LUT CASCADE-BASED IMPLEMENTATIONS OF ALLOCATORS

Václav Dvořák, Petr Mikušek

Brno University of Technology, CZ

ABSTRACT

This paper presents a new technique for iterative decomposition of multiple-output Boolean functions with an embedded heuristics to order variables. The algorithm produces a cascade of LUTs that implements the given function and simultaneously constructs a sub-optimal Multi-Terminal Binary Decision Diagram (MTBDD). The LUT cascade can be used for pipelined processing on FPGAs or at a non-traditional synthesis of large combinational and sequential circuits. On the other hand, suboptimal MTBBDs can serve as prototypes for efficient firmware implementation, especially when a micro-programmed controller that firmware runs on supports multi-way branching. A novel technique is illustrated on a practical example of the m × n wavefront allocator (m = n = 4, 20 inputs, 16 outputs). It may be quite useful as a more flexible alternative implementation of digital systems with increased testability and improved manufacturability.

Index Terms— LUT cascades, Multi-Terminal BDDs, iterative disjunctive decomposition, a wavefront allocator

1. INTRODUCTION

Design of digital systems with a degree of regularity in physical placement of subsystems 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. A one-dimensional cascade of look-up tables (LUT cells) is such a regular structure.

LUTs are in fact multiple-input, multiple-output universal logic blocks. LUTs in block RAMs may provide support for reconfigurable architectures, asynchronous cascades or clocked pipelines; speed is competitive with other FPGA designs [1], layout and wiring are very easy. The LUT cascade is a promising reconfigurable logic device for future sub-100nm LSI technology [1]. Sequential processing of LUT cascades by means of micro-engines with multi-way branching can improve firmware performance a great deal [2].

A direct synthesis of non-redundant LUT cascades comes out easily from the known representation of integer functions of Boolean variables in a form of Multi-Terminal Binary Decision Diagrams (MTBDD), [5]. Cascaded LUTs are slices (layers) of this MTBDD. An optimum ordering of variables can be treated as a separate problem or it can be solved concurrently with LUT cascade synthesis by iterative decomposition [2].

Multiple-output Boolean functions have been more recently represented by BDD_for_CF diagrams [6]. Here the top-down iterative decomposition starts from the root and after a removal of a single variable the whole diagram has to be reconstructed. Another disadvantage of this approach is a large size of BDD_for_CF diagrams as they include input as well as output binary variables.

In this paper we present a heuristic technique of the iterative decomposition of integer-valued functions. Its main contribution is that the bottom-up synthesis of MTBDD/LUT cascade does not require knowledge of optimum ordering of variables, because the order of variables is generated concurrently. Obtained LUT cascades can be used in hardware, firmware and software implementation of combinational and sequential functions.

The paper is structured as follows. Our heuristic approach to construction of sub-optimal MTBDDs and LUT cascades is explained in Section 2. Section 3 deals with a wavefront allocator, its decomposition and implementation. Experimental results are summarized in Section 4 and commented on in Conclusion.

2. CONSTRUCTION OF LUT CASCADES AND OF SUB-OPTIMAL MTBDDS.

In this section we will present a heuristic technique of a sub-optimal LUT cascade construction. The given function is decomposed iteratively, one input variable at a time [2]. In each decomposition step, such an input variable is selected that produces a minimum local width of the cascade. Simultaneously we obtain a MTBDD, which is in fact revealing the internal structure of LUTs in terms of decision nodes.

For the sake of brevity, we prefer to illustrate the synthesis technique on an example of 4-input priority encoder (PE4). Input requests are denoted r0 to r3, output grants g0 to g3 are represented by one integer variable z specifying the address of the winning request (0 to 3) or no request (4).

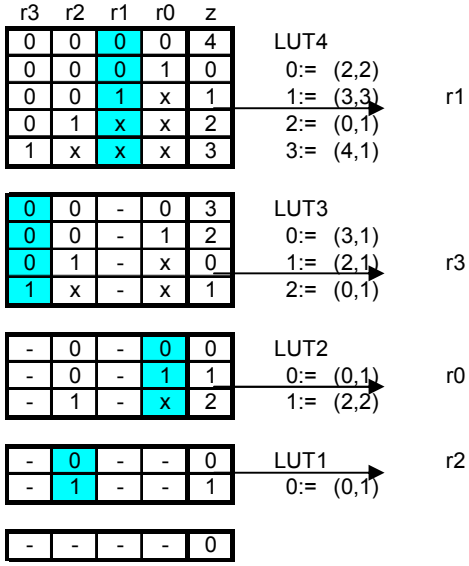

Fig.1. Iterative decomposition of an integer function of 4 binary variables (PE4)

The integer function z = F(r0, r1, r2, r3) of four binary variables is specified by a table with input values from {0,1,x}, Fig,1. We use symbol x to shorten the function specification; regardless whether an associated variable will attain the value of 0 or 1, the value of output z will be the same. In the meantime we will select a sequence of input variables for iterative decomposition randomly, e.g. r1, r3, r0, r2. A single variable will be removed from the function in one decomposition step. Starting with variable r1, we inspect column r1 (highlighted at the top table in Fig. 1) to see how many distinct pairs of function values (also equivalently sub-functions of a single variable [2])

$$[F(r0, 0, r2, r3), F(r0, 1, r2, r3)] \qquad (1)$$

are produced by this variable.

We have to analyze pairs of rows in the table such that one row has value 0 in column r1 and another one value 1. Two such rows which are not in conflict (do not have complementary values in the same column except column r1) can be replaced by a single row in the new table of a residual function F_1(r0, -, r2, r3). Bit values 0, 1 and symbol x in columns are combined as shown in Table 1.

A symbol "–" in a certain column means that the variable at that place has already been removed and does not exist. A pair of function values (1) from two rows will be replaced later by a new integer value (id).

There are two rows at the top table in Fig.1, 1st and 3rd, that can be combined into

00-0 (4,1).

Also the 2nd row can be combined with the 3rd row into

00-1 (0,1).

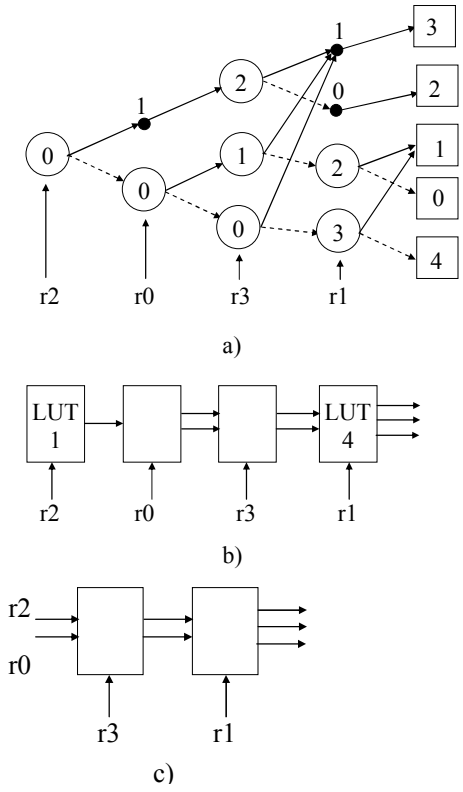

Fig. 2. A MTBDD (a), a generic LUT cascade (b), and a compact LUT cascade with 3-input cells (c) obtained by iterative decomposition (PE4 example)

Table 1. Combining rows in the function table

row (v = 0)	!v	1	0	x	1	x	0	x	—
row (v = 1)	v	1	0	x	x	1	x	0	—
new row	—	1	0	x	1	1	0	0	—

Two pairs of function values (4,1) and (0,1) with new identities "3" and "2" correspond to two decision nodes at the lowest level of a MTBDD that is just being created, see Fig.2a. For incomplete functions with "function value" z = DC (don´t care) we should combine (*a*, DC) = (DC, *a*) = (*a*, *a*), i.e. replace DC by value of *a*.

If a row contains symbol "x" in column r1, it will be carried over to the new residual table with "x" replaced by symbol "-". Function values (1) are now identical (similarly as z = DC combined with value *a*) and variable r1 in fact does not decide anything. A corresponding decision node in the MTBDD has only one output and can be replaced by a shortcut from the input to the output. There are two such rows in the first decomposition step, rows 4th and 5th. They produce degenerate decision nodes 0 and 1 in the lowest level of MTBDD shown in Fig. 2a as black dots.

By now, we have exhausted all possible pairs of rows and have replaced them by new rows in the next (residual) function table. As a result of the removal of variable r1 from

the original function, obtained pairs of function values can be assigned the shortest possible code by enumeration. This transform is shown next to the top table at Fig.1 and in fact it is realized by the last LUT4 (e.g. for cell input 3, cell output will become 4 or 1 depending on whether r1 is 0 or 1).

The same procedure is repeated in the following decomposition steps until all variables have been removed. We proceed in a backward direction, from the leaves to the root of the MTBDD or from LUT4 to LUT1, Fig. 2a, b. In the case of LUT cascades, it is sufficient to go on with iterative decomposition until the number of remaining variables equals to the required number of address inputs to the first LUT in the cascade.

The remaining question not addressed as yet is, which variable should be used in any given step. We use a heuristics that strives to minimize the LUT cascade width. At each step a variable is selected, that generates the minimum number of rows in the sought LUT or equivalently the minimum number of decision nodes (including degenerate ones) in the sought level of the MTBDD. In the case of a tie the lowest cost criterion is applied: a variable producing the lowest number of true (non-degenerate) decision nodes in the current level of the MTBDD is taken (this corresponds to a minimum number of rows with different function values in the pair (1)). In the case of a tie again, a variable is selected randomly.

To aid LUT cascade synthesis, the program tool HEDIT (Heuristic Iterative Decomposition Tool) has been developed [5].

3. ALLOCATORS

An $m \times n$ allocator is a unit that accepts m requests on its inputs for n distinct resources and generates grants on its outputs. It is therefore more general than an arbiter that makes decision on only a single resource. Any particular problem and its solution can be represented in terms of two binary-valued $m \times n$ matrices, a request matrix R and a grant matrix G. One requester may ask one or more resources, but

1. at most one grant for each input (requester) may be asserted;
2. also, at most one grant for each output (resource) can be asserted.

By means of a bipartite graph representation, the allocation can be formulated as bipartite matching problem and solved exactly, yielding the maximum possible number of assignments (maximum matching). Maximal matchings are those where no additional requests can be serviced without removing one of the existing grants.

As the exact solution of the allocation problem is too time consuming in software or too costly in hardware, we usually put up with approximate solution in hardware. So called input-first separable allocators use a set of arbiters to select one input per requester and then an arbiter for each resource to solve simultaneous requests for the same resource. Output-first separable allocators do these decisions in the opposite order. Some allocators (PIM, iSLIP) reach a decision in several iterations. Here we will design a popular $n \times n$ wavefront allocator that arbitrates among requests for inputs and outputs simultaneously, [6]. It works by granting row and column tokens to a diagonal group of cells, in effect giving this group priority.

A diagonal group k contains cells xij so that

$$(i + j) \bmod n = k.$$

Let us follow the example on Fig.3. Let the priority be initially given to a diagonal group $(i + j) \bmod n = 3$. This is done by asserting signal p3. Element 21 got the grant, remaining elements send tokens down and to the right.

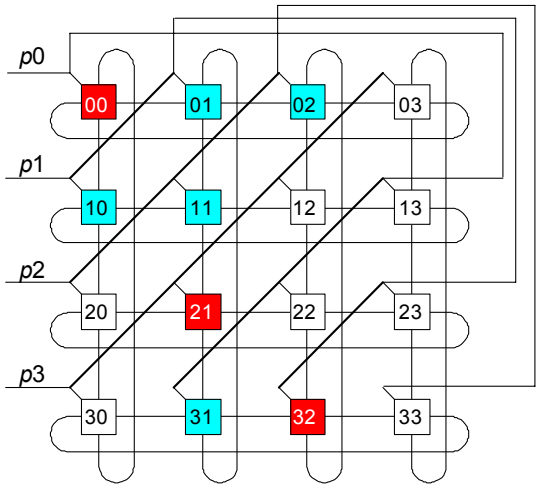

Fig. 3. 4 × 4 wavefront allocator

An element will get a grant if it received both row and column tokens. If only one token arrives or if the element has no request, tokens will pass through it. The highlighted grants g21, g00, g32 will be issued in response to 8 requests shown in the example. In the next round the next diagonal group is selected, and so on. This ensures fairness of allocations.

For example grant g00 is asserted under the following conditions:

```
g00  = p0*!p1*!p2*!p3*r00 +
+!p0 *!p1*!p2*p3*r00*!r03*!r30 +
+!p0*!p1*p2*!p3*r00*!r02*!r03*!r20*!r30+
+!p0*p1*!p2*!p3*r00*!r01*!r02*!r03*!r10*
!r20*!r30;
```

Similar equations can be written for remaining 15 grant signals.

4. EXPERIMENTAL RESULTS

The above wavefront allocator represents 16 Boolean functions of 20 variables. The iterative decomposition has been done with output grouping [4]. Logically grants related

to one requester have been included in a single group. For example, in VHDL we have had

```
entity g0x is
   port(p0, p1, p2, p3, r00, r01, r02,
r03, r10, r11, r12, r13, r20, r21, r22,
r23, r30, r31, r32, r33 : in std_logic;
g00, g01, g02, g03 : out std_logic);
end g0x;
```

The generic iterative decomposition using HIDET tool gave us an optimal (in terms of our heuristics) order of individual variables removed one after another from the leaves to the root of the MTBDD (from the last cell in the cascade backwards). Beside all intermediate function tables, we have obtained the following order of input variables (group g0x ordered as shown above, p0 ≡ 0, …, r33 ≡ 19):

0 4 8 1 2 3 5 6 7 9 12 13 14 15 11 10 16 17 19 18.

Parameters of a generic cascade with a single input variable entering each cell are displayed, cell after cell, in natural order (i.e. in the opposite order than they were obtained by the decomposition procedure) in Table 2. Each column represents one cell, with the following info from the top down:

1. index of a variable
2. the number of all rows in the LUT (the local width of the MTBDD)
3. the number of LUT rows with the same values in the pair (degenerate decision nodes).

Table 2. Parameters of a generic LUT cascade for 4 × 4 wavefront allocator

WFALOC4

										Legend:
18	19	17	16	10	11	15	14	13	12	1
1	2	3	4	4	4	4	7	10	12	2
0	1	1	1	3	4	0	3	5	7	3

9	7	6	5	3	2	1	8	4	0
11	11	10	9	8	7	6	5	5	5
3	10	9	8	7	6	5	5	5	4

From Table 2 one can obtain two global parameters of the MTBDDs/cascades: cost (the number of true decision nodes) and the maximum number of rails between cells, Table 3. LUT cascades with LUT size limited from 5 to 8 inputs (cell inputs) and with a specified cascade length ("cells") are listed here, too. For comparison Xilinx FPGA synthesis tool produced an implementation for group g0x with 24 4-input LUTs in 6 logic levels. It turns out that we should be able to get the same or better performance with LUT cascades starting with 6 LUT inputs. The cascades with 6-, 7-, and 8-input LUTs are shown in Fig.4.

The area for wiring LUT cascades promises to be much lower, whereas the area of cascaded LUTs themselves slightly higher due to their coarser granularity. The large area for the interconnections in an FPGA is thus absorbed in the larger LUTs in the cascade. LUT cascades are easy to

Table 3. Comparison of LUT cascade designs of the 4 × 4 wavefront allocator

MTMDD			LUT cascade		
node	cost	levels	cells	width	cell inputs
1 bin.	34	17	11	≤ 4	≤ 5
2 bin.	11+11	9	6	≤ 4	≤ 6
3 bin.	6+6+5	6	4	≤ 4	≤ 7
4 bin.	5+3+3+4	5	3	≤ 4	≤ 9

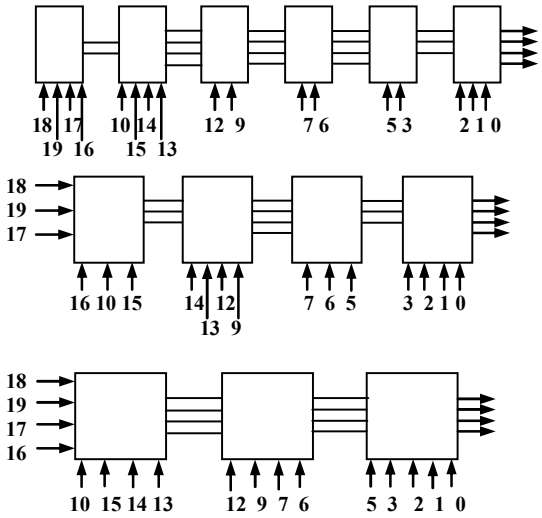

Fig. 4. The wavefront allocator - cascades for group g0x with 6-, 7-, and 8-input cells

pipeline. It is sufficient to insert pipeline registers between cascade cells. The allocation latency is still given by the number of cells, but one allocation is generated each memory cycle.

Generic MTBDDs obtained in the allocator synthesis can easily be converted to multi-valued heterogeneous diagrams (MTMDDs) and used for firmware synthesis [2], [8]. Four alternatives with different cost and speed (as given by DD levels) are shown in Table 3. The number of dispatch tables of size 2^1, 2^2, 2^3, and 2^4 (added up in this order into the cost) and the number of executed microinstructions (levels) of four possible micro-programs are given in Table 3. The size of the g0x micro-program is obtained as the size of all dispatch tables together, e.g. for nodes with 2 binary decision variable we have

11*2 +11*4 = 66 microinstructions.

For decision nodes with three and four decision variables we get similarly 76 and 110 microinstructions. The allocation delay is given by the number of executed microinstructions (MTMDD levels in Table 3). The best delay*size product is obtained for the MTMDD with up to 3

binary variables per node. In order to utilize nodes with variable number of control variables, the micro-program controller with 16-way branch support [2] should be used.

However, to implement not only g0x part but the whole allocator, the complete MTBDD has to be constructed. The allocation delay is the same, but the size of the complete microprogram is almost 4-times larger.

5. CONCLUSIONS

The presented method of LUT cascade synthesis aided by HIDET tool proved to be suitable for synthesis of wavefront allocators. Allocators, as well as other digital systems frequently used in practice, have relatively low complexity, what makes their cost-effective cascade implementations possible. Beside easy interconnection there are other advantages of cascade implementation. Testing of LUT cascades reduces to a problem of testing RAM modules. Fault tolerance techniques for memories such as SECDED are also applicable. Due to a highly developed memory technology the power consumption is very low for RAMs and it only remains to verify experimentally real power savings for specific applications.

Effectiveness of LUT cascades can be derived from the size of complete function table and the aggregate capacity of all LUTs in a cascade. Most of the functions that occur in digital design are decomposable effectively into cascades. One exception is the class of binary multipliers: for all possible variable orderings is the BDD size exponential [7].

Future research should address other classes of allocators such as separable allocators, iSLIP, Lonely output allocator, etc. [6]. Also a more general specification of multiple-output Boolean functions as a set of Boolean expressions for individual binary outputs with overlapping terms is not supported by the present version of the HIDET tool as yet and is planned as one option in the next version. Finally the quality of our optimization heuristics for variable ordering should be compared with exhaustive testing of all permutations of variables and with other heuristic approaches.

6. REFERENCES

[1] 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, Yokohama, Japan, 2005.

[2] V. Dvořák: "LUT Cascade-Based Architectures for High Productivity Embedded Systems", In: *International Review on Computers and Software*, Vol. 2, No 4, Naples, Italy, pp. 357-365, 2007.

[3] S.N. Yanushkevich, D.M. Miller, V.P. Shmerko, R.S. Stankovic: *Decision Diagram Techniques for Micro- and Nanoelectric Design Handbook*. CRC Press, Taylor & Francis Group, Boca Raton, FL, 2006.

[4] T. Sasao and M. Matsuura: "BDD representation for incompletely specified multiple-output logic functions and its applications to functional decomposition", *Design Automation Conference*, pp.373-378, June 2005.

[5] Mikušek Petr, Dvořák Václav: "On Lookup Table Cascade-Based Realizations of Arbiters:, In: *11th EUROMICRO Conference on Digital System Design* DSD 2008, Parma, IT, IEEE CS, pp. 795-802, 2008.

[6] W.J. Dally, B.Towles: *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers/ Elsevier, San Francisco, CA, 2003.

[7] R.E.Bryant: "On the complexity of VLSI implementations and graph representations of boolean functions with applications to integer multiplication". *IEEE Transactions on Computers*, Vol. 40, pp.205–213, 1991.

[8] S. Nagayama, T. Sasao: "Code Generation for Embedded Systems Using Heterogenous MDDs". *11th Conference on Synthesis And System Integration of Mixed Information technologies* (SASIMI 2003), Hiroshima, April 3-4, 2003, pp.258-264.

7. 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).