# Implementation of Combinational and Sequential Functions in Embedded Firmware

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

***Abstract***

*The paper addresses firmware implementation of multiple-output combinational and sequential Boolean functions based on cascades of Look-Up Tables (LUTs). A LUT cascade is described as a means of compact representation of a large class of Boolean functions, which reduces their evaluation to multiple indirect memory accesses. A LUT-oriented decomposition technique is illustrated on several examples. A specialized micro-engine is proposed for sequential processing of LUT cascades by means of multi-way branching. The presented method provides high performance micro-programmed control for embedded applications.*

## 1. Introduction

Efficient evaluation of Boolean functions is an important part of many embedded firmware or software systems. Application-specific functions most frequently used in embedded systems practice have typically low complexity. They include applications such as encryption, data compression and conversion, pattern matching and searching, sliding window functions on data streams, etc. We will address Boolean functions of large numbers (tens, hundreds) of variables because small size systems can be implemented directly in hardware, e.g. in various PLDs, PLAs, ROMs or TCAM (Ternary Content Addressable Memory).

Firmware implementation of Boolean functions will be assumed in a form of data structures describing the function and of a micro-program that reads the input vector and evaluates the function with the use of this data structure. The size of the code and of the data structure is one figure of merit; another one is the evaluation time from reading the input to generating the output.

Hereafter we will use two complementary representations: Look-Up Tables (LUTs) and binary decision diagrams (BDDs). The BDDs are well known, especially the reduced ordered BDDs (ROBDDs), [1]. On the base of ROBDDs we will develop a more practical representation – cascades of LUTs.

Firmware implementation of Boolean functions has been up to now studied especially in connection with programmable logic controllers - PLCs ("ladder diagrams") or specialized event processing, where either a speed (PLC) or a required memory were not that important. On the contrary, in embedded systems we do care for performance, memory space as well as for power consumption. We will demonstrate that presently used algorithms (binary programs, BDD traversal or sequential evaluation of Boolean expressions) are generally too slow and that the use of LUT cascades enables faster evaluation. The longer cascades with simpler LUTs are slower than shorter cascades with larger LUTs, and thus the processing speed can be even adjusted to requirements.

The idea of using a specialized micro-engine for sequential processing of LUT cascades was conceived in [2]. In the present paper we use a modified micro-engine architecture based on micro-program sequencer (Am 2910) and its multi-way branch control unit (Am 29803A), that can be easily implemented in FPGA. In the meantime a different architecture under the name LUT ring was developed and implemented in VLSI technology [3] from the scratch. However, it is more complicated and in some way less general (the use a barrel shifter instead of the branch control unit).

The paper is structured as follows. In the following Section 2 we introduce basic notions and terminology concerning Boolean functions and their representation. Binary decision diagrams (BDDs) and LUT cascades are reviewed in Section 3, and the way how to obtain the LUT cascade for a Boolean function is given in Section 4. A micro-engine for sequential LUT cascade processing is presented in Section 5 with illustration of trade-offs between speed of evaluation and required memory space. Obtained results, some generalizations and future research are commented on in Conclusions.

## 2. Basic notions and terminology

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

$$f_{\mathrm{n}}^{(\mathrm{i})} : (Z_2)^{\mathrm{n}} \to Z_2\,,\ \ i = 1, 2, ..., m \tag{1}$$

will be simply referred to as multiple-output Boolean function $F_{\mathrm{n}}$ with output values from $Z_{\mathrm{R}} = \{0, 1, 2, \ldots, R\text{-}1\}$,

$$F_{\mathrm{n}}: (Z_2)^{\mathrm{n}} \to Z_{\mathrm{R}}\,, \tag{2}$$

where $R$ is the number of distinct combinations of $m$ output binary values enumerated by values from $Z_{\mathrm{R}}$. Function $F_{\mathrm{n}}$ is incomplete if it is defined only on set $X \subset (Z_2)^{\mathrm{n}}$; $(Z_2)^{\mathrm{n}} \setminus X = D$ is the don't care set.

The behavior of a combinational circuit can be described by the system of $m$ complete functions of $n$ variables

$$y_{\mathrm{i}} = f_{\mathrm{n}}^{(\mathrm{i})}(x_1\,, x_2\,, \ldots, x_{\mathrm{n}}), \qquad i = 1, 2, ..., m \tag{3}$$

or $\boldsymbol{y} = \boldsymbol{F}(\boldsymbol{x})$ in vector notation.

Computer representation of Boolean functions uses binary decision diagrams (BDDs), which can have many forms. Bit-level binary decision diagrams (BDDs), ordered binary decision diagrams (OBDDs) and reduced ordered binary decision diagrams (ROBDDs) are the best known representations of a single Boolean function in a form of a directed acyclic graph [1]. The ROBDD is a canonical (unique) representation for any given complete function and for a given order of variables.

Important parameters of a BDD are its size and width, i.e. the total number of decision nodes and the maximum number of edges between adjacent levels, where the edges pointing to the same node are counted as one. The size determines the memory space needed to store the BDD data structure while the width $K$ (also a C-measure, [11]) determines a BDD form factor since the height is given by the number of variables. The construction of minimum-size or by the same token minimum-width ROBDDs belong among NP-complete problems [4]; the size and width of the ROBDD depend on variable ordering and there are $n!$ possible orderings of $n$ variables. A heuristic approach can be used in a search for near-optimal orderings [5]. Upper bounds on the OBDD's size and width for general random complete Boolean functions grow exponentially with number of variables $n$ for any ordering, but functions used in digital systems design with few exceptions do have a reasonable BDD size and small width.

To represent a system of Boolean functions (1) by means of decision diagrams, we can use either $m$ bit-level BDDs, one for each of $m$ Boolean functions (possibly sharing some of their sub-diagrams in Shared BDDs or SBDDs, [6]), or one word-level BDD (WLBDD) with $n$ Boolean decision variables and with $R$ integer terminal values [7]. Out of many types of WLBDDs we will use Multi-Terminal BDDs (MTBDD) which represent functions from Booleans to integers.

As the LUT cascades are the main concern of this paper, we will provide a formal definition. A LUT will be also interchangeably referred to as a "cell".

Def. 1. A cascade of a form $k \times m$ is the system of $B$ cells with $k$ horizontal rails and $m$ vertical cell inputs supporting $K \leq 2^k$ ($M \leq 2^m$) Boolean input vectors. Individual cells implement functions

$$H_i\colon Z_2^k \times Z_2^m \rightarrow Z_2^k,\ 1 \leq i \leq B.$$

The last cell in the cascade may have $r \neq k$ outputs.

Def. 2. A cascade is said to be non-redundant if each variable used at vertical input enters one and only one cell. Otherwise the cascade is redundant.

## 3. MTBDDs and LUT cascades

Whereas BDDs and MTBDDs proved useful in many areas of digital design [7] where they provide compact data structures and a degree of flexibility in manipulating them, they are not as useful for the purpose of function evaluation. The primary reason is the slow speed, since the evaluation by branching program inspects one Boolean variable at a time. There is though a certain speedup in comparison to direct evaluation of Boolean expressions, because each variable is processed only once. Straightforward remedy how to speed up the traversal of a BDD is to process several variables at a time. This way we will derive LUT cascades, in fact a special case of LUT networks.

A close relation between both these representations of multiple-output Boolean functions will be illustrated on a bit-counting example. The combinational function $F_n$: $(Z_2)^n \rightarrow Z_{n+1}$ gives the number of 1´s presented at $n$ inputs in a form of a binary number. The MTBDD and associated LUT cascade are displayed in Fig. 1 for $n = 4$.

Generalization for larger values of $n$ is easy. As the number of nodes grows linearly from the root to leaves, the width of the MTBDD is given by the last level of decision nodes and has the value of $K = n$.

What connects two representations is the concept of sub-functions. Informally, the sub-function $f$ of $F_n$ is a function of $s$ variables obtained from $F_n$ by setting $n - s$ variables to fixed constant values. The number of distinct sub-functions of $s$ variables, $s$ = 1, 2,…, n-1, a so called profile, characterizes the Boolean function and its complexity. In Fig.1 we can recognize distinct sub-functions as edges crossing boundaries between MTBDD layers, counting edges incident with the same node only once. Edges are labeled by ID codes of distinct sub-functions. From the top down, there are 2 sub-functions of variables *a*, *b*, *c* (ID codes 0, 1), 3 sub-functions of variables *a*, *b* (ID codes 0, 1, 2), 4 sub-functions of variable *a* (ID codes 0, 1, 2, 3), and 5 sub-functions of zero variables (constant terminal values 0 to 4). LUT contents are defined as input/output pairs, where inputs are binary ID codes and a value of a side variable entering a cell and outputs are binary ID codes generated by the cell. Co-synthesis of MTBDD and LUT cascade can be done for small problems by hand (as illustrated later) and for large incomplete functions by a program tool [8].

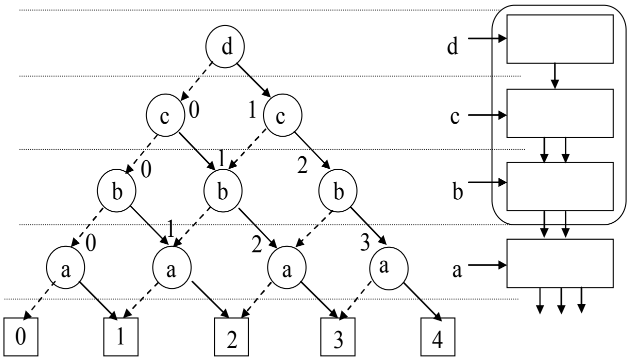


**Fig.1.** Bit counting example

As can be seen, the difference between the MTBDD and the LUT cascade is in communication among the MTBDD layers and LUTs in the cascade: in a MTBDD each sub-function ID code requires an individual edge (”one-hot” coding), whereas the ID codes being sent between LUTs are binary coded. The number of rails *k* in the cascade (a cascade “width”) is therefore

$$k = \log_2 \lceil K \rceil. \quad (4)$$

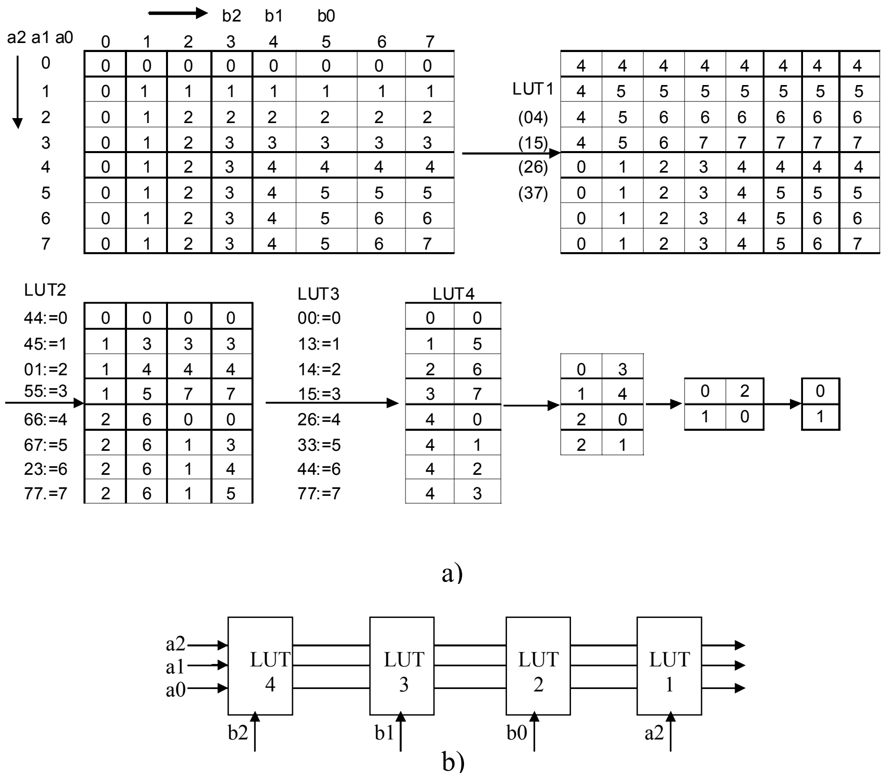
This difference of two representations reflects itself in the way how the program interprets a certain application-specific MTBDD or a LUT cascade. In case of the MTBDD we may use for each node a record with 3 fields. A format indicator is one-bit field specifying the leaf node (leaf nodes may generally occur at any level of the diagram). Two other fields of the leaf node are then used for an output. If the node is not a leaf, two fields (adjacent words) contain pointers to the base addresses of other nodes. The base address is then modified by the value of a current control variable(s) and is used to extract the correct field with the pointer to the next node. The program traverses a certain path in the MTBDD from the root to a leaf in at most *n* steps.

LUTs are interpreted similarly, only the pointer to the next LUT is obtained from the current output by concatenating it with the control variable value and adding it up to the next LUT base address. If suitable, some LUTs can be combined to provide even faster processing (see first three cells in Fig.1).

## 4. LUT cascades synthesis by iterative decomposition

The decomposition of the multiple output Boolean function (or a combinational part of a sequential system) can be done by identifying distinct sub-functions in the original function. If their count is slightly above a power of two, we can first try to make it equal or less than that value by transforming the function and resulting in a narrower cascade. Then the iterative decomposition removes one variable from the residual functions at a time. We will stop when the desired number of remaining variables for the first LUT is obtained.

We will consider the following combinational function: from two n-bit binary numbers on inputs the smaller one should be passed on to the output. For simplicity we will take n = 3 and compare numbers (a2 a1 a0) and (b2 b1 b0). The full function table is at the top of Fig. 2a. Sub-functions of b0 are the pairs of horizontally adjacent integers in the function table. The pairs of different integer values represent proper sub-functions, whereas pairs of the same integer values are constant sub-functions. Since the number of single-variable sub-functions is greater than 8 for any variable, we will do a permutation (04)(15)(26)(37) in the upper half of the table (for a2 = 0). By means of this permutation the number of sub-functions of b0 becomes 8 and the cascade width 3 rails will do. Enumeration of sub-functions of b0 is done by giving each and every distinct sub-function a new ID from 0 to 7. This way a variable b0 will not appear in the residual function.



| a2 a1 a0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 0 | 1 | 2 | 3 | 3 | 3 | 3 | 3 |
| 4 | 0 | 1 | 2 | 3 | 4 | 4 | 4 | 4 |
| 5 | 0 | 1 | 2 | 3 | 4 | 5 | 5 | 5 |
| 6 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 6 |
| 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

|  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
|  | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| LUT1 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| (04) | 4 | 5 | 6 | 6 | 6 | 6 | 6 | 6 |
| (15) | 4 | 5 | 6 | 7 | 7 | 7 | 7 | 7 |
| (26) | 0 | 1 | 2 | 3 | 4 | 4 | 4 | 4 |
| (37) | 0 | 1 | 2 | 3 | 4 | 5 | 5 | 5 |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 6 |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| LUT2 |  |  |  |  |
|---|---|---|---|---|
| 44:=0 | 0 | 0 | 0 | 0 |
| 45:=1 | 1 | 3 | 3 | 3 |
| 01:=2 | 1 | 4 | 4 | 4 |
| 55:=3 | 1 | 5 | 7 | 7 |
| 66:=4 | 2 | 6 | 0 | 0 |
| 67:=5 | 2 | 6 | 1 | 3 |
| 23:=6 | 2 | 6 | 1 | 4 |
| 77.=7 | 2 | 6 | 1 | 5 |

| LUT3 | LUT4 |  |
|---|---|---|
| 00:=0 | 0 | 0 |
| 13:=1 | 1 | 5 |
| 14:=2 | 2 | 6 |
| 15:=3 | 3 | 7 |
| 26:=4 | 4 | 0 |
| 33:=5 | 4 | 1 |
| 44:=6 | 4 | 2 |
| 77:=7 | 4 | 3 |

|  |  |
|---|---|
| 0 | 3 |
| 1 | 4 |
| 2 | 0 |
| 2 | 1 |

|  |  |
|---|---|
| 0 | 2 |
| 1 | 0 |

|  |
|---|
| 0 |
| 1 |

b)

**Fig.2.** Redundant iterative decomposition a) and an associated LUT cascade b)

Note that we have started building the cascade from the LUT 1 at the end, Fig.2b. Repeating the decomposition for variable b1, we will obtain a residual function of variables a2, a1, a0, and b2, in fact LUT4. Next three decomposition steps shown in Fig. 2a are not needed. Note that the LUT cascade in Fig. 2 is a redundant one.

Design of LUT cascades by iterative decomposition (or alternatively by slicing MTBDDs as in Fig.1) has a catch: the size and the width of MTBDD strongly depends on variable ordering. Optimum variable ordering is, however, a separate problem. Recently, heuristic minimization algorithms have been proposed [7] that allow reduction of the WLDD size analogously as for BDDs. A co-synthesis of both MTBDD and LUT cascade for incompletely specified multiple-output Boolean functions has been developed in [9].

There are other heuristic approaches for MTBDD optimization, e.g. a sifting method or the application specific variable ordering (ASVO) [7]. For example in sifting method all positions of a given variable in the given ordering are checked successively. The variable is then left in an optimal position with the lowest MTBDD size and the process repeats for all variables. Thorough comparison of all heuristic methods of optimization, as regards quality of results and an amount of the required execution time, remains still to be done.

## 5. A micro-programmed controller with multi-way branching

Evaluation of Boolean functions at the firmware level can use the LUT cascade paradigm. By making use of hardware micro-engines with a support for multi-way branching, we can speed up evaluation of Boolean functions with respect to a general purpose CPU core. A suitable architecture of a micro-engine, a modified version of the one in [2], is depicted in Fig.3.

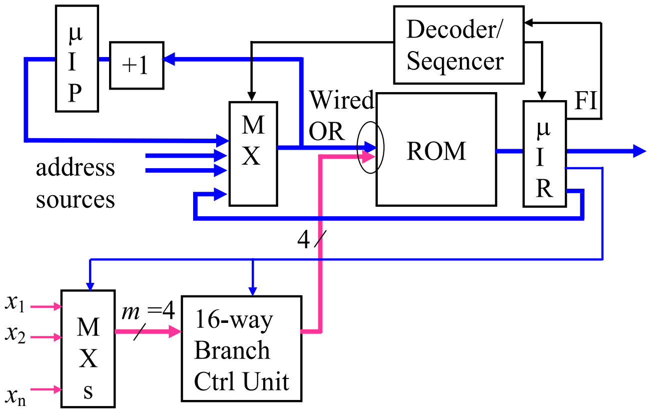


**Fig.3.** Micro-programmed controller architecture with multi-way branching

There are three microinstructions formats determined by a format indicator field FI:

FI = 01: state output (control signals), μIP := μIP+1

FI = 10: MXs and BCU control, jump to an address specified in micro-instruction modified by BCU

FI = 11: conditional output and jump to an address specified in micro-instruction (no modification).

The second format includes all kinds of 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 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. If there are more external variables, LUT cascade paradigm is used. The LUT output contains not only the rail variables, but the whole address of one of the input nodes in the next section of the MTBDD encapsulated in the next LUT.

We will illustrate the transformation of a general multi-way branch microinstruction into a micro-program. The multi-way branch has the same structure as a switch. Let us have the statement

S0: if F = 0 then v0 exit S0
if F = 1 then v1 exit S1
if F = 2 then v2 exit S1
if F = 3 then v2 exit S2
if F = 4 then v3 exit S3
else don´t care; (5)

Si´s are state labels, vj´s are conditional output vectors, $F(A,B,C,D)$: $X \rightarrow Z_5$, $X \subset (Z_2)^4$ is an incomplete multiple-output Boolean function, its map is in Fig.4a. The switch statement (5) describes a transition from present state *S*0 to one of next states *S*0 to *S*3 depending on the values of 4 external variables *A*, *B*, *C* and *D*. During the transition a certain conditional output vector vj is generated.

*F*(*A*,*B*,*C*,*D*) *CD* →

| *AB* | 00 | 01 | 10 | 11 |
|---|---|---|---|---|
| 00 | 0 | 1 | 2 | 2 |
| 01 | x | 2 | 2 | 2 |
| 10 | x | 4 | 0 | 1 |
| 11 | 4 | x | 2 | 3 |

S0: if F = 0 then v0 exit S0
if F = 1 then v1 exit S1
if F = 2 then v2 exit S1
if F = 3 then v2 exit S2
if F = 4 then v3 exit S3
else don´t care;

```
S0 exit L@ABCD
...
L@0000 v0 exit S0
L@0001 v1 exit S1
...
...
L@1110 v2 exit S1
L@1111 v2 exit S2
```

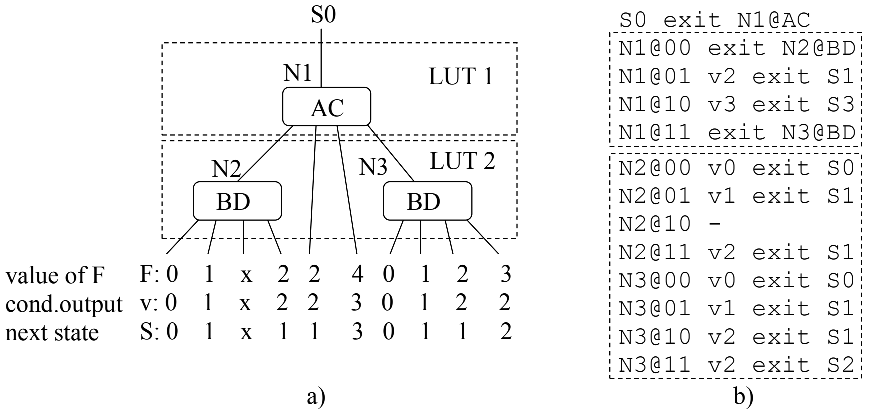
a) b) c)

**Fig.4.** The map of a sample function (a), a switch statement (b), and a dispatch table in a form of the symbolic micro-program (c)

If the speed of the micro-engine is the utmost priority, we should do the testing of external variables in one step. The 16-way branch is then translated to the dispatch table in Fig.4c.

Replacement of 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.

If saving in hardware (chip area) is more important than overall speed, we can test variables *A*, *B*, *C* and *D* in groups of two. The optimum MTBDD found by the iterative decomposition is shown in Fig. 5a, together with the symbolic micro-program derived from it (Fig.5b). It can be seen that the second LUT is only partial as two sub-functions of two variables *A*, *C* are constants (2 and 4). Control store capacity is almost half of the capacity in the previous case and also the BCU can be simplified.


S0
N1
LUT 1
AC
N2
N3
LUT 2
BD
BD
value of F F: 0 1 x 2 2 4 0 1 2 3
cond.output v: 0 1 x 2 2 3 0 1 2 2
next state S: 0 1 x 1 1 3 0 1 1 2
a)
b)

```
S0 exit N1@AC
N1@00 exit N2@BD
N1@01 v2 exit S1
N1@10 v3 exit S3
N1@11 exit N3@BD
N2@00 v0 exit S0
N2@01 v1 exit S1
N2@10 -
N2@11 v2 exit S1
N3@00 v0 exit S0
N3@01 v1 exit S1
N3@10 v2 exit S1
N3@11 v2 exit S2
```

**Fig. 5.** LUT cascade (a) and the symbolic micro-program (b) for a multiway branching example

## 6. Examples

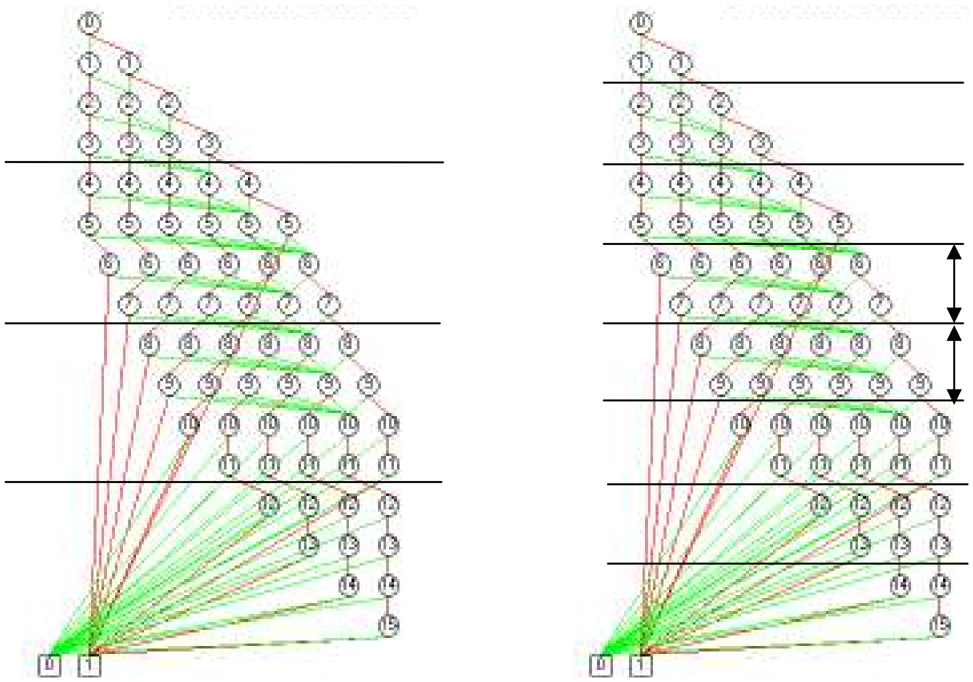
As the first example we shall consider evaluation of the following Boolean function of 16 variables: it attains the value 1 if the given 6-bit string is detected anywhere within an input string of 16 Boolean values; otherwise the function has the value 0.

Since the string of 6 consecutive values of variables may be located in 11 positions (we do not assume that the pattern wraps around), we can specify the function by the sum of 11 products. The CPU evaluation of Boolean expressions would take in the worst case 11 × 6 steps, whereas a traversal of the ROBDD would need 16 steps. We can do much better with LUTs, though. First the ROBDD of this function may be obtained using the applet [10], since the Boolean expression with 11 terms, each with 6 literals, is easy to write. For the pattern of six 1´s we have:

a1*a2*a3*a4*a5*a6+a2*a3*a4*a5*a6*a7+a3*a4*a5*a6*a7*a8+a4*a5*a6*a7*a8*a9+
a5*a6*a7*a8*a9*a10+a6*a7*a8*a9*a10*a11+a7*a8*a9*a10*a11*a12+a8*a9*a10*a11
*a12*a13+a9*a10*a11*a12*a13*a14+a10*a11*a12*a13*a14*a15+a11*a12*a13*a14*
a15*a16 (6)

The ROBDD is in Fig.6a, from which an optimal size and count of LUTs can be determined. We have used 4 ROBDD slices in Fig. 6a and obtained LUTs with 3 rails and 4 vertical inputs (Fig. 6c) for the target micro-controller architecture in Fig.3.

The size of the micro-program is determined by the total number of inputs into all LUTs, which is 1 + 5 + 6 + 4 = 16; each of these inputs corresponds to the block of 16 microinstructions – a dispatch table for 4 variables. The total number of jump micro-instructions is thus 16 × 16 = 256, but only up to 4 of them would be executed for the given input vector. The pattern detection time will be equal to the execution time of 4 jump micro-instructions.



a)

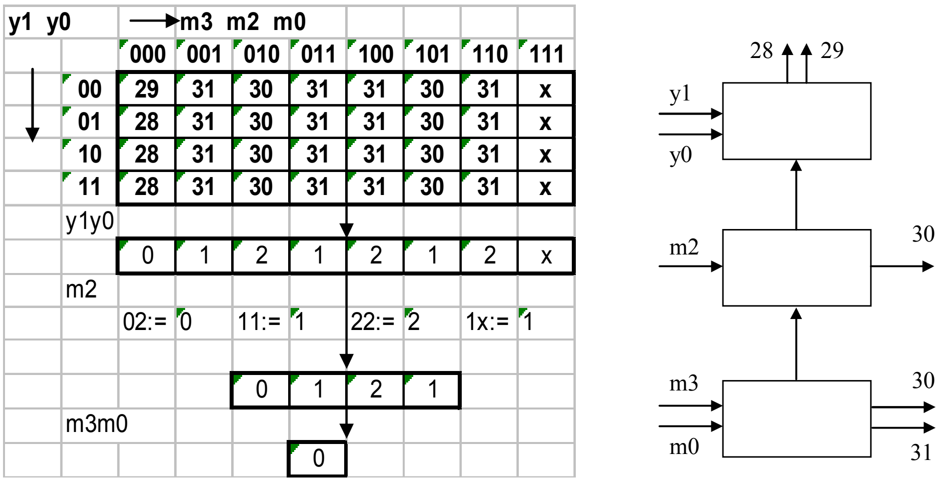
b)

c)

**Fig. 6.** The LUT cascade detecting a 6-bit string in 16 bits (a) 4-bit slices of the ROBDD (b) 2-bit slices of the ROBDD (c) related LUT cascades

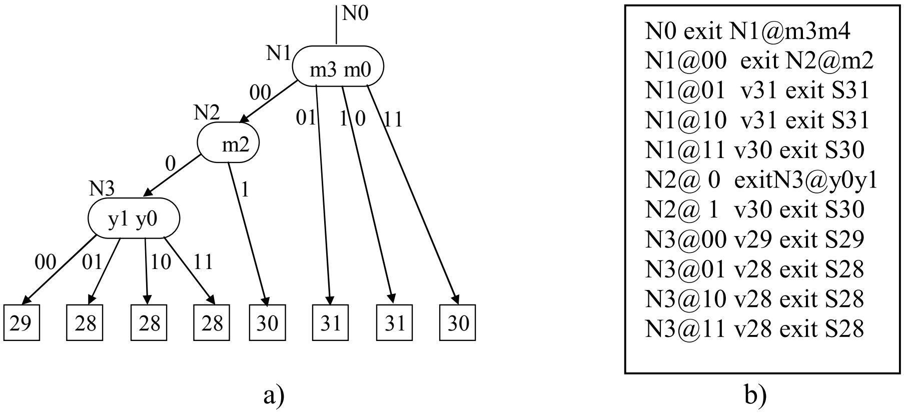
We can trade execution time for micro-program size. By creating 2-bit ROBDD slices we obtain a longer cascade (Fig. 6c), shorter micro-program and a simpler BCU. Moreover, we can detect 6-bit strings in sequences of arbitrary length. There is a 2-bit slice in the ROBDD at Fig. 6b that repeats itself, see the side arrows. The related LUT repeats itself in the cascade, see the highlighted cell in Fig.6c. If another micro-instruction format is provided, namely repeating the previous micro-instruction and decrementing an auxiliary counter, then any chain of identical LUTs can be reduced to a single repeated LUT only. In our case there are only two identical LUTs depicted as one, the number of input nodes is now 27 and the block of 4 micro-instructions corresponds to each one. The size of the micro-program is therefore 27 × 4 = 108 micro-instructions.

In the second example our 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). In this case we have the integer 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.7.



| y1 y0 | | →m3 m2 m0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| ↓ | 00 | 29 | 31 | 30 | 31 | 31 | 30 | 31 | x |
| | 01 | 28 | 31 | 30 | 31 | 31 | 30 | 31 | x |
| | 10 | 28 | 31 | 30 | 31 | 31 | 30 | 31 | x |
| | 11 | 28 | 31 | 30 | 31 | 31 | 30 | 31 | x |
| | y1y0 | | | | | | | | |
| | | 0 | 1 | 2 | 1 | 2 | 1 | 2 | x |
| | m2 | | | | | | | | |
| | | 02:= | 0 | 11:= | 1 | 22:= | 2 | 1x:= | 1 |
| | | | | | | | | | |
| | | | | 0 | 1 | 2 | 1 | | |
| | m3m0 | | | | | | | | |
| | | | | | 0 | | | | |

**Fig.7.** Decomposition of the sample 4-valued function (Example 2)



N0 exit N1@m3m4
N1@00 exit N2@m2
N1@01 v31 exit S31
N1@10 v31 exit S31
N1@11 v30 exit S30
N2@ 0 exitN3@y0y1
N2@ 1 v30 exit S30
N3@00 v29 exit S29
N3@01 v28 exit S28
N3@10 v28 exit S28
N3@11 v28 exit S28

**Fig. 8.** 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 for the first decomposition step, 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.8a. The symbolic micro-program targeted for the micro-engine in Fig. 3 is shown in Fig. 8b.

## 7. Conclusions

Firmware evaluation of multiple-output Boolean functions on the base of Boolean expressions or BDDs can be in many cases dramatically accelerated using the LUT cascade paradigm. Complexity of (incomplete) functions with many variables that can appear in embedded systems is usually low and related LUT cascades have much lower memory space requirements then the full table.

Obtaining the LUT cascade by slicing the MTBDD or by iterative decomposition is relatively easy. Optimum variable ordering in MTBDD is, however, a separate problem and can have a great impact on cascade width and space efficiency. An original heuristics that selects such a variable, that the width of the cascade is always kept at minimum, has been presented. LUTs obtained from the optimum MTBDDs seem to be a very good and effective data structure and should always be considered for evaluation of Boolean functions. They are flexible in making trade-offs between response time and memory consumption – two or more LUTs can be compacted into one larger LUT and the evaluation then reduces to a shorter chain of accesses into dispatch tables. Combinational LUT cascades implemented directly in hardware can support the fastest asynchronous or synchronous pipeline processing.

Future research will be oriented to study of evolutionary techniques for the optimum iterative decomposition of incomplete Boolean functions of many variables where the exhaustive search is out of question. The goal is to decompose large systems specified either by the set of specified points or systems fully specified by Boolean expressions into LUT cascades with the aid of parallel processing. Algorithmic synthesis of redundant cascades and of multiple cascades will be other targets of the research in a near future. Applications mainly in safety/security area will be sought.

## References

[1] B. M. Moret: Decision Trees and Diagrams, *Computing Surveys*, Vol.14, No.4, Dec. 1982, pp. 593-623.

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

[3] H. Qin, T. Sasao, M. Matsuura, K. Nakamura S. Nagayama and Y. Iguchi: "A realization of multiple-output functions by a look-up table ring," *IEICE Transactions on Fundamentals of Electronics*, Vol.E87-A, Dec. 2004, pp. 3141-3150.

[4] B. Bollig, I. Wegener: “Improving the Variable Ordering of OBDDs Is NP-Complete”, *IEEE Transactions on Computers*, 45(9), September 1996, pp. 993—1002.

[5] 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.

[6] A. Mishchenko, T. Sasao: Logic Synthesis of LUT Cascades with Limited Rails－ A Direct Implementation of Multi-Output Functions, *Technical report of IEICE*, The Institute of Electronics, Information and Communication Engineers, Vol.102, No.476 (20021121), pp. 103-108. VLD2002-99, ISSN:09135685.

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

[8] V**.** Dvořák: A cascade implementation of digital systems, *Microprocessing and Microprogramming*, North-Holland, Vol. 29, No. 1, 1990, pp. 151-163.

[9] V. Dvořák: Time- and Space-Efficient Evaluation of Sparse Boolean Functions in Embedded Software, *Proceedings of 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, Los Alamitos, IEEE CS US,, 2007, pp. 178-185.

[10] University of Hamburg, Department of Informatics, http://tams-www.informatik.uni-hamburg.de/applets

[11] T. Sasao and J. T. Butler, "Implementation of multiple-valued CAM functions by LUT cascades," ISMVL-2006, Singapore, May 17-20, 2006.

## Acknowledgement

This research has been carried out under the financial support of the research grants “Design and hardware implementation of a patent-invention machine”, GA102/07/0850 (2007-9), “Safety and security of networked embedded system applications”, GA102/08/1429 (2008-10), both care of Grant Agency of Czech Republic, and “Security-Oriented Research in Information Technology”, MSM 0021630528 (2007-13).

## Author

Vaclav Dvorak obtained M.Sc. degree in Electrical Engineering and Ph.D. degree in Applied Cybernetics from Brno University of Technology, Czech Republic, in 1963 and 1968. He was awarded a distinguished DrSc degree in Computer Science and Engineering in 1990.

Since 1963 he was 10 years with the Research Institute of Mathematical Machines Prague. Then he joined Brno University of Technology, Faculty of Information Technology, as a research associate and later as Associate and Full Professor. The major field of his interest has been computer hardware and architecture. He interleaved the work at the home university with acting as visiting scientist, lecturer and professor at a number of foreign institutions, over 8 years in all. (Canada, Malta, Libya, New Zealand, Australia, Tenerife-Spain). His research is recently oriented into application specific and parallel architectures.

Prof. Dvorak is a member of Computer Society and IEEE, a member of the Scientific Board of the Faculty of Information Technology, committees for Bc, MSc and Ph.D. studies in Information Technology and a member of JUCS Editorial Board.