Logic Synthesis for Cellular Architecture FPGA using BDD

Gueesang Lee
Dept. of Computer Science
The Chonnam National University
Kwangju, Korea
(Tel)082-62-520-6895, (FAX)082-62-524-0020
e-mail gslee@chonnam.chonnam.ac.kr

Abstract— In this paper, an efficient approach to the synthesis of CA(Cellular Architecture)-type FPGAs is presented. To exploit the array structure of cells in CA-type FPGAs, logic expressions called *Maitra terms*, which can be mapped directly to the cell arrays are generated. In this approach, a BDD is modified so that each node of the BDD has another branch which is an exclusive-OR of the two branches of a node. Once the modified BDD is obtained, a traversal of the BDD is sufficient to generate the Maitra terms needed. Since a BDD can be traversed in $O(n)$ steps, where n is the number of nodes in the BDD, Maitra terms are generated very efficiently. This also removes the need for generating minimal SOP or ESOP expressions which can be costly in some cases. The experiments show that the proposed method generates better results than existing methods.

I. Introduction

Because of high programmability and short turn around time, FPGAs have been considered as the most attractive devices for fast prototyping. Various architectures are employed to realize such devices. The most popular ones include Multiplexor based FPGAs and LUT-type FPGAs for which there have been extensive studies in developing their synthesis methods. CA(Cellular Architecture)-type FPGAs are introduced rather recently and this category of FPGAs are characterized by relatively small logic blocks and local connectivity between them. Figure 1 shows the basic structure of two dimensional array of cells in a CA-type FPGAs[1]. A generic model of the CA-type FPGAs consists of the array of cells which are connected to their neighbours and local buses, vertical and horizontal, to carry the signals to and from the other cells. For simplicity, we assume that the number of inputs to a cell is limited to two and the number of outputs of a cell is limited to one. Furthermore, only one input is taken from the local bus and the logic blocks can realize an inverter, an AND, OR, EXOR, NAND gate, a wire and their combinations. Figure 10 shows one such example. In Atmel 6000 series FPGAs, the architecture limitation is the same as the above except that the number of inputs to a logic block is limited to three and outputs to two.

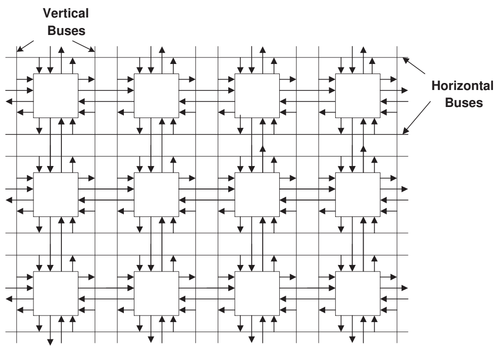

Fig. 1. A generic structure of CA-type FPGAs

For the synthesis of CA-type FPGAs, several approaches have been presented. The first group of works utilize various decision diagrams including FDD(Function Decision Diagram)[2] and KDD(Kronecker Decision Diagram)[3, 4] and tree structures[5]. However, these approaches suffer from the drawback that they can waste a large amount of cells due to its tree like structure. An algebraic approach presented by Perkowski et al.[1, 6] alleviates such deficiencies and provides a well-defined theoretical background for the manipulation of Boolean functions applicable to Cellular Architecture two dimensional arrays. The synthesis model of [1] is composed of two planes: the complex(input) and collecting (output) plane. It is similar to the conventional PLA architecture, but a linear array of cells can implement a broader class of Boolean functions than a simple *product term* in PLA. Since each cell can realize an AND, OR, EXOR or their combinations, the outputs of the cell arrays constitute a special class of Boolean functions called *Maitra terms* which are named from *Maitra Cascade*[7].

The problem of interest in this paper is the generation

[0]This work was supported by Korean Ministry of Education through Inter-University Semiconductor Research Center(ISRC 96-E-2013) in Seoul National University.

of the minimal number of Maitra terms particularly when the BDD of a function is given. After the Maitra terms are generated, folding techniques[1] can be applied to further reduce the number of cells needed. We further assume that the Maitra terms are collected by exclusive-OR operations as in [1] to compare the results. However we expect it will not be difficult to generalize our approach so that Maitra terms are collected by arbitrary cell functions.

II. Preliminaries

Boolean expressions which can be directly mapped to linear cell arrays are defined and are called Maitra terms. The following definitions are from [1].

Def) A *forward Maitra term* is defined recursively as follows.

1. a literal is a forward Maitra term.
2. if M is a forward Maitra term and a is a literal then $M \cdot a, M \oplus a, M + a$ are forward Maitra terms, where a variable never appears more than once in the string.

Def) A *reverse Maitra term* is defined recursively as follows.

1. a literal is a reverse Maitra term.
2. if M is a reverse Maitra term and a is a literal then $a \cdot M, a \oplus M, a + M$ are reverse Maitra terms, where a variable never appears more than once in the string.

Def) A *bidirectional Maitra term* is $M_1 \alpha M_2$ where α is a two input function, M_1 is a forward Maitra term, M_2 is a reverse Maitra term and a variable never appears more than once in the string.

Def) A *complex term* is a forward, reverse or bidirectional Maitra term.

Note that a specific ordering is imposed on input variables to form Maitra terms. It is not difficult to see that a complex term can be implemented in a cell array.

Example 1) A logic expression $(a + b)c + d$ can be implemented in cell arrays as shown in Figure 2. However, $(a + b)(c + d)$ is not a Maitra term and it needs another routing wire to be realized in the cellular architecture[6]. Note that the input ordering of a,b,c,d is imposed in this example.

In this paper, forward Maitra terms are used for simplicity. Therefore *Maitra terms*(or *complex terms*) will refer to *forward Maitra terms* in the rest of this paper.

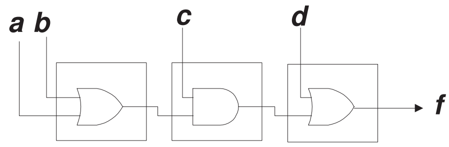

Fig. 2. Maitra term $(a + b)c + d$ implemented in a cell array

Given a Boolean function $f(V)$ and a variable $v \in V$, $f(V)$ can be expressed as:

$$\begin{aligned} f(V) &= v \cdot f(v=1) + v' \cdot f(v=0) \\ &= v \cdot f(v=1) + v' \cdot f(v=0) \\ &= v \cdot f(v=1) \oplus (1 \oplus v) \cdot f(v=0) \\ &= v \cdot (f(v=1) \oplus f(v=0)) \oplus f(v=0) \end{aligned}$$

and

$$\begin{aligned} f(V) &= v \cdot f(v=1) + v' \cdot f(v=0) \\ &= (1 \oplus v) \cdot f(v=1) \oplus v' \cdot f(v=0) \\ &= f(v=1) \oplus v \cdot (f(v=1) \oplus f(v=0)) \end{aligned}$$

In short, a logic function can be represented as:

$$f(V) = v \cdot f(v=1) \oplus v' \cdot f(v=0) \quad (1)$$
$$= v \cdot g \oplus f(v=0) \quad (2)$$
$$= f(v=1) \oplus v' \cdot g \quad (3)$$

where $g = f(v = 1) \oplus f(v = 0)$. We call these equations as *Davio expansions*[3] and particularly equation 2 and equation 3 are called as *positive davio decomposition* and *negative davio decomposition* respectively.

III. Generation of Maitra terms using modified BDDs

Our approach is based on the fact that terms derived by Davio expansions are Maitra terms. Davio expansions generate product terms summed by exclusive-OR operations and by definition product terms are Maitra terms. Therefore one of the decompositions of Davio expansions is selected which results in the smallest number of terms.

Consider a function f and an input variable v, where the number of minimal Maitra terms needed for $f(v = 0)$, $f(v = 1)$ and $f(v = 0) \oplus f(v = 1)$ are K,L and M respectively. If M is larger than K or L, using Shannon decomposition will result in smaller number of terms, while positive(or negative) Davio decomposition is better if M is smaller than K or L. More precisely, if smaller two of K,L and M are M and K, it is better to use positive Davio decomposition and if smaller two of K,L and M are M and L, it is better to use negative Davio decomposition. For example, given a function f in Figure 3,

$$f(a = 0) \quad = \quad (b \oplus c'd') \oplus b'cd \rightarrow 2 \ terms$$

ab \ cd	00	01	11	10
00	1		1	
01		1	1	1
11			1	
10				

Fig. 3. Function f in which $f(a=0) \oplus f(a=1)$ needs only one Maitra term

$$\begin{aligned} f(a=1) &= cd \rightarrow 1\ term \\ f(a=0) \oplus f(a=1) &= b \oplus c \oplus d \rightarrow 1\ term \end{aligned}$$

Therefore if Shannon decomposition is used, it will result in 3 Maitra terms as shown below.

$$\begin{aligned} f &= a' \cdot f(a=0) + a \cdot f(a=1) \\ &= a'(b \oplus c'd') \oplus ab'cd \end{aligned}$$

However if negative Davio decomposition is used, it needs only two Maitra terms.

$$\begin{aligned} f &= a' \cdot (f(a=0) \oplus f(a=1)) \oplus f(a=1) \\ &= a'(b \oplus c \oplus d) \oplus cd \end{aligned}$$

Furthermore when a cofactor is a constant, two terms are combined to make one Maitra term. Without loss of generality, assume that $f(v=1) = 1$. Then,

$$\begin{aligned} f(V) &= v \cdot f(v=1) + v' \cdot f(v=0) \\ &= v + v' \cdot f(v=0) \\ &= v + f(v=0) \end{aligned}$$

Also if $f(v=0) \oplus f(v=1) = 1$, the positive and negative Davio decompositions are simplified to get reduced number of Maitra terms as:

$$f(V) = v \oplus f(v=0) \rightarrow 1\ term \tag{4}$$

or

$$f(V) = f(v=1) \oplus v'. \rightarrow 1\ term \tag{5}$$

Therefore terms generated by Davio expansions can be used to form Maitra terms and by selecting appropriate decomposition type, the smallest number of Maitra terms can be generated. Another simple example was given in equation 4 and equation 5. If two cofactors are complementary to each other, or $f(v=0) \oplus f(v=1) = 1$, it is obvious that positive or negative Davio decomposition is advantageous to Shannon decomposition in generating smaller number of Maitra terms. Moreover if the number of Maitra terms need for $f(v=0)$ is smaller than that for $f(v=1)$, it will be better to use equation 4 rather than equation 5 and vice versa.

```
cost(d) { /* returns the no. of Maitra terms
needed for a BDD node d */

    if (d is visited) return d-> cost;
    if (d is the last variable in the path) return 1;
    l = cost( d->then);
    r = cost( d->else);
    x = cost( d->xor);
    /* d->xor = d->then ⊕ d->else; */
    return d->cost = l+r+x-max(l,r,x);
}
```

Fig. 4. Estimation of the number of Maitra terms

To implement this procedure, the BDD representing the given function is modified so that each node has another edge representing the exclusive-OR of two cofactors($f(v=0) \oplus f(v=1)$) to use Davio expansions. Once a BDD is modified, the smallest number of Maitra terms which can be generated from the modified BDD(we call it the *cost* of a node) is calculated for each node of the BDD by the algorithm given in Figure 4.

Since traversing the nodes only once is enough to calculate the costs, it takes $O(n)$ steps, where n is the number of nodes in the modified BDD. Once the cost of a node is decided, the set of Maitra terms in the modified BDD can be generated also in $O(n)$ time by the algorithm shown in Figure 5.

This approach is very similar to the construction of KDD which uses one of Davio expansions in each node of the decision diagram, but the cost function by which a decomposition type of a node is to be decided is the number of Maitra terms in our approach while the number of nodes in the diagram is the most important factor considered in constructing a KDD[3, 4].

Although our method works very fast for single output functions, in the case of multi-output Boolean functions generating Maitra terms for each output separately results in quite large number of Maitra terms in total. In our experiment, for each output of the function the Maitra terms of other outputs are considered and they are adopted when helpful(reduce the cost of the subfunction). This may look time-consumable, but since cost estimation can be done very fast, the entire time to generate the Maitra terms were manageable.

IV. Variable ordering for the modified BDD

Since the performance of our approach depends on the variable ordering, finding a good variable ordering is an important problem. In this paper, we utilized the structure of cell arrays in which primary outputs can be redi-

```
gen_Maitra(d)
/* generates Maitra terms for a BDD node d*/
{
    if (d is the terminal node ) {
            generate the path from the root to d;
            return;
    }
    l = cost( d->then);
    r = cost( d->else);
    x = cost( d->xor);
    select the smallest two of l,r and x;
    if( l and r are selected) {
    /* use Shannon decomposition*/
            buf = buf + "v· ", gen_Maitra(d->then);
            /* buf records the path */
            buf = buf + "v'· ", gen_Maitra(d->else);
            /* v is the variable in d*/
    }
    if( l and x are selected) {
    /* use positive Davio decomposition*/
            gen_Maitra(d->then);
            /* nothing to add to the path*/
            buf = buf + "v'· ", gen_Maitra(d->xor);
    }
    if( r and x are selected) {
            /* use negative Davio decomposition*/
            gen_Maitra(d->else);
            buf = buf + "v· ", gen_Maitra(d->xor);
    }
    return ;
}
```

Fig. 5. Algorithm for the generation of Maitra terms

```
main(f) /* generates Maitra terms for f */
{
    lst = list of Maitra terms generated;
    for ith output f_i of f {
            if (cost(BDD(f_i)) > cost(BDD(l ⊕ f_i))
                f_i = f_i ⊕ l; /* l ∈ lst */
            gen_Maitra(BDD(f_i));
    }
}
```

Fig. 6. Generation of Maitra terms for multi-output functions

rected to vertical buses and used as if they were primary inputs to the Maitra terms. Therefore Maitra terms are redefined so that they contain other outputs as one of its input variables. To exploit this property, BDD should be constructed so that the BDD of a subfunction is contained to that of other functions. Consider two subfunctions f_i and f_j of a multi-output function f such that support(f_i) $\subset$ support(f_j), where support(h) is the set of variables used in function h. In this case, the BDD of f_i should be placed at the bottom of the BDD of f_j in the ordering of their variables as shown in Figure 7. Suppose that

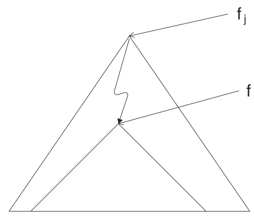

Fig. 7. Construction of BDD so that f_j contains the BDD of f_i

$$
\begin{aligned}
f_0 &= ab \oplus a'cd \\
f_1 &= e(ab \oplus a'cd) \\
&= ef_0.
\end{aligned}
$$

As shown in Figure 10, Maitra terms ab and $a'cd$ are connected to horizontal lines and they are collected to form f_0. Now f_0 is connected to a horizontal line and again it is redirected to the left vertical line which can be fed to other cells just as if it was another input line. Note that f_1 is composed of only one Maitra term ef_0 instead of two Maitra terms, abe and $a'cde$.

To construct a BDD in which subfunctions are shared by other outputs, the variables are ordered by computing the partial orders for variable subsets as shown in Figure 8. Suppose that support(f_0)={a,b,c,d,e} and support(f_1)={c,d,e}. Then a partial ordering between supports are established as

$$\{a, b, c, d, e\} \succ \{c, d, e\}.$$

Note that if support(f_0) $\not\subseteq$ support(f_1) and vice versa, no such partial ordering exists. After partial orderings are obtained, a graph $G = (V, E)$ is constructed, where $v \in V$ are subsets of input variables and $e = (u, v) \in E$ iff $u \succ v$. Then a path from the root to a terminal node represents a total ordering which can be directly used to generate the variable order of the BDD. For example, if

$$\{a, b, c, d, e\} \succ \{c, d, e\} \succ \{c, e\},$$

then variables can be selected from the subsets $\{a, b\}, \{d\}$ and $\{c, e\}$. There could be many paths from the root to a terminal node in graph G. Certain cost functions, for

```
order(f) /* get variable ordering for f*/
/* f_i is the ith output of f*/
{
  for i = 1, num_POs {
    for j = 1, num_POs {
      if support(f_i) ⊂ support(f_j) {
        support(f_j) ≻ support(f_i);
      }
  }
  /* ≻ is a partial ordering between supports*/
  Construct a graph G=(V,E)
    v ∈ V = supports, e = (u, v) ∈ E iff u ≻ v.
  Find a path from the root to a leaf with minimal cost
  Select variables from the total ordering(the path of G)
}
```

Fig. 8. Variable ordering by partial orders of the supports

example the number of product terms of the subfunctions, can be associated with the nodes of G to select a path which derives the most effective variable ordering.

With the variable ordering described above, we adopted the *sifting algorithm*[8] so that the ordering minimizes the cost function of the BDD as shown in Figure 9. In this algorithm, a variable is selected which guarantees the minimum cost when it is moved to the top of the BDD with other nodes remaining in the current position. Each time a variable is selected, current BDDs are replaced by new BDDs which are generated by substituting a constant value to the selected variable. This method resembles the sifting algorithm in [8], but we used the basic operations provided by the BDD package instead of exchanging adjacent variables.

V. Experimental Results

Suggested algorithm in this paper was implemented in C and run on IBM PC (Pentium 133MHz) Unix system for experimentation. The results were verified by comparing the BDD constructed by Maitra terms with the original function for each benchmark circuit. For the construction of modified BDDs, we employed the BDD package developed in Carnegie Mellon University. Since the BDD package uses complementary edges, it is well suited to our approach in which the cost of a function and its complement should have the same value. Note that the inputs to a cell can be inverted if necessary in our circuit model.

Table I show the comparison of Maitra terms(complex terms in [1]). To compare the results, Table I listed benchmarks which appear in [1] only. The column "C1" refers to the result with ordering obtained by considering the containment of subfunctions as shown in in section IIIand

```
modified-sift(f) /* get variable ordering for f*/
/* f_i is the ith output of f*/
{
  FUNC_LIST = all POs initially
  for i=1,num_PIs {
     for j=1,num_PIs {
        if((v_j is selected) continue;
        new_cost_j = 0;
        for each function in FUNC_LIST {
           new_cost_j += min23( cost(f(v_j=0)),
              cost(f(v_j=1)), cost(f(v_j=0)⊕f(v_j=1)))
           /* min23() : sum of the largest two */
        }
     }
     order[i] = Select v_j of the smallest new_cost_j
     for each function f in FUNC_LIST {
        delete f from FUNC_LIST
        Put f(v_j=0) and f(v_j=1) into FUNC_LIST;
     }
  }
}
```

Fig. 9. Modified sifting for the variable ordering

"R1" refers to the best result obtained from 20 random orderings. Also the column "MS" shows the results with modified sifting. The results show that modified sifting is more efficient than the other ordering techniques even though they need still less number of terms than [1]. Also the column "C1" shows the best results in many benchmarks while a lot worse in other circuits.

VI. Discussions and future research

This paper suggested a synthesis method for Cellular Architecture-type FPGAs using modified BDDs. For the decomposition of a node in the BDD of the given function,

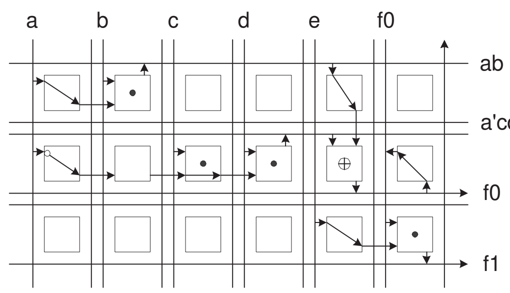

Fig. 10. An example of a Maitra term implemented in a cell array

TABLE I
COMPARISON OF THE NUMBER OF MAITRA TERMS

name	[1]	C1	time	R1	time	MS	time
5xp1	33	23	0.14	23	1.18	24	0.58
card4	30	20	0.15	20	1.98	20	0.74
clip	57	86	0.91	58	3.95	52	1.93
clog8	84	98	0.95	98	5.53	76	2.35
cmlp4	54	68	0.46	65	4.51	70	2.49
cnrm	52	55	0.65	68	6.25	55	2.18
cu	15	19	0.11	17	0.82	18	0.86
f51m	30	21	0.11	21	1.25	21	0.59
inc	26	30	0.25	30	1.87	31	0.86
mlp3	17	17	0.08	17	0.74	17	1.40
rd53	13	9	0.05	9	0.54	9	0.16
rd73	36	19	0.20	19	2.66	19	0.54
sao2	26	31	0.28	30	2.37	32	1.62
t481	18	8	0.88	10	18.33	10	2.13
vg2	179	138	1.09	89	4.83	90	15.10
Total	670	642	6.31	574	56.14	544	32.53

Davio expansions are employed to select the smallest number of Maitra terms derivable from the BDD. To apply Davio expansions, a BDD is modified so that each node of the BDD has another edge pointing to the exclusive-OR of two cofactors of the node. Once the BDD is modified, the cost function which is the number of Maitra terms derivable from the Davio expansions are calculated for each node. Since traversing a BDD(or a DAG)needs only $O(n)$ steps, where n is the number of nodes in the BDD(or a DAG), computing cost functions in the nodes and the generation of the smallest number of Maitra terms in the modified BDD are performed very efficiently.

The contribution of this paper is the presentation of a method which is simple and efficient to generate Maitra terms directly from a BDD and it does not require minimization tools for SOP or ESOP expressions which can be costly in some cases. Since BDD is widely used as an effective tool for the manipulation of Boolean functions, our approach is expected to be applicable to a large area of design tools.

However, for multi-output functions, sharing of the Maitra terms among different output functions should be considered. In this paper, previously generated Maitra terms are considered for sharing them with other output functions. Since the cost estimation of a BDD node can be done vaery fast, this scheme works well for most benchmark circuits. We believe that there are also linear time (in the number of node of the BDD) algorithm to generate Maitra terms for multi-output functions which employs the technique given in this paper and we are currently working on it.

For future research, decision of the appropriate variable ordering should be studied. Even though we tried random ordering and still acceptable results are obtained, a systematic method of selecting the best variable ordering for the BDD should be developed. Also we assumed that the collecting plane of the Maitra terms only exclusive-OR operation is allowed. However, since the cells in CA-type FPGA provides virtually any two input functions, our approach will be generalized to use arbitrary operations including inclusive and exclusive-OR to collect the Maitra terms.

REFERENCES

[1] A. Sarabi, N. Song, M. Chrzanowska-Jeske, and M. A. Perkowski. A comprehensive approach to logic synthesis and physical design for two-dimensional logic arrays. In *Design Automation Conference*, pages 321–326, June 1994.

[2] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagram. In *EDAC*, pages 43–47, 1992.

[3] I. Schafer, M.A. Perkowski, and H. Wu. Multilevel logic synthesis for cellular fpgas based on orthogonal expansions. In *Proc. IFIP WG 105 Workshop on Applcations of the Reed-Muller Expansion in Circuit Design, Hamburg,Germany*, pages 42–51, September 1993.

[4] R. Drechsler, A. Sarabi, M.Theobald, B.Becker, and M.A.Perkowski. Efficient representation and manipulation of switching functions based on ordered kronecker functional decision diagrams. In *Design Automation Conference*, pages 415–419, 1994.

[5] L. F. Wu and M. A. Perkowski. Minimization of permuted reed-muller trees for cellular logic programmable gate arrays. *H. Gruenbacher and R, Hartenstein (eds.),LNCS*, pages 78–87, 1993.

[6] N. Song and M. A. Perkowski. A new design methodology for two-dimensional logic arrays. In *Proc. of IWLS, Tahoe City, CA*, May 1993.

[7] K. K. Maitra. Cascaded switching networks of two-input flexible cells. *IRE Trans. Electron. Comput.*, pages 136–143, 1962.

[8] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *ICCAD*, pages 42–47, 1993.