LUT Cascade-Based Architectures for High Productivity Embedded Systems

V. Dvorak[1]

Abstract – *Fast, flexible, cheap in hardware or low-power implementations of multiple-output Boolean functions are often required in embedded systems. The paper describes digital system architectures which embody some of these attributes. They are based on already known, and recently reinvented, representation of combinational logic by Look-Up Table (LUT) cascades. Theoretical background of cascade decomposition is revised and a relation to decision diagrams is pinpointed. The design of LUT cascades is discussed and a heuristic method of cascade synthesis is given. Three possible applications of LUT cascades are presented: combinational logic pipelines, efficient micro-programs with multi-way branching and fast logic simulation in software. It is shown that LUT cascades are quite flexible in making trade-offs between performance and cost by adjusting cascade length, complexity of its cells and multiplicity of cascades. The method of LUT cascades may be quite useful not only for high performance pipelined stream processing or embedded microprocessor or microcontroller firmware, but also in digital system simulation.*

Keywords*: LUT cascades, binary decision diagrams BDD, MTBDD, iterative disjunctive decomposition, multi-way branching.*

Nomenclature

Z_R, the set of integers $\{0, 1, \ldots, R\text{-}1\}$
Complete Boolean functions:
$f_n : (Z_2)^n \rightarrow Z_2$, a single output Boolean function
$f_n^{(i)} : (Z_2)^n \rightarrow Z_2 \quad i = 1, 2, ..., m$ or equivalently
$F_n : (Z_2)^n \rightarrow Z_R$, a multiple output Boolean function
or an R-valued function of n Boolean variables
R, the number of distinct m-tuples of binary values
Explicit form: $i = 1, 2, ..., m$
$y_i = f_n^{(i)}(x_1, x_2, \ldots, x_n)$ (scalar notation),
$\boldsymbol{y} = \boldsymbol{F}(\boldsymbol{x})$ (vector notation)
Implicit form:
$\phi_0 (\boldsymbol{x}, \boldsymbol{y}) = 1$, an output characteristic function
Incomplete multiple-output Boolean functions:
$F_n : X \rightarrow Z_R, X \subset Z_2^n$
$Z_2^n \setminus X = D$, the don't care set
Discrete M-valued functions of n K-valued variables
$F_n : (Z_K)^n \rightarrow Z_M$.

I. Introduction

The implementation of combinational logic circuits in a form of one-dimensional unidirectional arrays (cascades) of modules (cells) attracted attention of researchers already many years ago. The reason might have been the simplicity, regularity and a hidden potential for future VLSI technologies. Soon it became known that the simplest single-rail (Maitra) cascade of horizontally connected cells, with individual (side) inputs, is not sufficient for realization of every single Boolean function of $n > 2$ variables, even though each variable is allowed to enter several cells [1]. However, if K- valued signals are used at horizontal line and M-valued signals at side inputs, every K-valued integer function of M-valued integer variables is then realizable by uniform redundant cascade (i.e. with repeated use of some input variables), [2]. Binary coded integer values can then be substituted for integer values if multiple output Boolean functions are required. Cascade cells can be realized either by random logic based on gates, multiplexers/de-multiplexers, or by look-up tables (LUTs) stored in ROM or RAM.

Redundant cascades of this kind were designed using algebraic approach [2] leading to an excessive number of cells, since this approach disregarded the complexity of synthesized functions. Long cascades were impractical due to a high cost and large delays. E.g. two Boolean functions of 4 variables required a two-rail cascade with 22 "permutation" cells or 12 general cells [2]. (Author proved that only 8 cells would do, [3]). That was the reason why the redundant cascades were more or less abandoned and shortest possible irredundant cascades were sought. Such cascades can be related to ordered multi-terminal binary decision diagrams (MTBDDs) that are used to represent binary-input, integer-valued output functions [3]. One level of the MTBDD is mapped into one cell in the cascade. If the order of variables is given, the MTBDD has a canonical form. However, finding a good order even

[1] Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic

for BDDs is an NP-complete problem [4]. Different variable orderings may produce formidable difference in size of resulting DDs.

Nevertheless, the sub-optimal LUT cascade synthesis can be based on the concept of sub-function and sub-function counting. The cascade is being built backwards from the last cell to the first, and simultaneously, as a byproduct, the MTBDD from leaves up to the root [5], [6].

The recent renewed interest in LUT cascades is due to a demand for more efficient realization of digital systems than provided by PLA or FPGA, as far as chip area or power is concerned, with a lower or even competitive speed [7]. It is therefore the right time to revise former results [8], [9] and compare them to the most recent ones [10]. As we will see, their synergy could produce some new ways of designing LUT cascades, clarify the relevant figures of merit, and specify other function classes realizable by them.

The rest of the paper is organized as follows. Section II presents the basic notions. Section III reviews some important theorems useful for estimation the cascade size and defines classes of realizable functions. Heuristic cascade synthesis by iterative decomposition is exemplified in Section IV. Synthesis of LUT cascades and LUT-cascade/ MTBDD co-synthesis is presented in sections V and VI. Multi-way branching micro-programs and a specialized micro-engine are discussed in Section VII and further optimization of LUT cascades in Section VIII. Results are commented on in Conclusion.

II. Basic Notions

Machine 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 well known representation of a single Boolean function in a form of a directed acyclic graph [4]. Whereas ROBDD is canonical (unique) representation for any given function and an order of variables, in case of incomplete Boolean functions we may have apparently more choices.

An important parameter is a size of BDD, i.e. the total number of decision nodes, as it determines the size of data structure needed to store a BDD. The construction of minimum-size ROBDDs belongs among NP-hard problems [4]: the size of the ROBDD depends on variable ordering and there are $n!$ possible orderings of n variables. A heuristic approach can be used in search for near-optimal orderings [6]. Even though the upper bounds on the OBDD's size for general Boolean functions are not too encouraging, many practical functions do have a reasonable BDD size.

M-ary decision diagrams 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_M (generally Z_R, $R \neq M$) to output $y = F_n(x_1, x_2, \ldots, x_n)$.

To represent a system of Boolean functions 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, Shared BDDs or SBDDs, [19]) or one word-level BDD (WLBDD) with n Boolean decision variables and with R integer terminal values. The latter form is more concise, but to obtain it from the bit-level BDD is not easy. There are many types of WLDDs. Multi-terminal BDDs have integer leaves and therefore represent functions from Booleans to integers. A BMD (Binary Moment Diagram) is another representation for functions that map Boolean vectors to integers. This representation is more compact for some useful arithmetic functions which have exponential size if represented by MTBDDs. Hybrid decision diagrams HDDs are a combination of MTBDDs and BMDs.

Encoded Characteristic Function of Non-zero outputs (ECFN) is yet another representation of multiple-output functions, which uses the shortest encoding of output vectors $\boldsymbol{y}$ using auxiliary variables. The auxiliary variables can be intermingled with normal variables arbitrarily [11]. Auxiliary variables can also be used in connection with MTBDDs and SBDDs with resulting diagrams MTBDD+ and SBDD+, [11]. In what follows, we will use the most frequent type – MTBDDs.

As the LUT cascades are the main concern of this paper, we will provide a formal definition.

Def. 1. A cascade C of a form $k \times m$ is the system
$C = [K, M, H_1, H_2, \ldots, H_B, \mu]$
where
$K \leq 2^k$ ($M \leq 2^m$) is the number of specified Boolean input vectors at k horizontal (m vertical or side) cell inputs,
H_i: $(Z_2)^k \times (Z_2)^m \rightarrow (Z_2)^k$, $1 \leq i \leq B$ are functions implemented by individual cells,
B is the total number of cells and
μ: $\{1,2,\ldots, B\} \rightarrow (Z_2)^k$ assigns k-tuples of input variables x_i , $i = 1,2,\ldots, n$ to B cells in the cascade.

The above cascade has k horizontal rails carrying Boolean values and each cell has m vertical (side) inputs. The last cell in the cascade may have $r \neq k$ outputs.

Note. Cascades considered at [12] use cells with additional vertical outputs. They will be introduced by example in Section VI.

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

III. Complexity of Some LUT Cascades

As an arbitrary system of Boolean functions can be implemented by a single memory look-up table, it is natural to compare the capacity of this single LUT with the total capacity of all LUTs in the cascade. So the simplest figure of merit of different cascades implementing the same Boolean system is a total number of bits of all LUTs.

The LUT of the original Boolean system with n input and r output variables requires $r2^n$ bits, whereas each but last LUT in a cascade requires $k2^{k+m}$ bits. We can therefore realize saving if

$$(B\text{-}1)k2^{k+m} + r2^{k+m} < r2^n. \quad (1)$$

With $k+m$ input variables entering the first cell, there will be $B = \lceil (n\text{-}k)/m \rceil$ cells in the cascade and condition (1) becomes

$$r > k\frac{\lceil (n-k)/m \rceil - 1}{2^{n-k-m}-1}. \quad (2)$$

LUT cascades satisfying above condition (1) will be referred to as cost-effective ones. In case that $r \le k$ saving may start at no less than 6 (5) variables for r=1 (r=2). We always save with cascades having $r > k$. The problem is that not all functions are realizable by cost-effective cascades. Fortunately, important classes of functions used in digital design are LUT cascade-realizable.

Incomplete Boolean functions that are frequently used in applications are one such class, for which the form factor of LUT cascades can be estimated [13]:

Theorem 1. Every R-valued incomplete function of n Boolean variables defined on set $X \subset Z_2^n$ is realizable as the output function of a $k \times m$ cascade with $k = \lceil \log_2 |X| \rceil$.

Upper bound on the cost of such LUT cascade is derived in [13].

Functions with large areas of the domain mapped to a certain constant and with remaining points mapped to R-1 other values show similar properties as incompletely specified functions [13]:

Theorem 2. Every function F: $Z_2^n \to Z_R$ such that F = const in K-1 input vectors and $F \ne$ const otherwise is realizable by $\lceil \log_2 K \rceil \times m$ cascade.

Another measure of multiple-output Boolean functions is the width of the MTBDD for the given ordering of input variables. This is called a C-measure in a recent literature [12]. In Section VI we will see, that C- measure is directly related to the number of rails in the LUT cascade and the lower its value, the better the chance that cost effective cascade exists. Theoretically, for the most random functions and any permutation of input variables, C-measures increase exponentially and the related number of rails k may be too large to provide saving. For random logic the following Theorem 3 gives the upper bound on number of rails k (the C-measure has a value of 2^k), [12]:

Theorem 3. Every multiple output Boolean function F: $Z_2^n \to Z_R$ is realizable as the output function of an irredundant cascade $k \times m$ with $B = \lceil n/m \rceil$ cells and with number of rails

$$k \le \left\lceil \log_2 \max \left[\min_i (R^{2^i}, 2^{m(B-1)}) \right] \right\rceil , \quad (3)$$

i = 0, 1,…, B-1. Fortunately, many real-world functions have small C-measures and lend themselves to cost-effective cascade realization. Some of these classes are listed below:

1. Symmetric functions (any permutation of n input variables does not change the value of the function)
2. Threshold functions (special case of symmetric functions) [12]
3. Detectors of bit patterns in data streams [14] important for Intrusion Detection Systems (IDS)
4. Numerical function generators (trigonometric functions, logarithm functions, square root, reciprocal) using linear [15] or quadratic approximations
5. Weighted Sum functions [12]
6. The Advanced Encryption Standard (AES) encryption using a 128-bit key [16]
7. Multiple-valued CAM functions [17]
8. Code converters and checkers
9. Radix converters
10. Sorting networks [8].

IV. Iterative Disjunctive Decomposition and LUT Cascades

Storing the full map of the multiple output function as a single LUT in the memory is in embedded systems acceptable approach up to about 10 variables. For several tens of variables we have to use more compact data structures and one way of obtaining them utilizes a disjunctive decomposition of original functions. The basic idea is shown in Fig.1.

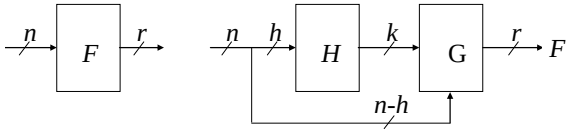

Fig.1. Disjunctive decomposition of multiple output Boolean function F of n variables

The original function F is split into two functions H

and G, so that $F = G(Y, H(X))$, where multi-valued variables X, Y, H and G are binary coded using h, n-h+k, k *and* r bits, respectively:

$$X \in Z_2^h,\ Y \in Z_2^{n-h+k},\ H \in Z_2^k, \text{ and } G \in Z_2^r, \qquad (4)$$

Fig.1. Of course, we are interested only in non-trivial decompositions for which $k < h$, i.e. tables of functions H and G are more thrifty in memory space than the table of original function F, i.e.

$$k\,2^h + r\,2^{n-h+k} \le r\,2^n. \qquad (5)$$

Sometimes we prefer to get tables describing H and G of the same size, to be stored in the same memory area (e.g. two table items in one word). This requirement translates to

$$h = k + n - h \qquad (6)$$

and (9) is then rewritten into

$$2^k \le r2^h/(r + k). \qquad (7)$$

The lower a value of k (with values n, h, r fixed), the better. In a special case k=r, (11) turns to $k \le h - 1$.

The value of k cannot be selected arbitrarily, it is given by complexity of the function under consideration. The minimum value of k is given by modified decomposition Theorem 4 [18], which under notation (4) and according to Fig.1 says:

Theorem 4.
Function F is decomposable into

$$F = G(Y, H(X))$$

(Fig.1) if and only if the value of 2^k is equal or greater than the number of distinct sub-functions of n-h variables.

Note. A *sub-function* f_{n-h} of n-h variables is an instance of function F_n with h remaining variables fixed at certain values. In the following sections we will use a technique of minimizing *sub-function count*; it requires enumeration of the distinct sub-functions in the set of all 2^h sub-functions. Then we also need to count separately distinct *non-constant* sub-functions.

Decomposition shown at Fig.1 can be repeated iteratively with functions H and G. Provided that m variables are removed at a time, $n - h = m$, the cascade will be ultimately composed of $\lceil n/m \rceil$ cells, each cell with m side inputs and cells interconnected horizontally to one another. The procedure of obtaining this cascade will be referred to as iterative disjunctive decomposition. We will illustrate it on examples in the next section.

V. Synthesis of LUT Cascades

Using the concept of sub-functions, we will now illustrate iterative decomposition. The number of distinct sub-functions of s variables, s = 1, 2, …, n-1, characterizes the Boolean function and its complexity. Sub-functions themselves may also be incomplete (don't care values for some binary s-tuples). A compatibility relation can be defined on the co-domain of such sub-functions: don't care (denoted by "x") is compatible with any value from Z_R .

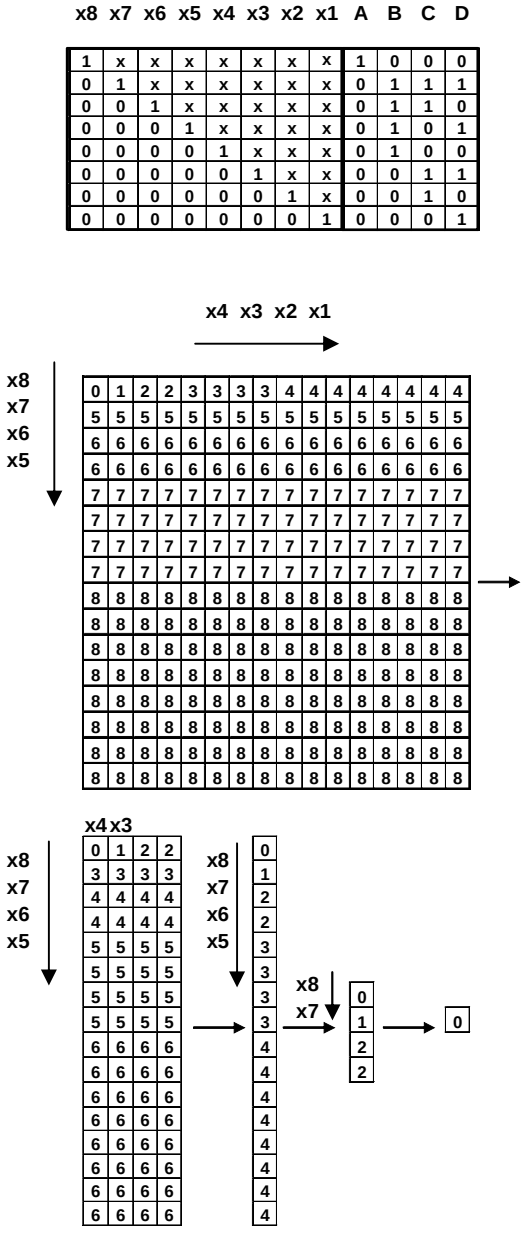

Fig.2. Iterative disjunctive decomposition of 8-bit priority encoder

First we shall decompose iteratively the 8-bit priority encoder (PE) fully specified by the compact table at the top of Fig.2, which is expanded to the full size function table under it. Two variables will be removed from the function at a time. If we start with variables x2 and x1, sub-functions of x2 and x1 are easily detected as 4 adjacent fields in rows. There are 7 distinct sub-functions that will be renamed as:
0122 := 0, 3333 := 1, 4444 := 2, 5555 := 3, 6666 := 4, 7777 := 5, and 8888 := 6.

Replacing sub-functions by new IDs, we obtain the map of a residual function of 6 variables as shown in Fig.2. This residual function has 5 distinct sub-functions, regardless the choice of two variables. If we use x4 and x3, sub-functions are:
0122 := 0, 3333 := 1, 4444 := 2, 5555 :=3, 6666 := 4
and a residual function of 4 variables remains. In two additional steps we will rename 3 sub-functions of variables x6 and x5:
0122 := 0, 3333 := 1, 4444 := 2
and in the last step we have a single function of variables x8 and x7 (0122 renamed to a constant 0).

The complexity of the cascade for the priority encoder function is given by numbers of sub-functions in all residual functions. If we take all the distinct sub-functions, the "sub-function profile" is {7, 5, 3}. Number of bits required for encoding sub-functions IDs gives the number of rails connecting neighbor cells. In the next section we will show that number of rails between adjacent cells can be reduced by eliminating constant sub-functions.

The design of the LUT cascade is completed by creating look-up tables of individual cells. The contents of these tables are obtained by reversing the previous renaming assignments. Contents of four LUTs in the priority encoder example are given in Fig.3b. First two LUTs are combined into one in cell 1.

Let us note that capacity of 3 LUTs is $16 \times 3 + 32 \times 3 + 32 \times 4 = 272$ bits, whereas the full table of the priority encoder would require $256 \times 5 = 1280$ bits. Cascades with smaller aggregate LUT capacity than the full table of the original function will be denoted as *cost-effective*. Fortunately, many functions used in real life may be realized by cost-effective cascades. There are some exceptions though, e.g. integer multipliers.

LUT cascade just described can be used for pipelined implementation of the PE. The LUT cascade would have to be completed by pipeline registers between cells. These registers would serve also for storing variables used at cell side inputs (4 bits between cells 1 and 2, 2 bits between cells 2 and 3). The performance of the PE in the continuous stream of input vectors would then be determined by the slowest cell, be it memory block or logic gate network.

The first implementation of a new programmable logic device using LUT cascade architecture developed in 0.35um CMOS logic process has been announced recently, [7]. Eight 64Kb asynchronous SRAMs are simply connected to form an LUT cascade with a few additional circuits. Benchmark results show that it has a competitive performance to FPGAs. The latency of an internal LUT is 3.8ns. A total latency of 11.6ns for a 2-LUT cascade, 34.4ns for an 8-LUT cascade in asynchronous operation, and the operating frequency of 200 MHz in an 8-stage pipeline operation were experimentally confirmed, [7].

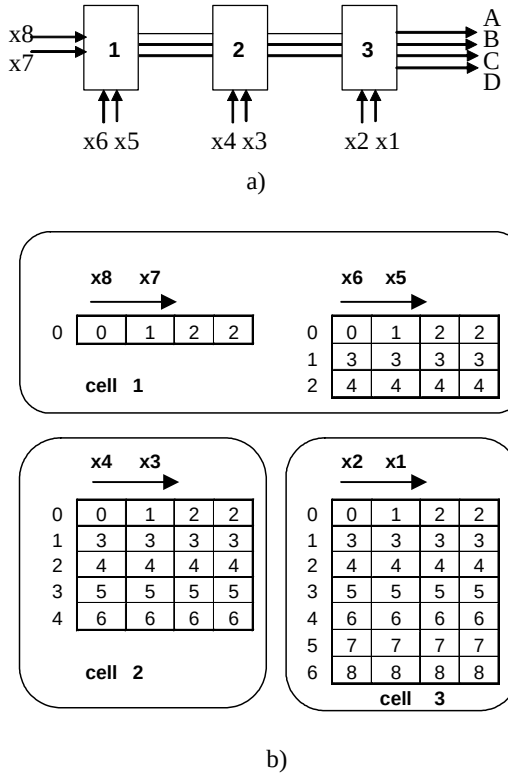

Fig.3. Iterative disjunctive decomposition of the 8-bit priority encoder. a) LUT cascade b) cell functions

The PE example was easy. The number of sub-functions in each step was the same, regardless which pair of variables had been selected. The next example will illustrate a typical LUT cascade for an incompletely specified function, where the number of sub-functions in a certain step of iterative decomposition depends on a selected variable.

VI. LUT Cascades and MTBDDs

The central concept of a sub-function used previously in iterative decomposition has another representation – a decision node in MTBDDs. There is 1:1 mapping between a level of decision nodes in the MTBDDs and a set of sub-functions recognized in a corresponding decomposition step. In this section we will revise a heuristic algorithm [6] for constructing sub-optimal MTBDDs and simultaneously the LUT cascade [5]. Before exact formulation of an algorithm we prefer to illustrate the technique on a small example.

Let us consider the 5-valued function $F(a,b,c,d)$ of four Boolean variables specified by the map in Fig. 4. (An algorithm described later on operates, however, on a list of defined input vectors). Selection heuristics used in decomposition steps is

based on counting the number of *true* (non-constant) distinct sub-functions of every variable and selecting the variable with the lowest count. In case of ties, we use a lower count of *constant* sub-functions, then an arbitrary choice.

In the first decomposition step we have

a: 2, *b*: 2, *c*: 4 and *d*: 3

true sub-functions and we select variable *b* because it has only one constant sub-function whereas *a* has four. The list of all distinct sub-functions of variable *b* with the new IDs follows:

44 := 0, 03 := 1, 21 := 2, *x*3 := 03 :=1, *xx* := *x* .

Don´t cares in incomplete sub-functions are either replaced by output values to make them equivalent with other complete sub-functions, e.g. *xb* and *ax* can be made equivalent to *ab*, or are left don´t cares (*xx* := *x*). The goal is to minimize the total count of true and constant sub-functions.

We can interpret distinct sub-functions of a single variable as binary decision nodes. True sub-functions are represented by decision nodes with two edges, whereas for constant sub-functions these edges coincide and the decision node may be omitted. Using new IDs for sub-functions of variable *b*

(44 :=0) 03 :=1 21 :=3,

we can draw the lowest level of the MTBDD, Fig. 4a, and transform the original map into a map of the residual function of 3 variables.

In the second decomposition step variable *d* is an optimum choice, requiring also only two decision nodes. Two remaining variables *c* and *a* need one decision node each. The complete MTBDD of the given function in Fig. 4a thus contains 6 nodes what is a minimum count in this case. Three more nodes corresponding to constant sub-functions are replaced by a single output edge. The LUT cascade complementary to the MTBDD is shown in Fig. 4b. It is easily obtained by cutting the MTBDD into 3 slices.

Returning to the former example of 8-bit priority encoder, we can obtain the MTBDD from previous decomposition at Fig.2 when we remove only one variable at a time instead of two. This diagram in Fig. 5a has a very simple linear form and terminal values are generated very early along the main path. If we present the terminal values at the cell side outputs as soon as they are generated, the number of rails between cells in the LUT cascade can be reduced, as the terminal values (i.e. constant sub-functions IDs) do not have to propagate to the end of cascade and the shorter code carried on the rails identifies only the true sub-functions. This reduced form of the LUT cascade is shown in Fig.5b and it corresponds to implicit representation of the multiple output function. All terminal values are wire-ORed on the output bus. What terminal value will be used is determined by a Boolean function ϕ_0 (***x***, ***y***) = 1.

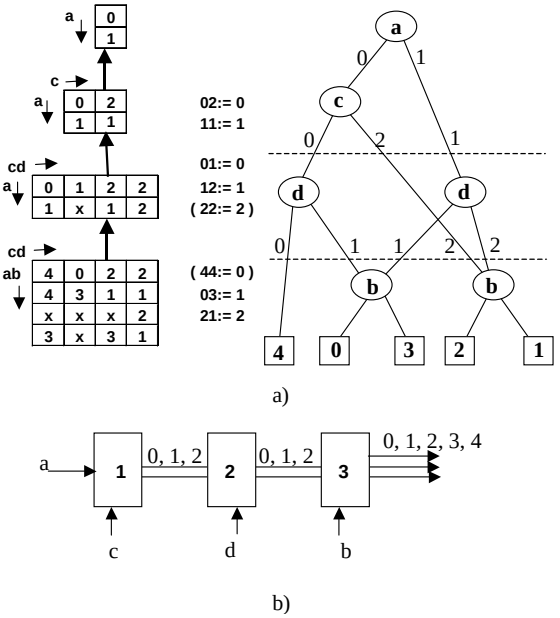

Fig.4. LUT cascade and MTBDD co-synthesis by means of the bottom -up iterative decomposition.
a) leaves to root MTBDD construction b) backward construction of the cascade from cell 3 to 1

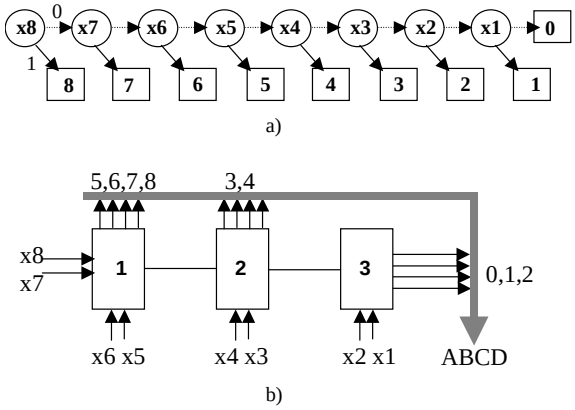

Fig.5. The 8-bit priority encoder
a) ROBDD b) variant of the LUT cascade with a single rail

As we have seen, instead of undertaking a global search of optimum order of variables, we go on in steps and select *m* variables at a time. In each step we look for an *m*-tuple of variables which has the lowest minimum cover of all true distinct sub-functions associated with it, over all *m*-tuples of remaining variables. The maps of small functions are good for illustration purposes only. The large functions must be handled automatically. The software packages for MTBDDs and multi-valued DDs and their optimization are available and could be employed for LUT cascade synthesis. A special algorithm for a class of incomplete specified functions given by the list of input vectors is described in detail in [8]. As already mentioned, completely specified functions with one dominant value also behave as incomplete ones.

VII. Multi-way Branching Micro-programs and a Micro-sequencer

LUT cascades can also serve as a paradigm of operation for specialized micro-programmed controllers with frequent multi-way branching. The required speed is determined by the size of a group of condition variables tested in one micro-instruction. For example, if the micro-program should branch to 5 targets according to a function of 4 Boolean variables, it can do four 2-way branching, two 4-way branching or a single 16-way branching. Micro-sequencers available nowadays as off-the-shelf components or IP cores support typically two-way branching which assumes testing either a single condition bit or a hard-wired combination of selected condition bits. Multi-way branching based on certain binary patterns of subgroups of condition bits can be implemented as a series of two-way branches along a certain path in an associate BDD or using an additional multi-way branch control unit such as the 16-way Am 29803A by AMD Inc.

The operation of a micro-sequencer with multi-way branch unit will be explained in Fig.6. Here μIP and μIR is the micro-instruction pointer and register, ROM is a control store, +1 is an incrementer, MX is a multiplexer. The 16-way branch control unit enables to move values of 0 up to 4 variables selected by input multiplexers to the lowest significant bits of the output code. This code is then wire – ORed with the lowest part of the target address.

As an example, we will continue our previous PE example at Fig.3. A general multi-way branch micro-instruction, not related to any particular architecture, has the same structure as a switch:

S0 if $F^{(0)}$ then *cv*0 exit *S*0
if $F^{(1)}$ then *cv*1 exit *S*1
if $F^{(R-1)}$ then *cv*R-- exit *S*R--
…
else don´t care

where *S*i´s are state labels, *cv*j´s are conditional output vectors (ABCD) and $F^{(i)}$ = 1 iff *F*(x1, x2, …, xn) = *i*. In case of the 8-bit PE we may use two LUTs, each with 16 items (cell 1 and 2+3 in Fig. 3a). The symbolic micro-program will look like this segment:

*N*0 exit *N*1@x8x7x6x5
*N*1@0000 exit *N*2@x4x3x2x1
*N*1@0001 5 exit *S*5
*N*1@0010 6 exit *S*6
*N*1@0011 6 exit *S*6
….
*N*1@1111 8 exit *S*8
*N*2@0000 0 exit *S*0
*N*2@0001 1 exit *S*1
…
*N*2@1111 4 exit *S*4.

Here conditional output is the priority level from 0 to 8 and operator @ means modification of target address *N*i (with the lowest part cleared) by wired-ORing it with the code at the right of @.

It is apparent that we have used a cascade of two look-up tables with starting addresses *N*1, *N*2 and that 2 or 3 microinstructions have to be executed in order to emulate the PE.

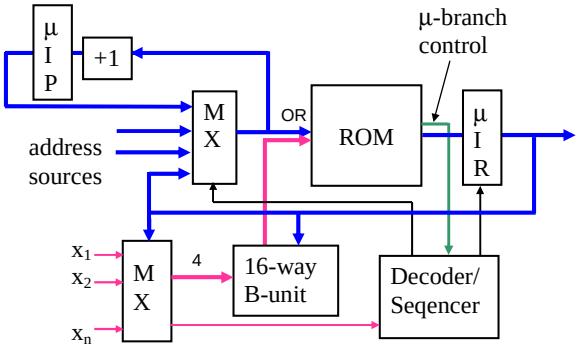

Fig.6. Micro-sequencer architecture with multi-way branching

VIII. Further Optimization of LUT Cascades

Two optimization techniques will be presented in this last section: irredundant cascades and output grouping. Both these techniques can reduce the number of rails or cell complexity and can make the cascade cost-effective.

First example, Moore-type state machine, is the arbiter circuit with dynamic priority allocation scheme based on Last Granted Lowest Priority (LGLP). It has 3 input requests x3, x2, x1, 6 states, and 3 outputs (grants). Its behavior is described by next state/output table in Fig.7a. The arbiter is scalable, for n inputs it will have $n(n+1)/2$ states. We are to find LUT cascade implementation for n = 3.

Let us consider the next state function. The number of single-variable sub-functions is 9 for any variable and we would need 4 rails on entrance into the last LUT. However, the number of sub-functions of x1 can easily be reduced to 7 (3 rails) by making use of permutation (12) in the right half of the next state table. By applying this permutation we do not remove any variable, but simplify the following decomposition. Permuted values are denoted in bold in Fig.7a, b. Three following decomposition steps in Fig. 7b will do to obtain the resultant cascade in Fig. 7c. The output is generated by another cell as a function of the state. The cascade delay can be cut in half by combining two adjacent LUTs into one. One more register (beside the state register) will enable pipeline operation. This small size arbiter could use

one LUT only, but serves as an exercise example. Larger arbiters could yield more interesting cascade implementations. The same architecture, a LUT cascade plus pipeline registers, can implement any sequential circuit.

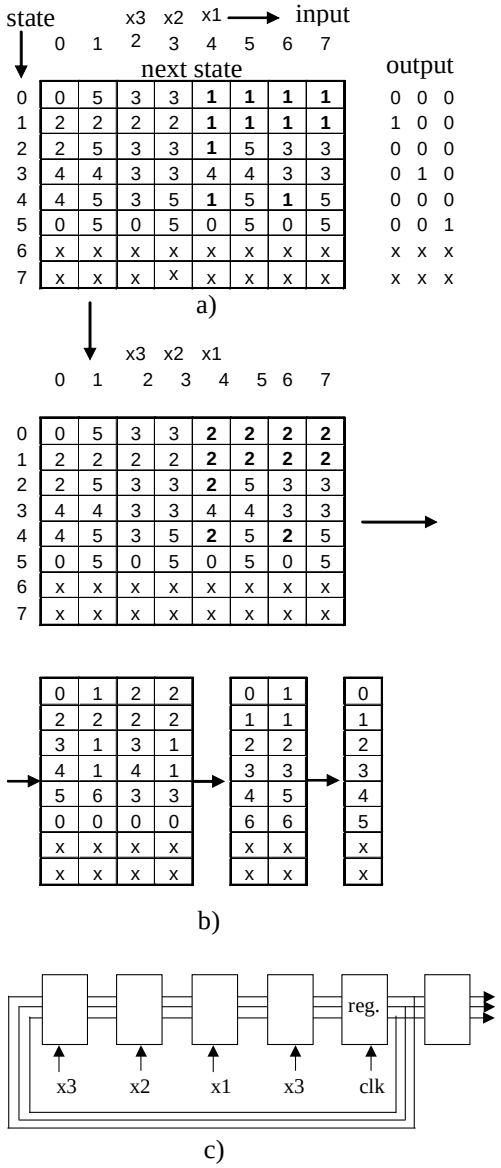

Fig.7. LUT cascade implementation of the LGLP arbiter. a) transition table b) decomposition procedure c) redundant cascade.

If the LUT cascades tend to have too many rails, we can partition the outputs into several groups and realize each group by a separate LUT cascade. Immediately a new problem is generated, namely how to do output grouping, i.e. which outputs should be considered together. Suggested heuristics [19] allow cascade realization of large designs, lead to faster responses and to a better cost-effectiveness.

IX. Conclusion

Design of digital systems with a degree of regularity in physical placement of subsystems (cells) and in their interconnection has always been a much desired goal and is even more so at present. A regular logic has advantages which make it more attractive: short development time, better utilization of chip area, easy testability and easy modifications all end up in a lower cost.

Digital systems based on LUT cascades have desired regularity and may therefore result in high productivity. The method of LUT cascade synthesis of Boolean functions is suitable for designs with many input- and/or output variables in the following cases:

- LUTs in block RAMs: provide support for reconfigurable architectures, asynchronous cascades or clocked pipelines; speed is competitive with other FPGA designs [7], layout and wiring are very easy. The LUT cascade LSI is a promising reconfigurable logic device for future sub-100nm LSIs [7].
- LUTs in control ROM. Sequential processing of LUT cascades by means of multi-way branching (also known as LUT ring, [20]); it can speed up branching programs or micro-programs. It can be useful for micro-sequencers and micro-program controllers not only on FPGAs, but also for controllers on ASICs or SOCs. Comparison with traditional design methods on a set of benchmarks [20] demonstrated better performance and a smaller chip area.
- LUTs in RAM, sequential processing of LUT cascades in software by universal CPU cores. LUTs serve as a means of software description of large systems for the purpose of simulation and verification. High speed-ups (from 16 to 64) at evaluation of logic functions with respect to a LCC simulator (Levelized Compiled Code simulator) were reported [21].

Cost-effective cascade implementation is restricted to functions with low complexity or with don't cares, that are frequently used in practice. Synergy of present and old LUT cascade synthesis techniques can broaden the field of applications. Output grouping is one such technique. It provides multiple LUT cascades that can be processed in parallel or one after another sequentially.

Future research should address new application areas by employing optimization techniques - introducing redundant cascades or new heuristics for output grouping. These techniques could provide cost-effective cascades for new classes of functions, especially for encrypting devices, intrusion detectors, code checkers and fault-tolerant systems. Security and safety oriented applications will be a subject of a future research.

Acknowledgements

This research has been carried out under the financial support of the research grants “Architectures of Embedded Systems Networks”, GACR GA102/05/0467, “Design and hardware implementation of a patent-invention machine”, GACR 102/07/0850, Grant Agency of Czech Republic and “Security-Oriented Research in Information Technology”, MSM 0021630528.

References

[1] K.K. Maitra: Cascaded switching networks of two-input flexible cells, *IRE Trans. Electron. Comput*., pp, 136-143, 1962.

[2] M. Yoeli : The Synthesis of Multivalued Cellular Cascades. *IEEE Trans. On Computers*, Vol. C-9, Nov. 1970, pp.1089-1090.

[3] V. Dvořák: *Decomposition Theory with Applications in Programmable Digital Systems*. A thesis required for Doctor of Sciences (DrSc) degree. Faculty of Electrical Engineering, Technical University of Brno, May 1989. (224 pages, in Czech).

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

[5] V. Dvořák: Automated Cascade Partitioning of Combi-national Logic. *Proc. of the 1st conf. CompEuro 87, VLSI and Computers.* Eds. W.E.Proebster and H.Reiner, Hamburg, 1987, pp. 315 - 318.

[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] K. Nakamura, T. Sasao, M. Matsuura, K. Tanaka, K. Yoshizumi, H. Qin, and Y. Iguchi, "Programmable logic device with an 8-stage cascade of 64K-bit asynchronous SRAMs," Cool Chips VIII, *IEEE Symposium on Low-Power and High-Speed Chips*, April 20-22, 2005, Yokohama, Japan.

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

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

[10] T.Sasao´s group publications:
www.lsi-cad.com/sasao/Papers/pub2001.html to 2007.html

[11] A. Mishchenko, T. Sasao: Logic Synthesis of LUT Cascades with Limited Rais – 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.

[12] T. Sasao, Y. Iguchi, M. Matsuura, "LUT cascades and emulators for realizations of logic functions," RM2005, Tokyo, Japan, Sept. 5 - Sept. 6, 2005, pp.63-70.

[13] V. Dvořák: Bounds on Size of Decision Diagrams, *JUCS - The Journal of Universal Computer Science*, Vol..3, 1997, pp. 2-23.

[14] V. Dvořák: Time- and Space-Efficient Evaluation of Sparse Boolean Functions in Embedded Software. *Proc. of the 14th IEEE Int. Conf. And Workshops on the Engineering of Computer-Based Systems*. IEEE CS Press, Los Alamitos, CA, 2007, pp.178-185.

[15] T. Sasao, S. Nagayama, and J. T. Butler, "Programmable numerical function generators: Architectures and synthesis system," *FPL 2005* ,Tampere, Aug.24-26, 2005, pp.118-123.

[16] H. Qin, T. Sasao, and Y. Iguchi, "An FPGA design of AES encryption circuit with 128-bit keys," *GLS VLSI 2005*, Chicago, IL, April 17-19, 2005, pp. 147-151.

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

[18] H.A. Curtis: *A New Approach to the Design of Switching Circuits* (Van Nostrand Comp. Inc., Princeton, N.J., 1962).

[19] R. Drechsler, M. Herbstritt, B. Becker: Grouping heuristics for word-level decision diagrams. *Proceedings of the 1999 IEEE International Symposium on Circuits and System* ISCAS '99, pp. 411--415.

[20] T. Sasao, J. T. Butler, and M. D. Riedel, "Application of LUT cascades to numerical function generators," *The 12th Workshop on Synthesis And System Integration of Mixed Information technologies* (SASIMI2004), Oct. 18-19, 2004, Kanazawa, Japan, pp.422-429.

[21] H. Nakahara, T. Sasao and M. Matsuura, "A fast logic simulator using an LUT cascade emulator," *ASPDAC 2006*, Yokohama Jan. 2006, pp.466-465.

Authors’ information

[1]Brno University of Technology, Faculty of Information Technology, Brno, Czech Republic, CZ 612 66

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 and JEE Editorial Boards.