A Machine to Evaluate Decomposed Multi-Terminal Multi-valued Decision Diagrams for Characteristic Functions

Hiroki Nakahara*, Tsutomu Sasao†, and Munehiro Matsuura†
*Kagoshima University, Japan, †Kyushu Institute of Technology, Japan

Abstract—A decomposed multi-terminal multi-valued decision diagrams for characteristic function (MTMDDs for CF) represents decomposed circuits. It can represent complex functions compactly. This paper shows a machine that evaluates decision diagrams. First, we introduce the decomposed MTMDDs for CF. Then, we consider two instructions to evaluate the decomposed MTMDDs for CF. Next, we show a machine that evaluates the decision diagrams. We compare that machine with embedded processors. As for the power-delay product, our machine running at 100 MHz is 60.84 times smaller than Nios II processor running at 100 MHz, and it is 18.66 times smaller than Atom N455 processor running at 1.67 GHz.

I. INTRODUCTION

Before the era of deep submicron, the performance of microprocessors was the major concern, and it was improved by the scaling down of device dimensions. However, in the era of deep submicron, the power-efficiency is the major concern of the modern processors [4]. To improve the power-efficiency, this paper proposed a decision diagram machine (DDM) [7], [1] which evaluates decision diagrams (DDs). A heterogeneous multi-valued decision diagram (HMDD) may have nodes with different number of binary variables for different super variables [10]. Since the HMDD can use the optimal partition of the input variables to reduce the path length, the evaluation time of the HMDD is shorter than that of the corresponding binary decision diagram (BDD) [15]. We showed that, the HMDD machine is faster and dissipates lower power than the conventional processor [12]. Applications for the HMDD machine include a sequencer [22]; an accelerator for a logic simulation [5]; and a packet classifier [14]. However, for general applications including arithmetic circuits, since the number of the nodes in a conventional HMDD increases exponentially with the number of inputs, such HMDDs are difficult to represent, when the number of inputs is large [20]. Thus, the conventional DDM is unapplicable for general purpose.

To represent a logic circuit by DDs with a reasonable size, we proposed decomposed MTMDDs for CF representing a decomposed circuit [11]. With an enough size of memory, the DDM for the decomposed MTMDD is faster than that for the conventional HMDD. When the memory was expensive, it was difficult to use a memory with large size. However, a memory with more than one Giga bytes is available at a low price today. Therefore, adopting the DDM for the decomposed HMDDs for CF with a large memory is practical.

In this paper, we realize DDM for the decomposed HMDDs for CF using an FPGA and an off-chip SRAM. Also, we compare the proposed one with embedded processors with respect to delay time, power consumption, and power-delay product.

The rest of the paper is organized as follows: Chapter 2 shows definitions; Chapter 3 introduces the multi-terminal multi-valued decision diagrams for characteristic function representing a decomposed circuit (decomposed MTMDDs for CF); Chapter 4 realizes the DDM for decomposed MTMDDs for CF; Chapter 5 shows the experimental results; and Chapter 6 concludes the paper.

II. DEFINITION OF DECISION DIAGRAMS

A. Decision Diagram (DD)

Definition 2.1: A binary decision diagram (BDD) is obtained by applying Shannon expansions repeatedly to a logic function \( f \) [2]. Each non-terminal node labeled with a variable \( x_i \) has two outgoing edges which indicate nodes representing cofactors of \( f \) with respect to \( x_i \). When the Shannon expansions are performed with respect to \( k \) variables, all the non-terminal nodes have \( 2^k \) edges. In this case, we have a multi-valued decision diagram (MDD(\( k \))) [6].

Definition 2.2: In a DD, a sequence of edges and non-terminal nodes leading from the root node to a terminal node is a path. An ordered BDD (OBDD) has the same variable order on any path. A reduced ordered BDD (ROBDD) is derived by applying as follows:

1. Share equivalent sub-graphs.
2. If all the outgoing edges of a non-terminal node \( v \) point the same succeeding node \( u \), then delete \( v \) and connect the incoming edges of \( v \) to \( u \).

An ROMDD(\( k \)) can be defined similarly to the ROBDD. Note that, an MDD(1) means a BDD. In this paper, a BDD and an MDD(\( k \)) mean an OBDD and an ROMDD(\( k \)), respectively, unless stated otherwise.

If the evaluation time for all the nodes are the same, then the average evaluation time for an HMDD is proportional to the average path length (APL) [3].

B. Representation of Multi-output Logic Function Using Decision Diagrams for Characteristic Function

Many practical applications use multiple-output functions. Here, we represent an \( n \)-input \( m \)-output logic function using
a decision diagram (DD).

**Definition 2.3:** Let $\vec{X} = (x_1, x_2, \ldots, x_n)$ be the input variables, $\vec{Y} = (y_1, y_2, \ldots, y_m)$ be the output variables, and $\vec{f} = (f_1(\vec{X}), f_2(\vec{X}), \ldots, f_m(\vec{X}))$ be a multiple-output function. The characteristic function (CF) of a multiple-output function is $\vec{\chi}(\vec{X}, \vec{Y}) = \bigwedge_{i=1}^{m} (y_i \equiv f_i(\vec{X}))$.

The characteristic function of an $n$-input $m$-output function is a two-valued logic function with $(n + m)$ inputs. It has input variables $x_i$ ($i = 1, 2, \ldots, n$), and output variables $y_j$ for outputs $f_j$. Let $B = \{0, 1\}$, $\vec{a} \in B^n$, $\vec{f} = (f_1(\vec{a}), f_2(\vec{a}), \ldots, f_m(\vec{a})) \in B^m$, and $\vec{b} \in B^m$. Then, the characteristic function satisfies the relation $\vec{\chi}(\vec{a}, \vec{b}) = \begin{cases} 1 & \text{when } \vec{b} = \vec{f}(\vec{a}) \\ 0 & \text{otherwise} \end{cases}$

**Definition 2.4:** A support variable of a function $f$ is a variable on which $f$ actually depends.

**Definition 2.5:** [19] A multi-terminal binary decision diagram for characteristic function (MTBDD for CF) of a multiple-output function $\vec{f} = (f_1, f_2, \ldots, f_m)$ represents the characteristic function $\vec{\chi}$. We assume that the root node is in the top of the MTBDD, and the variable $y_i$ is below the support variable of $f_i$, where $y_i$ is the variable representing $f_i$.

### III. DECOMPOSED MULTI-TERMINAL MULTI-VALUED DECISION DIAGRAMS FOR CHARACTERISTIC FUNCTION

#### A. Cluster Decomposition of a Circuit

In this paper, a combinational circuit is represented by a directed acyclic graph (DAG). In a DAG, a primary input node denotes a primary input; a primary output node denotes a primary output; and an intermediate node denotes a 2-input logic gate. When the output of a logic gate is connected to a logic gate $j$, the DAG has an edge from a node $i$ to a node $j$. In the DAG, we assume that a primary input does not fan-out. Thus, in general, we need to duplicate the input variables. Therefore, the number of the primary input nodes can be greater than the number of the primary inputs. We denote an input node by (primary input name)_unique number.

---

1 A NOT gate is converted into a NAND gate having the same inputs. Also, a gate with more than two inputs is decomposed into multiple gates with two inputs.
Definition 3.10: Fig. 4 shows the circuit representing a set of clusters \( \{C_1, C_2, \ldots, C_q \} \). Let \( i < j \). The outputs of the cluster \( C_i \) directed to the cluster \( C_j \) are rail outputs, and the inputs of the cluster \( C_j \) directed from the cluster \( C_i \) are rail inputs.

Example 3.4: Fig. 5 shows the circuit realizing the cluster decomposition of the two-bit multiplier shown in Fig. 3.

B. MTBDDs for CF Representing a Cluster Decomposition

In a set of clusters \( \{C_1, C_2, \ldots, C_q \} \), \( C_i \) is represented by an MTBDD for CF, where the inputs consist of primary inputs and rail inputs, and the outputs consist of primary outputs and rail outputs.

Example 3.5: Fig. 6 shows MTBDDs for CF representing clusters shown in Fig. 5.

Definition 3.11: Let \( \{C_1, C_2, \ldots, C_q \} \) be a set of clusters, where \( C_i \) is represented by an MTBDD for CF. The MTBDDs for CF representing a cluster decomposition (decomposed MTBDDs for CF) is obtained by connecting MTBDDs for CF in the topological order of \( \{C_1, C_2, \ldots, C_q \} \).

Example 3.6: Fig. 7 shows the decomposed MTBDDs for CF representing the two-bit multiplier in Fig. 3.

C. Decomposed MTMDDs for CF

A decomposed MTBDDs for CF can be extended to a decomposed multi-valued decision diagrams for CF (decomposed MTMDDs for CF). To reduce its evaluation time, the DDM for decomposed MTMDDs for CF can use more memory than that for the decomposed MTBDDs for CF.

Example 3.7: Fig. 8 shows the decomposed MTMDDs for CF converted from the MTBDDs for CF shown in Fig. 7. Note that, the decomposed MTMDDs for CF is obtained by merging the cluster \( C_1 \) and a part of cluster \( C_2 \), and by changing the variable order of a node.

As shown in Example 3.7, in MTBDDs for CF, multiple clusters are merged and the variable order of a node is changed to obtain MTMDDs for CF.

Note that, the decomposed MTBDDs for CF lost canonicity of the BDD. Thus, the application of decomposed MTBDDs for CF to formal verification is difficult. However, the application to logic evaluation is straightforward, since the canonicity is not used.

IV. DDM FOR DECOMPOSED MTMDDS FOR CF

A. Instructions to Evaluate the Decomposed MTMDDs for CF [13]

In a DDM, for a non-terminal node, we use the indirect branch instruction shown in Fig. 9. The index and branch addresses are stored in the separated words to use the memory efficiently as shown in Fig. 10. For a non-terminal node, first, the current index is read. Then, the jump address corresponding to the value of the current input variables is read. To evaluate a terminal node, we use the output and jump instruction shown in Fig. 11. To evaluate a terminal node, first, the output value is read. Then, the jump address is read. Since we use only two instructions, only one bit field is necessary to distinguish them.

B. Interconnections Among the Decomposed MTMDDs for CF

Since the decomposed MTMDDs for CF allows fan-outs of both the primary inputs and the rail outputs, interconnections...
are necessary. They are implemented by the random logic in the FPGA. Since the most parts of the circuit is realized by the nodes of the decomposed MTMDDs for CF, hardware for the interconnections tends to be small. Thus, the interconnections can be realized with a small FPGA. Experimental results will demonstrate this.

C. DDM for Decomposed MTMDDs for CF

Fig. 12 shows the DDM for a decomposed MTMDDs for CF. The instruction memory stores the instructions; the instruction register stores the instruction from the instruction memory; the program counter (PC) retains the address for the instruction memory; the interconnection module selects both the primary inputs and the rail inputs; and the double-rank shift register retains the output value. Fig. 13 shows the double-rank shift register consisting of double-rank flip-flops [18]. The register retains outputs of the decomposed MTMDDs for CF by the C_Clock. When all outputs are evaluated, values are sent to the primary outputs by the S_Clock.

The following algorithms show the execution of the indirect branch instruction and the output and jump instruction.

Algorithm 4.1: \( 2^k \) indirect branch instruction
1. Obtain indirect branch address
   1.1 Read the index corresponding to the index filed of the branch instruction.
   1.2 Add it to the PC.
2. Perform the jump operation
   2.1 Read the jump address corresponding to the PC.
   2.2 Set the jump address to the PC.
3. Terminate.

Algorithm 4.2: (Output and jump instruction).
1. Output the value.
   1.1 Read the output value and the jump address corresponding to the PC.
   1.2 Set the value to the double-rank register.
   1.3 If all outputs are evaluated, then send them to the primary outputs.
2. Perform the jump operation, similarly to the Step 2 of Algorithm 4.1.
3. Terminate.

Example 4.8: The left column of Fig. 14 shows the interconnections for the decomposed MTMDDs for CF shown in Fig. 8.

1. To evaluate the root node, select primary inputs \( x_0 \) and \( x_2 \). Then, set the primary output \( y_0 \) to the double-rank register (Fig. 14 (1)).
2. To evaluate the next node, select primary inputs \( x_1, x_2, \) and \( x_3 \). Then, set rail outputs \( w_1 \) and \( w_5 \) to the rail register (Fig. 14 (2)).
3. To evaluate the next node, select primary inputs \( x_0 \) and \( x_3 \), and the rail input \( w_1 \). Then, set the primary output \( y_1 \) to the double-rank register (Fig. 14 (3)).
4. To evaluate the leaf node, select the rail input \( w_5 \). Then, set primary outputs \( y_2 \) and \( y_3 \) to the double-rank register. Since all the outputs are evaluated, the double-rank register send them to the primary outputs (Fig. 14 (4)).

V. EXPERIMENTAL RESULTS

A. Comparison with Embedded Processors

We implemented the DDM for decomposed MTMDDs for CF on the Altera Cyclone III starter kit (FPGA: Cyclone III, EP3C25). For the FPGA synthesis tool, we used
QuartusII (v.9.1). We compare it with the Intel’s Atom processor running at 1.67 GHz and the Altera’s Nios II embedded processor running at 100 MHz. We obtained the delay time per one random test vector [nsec/work] as shown in Table I using MCNC benchmark functions [21]. To obtain the decomposed MTBDDs for CF, first, we generated the netlist from the Verilog-HDL description by using Quartus II logic synthesis tool ver.9.1 with area minimization option. Then, we decomposed the netlist into clusters by using the greedy algorithm [16]. Next, we converted clusters into the decomposed MTBDDs for CF. To reduce the number of nodes in MTBDDs, we used the sifting algorithm [17]. To obtain the decomposed MTMDDs for CF, first, we set the memory size limitation to 1 Mega bytes (MB). Then, we converted the decomposed MTBDDs for CF into an MTMDDs for CF by using the dynamic programming [9]. To obtain delay time for the Nios II processor and the Atom N455 processor, we generated C-code for the decomposed MTMDDs for CF. Then, we generated the executable code using gcc compiler with optimize option -O3. The delay time of the DDM for the decomposed MTMDDs for CF is 1.91 times shorter than that of the Atom processor, and it is 13.12 times shorter than that of the Nios II processor.

We measured the power consumption [W] to obtain the power-delay product. The power-delay product corresponds to the energy [Wnsec] to evaluate the function. To make the comparison fair, we tried to make the temperature of the DDM for decomposed MTMDDs for CF, the Nios II processor, and the Atom N455 processor the same. A linear shift register for decomposed MTMDDs for CF, the Nios II processor, and it is 13.12 times faster than the Atom processor, also, as for the power-delay product, the DDM dissipates 18.66 times lower energy than the Atom processor, and dissipates 66.84 times lower energy than the Nios II processor. Thus, the DDM for the decomposed MTMDDs for CF is more power-efficient than existing embedded processors.

VI. CONCLUSION

This paper showed the DDM for the decomposed MTMDDs for CF. To evaluate a non-terminal node, it uses an indirect branch instruction, while to evaluate a terminal node, it uses an output and jump instruction. The FPGA realizes the interconnections and the controller, while the SRAM stores the instructions. As for the speed, the DDM for the decomposed MTMDDs for CF is 1.91 times faster than the Atom processor, and is 13.12 times faster than the Nios II processor. Also, as for the power-delay product, the DDM dissipates 18.66 times lower energy than the Atom processor, and dissipates 66.84 times lower energy than the Nios II processor. Thus, the DDM for the decomposed MTMDDs for CF is more power-efficient than existing embedded processors.

VII. ACKNOWLEDGMENTS

This research is supported in part by the Grants in Aid for Scientific Research of JSPS. Discussions with Prof. J. T. Butler and Prof. Shinobu Nagayama were quite useful.

REFERENCES

Fig. 14. Example of the input selector.