Branching Program-Based Programmable Logic for Embedded Systems

Vaclav Dvorak
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
dvorak@fit.vutbr.cz

***Abstract*—The paper considers realization of logic functions by branching programs running on special purpose Decision Diagram Machines (DDMs). It is not the fastest way to implement logic, but it enables different versions and frequent modifications, e.g., in embedded systems. First, this paper derives upper bounds on the cost of multi-terminal binary decision diagrams (MTBDDs); the cost is directly related to the size of branching programs derived from MTBDDs. Second, optimization of heterogeneous branching programs is undertaken that makes a space-time trade-off between the amount of memory required for a branching program and its execution time. As a case study, optimal configurations of branching programs are found for a set of benchmark tasks. Beside DDMs, the technique can also be used for micro-controllers with a support for multi-way branching running logic-intensive embedded firmware.**

Keywords- Boolean functions; multi-terminal binary decision diagrams MTBDDs; branching programs; MTBDD complexity; decision diagram machines DDMs

I. INTRODUCTION

The popularity of programmable architectures is due to the savings in hardware development time and cost. Various methods exist to realize multiple-output logic functions by programmable architectures. The FPGAs are widely used; however, they require layout and routing in addition to logic design. Look-up table (LUT) cascades, i.e., a series connection of memories, are more flexible, since the architecture is simple; various classes of functions can be realized by LUT cascades efficiently [1], [11]. Finally, Decision Diagram Machines (DDMs) are special purpose processors that evaluate decision diagrams. Branching programs that evaluate single- or multiple-output Boolean functions on DDMs can be directly constructed from decision diagrams (DDs).

A binary DD (BDD) represents a single Boolean function in a form of the directed acyclic graph with internal decision nodes controlled by input variables and with terminal nodes valued 0 or 1. Generalization to integer-valued terminal nodes leads to multi-terminal BDDs (MTBDDs) [1]. As the number of decision nodes in the ordered (MT)BDDs depends dramatically on the order of variables, we strive to find such variable ordering that reduces the node count as much as possible, and proportionally also the size of the branching program. As the optimal ordering belongs among NP-complete problems [1], heuristic methods have been suggested and used toward this goal. For example, the sub-optimal ordering of variables and (MT)BDD synthesis can be done simultaneously by the iterative decomposition of the original function, i.e., by repeatedly removing variables that minimize the node count at the current level of the diagram [2].

Mapping of optimal (MT)BDDs to branching programs is straightforward; non-terminal nodes are mapped to branch instructions, whereas terminal nodes to output instructions. Branching programs run faster on a special purpose processor (DDM) than on a general-purpose CPU [3]. Optimization criteria for branching programs are the execution time, memory size (area) or the area – time product. Some parameters subject to optimization are: testing more than 1 variable at a time, a number of instruction addresses, and a number of parallel DDMs. With the help of above optimizations, the execution speed of branching programs can be even adjusted to achieve very high performance [4]. Among applications of DDMs, let us mention micro-program sequencers, logic simulators, industrial programmable logic controllers and recently packet filters [5].

In this paper, we first analyze the MTBDD cost for general *R*-valued functions of Boolean variables. Then the class of sparse functions often used in real life is defined. The new results on upper bounds of MTBDD cost and profile of sparse functions are derived. This is generalization of results for single-output functions in [6]. In the second part, we show optimization of branching programs with respect to the area – time product. Heterogeneous MTBDDs for arbiters and controlled shift circuits serve to illustrate this optimization.

The paper is structured as follows. Section II introduces related works, whereas Section III gives the preliminaries. MTBDD profiles and costs for sparse logic functions are derived in Section IV. Mapping MTBDDs to branching programs is dealt with in Section V and branching program optimization in Section VI. The experimental results and future research directions are commented on in Conclusion.

II. RELATED WORKS

Various DDMs have been proposed in literature for evaluation various types of decision diagrams - ordered BDDs and Quaternary Decision Diagrams (QDDs), quasi-reduced, BDDs and QDDs (QRBDDs and QRQDDs) and ordered Heterogeneous Multi-valued Decision Diagrams (HMDDs) as well as Quasi-Reduced HMDDs (QRHMDDs). Six DDM architectures have been compared with respect to area-time complexity, throughput and compatibility to the existing memory [3].

Area-time complexity is important for embedded systems, because DDM with low area-time complexity dissipates low

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-184-7

power. Since the instruction memory occupies the most area for the DDM, we assume that the area is proportional to the memory size. The QDD Machine was found the best for area-time complexity [3]. Quasi-Reduced diagrams contain not only true decision nodes, but also degenerated nodes with one output edge only. This leads to higher memory consumption but enables pipelining and thus leads to the best throughput for QRQDD Machines [3].

The main problem with the above DDM architectures is that multiple-output functions are implemented by partitioning into single output functions. That is why we study the direct use of MTBDDs for branching programs. In Section IV we formulate hypothesis 4.1 suggesting that for a multiple-output sparse logic function the cost and the memory area to store the MTBDD are much lower than those for *r* BDDs of its *r* single-output component logic functions. Thus the architecture of the MTBDD Machine proposed in this paper should be superior.

Six DDM architectures mentioned above all use fixed number (1 or 2) of control inputs at decision nodes. The MTBDD machine is more flexible - the number of tested variables can be varied from one node to another, e.g., between 1 and 4. This lowers memory requirements and power consumption even further.

Code optimization for QDD Machines [10] has been achieved by means of 3 instead of 4 addresses in the instruction and by means of four types of branching instructions. In the other hand, in the MTBDD Machine we use only two instructions and only one base address that gets modified by the values of tested variables.

Applications for DDMs include industrial process controllers and logic simulators. Also a parallel DDM with 128 QDD Machines implemented on FPGA and running at 100 MHz has been proposed [4], that is about 100 times faster at the peak performance than Intel´s Core2 Duo microprocessor (@ 1.2 GHz) and requires a quarter of the memory.

III. BASIC DEFINITIONS AND NOTIONS

To begin our discussion, we define the following terminology. A system of *m* Boolean functions of *n* Boolean variables,

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

will be described as a logic function F_n with output values from $Z_R = \{0, 1, 2, \ldots, R\text{-}1\}$,

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

where R is the number of distinct combinations of m output binary values enumerated by values from Z_R.

Function F_n is incomplete if it is defined only on set $X \subset (Z_2)^n$; $(Z_2)^n \setminus X$ is the don't care set. (We assume that all component functions (1) have the same don't care set.)

Definition 3.1 Under the **sparse functions** F_n: $(Z_2)^n \rightarrow Z_R$ we will understand functions with the domain $(Z_2)^n$ divided into two subsets X and D, $(Z_2)^n = X \cup D$, $|X| << 2^n$, if one of the following conditions hold:

1) F_n is a fully specified function in $(Z_2)^n$,
$F_n: [X \rightarrow Z_R \setminus \{0\}, D \rightarrow \{0\}]$
(without loss of generality, value 0 is taken as the dominant value);

2) F_n is an incomplete function in $(Z_2)^n$, F_n: $X \rightarrow Z_R$ and $(Z_2)^n \setminus X = DC$ is the don't care set.

In this second case we can artificially define mapping $DC \rightarrow \{0\}$ and come back to the first case. Further on we therefore consider only the first case.

Definition 3.2 The **weight** of function F_n, denoted by u, is the cardinality of set X in Def. 3.1, $u = |X|$.

Definition 3.3 Let F_n: $(Z_2)^n \rightarrow Z_R$ be the function of binary variables $x_1, x_2, \ldots, x_n$. **Sub-function** $f(x_{n-k+1}, \ldots, x_{n-1}, x_n)$ of k variables is the function $f = F_n(v_1, v_2, \ldots, v_{n-k}, x_{n-k+1}, \ldots, x_{n-1}, x_n)$ for any given combination of binary constants $v_1, v_2, \ldots, v_{n-k}$.

Lemma 3.1 There are up to min $(2^{n-k}, R^{2^k})$ sub-functions of k variables, $k = 1, 2, \ldots, n$, but not all of them are necessarily distinct.

(Proof) According to Def. 3.3, each k-variable sub-function $(Z_2)^k \rightarrow Z_R$ is related to a particular binary vector $(v_1, v_2, \ldots, v_{n-k})$. There are 2^{n-k} such vectors and related sub-functions. On the other hand, the number of k-variable sub-functions is limited by the number of function values R. Maximum number of single variable (k=1) sub-functions is the same as the number of distinct pairs of function values, i.e., R^2. Two-variable sub-functions (k=2) are 4-tuples of function values and there are up to R^{2^2} of them. Continuing in the same way, we have up to R^{2^k} sub-functions of k variables (2^k-tuples of function values). A lower value of the two limits gives the bound, QED.

Definition 3.4 Let the order of variables in the MTBDD be $x_1, x_2, \ldots, x_n$ and the set of nodes controlled by x_j be the **level** j of the diagram. The **local width** w_j of the MTBDD at level j, $j = 1,2,\ldots, n$, is the number of all nodes at level j, i.e., the number of all distinct sub-functions of n-j+1 variables $x_{n-(n-j)}, \ldots, x_{n-j}, x_n$. The **width** w of the MTBDD is the maximum width of the MTBDD among the levels (w is referred to as the C-measure in [1]).

Note that k sub-function variables are counted from x_n backwards, whereas local widths $w_1, w_2, \ldots, w_j$ are indexed from x_1 onwards, i.e., the same way as are MTBDD levels. The relation between indices j and k is thus $j = n$-k+1.

Definition 3.5 Let F_n:$(Z_2)^n \rightarrow Z_R$ be the function of binary variables $x_1, x_2, \ldots, x_n$. The **profile** of the function F_n is the vector $(w_1, w_2, \ldots, w_n)$. Note that always $w_1 = 1$, $w_2 = 2$. The total sum of all non-terminal nodes is $W = w_1 + w_2 + \ldots + w_n$.

Definition 3.6 The **local cost** c_j of the MTBDD at level j is the number of true decision nodes (distinct non-constant sub-functions) in that level. The **cost** C of the MTBDD is the sum $C = c_1 + c_2 + \ldots + c_n$.

The local cost c_j is always less or equal the local width w_j because c_j includes only decision nodes with two output edges whereas w_j is the number of all nodes at level j including those with a single output edge (depicted by black dots in the sample MTBDD at Fig 3.1).

Example 3.1 A sample MTBDD is in Fig.1. The profile of the related function is {2, 3, 3, 4}. The number of true decision nodes is at the minimum (C =4); if the function is to depend on all its variables, at least one true node per variable is required. Note that decision nodes with a single output edge

do not decide anything. We can shift terminal nodes up to the root over a sequence of such nodes and branch to terminal nodes not only in the last level. For example, the terminal node 2 is reached after testing variable x_1 and x_2 only.

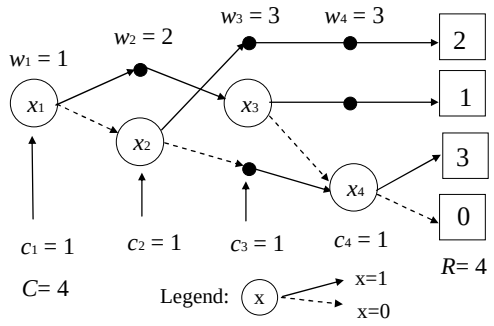

Figure 1 An example MTBDD for the 4-valued function of 4 Boolean variables.

Each of two characteristics, the profile and cost, is important in one of two different implementations of F_n. Whereas the profile, and especially the global width w, determine the LUT cascade configuration for F_n (hardware implementation, [11]), the size of the branching program is proportional to cost C; remaining W–C nodes just shrink to edges.

Most often two parameters of MTBDDs are optimized: cost C and width w. For branching programs based on MTBDDs, the cost optimization is of interest. Minimization of two parameters, cost C and width w, cannot be strictly separated. In the bottom-up synthesis of MTBDDs using heuristics [7], we select the variable so as to minimize the number of nodes in the next higher level of the MTBDD. If two variables produce the same number of nodes, we take the one with a lower number of true decision nodes. This goes on iteratively level by level, from leaves to the root. We expect that the total cost will be close to the minimum total cost. This has been confirmed for functions of up to 10 variables by an exhaustive search [11].

IV. Complexity Issues

Before analyzing complexity of sparse functions, we first review the complexity of general multiple-output Boolean functions. The knowledge of a function profile is essential for complexity analysis. By summing up local costs and by inspecting local widths of a MTBDD, we can arrive at global cost and width given in the following

Theorem 4.1 Cost C and width w of the MTBDD for function F_n: $(Z_2)^n \rightarrow Z_R$ are upper-bounded by

$$w \le \max_k \min(2^{n-k}, R^{2^k})$$
$$C \le \min_k (2^{n-k} + R^{2^k}) - R - 1, \quad (3)$$

where $k = 0, 1, \ldots, n-1$.

(Proof). The first relation follows directly from Lemma 3.1 and Def. 3.4, if we include sub-functions of k=0 variables (terminal values). In the case of cost C we must subtract constant sub-functions (nodes with a single output edge), see Lemma 3.1:

$$C = 1 + 2 + 4 + \ldots + 2^{n-(k+1)} + \\ + (R^{2^k} - R^{2^{k-1}}) + \ldots + (R^{2^2} - R^2) + (R^2 - R), \quad (4)$$

By computing the sum and taking the minimum we arrive at the total cost C:

$$C \le \min_k (\sum_{i=k+1}^{n} 2^{n-i} + \sum_{i=1}^{k} (R^{2^i} - R^{2^{i-1}}) = \min_k \left[(2^{n-k} - 1) + (R^{2^k} - R) \right],$$

QED.

Example 4.1 The profile of a general 4-valued function of 12 variables is according to Lemma 3.1 limited by
2, 4, 8, 16, 32, 64, 128, 256, 512, 256, 16, 4.
The MTBDD cost is

$C \le (1+2+4+\ldots +512) + (256-16)+(16-4) = 1275$ and width $w \le 512$. These bounds are too weak for real-life functions which have typically low values of w.

Random functions $(Z_2)^n \rightarrow Z_R$, $R = 2^r = 2^n$ are the most difficult functions to implement. Their MTBDDs have a form of the full binary tree and the number of all sub-functions is $W = C = 1+2+4+\ldots 2^{n-1} = 2^n-1$.

TABLE 1 Upper Bounds on MTBDD Cost

R ↓	n									
	1	2	3	4	5	6	7	8	9	10
2	1	3	5	7*	15*	29	45	77	141	269
4		3	7	15	27	43	75	139	267	507
8			7	15	31	63	119	183	311	567
16				15	31	63	127	255	495	751
32					31	63	127	255	511	1023
64						63	127	255	511	1023

Binary n-bit multipliers with $2n$ binary inputs and $2n$ outputs have R close to 2^{2n}. (End of Example)

The upper bounds for cost C for selected classes of multiple-output logic functions are summarized in Tab.1. They were calculated from (3), except for two items marked by asterisk which should have been 9 and 17, see Theorem 4.2 and Corollary 4.1 below. Separate regions in Tab.1 are interpreted as follows:
- the top region: minimum in (3) occurs at i = 2,
- the middle region: minimum in (3) occurs at i = 1,
- the bottom region: minimum in (3) occurs at i = 0.

Theorem 4.2 The cost of the BDD of the arbitrary logic function of 4 variables is $C \le 7$.

(Proof by construction) There are 65536 functions of 4 variables. Under the group of negations and permutations, we can reduce this count to only 222 equivalence classes. Now it is sufficient to prove the theorem for one representative out of each equivalence class. This was done by exhaustive search considering all 24 variable orderings. The upper bound 7 was reached in only 8 cases (the average node count was 4.6), QED.

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-184-7

Corollary 4.1 The cost of the BDD of the arbitrary logic function of 5 variables is $C \le 15$. (The first decision node can fork to two BDDs of two 4-variable sub-functions.)

The upper bounds on cost are too weak for most of functions in digital engineering practice. Very often the functions are defined only in a small fraction of all 2^n binary input vectors. Therefore, from now on, we will consider sparse logic functions and will attempt to obtain stronger upper bounds for them. Such bounds on local values of $w_{n\text{-}k}$, k = 2, 3,…,6 are known [6] for single output Boolean functions (R=2). Here we will analyze multiple-output Boolean functions and generalize the previous results.

Lemma 4.1 Let sparse function F_n: $(Z_2)^n \rightarrow Z_R$ attains non-zero values 1, 2, …, R-1 in $|X| = u << 2^n$ points, $X \subset (Z_2)^n$. Consider distinct k-variable sub-functions of F_n. Such sub-functions are specified by column vectors of $t = 2^k$ elements (rows). The maximum number of distinct column vectors is (5)

$$w_{n-k+1} = \lambda(t,u,R) = \sum_{i=0}^{\sigma} \binom{t}{i} (R-1)^i + q \text{ for } 0 < u < \sum_{i=1}^{t} i \binom{t}{i} (R-1)^i = u_m$$

where σ is the integer satisfying the relation

$$u_1 = \sum_{i=1}^{\sigma} i \binom{t}{i} (R-1)^i \le u \le \sum_{i=1}^{\sigma+1} i \binom{t}{i} (R-1)^i = u_2 \qquad (6)$$

and $q = \lfloor (u-u_1)/(\sigma+1) \rfloor$.

Note that $\lambda(u)$ is piece-wise linear, monotone increasing for $0 < u < u_m$. In the first interval $1 \le u \le t(R\text{-}1)$ the value of $\lambda(u) = u+1$. On the other hand, when $u \ge u_m$, $\lambda(u)$ takes up the constant value

$$\sum_{i=0}^{t} \binom{t}{i} (R-1)^i. \qquad (7)$$

Theorem 4.3 Let $\lambda(2^k, u, R)$ be the number of distinct k-variable sub-functions for an n-variable sparse function $(Z_2)^n \rightarrow Z_R$ and with weight u. Then

$c_{n\text{-}k+1} \le \lambda(2^k, u, R) - \varepsilon, \quad k = 1,..., n\text{-}1, \quad$ and $c_1 = 1$.

This theorem is immediately derived from Lemma 4.1: local cost $c_{n\text{-}k+1} \le w_{n\text{-}k+1}$, because of constant sub-functions. The upper limit on $c_{n\text{-}k+1}$ is the upper limit on $w_{n\text{-}k+1}$, i.e., $\lambda(2^k, u, R)$, decreased by ε, that is by the number of constant sub-functions of k variables for the given u and R. In general, if $u \ge u_m$, λ attains the saturated value $\lambda = R^{2^k}$ and we must subtract $\varepsilon = R^{2^{k-1}}$ constant sub-functions from λ to get the value of $c_{n\text{-}k}$, as seen from (4). However, if $u < u_m$ and $\lambda < R^{2^k}$, correction ε depends on u, = ε(u); we must always subtract 1 (all zeros pattern) and incidentally some other constant patterns if they appear in the range of u, QED.

Example 4.2 Let us have t = 2, R = 4, u = 10. All distinct sub-functions of a single variable are columns in the Table 2. Now, we can compare upper bounds on local costs c_n and values of $\lambda(2^1, u, 4)$ for some values of u.

For u = 10, we have λ= 9, but c_n = 8 only (all zero pattern does not count). For upper bound we have to consider the worst case when constant sub-functions are taken only if there is no other choice (see the last 3 columns in Table 2). So if u = 18 we get λ= 13 and c_n =12. This count does not grow any further, for u > 18 we get always c_n =12. (End of Example)

Local costs are functions of three parameters $t = 2^k$, u and R. With some computations according to Lemma 4.1, one can figure out the upper bound on the cost of any given sparse function.

Example 4.3 Cost profiles of sparse function of 13 variables, $(Z_2)^{13} \rightarrow Z_R$, R = 2, 4, 8, 16 and u = 100 are depicted in Fig. 2. For illustration, let us calculate $\lambda(t, u, R)$ for u = 100, R = 8 and t = 2, 4, 8, 16, …:

$t = 2: u_1 = 14 \le u \le 14 + 98 = 112 = u_2$,
$\lambda(2,100,8) = 1 + 14 + \lfloor (100-14)/2 \rfloor = 57,\ c_{13} = 56$

$t = 4: u_1 = 28 \le u \le 28 + 294 = 322 = u_2$,
$\lambda(4,100,8) = 1 + 28 + \lfloor (100-28)/2 \rfloor = 65,\ c_{12} = 63$

$t = 8: u_1 = 56 \le u \le 56 + 2744 = 2800 = u_2$,
$\lambda(8,100,8) = 1 + 56 + \lfloor (100-56)/2 \rfloor = 79,\ c_{11} = 78$

$t = 16,\ 32,\ 64$: $c_{10} = c_9 = c_8 = u = 100$.

The remaining local costs from c_7 to c_1 are 64, 32, 16, 8, 4, 2, 1. (End of Example)

TABLE 2 COMPUTATION OF $\lambda(t, u, R) = \lambda(2, 10, 4) = 9$

0	1	2	3	0	0	0	1	1	2	2	3	3	1	2	3
0	0	0	0	1	2	3	2	3	1	3	1	2	1	2	3
1	6								9						
←	u=10														→

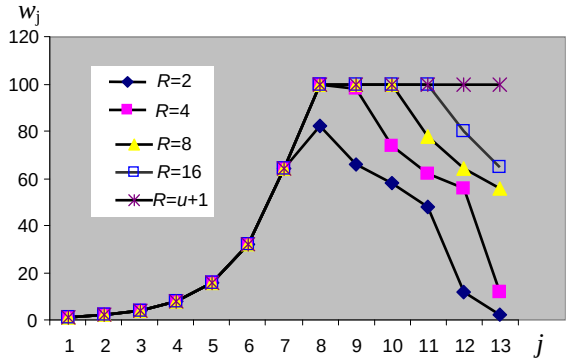

Figure 2 Profiles of sparse functions of 13 variables (u = 100)

From the inspection of Fig. 2, the following hypothesis can be formulated:

Hypothesis 4.1 The cost of the MTBDD for multiple-output sparse function F_n: $(Z_2)^n \rightarrow Z_R$, $R = 2^r$ and the memory area to store the MTBDD is much lower than the cost and memory area to store r BDDs for its r single-output component logic functions.

V. MAPPING MTBDDS TO BRANCHING PROGRAMS

Branching programs have typically two instructions: (multi-way) branch and output instruction. Their format depends on the architecture of a DDM as well as on the type of the DD. For a multi-output function, a partition into single output

functions (BDDs) or into groups of functions (multiple MTBDDs) has been used [3]. Due to our hypothesis we will use partition into groups of functions. Our starting point will be the MTBDD with sub-optimal ordering of variables obtained by heuristic [7], that can be easily converted to a 2^k-valued DD with generally non-uniform k (heterogeneous DDs). The architecture of a suitable DDM is in Fig. 3.

The code memory stores instructions that evaluate nodes of the DD. Each node is represented by a 2^k-way dispatch table that starts at a node base address and its items are indexed by k-bit offset. Each item contains a code for input multiplexers (group id) and the mask which together specify the offset for the next node. Fig. 4 shows two instruction formats. The multi-way branch instruction evaluates a non terminal 2^k-ary node, while the output instruction evaluates a terminal node:

1) the jump to an address in the PC modified by BCU;
   L_n: branch $L_m@x_1...x_k$;
2) output and the unconditional jump to the specified address
   L_n: output, go to L_m .

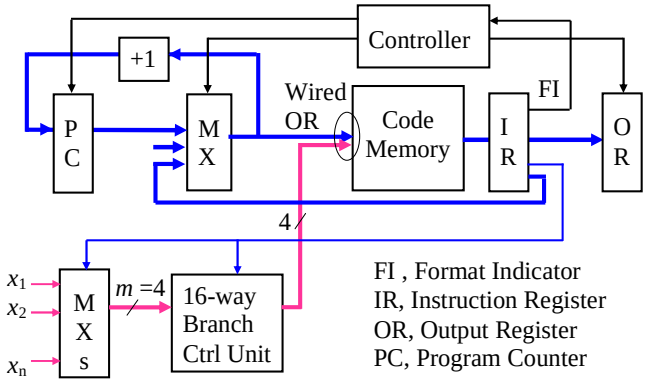

Fig. 3 Architecture of Decision Diagram Machine (DDM)

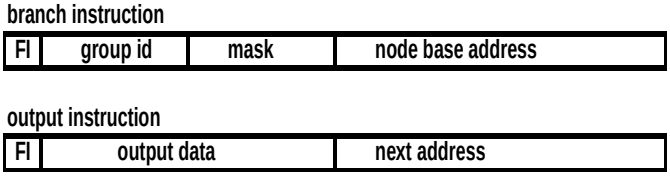

Fig. 4 Instruction formats for a DDM with the support for multi-way branching

The multi-way indirect branch is executed in 1 clock cycle, the current base address in the PC gets modified by external variables (operator @), by up to 4 variables at a time, including 0 variable (no modification, the unconditional jump), by means of 16-way Branch Control Unit (BCU). Input variables are selected by multiplexers, so that instructions contain MXs control field and a BCU mask. The task of the 16-way BCU4 is to shift up to 4 active inputs, selected by a 4-bit BCU mask, to the lowest positions of the 4-bit output vector and reset the rest of outputs. The output vector then serves as an offset from the base address of a dispatch table. This way the dispatch tables can be stored in code memory in a compact form; the bits of the base address supplied by the BCU must be reset to 0 if wired-OR is used for modification. Tri-state outputs of the BCU are wire-ORed to the address inputs of the code memory.

The advantages of above architecture are:

- the word lengths for the multi-way branches are the same
- relatively short word lengths for instructions (single base address only)
- easy extension for other instructions and addressing (incrementing PC for longer output sequences, support for a return address stack, etc.)

Example 5.1 Let us implement the Round Robin Arbiter (RRA) with 4 input requests r_0, r_1, r_2, r_3. The priority register [p_0, p_1, p_2, p_3] points to the requester i, currently with the highest priority (one-hot encoding). Priority decreases for subsequent inputs:

$$g_3 = p_3r_3 + p_0!r_0r_3 + p_1!r_1!r_0r_3 + p_2!r_2!r_1!r_0r_3$$
$$g_2 = p_2r_2 + p_3!r_3r_2 + p_0!r_0!r_3r_2 + p_1!r_1!r_0!r_3r_2$$
$$g_1 = p_1r_1 + p_2!r_2r_1 + p_3!r_3!r_2r_1 + p_0!r_0!r_3!r_2r_1$$
$$g_0 = p_0r_0 + p_1!r_1r_0 + p_2!r_2!r_1r_0 + p_3!r_3!r_2!r_1r_0.$$

The quaternary MTBDD for this RRA obtained by means of HIDET tool [2] is at Fig. 5. The sample of a branching program with inspection of two binary inputs at a time is shown at Fig. 6. The symbolic program is composed of 9 4-way and of 2 2-way dispatch tables. The base addresses of dispatch tables shown in Fig. 6 as L1 to L11 correspond to the same labels in the MTBDD in Fig. 5. The total number of instructions is

$9\times4 + 2\times2 = 40.$ (End of example)

VI. BRANCHING PROGRAM OPTIMIZATION

The most important parameters that are usually subject of optimization are memory size, execution time, and power consumption. Since code memory occupies the most area for the whole DDM, we assume that the area is proportional to the memory size. Area-time complexity, or the product of memory size and performance, is important for embedded systems. In this section, we will focus on optimizing the area-time product only. As a by-product, a processing with low area (time) means low dissipation of static (dynamic) power, too.

There are two possibilities for optimization, ordering of input variables (not discussed in this paper) and their grouping. Whereas variable ordering influences the size of a MTBDD and required code memory, grouping of input variables impacts the speed of processing. Testing several input variables simultaneously can also be visualized as converting the MTBDD into a multi-valued DD (MVDD), [8] – [9]. Very often the nodes of such MVDD are degenerated in a certain degree, i.e., not all the output edges are distinct. The DDM architecture at Fig. 3 can utilize this fact for minimization of memory requirements; it can vary the number of variables tested in each step according to the local structure of the MTBDD on the followed path. For example if we test

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-184-7

three variables at a time, the complete tree of 7 nodes could be converted to a single 8-valued node. However, there are

$$\sum_{i=0}^{7} C(7,i) = 2^7$$

possible configurations of i = 0, 1, …. 7 degenerate nodes and if they occur in the same level of the diagram, we can skip their testing and reduce the size of a dispatch table. Fig. 7 shows all possible sub-graphs rooted at a local node subject to such reduction. An extreme case is that there are no true decision nodes on the path and a dispatch table is eliminated completely. If we test k variables at a time, then there are

$$\sum_{i=0}^{k-1} C(k,i) = 2^k - 1$$

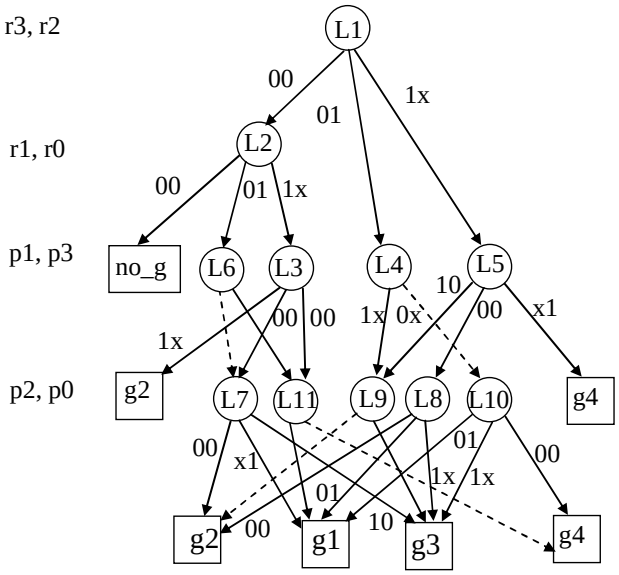

Fig. 5 MTBDD of the 4-input RR arbiter.

```
RRA:        exit L1@r3r2
L1@00:      exit L2@r1r0
L1@01:      exit L4@p1p3
L1@10:      exit L5@p1p3
L1@11:      exit L5@p1p3
L2@00:      no_g exit Next
L2@01:      exit L6@p3
L2@10:      exit L3@p1p3
L2@11:      exit L3@p1p3
        …..
L10@00:     g4 exit Next
L10@01:     g1 exit Next
L10@10:     g3 exit Next
L10@11:     g3 exit Next
L11@0:      g4 exit Next
L11@1:      g1 exit Next
Next:
```

Fig. 6 A symbolic microprogram for the RRA.

sub-graphs leading to dispatch tables of reduced size. Simplifying nodes with 2^k outputs wherever possible leads to a heterogeneous MVDD with nodes controlled by k or less variables.

Example 6.1 Continuing in Example 4.1, we can apply various grouping of input variables and then reduction of multi-valued nodes to the smaller ones. If we create groups of 2 input variables, we will do with 7 4-way nodes (7 dispatch tables of size 4) and 4 binary nodes (4 dispatch tables of size 2), altogether 36 instructions. Had we used only single variable tests (a binary program with 2-way branching), we would need 17 dispatch tables of size 2, i.e., 34 instructions in total. However, the performance would be 2- times lower due to execution of a chain of 8 instructions, one in each level of the MTBDD. Processing in three steps could test 2, 3, 3 or 2, 2, 4 decision variables, but the area-time product would get worse. The fastest execution tests 4 decision variables at a time (16-way branching). The features of various options are summarized in Table 3. The area × time product is a figure of merit of quality of the implementation. It gets its best (lowest) value for testing two and four variables at a time. (End of Example)

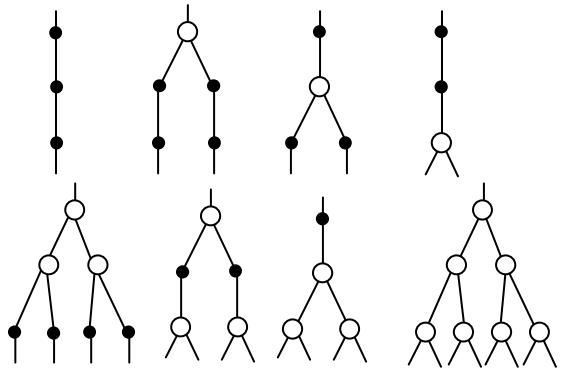

Fig. 7 MTBDD sub-graphs with 0, 1, 2 and 3 decisions on each path

TABLE 3 VARIOUS MICROPROGRAM OPTIONS

tested variables:	total micro-instructions	execution time	space x time
8 x 1	34	8	272
4 x 2	36	4	144
2, 3, 3	52	3	156
2, 2, 4	64	3	192
2 x 4	72	2	144
8	256	1	256

Similar optimization has been carried out by a home-made software tool for a number of logic modules such as branch control units (bcu), round robin arbiters (rra), least-recently-served arbiters (lrs) and priority encoders (pe) with a various number of inputs.

MTBDDs of these logic modules have been obtained and optimized by HIDET tool from cube specification [2].

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-184-7

MTBDD parameters (cost *c*, size *s*, and width *w*) for these modules are listed in Table 4, together with optimal grouping of input variables ordered by HIDET and resulting area × time (a×t) product. This product is calculated as the aggregate number of all instructions in dispatch tables (a dispatch table for a *k*-ary node has 2^k instructions) multiplied by the number of dispatch tables from the root to leaves.

The results show that the best area-time complexity is obtained for $k = 3$ or 4 variables tested simultaneously. This does not corresponds to the result in [6], where quaternary ($k = 2$) DDs were found best with respect to this figure of merit in a different set of benchmarks. The reason for this may be not only in benchmarks, but also in our different DDM architecture supporting variable multi-way branching.

VII. CONCLUSIONS AND FUTURE WORKS

In this paper, we have proposed a new MTBDD machine that could outperform known DDM architectures. The size of the branching programs and the maximum required code memory for this machine can be estimated using derived upper bounds on the cost of MTBDD for logic functions specified in *u* input vectors and attaining only a single value (0) for other input vectors. The bounds are not limited to sparse functions with $u << 2^n$, but are valid generally. The case of incomplete functions can be treated similarly, replacing don´t - cares by a dominant function value (0).

TABLE 4 OPTIMUM BRANCHING PROGRAM CONFIGURATIONS

	n	m	s	c	w	groups	a×t
bcu4	8	4	45	30	16	2x4	160
bcu6	12	6	189	126	64	3x4	1008
bcu7	14	7	381	254	128	4,4,4,2	2368
bcu8	16	8	765	510	256	4x4	5440
rra4	8	4	37	17	8	2x4	144
rra6	12	6	91	40	11	3x4	438
rra8	16	8	179	75	22	4x4	1248
rra12	24	12	489	189	44	8x3	4688
lrs4	10	4	39	17	6	3,3,3,1	168
lrs6	21	6	119	36	9	7x3	742
pe8	8	4	8	8	2	4x2, 2x4	64
pe12	12	5	12	12	2	4x3	128
pe16	16	5	16	16	2	8x2, 4x4	256

Firmware implementation of a MTBDD is usually a matter of trade-off between performance and the size of memory storing the code. The memory size can be derived as an aggregate size of all dispatch tables, and the performance is given by the number of dispatch tables on the path from the root to leaves of the MTBDD. The area-time complexity has been optimized for the MTBDD machine that supports multi-way branching in one clock cycle with variable number of ways (0 to 16). On the given set of benchmarks the optimum area-time product has been reached for DDM for *k*-ary nodes with $k = 3$ and 4, what is in contradiction to finding QDD ($k = 2$) as optimal in [3].

Future research should compare the MTBDD Machine to other published DDMs on the common set of benchmarks and thus verify hypothesis 4.1 and implied superiority of MTBDD Machines. The library of optimal MTBDDs for a such a benchmark suite should be created first, it is not available as yet. Another optimization problem is to pack dispatch tables of all MTBDD nodes into as small memory as possible. The final step would be a hardware implementation of the MTBDD machine and also of its parallel version in FPGA.

ACKNOWLEDGMENT

This research has been carried out under the financial support of the research grants GP103/10/1517 “Natural Computing on Unconventional Platforms”, and MSM 21630528 “Security-Oriented Research in Information Technology” and the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070).

REFERENCES

[1] T. Sasao, Memory-Based Logic Synthesis. Springer, New York, 2011, 189 pages.

[2] V. Dvořák and P. Mikušek, "Design of Arbiters and Allocators Based on Multi-Terminal BDDs". In: Journal of Universal Computer Science, Vol. 16, No. 14, 2010, AT, pp. 1826-1852.

[3] H. Nakahara, T. Sasao, and M. Matsuura, "A comparison of architectures for various decision diagram machines," International Symposium on Multiple-Valued Logic, Barcelona, Spain, May 26-28, 2010, pp. 229-234.

[4] H. Nakahara, T. Sasao, M. Matsuura, and Y. Kawamura, "A parallel branching program machine for sequential circuits: Implementation and evaluation," IEICE Transactions on Information and Systems, Vol. E93-D, No. 8, pp. 2048-2058, Aug. 2010.

[5] H. Nakahara, T. Sasao, and M. Matsuura, "Packet classifier using a parallel branching program machine," 13th EUROMICRO Conference on Digital System Design (DSD-2010) Lille, France, Sept. 1-3, 2010, pp. 745-752.

[6] T. Sasao, "On the number of LUTS to realize sparse logic functions," 18th International Workshop on Logic and Synthesis, (IWLS-2009), Berkeley, CA, U.S.A., July 31-Aug. 2, 2009, pp. 64-71.

[7] P. Mikušek and V. Dvořák, "On Lookup Table Cascade-Based Realizations of Arbiters". Proc. of the 11th EUROMICRO Conference on Digital System Design DSD 2008, Parma, IT, IEEE CS, pp. 795-802.

[8] S. Nagayama and T. Sasao, "On the optimization of heterogeneous MDDs," IEEE Transactions on CAD, Vol. 24, No.11, Nov. 2005, pp.1645-1659.

[9] H. Nakahara, T. Sasao, and M. Matsuura, "A Comparison of heterogeneous multi-valued decision diagram machines for multiple-output logic functions," International Symposium on Multiple-Valued Logic (ISMVL-2011), Tuusula, Finland, May 23-25, 2011.

[10] T. Sasao, H. Nakahara, K. Matsuura, Y. Kawamura, and J.T. Butler, "A quaternary decision diagram machine: Optimization of its code," IEICE Transactions on Information and Systems, Vol. E93-D, No. 8, pp. 2026-2035, Aug. 2010.

[11] V. Dvořák and P. Mikušek, "On the cascade realization of sparse logic functions". In: Euromicro Proceedings, Oulu, FI, IEEE CS, 2011, pp. 21-28.

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-184-7