Embedded Firmware Development with Multi-Way Branching

Václav Dvořák
Brno University of Technology, CZ
dvorak@fit.vutbr.cz

Abstract

This paper proposes a technique of firmware development based on Multi-valued Decision Diagrams (MDDs). Evaluation of multiple-output Boolean functions is faster than the one using Binary Decision Diagrams (BDDs) and has a small memory footprint often required in embedded systems. A micro-programmed controller that firmware runs on is supposed to support multi-way branching in hardware, whose implementation is known. A novel heuristic technique of a sub-optimal multivalued MDD synthesis is presented and a specific condition for spatial efficiency of MDD-based firmware is derived. The method is illustrated on practical examples. It may be quite useful for development of embedded microcontroller firmware as well as for fast digital system simulation.

Keywords: Embedded firmware, decision diagrams, iterative disjunctive decomposition, multi-valued functions, space complexity

1. Introduction

Fast and efficient embedded systems are being built with micro-programmed controllers on FPGA in order to reduce time to market. Small amount of memory, low power consumption and low cost are obvious requirements, so that development of firmware with a limited microcode size is gaining importance.

If the micro-program scans Boolean expressions one variable at a time and uses binary logic operations like PLC [1], reading/testing of input variables is redundant and multiple Boolean functions are evaluated sequentially one after another, which is not too efficient. In case of Reduced Ordered Binary Decision Diagrams (ROBDDs) [2], redundancy is implied by repeated testing of variables in each ROBDD if several functions of the same variables are evaluated. A BDD variant known as MTBDD (Multi-Terminal BDD) can remove this redundancy. It represents an integer function of Boolean variables. Nonetheless, if we do care for performance and memory space, testing of two or more binary variables at a time can provide a sufficient speedup. Generally testing *m* binary variables at a time can be considered as testing a single $M = 2^m$ valued variable and a multi-valued MDD can serve as a suitable representation [3].

The main contribution of the paper is a heuristic method of MDD synthesis of multi-terminal MDDs that can be directly rewritten to firmware. Code generation for embedded systems appeared also in [4], but has been limited to single-output Boolean functions only. The used heuristics optimized an average evaluation time with less regard to memory space. The ordering of variables in the ROBDD was assumed, whereas in our method we derive ordering of multi-valued variables directly.

The paper is structured as follows. In the following Section 2 we will review definitions and representation of Boolean functions with the use of decision diagrams (DDs). Our heuristic approach to construction of sub-optimal MDDs representing Boolean functions is explained in Section 3. Section 4 analyzes efficiency of micro-programs obtained from these MDDs. Some examples are included in Section 5. Results are commented on in Conclusion.

2. Basic definitions and notions

To begin our discussion, we define the following terminology. A system of *m* Boolean functions of n Boolean variables (also known as a multiple-output Boolean function),

$$f_n^{(i)} : (Z_2)^n \rightarrow Z_2 , \ i = 1, 2, ..., m \qquad (1)$$

will be simply referred to as a multiple-output Boolean function F_n. Equivalently, integer (*R*-valued) function of *n* Boolean variables can be used,

$$F_n : (Z_2)^n \rightarrow Z_R . \qquad (2)$$

with output values from $Z_R = \{0, 1, 2, \ldots, R\text{-}1\}$, $R \leq 2^m$. Function F_n is incomplete if it is defined only on set $X \subset (Z_2)^n$; $(Z_2)^n \setminus X = D$ is then the don't care set.

The above function (2) is a special case of a discrete

R-valued function of *n M*-valued (or *M*-ary) variables

$$F_n: (Z_M)^n \rightarrow Z_R; \quad (3)$$

a compact representation of (2) is obtained if integers from Z_M and Z_R are interpreted as $\lceil \log_2 M \rceil$ and $\lceil \log_2 R \rceil$ binary values. If $M < 2^{\lceil \log_2 M \rceil}$, unused combinations of binary input values are in fact don't cares in (2).

Machine representation of Boolean functions uses binary decision diagrams (BDDs), which can have many forms. Ordered BDDs (OBDDs) use the same order of variables along all paths, whereas free DDs relax this restriction. For a specific variable order and the given function can be the size of OBDD reduced to a canonical form of ROBDD [2] with a minimum number of decision nodes (i.e. BDD size). The same is true for multi-terminal binary decision diagrams (MTBDDs) representing binary-input, integer-valued output functions [5].

The DD size is the important parameter as it directly influences the size of data structure needed to store the DD. However, the size of a DD is very sensitive to variable ordering and finding a good order even for BDDs is an NP-complete problem [5]; there are *n*! possible orderings of *n* variables. We will refer to ROBDD or MTBDD with the best variable ordering as to the optimal DD. The term "sub-optimal DD" will denote a DD with the size near to the optimal DD. Such DDs result from heuristic synthesis techniques [6]. The size of DDs for random functions grows exponentially with number of variables *n* for any ordering, but functions used in digital systems design with few exceptions do have a reasonable DD size. The average path length (APL) of a DD, that relates to the average evaluation time, is another parameter subject to optimization [7].

M-ary MDDs are straightforward generalization of BDDs. They have two types of nodes: decision and terminal nodes. Decision node *L* is testing *M*-ary variable var(*L*) and its outgoing edges are marked by its values 0, 1, …, *M*-1. The terminal node assigns a single value from Z_R , (generally $R \neq M$) to the function value $y = F_n(x_1, x_2, \ldots, x_n)$. Ordered MDDs are better suited to evaluation of Boolean functions as the traversal from the root to a leaf can be much shorter than for OBDDs, depending on the value of *M*. If the *M*-ary variables are coded in *m* bits, evaluation process can be up to *m*-times faster with the MDD than with a BDD.

Def.1. The *M*-ary program associated with a MDD of function F_n consists of finite number of labeled instructions of the type

$L_u \; x_i \; S_0 \; S_1 \; \ldots \; S_{M-1}$

where L_u is instruction label, x_i is *M*-ary variable and S_k are symbols of two kinds of commands:

- go to label L_h (a non-terminal command)
- the value of the function is *v* (a terminal command).

Apparently, there is one-to-one correspondence between instructions of the *M*-ary program and MDD nodes. Each of *M* fields reserved for symbols S_k contains a tag specifying whether the content of the field is to be interpreted as a label of a next instruction or as a value of the function and the command stop. The ordering of instructions is not important, only the initial instruction must be specified. Instructions are interpreted this way: if the value of the variable x_i under the test is $x_i = q$, execute command S_q.

Def. 2. The ordered DD is redundant, if each test variable is used at one and only one level of the DD.

In what follows only irredundant ordered DDs will be considered, even though redundant testing may sometimes lead to a lower DD size.

Each of *N* nodes in OMDD is described by a table with $M \leq 2^m$ items. Each item has a format indicator (decision/ terminal node) followed either by a pointer to a successor node $\lceil \log_2 N \rceil$ bits wide or by the output value $r = \lceil \log_2 R \rceil$ bits. The size of the data structure is therefore in the worst case

$$\text{space} = NM\,[1 + \max(\lceil \log_2 N \rceil, r)] \text{ bits.} \quad (4)$$

3. Construction of sub-optimal MDDs

Evaluation of Boolean functions in firmware could rely on the full function map stored in the memory. This approach is in embedded systems acceptable for about less than 10 binary variables. For a larger number of variables we have to use a more compact data structure – a network of LUTs corresponding to MDD nodes. In this section we will present a heuristic technique of a suboptimal MDD construction. It is generalization of the approach taken in BDD construction [6], when we do iterative disjunctive decomposition of the original function, removing one input variable after another.

We prefer to explain the synthesis technique on an Example 1. The ternary function of three ternary variables F(x1, x2, x3) is specified by a map in Fig.1. The construction of the MDD starts from terminal nodes (leaves); 3 function values can be seen as "sub-functions" of zero variables and create the lowest level of the MDD. A next level are decision nodes that correspond to distinct triplets of ternary values of F ("output triplets") associated with values 0, 1, 2 of a certain input variable ("an input triplet"). If the output triplet consists of the same values, no decision node is needed. The distinct output triplets are in fact all single-variable sub-functions of F.

The question is which variable should be used in

any given step. Our heuristics selects the variable with a minimum number of sub-functions (and thus decision nodes). In case of ties the variable with the lowest number of non-constant sub-functions is taken. In case of ties again, one variable is selected randomly. In our example 1 we have the following numbers of sub-functions associated with 3 variables:

x1: 3, x2: 5, x3: 6.

The choice is clear, and the distinct output triplets generated by variable x1 are 222, 210, and 000. To remove x1 from input variables of F, it is sufficient to replace output triplets by new id codes resulting from their enumeration. The decomposition step can then be visualized as tiling a map of *F* with a smallest possible assortment of tiles (id codes), Fig.1.

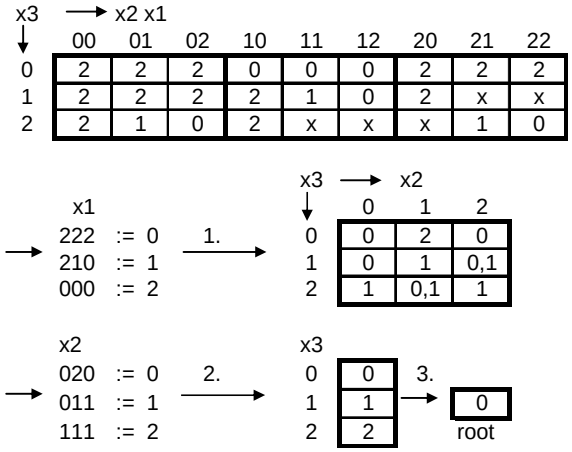


Fig.1. Disjunctive decomposition of ternary function *F* of 3 ternary variables (Example 1)

Since there are don't cares in the map at Fig.1, we have some choices in the second decomposition step. Variable x2 is chosen, because with suitable cover of don't cares only two decision nodes are sufficient, whereas x3 would need three. The top level of the MDD consists of one decision node, the root. The MDD is obtained by reversing the decomposition procedure and reversing assignments, Fig.2.

There are other heuristic techniques to obtain sub-optimal MDDs. E.g. the algorithm [4] can convert the BDD with the given order of variables into a MDD in which the average path length (APL) is minimized. Another technique for minimizing the BDD cost is known as sifting [5]. These techniques could probably be generalized for MDDs.

Let us note that in our technique incomplete functions can be decomposed the same way. However, the enumeration process must be done more carefully, because sub-functions can also be incomplete and can be combined with complete constant or non-constant sub-functions in different ways to reduce the total count as much as possible.

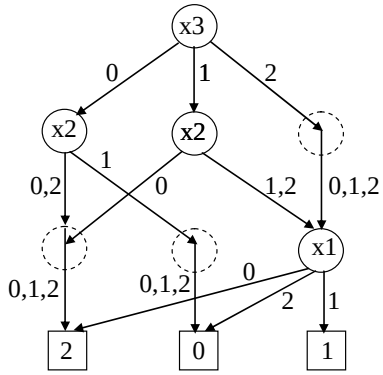


Fig.2. Ternary DD of function *F* of 3 ternary variables (Example 1)

4. Spatial efficiency of MDD-based micro-programs

To produce space-efficient MDD-based micro-programs, we will require that the memory capacity for storing the micro-program be less than the size of a full function table. In this section we will derive a necessary condition ensuring this property.

Theorem 1. Let discrete function F: $(Z_M)^n \rightarrow Z_R$ be specified by the MDD with $C(F)$ decision nodes. Then the M-ary program evaluating this function is space efficient if $C(F)$ fulfills the condition

$$Z \log_2 Z < U \tag{5}$$

where

$$Z = C(F)\sqrt{2}\sqrt[M]{n},$$

$$U = 2^{n\lceil \log_2 M \rceil}\lceil \log_2 R \rceil \sqrt[M]{n}\,/(M\sqrt{2})$$

Proof.

The M-ary program, as defined previously, will consist of $C(F)$ instructions, each w bits long, where

$$w = \lceil \log_2 n \rceil + M(\lceil \log_2 C(F) + 1 \rceil .$$

The size of the full function table with the use of binary coding will be

$$s = 2^{n\lceil \log_2 M \rceil}\lceil \log_2 R \rceil .$$

The condition of space efficiency is thus $wC(F) < s$. When we remove rounding to nearest integer from expression $wC(F)$, the upper bound can be increased to

$$wC(F) < [2\log_2 n + M(2\log_2 C(F) + 1)] =$$

$$= C(F)\left\lceil \log_2 n^2 + \log_2 C(F)^{2M} + \log_2 2^M \right\rceil =$$

$$= 2MC(F)\log_2\left\lfloor \sqrt[M]{n}\, C(F)\sqrt{2} \right\rfloor .$$

The original inequality will be certainly fulfilled when

$$2MC(F)\log_2\left\lfloor \sqrt[M]{n}\, C(F)\sqrt{2} \right\rfloor < s \quad \text{or}$$

$$\sqrt{2}\sqrt[M]{n}\left(2MC(F)\log_2\left\lfloor \sqrt[M]{n}\, C(F)\sqrt{2} \right\rfloor\right) < s\sqrt{2}\sqrt[M]{n}\ ,$$

what can be written in a form

$$Z\log_2 Z < s\sqrt[M]{n}\sqrt{2}/(2M) = U,$$

where $Z = \sqrt[M]{n}\sqrt{2}\,C(F)$,

Q.E.D.

Example.

Let us have function F: F_n: $(Z_2)^8 \rightarrow (Z_2)^8$, so that M=2, R=256. The value of U becomes $U = 2^{11}$ and

$$Z\log_2 Z < 2^{11}, \text{ i.e. } Z < 2^8.$$

As

$$Z = \sqrt[M]{n}\sqrt{2}\,C(F) = 4C(F) < 2^8\ ,$$

the result is $C(F) \le 64$.

End of example.

5. Micro-program synthesis examples

Two typical examples will be solved in this section, combinational and sequential logic circuits.

Example 2. The task is to detect a number of days in a month and a year from the state of binary counters for months (m3 m2 m1 m0) and years (y1 y0). (The fact that once in 400 years we do not have the leap year is not taken into account).

In this case we have the 4-valued function of 5 binary variables because it turns out that the number of days in a month does not depend on bit m1. The map of the function is shown in Fig.3.

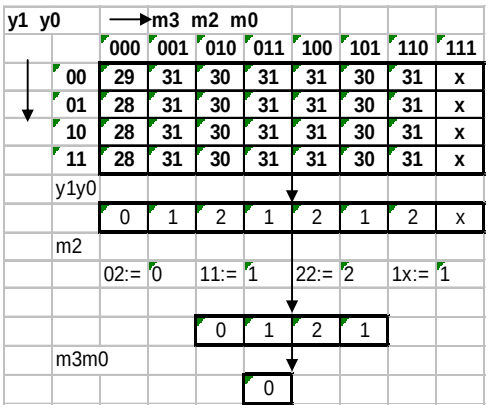
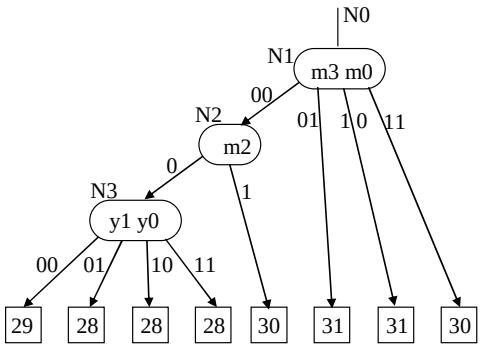


Fig.3. Decomposition of the sample 4-valued function (Example 2)



a)

N0 exit N1@m3m4
N1@00 exit N2@m2
N1@01 S31 exit N31
N1@10 S31 exit N31
N1@11 S30 exit N30
N2@ 0 exitN3@y0y1
N2@ 1 S30 exit N30
N3@00 S29 exit N29
N3@01 S28 exit N28
N3@10 S28 exit N28
N3@11 S28 exit N28

b)

Fig.4. Example 2. a) a heterogeneous MDD b) a symbolic micro-program

There are only 3 sub-functions of variables y1, y0 (three distinct columns), and only one of them different from a constant (the first column). A group y1, y0 is thus the best choice, because only one 4-way decision node results.

In the second decomposition step we can remove one or two variables simultaneously. The choice of m2 leads to only one binary decision node and m2 is therefore selected. Finally two remaining variables m3 and m0 are used to decide one of 4 ways. The resulting heterogeneous MDD mixes binary and quaternary nodes, Fig.4a.

To run effectively our M-ary program on a hardware micro-engine, faster than general purpose CPU core, a support for multi-way branching must be provided. A suitable architecture of a micro-engine, a modified version of the one in [8], is depicted in Fig.5.

Instead of an M-ary program, we will use more practical formats of shorter micro-instructions. The format of a microinstruction is specified by format indicator (FI) bits. These bits are decoded and then control internal components of a controller. The basic format gives a state output (control signals) and

increments the microinstruction pointer (μIP := μIP+1). For multi-way branching two microinstructions formats should be supported:

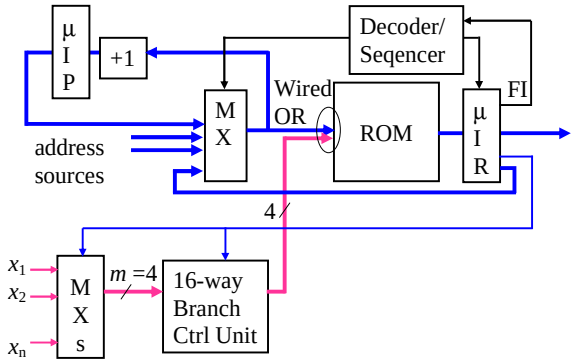


Fig.5. Micro-programmed controller architecture with multi-way branching

1) jump to an address specified in micro-instruction modified by BCU; MXs and BCU control,

2) conditional output, jump to an address specified in micro-instruction (no modification).

The first format includes jumps to the target address obtained from the address specified in the micro-instruction; this latter address is modified by external variables, by up to 4 variables at a time, including 0 variable (no modification), by means of 16-way Branch Control Unit (BCU). The task of this unit (such as Am 29803A) is to shift active inputs, selected by a 4-bit mask, to the lowest positions of the 4-bit output vector. This vector is then wire-ORed with the address obtained from the micro-instruction.

The symbolic micro-program targeted for the micro-engine in Fig. 5 is shown in Fig. 4b. Replacement of up to 4 bits in the address is denoted by operator "@". If wired-OR is used for replacement, the bits being replaced must be reset to 0.

Example 3. The sequential circuit with 2 state variables and 4 binary inputs with the state diagram at Fig.6 is to be implemented on the micro-engine of Fig.5.

The new state is given by equations

next Q1 = !w(Q1Q2+Q1!Q2!x+!Q2xy!z)+!Q1!Q2xy!z

next Q2 = xQ2+(x!yz+xy!z)(!Q1+!w)+Q1Q2w,

the state transition table and iterative decomposition are shown in Fig.7. We do testing input and state variables in groups of two in two steps. The lowest level of MDD degenerates to only a single node, other nodes (in dark) do not decide anything. The next MDD level up consists of 4 nodes and corresponds to 4 states of the state machine. The whole MDD is in Fig. 8a.

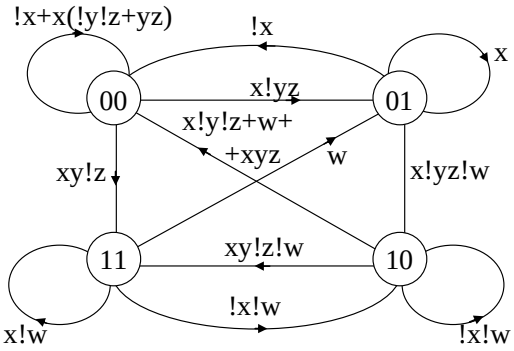


Fig.6. Example 3. State transition graph.

Implementation of a state machine is different from the combinational circuit in that the MDD has 4 roots. The state of the machine is kept in a next-address part of μIR and the variables x, y, z, w are available for inspection via multiplexers and BCU. The next state is "computed" in 2 steps, as shown by 4-ary program in Fig.8b. The real vertical micro-program with reasonably short micro-instructions is at Fig. 8c. Four states are given by addresses N1 to N3; state N4 does not generate any output since N4 fields in the 4-ary program of Fig.8b (in dark) are marked as pointers. The marker bit can be used to disable the output.

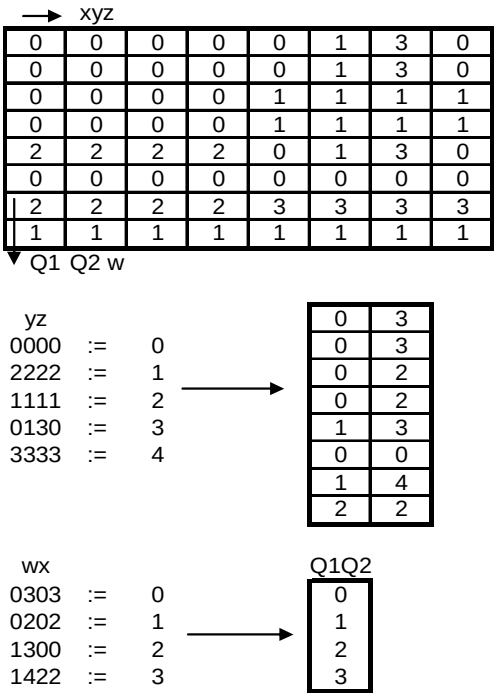
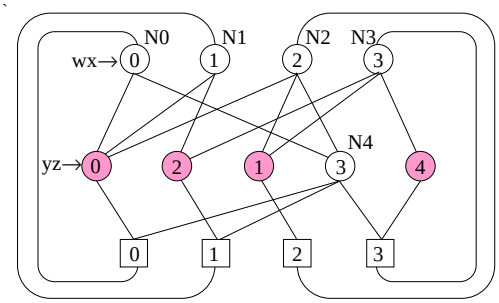


Fig.7. Example 3. The state transition table and iterative decomposition



a)

		0	1	2	3
N0	wx	N0	N4	N0	N4
N1	wx	N0	N1	N0	N1
N2	wx	N2	N4	N0	N0
N3	wx	N2	N3	N1	N1
N4	yz	N0	N1	N3	N0

b)

N0@00 exit N0 @wx	N2@10 exit N0 @wx
N0@01 exit N4 @yz	N2@11 exit N0 @wx
N0@10 exit N0 @wx	N3@00 exit N2 @wx
N0@11 exit N4 @yz	N3@01 exit N3 @wx
N1@00 exit N0 @wx	N3@10 exit N1 @wx
N1@01 exit N1 @wx	N3@11 exit N1 @wx
N1@10 exit N0 @wx	N4@00 exit N0 @wx
N1@11 exit N1 @wx	N4@01 exit N1 @wx
N2@00 exit N2 @wx	N4@10 exit N3 @wx
N2@01 exit N4 @yz	N4@11 exit N0 @wx

c)

Fig.8. Example 3. a) MDD with multiple roots b) 4-ary micro-program c) the symbolic micro-program

6. Conclusions

Complexity of functions that can appear in embedded systems varies a great deal and so do their space and time requirements in various evaluation techniques. There is no single optimal method for evaluation of all Boolean functions. Firmware evaluation of *M*-valued functions is faster than evaluation in software or on universal microprocessors, especially if hardware support for multi-way branching is provided. Also memory size to store micro-programs is sufficiently small. Binary programs due to fine granularity take indeed little less amount of memory than *M*-ary programs, but are almost *M*-times slower.

The presented synthesis of sub-optimal MDDs, when completely automated, will now be compared to other techniques (APL optimization [4], MDD cost minimization by sifting [5], etc). Future research should look also at non-disjunctive decompositions, redundant MDDs and free MDDs. Also an efficient procedure for finding sub-optimal MDDs for a set of Boolean functions given by expressions would be very valuable and is still missing.

6. References

[1] F. D. Petruzella: *Programmable Logic Controllers*, McGraw Hill Science/Engineering/Math, 2004.

[2] H.R Andersen, An Introduction to Binary Decision Diagrams. Lecture notes for 49285 Advanced Algorithms E97, http://www.itu.dk/~hra/notes-index.html

[3] T.Kam, T.Villa, R.K.Brayton, and A.L. Sagiovanni-Vincentelli, "Multi-valued decision diagrams: Theory and Applications," *Proc. of* the *Multiple-Valued Logic*, 1988, Vol. 4, No. 1-2, pp. 9–62, 1998.

[4] S. Nagayama and T. Sasao, "Code generation for embedded systems using heterogeneous MDDs," *Proc. of the 11th Coference Synthesis And System Integration of Mixed Information technologies* (SASIMI 2003), Hiroshima, April 3-4, 2003, pp.258-264.

[5] R. Drechsler, B. Becker, *Binary Decision Diagrams - Theory and Implementation.* Springer 1998

[6] V. Dvořák: An optimization technique for ordered (binary) decision diagrams, *Proceedings of the 6th Annual European Computer Conference CompEuro' 92*, Hague, NL, 1992, pp. 1-4.

[7] J. Butler and T. Sasao, "On the average path length in decision diagrams of multiple-valued functions," *Proc. of the 33rd International Symposium on Multiple-Valued Logic*, Tokyo, May 16-19, 2003. pp.383-390.

[8] V. Dvořák: Microsequencer architecture supporting arbitrary branching up to 2^m targets, *Computer Architecture News*, IEEE Publ., US, March 1990, pp. 9-16.

Acknowledgement

This research has been carried out under the financial support of the research grants "Design and hardware implementation of a patent-invention machine", Grant Agency of Czech Republic GA102/07/0850 (2007-2009) and "Security-Oriented Research in Information Technology", MSM 0021630528 (2007-2013).