Boolean Manipulation with Free BDDs. An Application in Combinational Logic Verification. *

Jordan Gergov and Christoph Meinel [OT1]

[a]FB IV – Informatik, Universität Trier, D-54286 Trier, Germany

Free Binary Decision Diagrams (FBDDs), with respect to a predefined type, provide a canonical representation of Boolean functions, and allow efficient solutions of the basic tasks in Boolean function manipulation similarly as the well–known OBDDs do. But in contrast to OBDDs, typed FBDDs allow more (sometimes exponentially more) succinct representations.

In the following, we demonstrate the power of the FBDD–based data structure by showing that the combinational logic circuit C_{HWB} proposed by Bryant that computes the hidden weighted bit function HWB *can be verified in polynomial time and space* $2n^4 \log n$ *in terms of FBDDs while, for principal reasons, verification in terms of OBDDs needs exponential space and time.*

Keyword Codes: B.6.3; F.2.3;
Keywords: Logic Design Aids, Data Structures

1. INTRODUCTION

By means of hardware description languages, circuits can be described at a very high level of abstraction which allows the designer to specify the behavior of a circuit beforehand realizing it. In order to validate these specifications and to verify a designed circuit, against its specification, formal methods were developed which lead to problem descriptions in terms of Boolean functions. Then the verification problem is solved by analyzing and manipulating these functions. Here we consider the problem of determining whether a combinational logic circuit C correctly implements a given specification S, i.e. to test whether $f_C = f_S$, if f_C and f_S are the functions realized by C and S, respectively. A common way for doing this [e.g. Eve91] is to construct representations for f_C and f_S (usually in terms of the primary inputs), and then to perform the equivalence test for both functions in terms of these representations. Hence, efficient algorithms for solving the verification task under consideration require (i) efficient algorithms for deriving the representations of the involved Boolean functions (e.g., from a circuit network description) as well as (ii) efficient algorithms for solving the equivalence test or similar tests such as satisfiability or tautology test in terms of these representations.

*Proc. of *13th IFIP World Computer Congress 1994*, Hamburg,
IFIP Transactions A-51 "Technology and Foundations", Vol. 1, 309–314, North Holland, 1994.

In the past a great variety of representations of Boolean functions has been investigated like *truth tables, disjunctive* (DNF) and *conjunctive normal forms* (CNF), *Reed–Muller–expansions*, and various types of *formulas, Boolean circuits*, or *branching programs* (BDD). However, a systematic inspection of the different succinct representation schemes [e.g. GJ79, GM92] has shown that almost all of them do *not* support efficient solutions of the mentioned basic tasks of Boolean function manipulation. Even worse, performing the basic tasks often requires the solution of NP–hard problems. The only exceptions seem to be BDD–based representations [e.g. Bry86, Mei92, GM92] that provide with so-called OBDDs (i.e., BDDs which test input variables at most once in a predefined order) the state-of-the-art data structure for Boolean function manipulation [e.g. FFK88, CMB90, MWBS88, Bry92].

The bottleneck of most OBDD–applications is the huge size of the representations since the total computation merely remains tractable as long as the involved OBDDs remain of reasonable size. Since it is well known that OBDDs are restricted FBDDs (i.e., binary decision diagrams that test, on each path, each input variable at most once), and that FBDD–representations are often much more (sometimes even exponentially more) concise than OBDD–representations, we proposed an FBDD–based data structure for Boolean function manipulation [GM92b, GM93].

In the following we demonstrate the power of this FBDD–based data structure in the case of the paradigmatic example of the verification of the well-known combinational circuit design C_{HWB} proposed in [Bry91] for the hidden weighted bit function HWB (Figure 1). We show that the entire circuit C_{HWB} can be verified in terms of FBDDs within space $2n^4 \, log \, n$, while it is not even possible to represent single gates of it (e.g. the output gate) by polynomial size OBDDs. Due to the limitation in space, all proofs are omitted. They can be found together with more details in [GM93b].

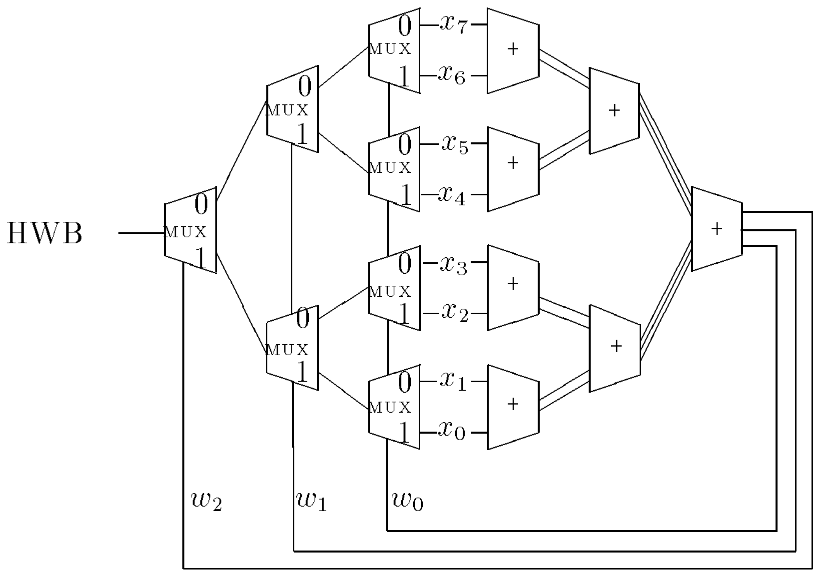

Figure 1. *Circuit design C_{HWB} for the hidden weighted bit function HWB for $n = 7$. The w_0, w_1, w_2 are the bits of the binary represented weight of $(x_1, x_2, \ldots, x_7)$.*

2. FBDDs AS DATA STRUCTURE FOR EFFICIENT BOOLEAN FUNCTION MANIPULATION

A *binary decision diagram (BDD)* over a set $X_n = \{x_1, \ldots, x_n\}$ of Boolean variables is a directed acyclic graph with one source and at most two sinks labelled 0 and 1. Each non-sink node v is labelled with a Boolean variable $l(v) \in X_n$ and has two outgoing edges, one labelled with 0 and the other with 1. The *then–son* of v is reached via the 1–edge, the *else–son* is reached via the 0–edge. (In pictural representations we do not indicate the edge-labels if the 0-edge is drawn left of the 1-edge.) The *computation path* for an input $a = (a_1, \ldots, a_n)$ starts at the source. At an inner node with label x_i, the outgoing edge with label a_i is chosen. The BDD P *represents* a Boolean function $f \in I\!B_n$ if the computation path for each input a leads to the sink labelled $f(a)$. f is sometimes denoted by f_P. A BDD is called *free binary decision diagram* (FBDD) if, on each path, each variable is tested at most once. An *ordered binary decision diagram* (OBDD) is an FBDD with the property that on each path the variables are tested in a predefined order.

An *(FBDD-)type* is defined similarly as an FBDD with the only exception that it possesses merely one sink labelled t. A type τ is said to be *complete* over $X_n = \{x_1, \ldots, x_n\}$ if, on each source-to-sink path, each variable is tested. An FBDD P is said to be of type τ (shortly called a τFBDD) if, for each input vector $a \in \{0,1\}^n$, the variables on the computation path of a in P appear in the same order as on the corresponding source-to-sink path in τ.

Theorem 1 [GM92b]. *Let τ be a complete type, and let P_1, P_2 be τFBDDs.*
1. *(Universality.) Each Boolean function f over X_n can be represented by means of a τFBDD.*
2. *(Canonicity.) The reduced τFBDD for f is uniquely determined.*
3. *(Efficient Composition.) Let f_1, f_2 be the Boolean functions computed by τFBDDs P_1, P_2, respectively. For every binary Boolean operation $*$, the reduced τFBDD P for $f = f_1 * f_2$ can be constructed in time $O(size(P_1) \cdot size(P_2) \cdot size(\tau))$.* $\Box$

3. HIDDEN WEIGHTED BIT FUNCTION HWB

The example of the *hidden weighted bit function HWB* demonstrates the efficiency of FBDD–representations in comparison with OBDD–representations.

Proposition 2. [Bry91] *For each variable ordering, OBDD-representations of HWB are of exponential size $2^{\Omega(n)}$.* $\Box$

In contrast, HWB can be represented by means of quadratic size FBDDs. To explain this we use a construction proposed by Bryant [Bry92b] which is based on the computation of appropriately chosen restrictions of HWB. If $x_{i:j}$ represents the variables $x_i, \ldots, x_j$ then

$$H(x_{i:j}) = \begin{cases} 0 & \text{if } wt(x_{i:j}) = 0 \\ x_{i+wt(x_{i:j})-1} & \text{otherwise,} \end{cases} \qquad G(x_{i:j}) = \begin{cases} 1 & \text{if } wt(x_{i:j}) = j - i + 1 \\ x_{i+wt(x_{i:j})} & \text{otherwise.} \end{cases}$$

One can easily verify that $HWB(x_1, x_2, \ldots, x_n) = H(x_{1:n})$, and

$$H(x_{i:j}) = \bar{x_j} H(x_{i:j-1}) + x_j G(x_{i:j-1}), \quad G(x_{i:j}) = \bar{x}_i H(x_{i+1:j}) + x_i G(x_{i+1:j}), \quad G(x_{i:i}) = H(x_{i:i}) = x_i.$$

The main idea of the FBDD construction is explained in Figure 2. Simultaneously, $H_{1:n}$ and $G_{1:n}$ are computed. Identical restrictions on the same level are merged together. Since, on each level, k is fixed, we have at most $2n$ nodes on each level. Since on any source–to–sink–path each variable is tested exactly once, we get

Proposition 3. *The FBDD–representation P_{HWB} of $HWB \in I\!B_N$ is of size at most $2n^2$.* $\square$

Figure 3 shows, for $n = 7$, the type τ_{HWB}. Since, on each source–to–sink path, each variable is tested, τ_{HWB} is complete. One checks easily that the FBDD P_{HWB} constructed above is of type τ_{HWB}.

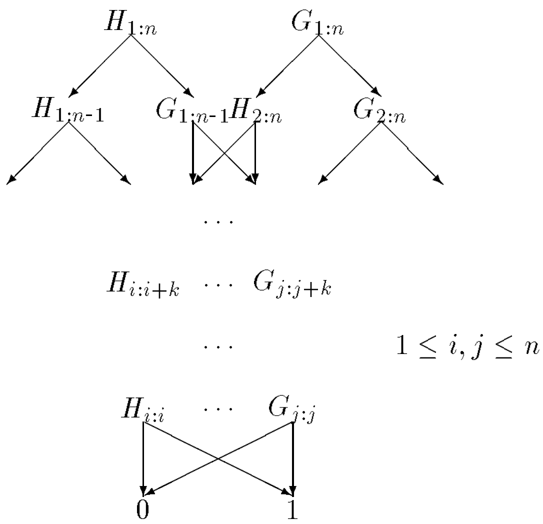

Figure 2: *Construction of an FBDD P_{HWB} that computes HWB.*

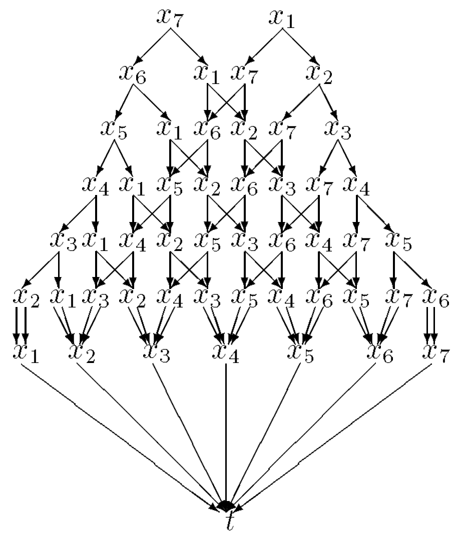

Figure 3: *The type τ_{HWB} for $n = 7$. (τ_{HWB} roots in the source labelled x_7.)*

4. COMBINATIONAL VERIFICATION WITH FBDDs

If we work with FBDDs of a fixed type τ, then it is possible to verify combinational circuits in terms of FBDDs similarly as in terms of OBDDs (e.g. [Bry92]): One derives a τFBDD-representation of the circuit and checks, due to canonicity, whether this representation equals the τFBDD for the specified Boolean functions the circuit is supposed to compute at its primary output gates. Due to Fact 1, the τFBDD-representation of circuit C can be constructed as follows: Starting with the primary inputs, for each gate g of C, an τFBDD for the function computed at g is generated by applying the Boolean operator performed in g to the τFBDDs of the inputs of g. Since any single operation has reasonable worst–case performance, this can be done as long as the FBDD–descriptions of the intermediate gates fit into the computer.

In the following we consider the problem of verifying a circuit realization C of HWB. Since, due to Fact 3, any OBDD-representation of the output gate of C is of exponential size, it is practically impossible to verify C in terms of OBDDs. Due to Fact 3, the question arises whether such a verification can be performed in terms of FBDDs. Since a bound on the size of an FBDD-representation P of the output gate of C does not impose any bound on the sizes of the FBDDs occuring in the course of the generation of P, one has to show that each FBDD occuring in the generation process of P remains small. Indeed, considering the well-known benchmark circuit C_{HWB} (Figure 1) proposed by Bryant [Bry91] we can do this if we chose to work with type τ_{HWB} (Figure 3) and some simple extensions of it.

Theorem 4. *The verification of the circuit C_{HWB} proposed in [Bry91] for computing the hidden weighted bit function HWB can be carried out in terms of τ_{HWB}FBDDs of size $2n^4 \log n$.*

4.1. Circuit Partitioning by Means of Internal Variables

In order to break down the problem of constructing the τ_{HWB}FBDD P_{HWB} of C_{HWB} into more easily managable tasks we partition C_{HWB}, due to its logical structure, into two subcircuits C_A and C_M. C_A embodies the adder tree (Figure 4), and C_M embodies the multiplexer tree (Figure 5). Subcircuit C_A outputs values $w_{m-1}, \ldots, w_0$ such that $wt(x) = \sum_{i=0}^{m-1} w_i 2^i$ for a primary input $x = (x_1, \ldots, x_{n-1})$. Together with the primary inputs these outputs are the inputs of subcircuit C_M which then computes HWB.

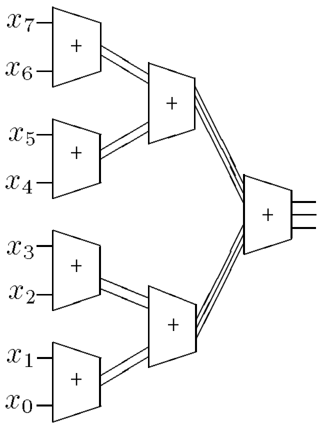

Figure 4. *Adder tree C_A.*

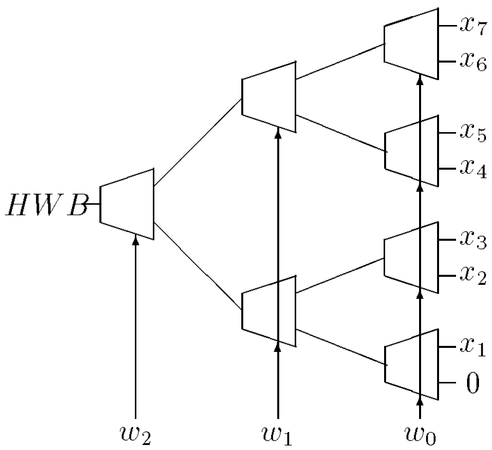

Figure 6. *Multiplexer tree C_M.*

Representing signals w_i, $0 \leq i < m$, by internal variables we can separately consider subcircuits C_A and C_M. In order to derive FBDD–representations P_A and P_M for C_A and C_M, respectively, and to be able to derive from P_A and P_M efficiently the desired FBDD P_{HWB}, we have to work with a common type that is (i) complete over $X_n \cup \{w_0, \ldots, w_{m-1}\}$ and (ii) has the property that all occurrences of the internal variables w_i are simple reducible. (The last assumption is to guarantee that restricting the internal variables doesn't change substantially the type of the FBDDs under consideration. For more details see [GM93b].) A type τ_0 with these properties can be easily obtained by concatenating an OBDD-type for the internal variables $w_{m-1}, \ldots, w_0$, and the basic type τ_{HWB}.

4.2. Deriving τ_{HWB}FBDDs for Carry–Lookahead Adder Trees

We start with the following observation that can be generalized to functions that are invariant under permutations of the input bits which respect a predefined partition of X_n.

Lemma 5. *Let τ be a complete type over X_n, and let f be a function $f \in I\!B_n$ that is symmetric in all its essential variables. Then f can be represented by means of an τFBDD of size at most $n \cdot size(\tau)$.* $\Box$

Now we have to go into the implementation details of C_A. In order to compute the weights w_i, $0 \leq i < m$, efficiently we discuss an implementation on the basis of carry–lookahead adders. Since the outputs of the carry–lookahead adders including the outputs of C_A itself are functions that are symmetric in their (primary) inputs, due to Lemma 4, they all can be represented by τ_{HWB}FBDDs of size at most $n \cdot size(\tau_{HWB}) \leq 2n^3$.

It remains to show that also the internal signals of the single carry–lookahead adders can be represented space–efficiently in terms of FBDDs.

Proposition 6. *The verification of the adder tree circuit C_A, area–time efficiently implemented by means of carry–lookahead adders, can be carried out in terms of τ_{HWB}FBDDs of size at most $2n^4 \log n$.* $\square$

4.3. Deriving τ_{HWB}FBDDs for Multiplexer Trees

Now we consider the subcircuit C_M of the circuit C for HWB which implements the multiplexer tree. It is quite easy to design a τ_0FBDD-representation P for C_M of size $size(P) = 2n$. More involved is to show that all FBDDs emerging in the process of eliminating the internal variables $w_{m-1}, w_{m-2}, \ldots, w_0$ are of small size. However, since each occurrence of an internal variable w_i in type τ_0 is simple reducible, one can apply the efficient algorithms for quantification, restriction and composition in the internal variables. The whole computation can be done in time and space $O(n^3)$. For details we refer to [GM93b].

REFERENCES

[Bry86] R. E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation, IEEE Trans. Comput. C-35, 6 (Aug.), 677–691, 1986.

[Bry91] R. E. Bryant: On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Applications to Integer Multiplication, IEEE Trans. Comput. 40, 2 (Feb.), 205–213, 1991.

[Bry92] R. E. Bryant: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams, ACM Computing Surveys, Vol. 24, No. 3 (Sep.), 293–318, 1992.

[Bry92b] R. E.Bryant, personal communication, (June), 1992.

[CMB90] O. Coudert, J.-C. Madre, C. Berthet: Verifying Temporal Properties of Sequential Machines without Building Their State Diagrams, Proc. of Computer-Aided Verification (Rutgers, N.J., June), 75–84, 1990.

[FFK88] M. Fujita, H. Fujisawa, N. Kawoto: Evaluation and Improvements of Boolean Comparison Methods Based on Binary Decision Diagrams, Proc. of IEEE Intern. Conference on Computer-Aided Design (Santa Clara, Nov.), 2–5, 1988.

[GJ79] M. R. Garey, D. Johnson: Computers and Intractability, W. H. Freeman & Co., New York, 1978.

[GM92] J. Gergov, Ch. Meinel: Analysis and Manipulation of Boolean Functions in Terms of Decision Graphs, Proc. of 18th International Workshop on Graph-Theoretic Concepts in Computer Science (June, 1992), Lecture Notes in Computer Science 657, 310–320, 1993.

[GM92b] J. Gergov, Ch. Meinel: Efficient Analysis and Manipulation of OBDDs Can Be Extended to Read-once–only Branching Programs, Tech. Rep. 92-10 (June), Univ. Trier, 1992. To appear in IEEE Transactions on Computers.

[GM93] J. Gergov, Ch. Meinel: Frontiers of Feasible and Probabilistic Feasible Boolean Manipulation with Branching Programs, Proc. of 10th Annual Symposium on Theoretical Aspects of Computer Science (Feb.), Lecture Notes in Computer Science 665, 576–585, 1993.

[GM93b] J. Gergov, Ch. Meinel: Combinational Logic Verification with FBDDs, Tech. Rep. 93-08, Univ. Trier, 1993.

[Mei92] Ch. Meinel: Branching Programs - An Efficient Data Structure for Computer-Aided Circuit Design, EATCS-Bulletin No. 46, 149–170, 1992.

[MWBS88] S. Malik, A. Wang, R. K. Brayton, A. Sangiovanni-Vincentelli: Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment, Proc. of IEEE Intern. Conference on Computer-Aided Design (Santa Clara, Nov.), 6–9, 1988.