several intrinsic features of MCU, the proposed structure allows to reach good solutions from the point of resulting overhead.

REFERENCES

1. S. Baranov. Logic Synthesis for Control Automata. Kluwer Academic Publisher. Dordrecht/Boston/London. 1994.

2. H. Dong. Modified Berger Codes for Detection of Errors. Digest of Papers 12th Annual Symp. on Fault-Tolerant Concurrent Computing., pp. 317-320, June, 1982

3. J. Khakabaz, E. J. McCluskey. Concurrent Error Detection and Testing for Large

vol. Ed-29, no. 4, April 1982, pp. 756-764.

4. G. P. Mak, J.A. Abraham and E. S. Davidson. The Design of PLAs with concurrent Error Detection. Digest 12th Int. Symp. Fault-Tolerant Computing, 1982, pp. 303-310.