Formal Equivalence Checking of Folded Architectures*

Tay-Jyi Lin, Chein-Wei Jen
Department of Electronics Engineering
National Chiao Tung University
1001 Ta-Hsueh Road, Hsinchu
Taiwan, R.O.C.
{tjlin, cwjen}@ee.nctu.edu.tw

Abstract: - A high-level synthesis system can time-multiplex operations onto different number of processing elements to meet various requirements by folding transformation with optimal scheduling and allocation. The resultant folded architectures usually have divergent I/O sequencing or timing and complicate the equivalence checking on them. One possible solution is with resort to the powerful combinational verification methods by unfolding transformation with proper retiming to completely remove the registers. In this paper, we propose a novel iterative construction algorithm of decision diagrams that implicitly transforms the folded architectures under verification to significantly reduce computation and memory usage. Experimental results show that our method has great improvement on memory usage represented in BDD nodes.

Key Words: - Formal Verification, Equivalence Checking, and High Level Synthesis

1 Introduction

Current practiced design validation methods are still techniques of simulation and testing. Although very effective in the early debugging stages, they require alarmingly increasing amount of time to uncover more subtle bugs. Equivalence checking (EC) verifies that two circuits, one of which is usually regarded as the "golden" model, are functionally equivalent at the I/O boundaries, at the internal state bits or on a cycle-by-cycle basis. Formal verification provides complete EC as opposed to simulation and testing, which checks the equivalence only to some extent that the test suite exercises the design. Binary decision diagram (BDD)-based [1] canonical representations play a major role in formal EC of combinational circuits, where circuits with the same functionality should have an identical form. Reduced, ordered BDD (ROBDD) is the basic canonical representation of Boolean functions with various extensions, such as word-level representations that efficiently represent arithmetic functions [2]. EC of two sequential circuits is much more complex, which involves state-traversal on their product machine and proves that no product state is reachable that gives different outputs [3][4]. FSMs of interest may have 10^{10} (~34 flops) to 10^{100} (~333 flops) or even a greater number of states that make it impractical to represent the states as individual entities. Symbolic state-traversal [5], as an alternative, uses BDD-represented state-transition functions to store billions of states with just thousands of BDD nodes. Exploration of the structural similarity between the circuits under verification is a frequently used technique to reduce the verification efforts both in combinational and sequential EC (e.g. circuit partitioning, register mapping, etc) [6][7][8].

Folding transformation [9] in a high-level synthesis system [10] time-multiplexes multiple algorithm-level operations to processing elements. Various optimal folded architectures can be automatically generated through different scheduling, allocation or even different number of processing elements for a single algorithm to meet divergent requirements. The resultant architectures usually have distinct timing and I/O sequencing, which are very problematic in sequential EC. One alternative is to encapsulate these folded architectures into stream interface units (SIU), which transforms the distinct I/O sequencing to an identical external interface [11], but the verification on the SIU itself is another unsolved problem. The powerful methods of combinational EC can be applied instead of the more complicated sequential EC methods [12], as another alternative, via unfolding transformation with proper retiming to completely remove the pipelining registers. The exploration of the structural regularity [13] in the unfolded representation (i.e. due to the temporal identity of folded architectures) significantly reduces the complexity in verification.

In this paper, we propose a novel approach to construct the BDD-based canonical representation for combinational EC on sequential circuits. The unified algorithm implicitly performs the required

* This work was supported by the National Science Council, Taiwan, under Grant NSC89-2218-E009-078.

unfolding transformation and retiming to reduce the computation and memory usage while exploring the structural regularity simultaneously. The proposed algorithm is detailed in section 2 with simple illustrating examples in section 3. Section 4 shows our experimental results and section 5 concludes this work.

2 Proposed Method

Sequential EC can be reduced to a combinational one by unfolding the circuit and retiming acyclic paths with robust register mapping on feedback [8][12]. For simplicity, we only discuss the terminating (i.e. with finite execution steps) folded architectures in this paper and demonstrate the verification of bit-level circuits with widely used ROBDD for clarity. The proposed iterative algorithm to incrementally build the canonical representation can be easily modified for the word-level extension [2]. Fig 1 depicts the 4-stage algorithm.

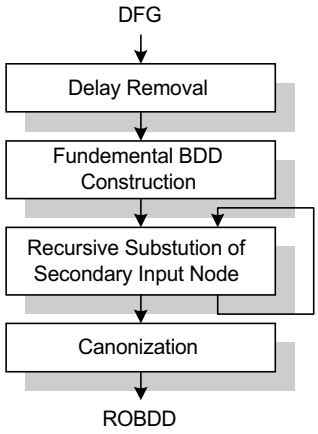


Figure 1 Incremental ROBDD Construction

Step 1: Delay removal

The folded architectures to be verified are first represented in synchronous dataflow graphs (DFG) [9] with the corresponding I/O sequencing. P_i and P_o denote the primary input and the primary output ports respectively. Remove all delay elements (registers) and label the secondary input and the secondary output ports with $S_i^{(n)}$ and $S_o^{(n)}$, where n is the number of removed delay units. The resultant acyclic DFG is referred as the ***fundamental DFG***.

Step 2: Fundamental BDD construction

Construct the ROBDD of the fundamental DFG. Except at the I/O boundary, the resultant ROBDD, which we call ***fundamental BDD***, is identical to the BDD-represented transition functions in symbolic state-traversal and model checking [3][4][5].

Step 3: Recursive substitution of secondary input nodes

Incrementally construct the complete ROBDD of the folded architecture backward from the final execution step. First, we begin with the fundamental BDD and associate each primary input node with its corresponding variable (according to the I/O sequencing) at the final time-slice. Recursively substitute the secondary input nodes superscripted by a matching n at the succeeding time-slices with the corresponding BDD sub-tree rooted by the same secondary output variable in the fundamental BDD. Increment the superscript n by the time index in the substituting sub-tree and associate all primary input nodes with the corresponding variables (according to the I/O sequencing, again) at each time-slice. Continue the substitution process until no secondary leaf node exists. If the BDD sub-tree has a node corresponding to a primary input variable appearing in its ancestor nodes, remove the conflicting path.

Step 4: Canonization

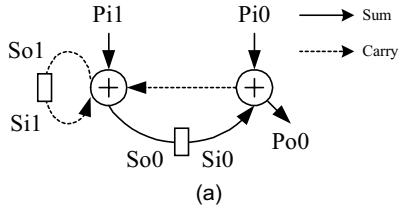
Redundant node removal and variable reordering [14] for different I/O sequencing are required to maintain the ROBDD canonization.

The proposed algorithm requires the computation-intensive BDD construction only for the folded DFG, which is usually much smaller than the flattened unfolded representations. The incremental ROBDD construction via recursive substitution enables DFS node traversal, which significantly reduces the memory usage during EC when the two architectures under verification have similar input sequencing (i.e. only requires limited-range variable reordering). Optimal variable ordering can be conducted along each path individually to further reduce the ROBDD size. The spatial structural similarity (cf. the regularity in the unfolded representation due to temporal identity) can be explored early in the folded architectures to reduce the verification efforts.

3 Examples

Multiplication generally consists of two portions: partial-product generation and reduction. Various bit-level multiplication architectures with different sizes, operating frequencies and power consumption differentiate themselves mainly in the partial-product reduction architectures. We assume the partial-product generators of the multipliers are identical (this is always the case) and ignore them for simplicity. Fig 2 and Fig 3 show two simplified reduction architectures (of a 2-bit multiplier) to

reduce the partial products p0~p3 from their primary inputs. Note that the left adder in Fig 3(a) is redundant and remains only to show the scalability for a longer wordlength.

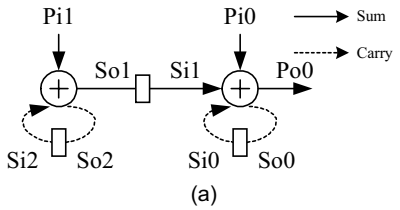


(a)

	Cycle 1	Cycle 2
Pi0	p0	p2
Pi1	p1	p3
Si0	0	p1
Po0	**p0***	**p0+p1**
Si1	0	C(p1)**
So0	p1	**p3+C(p0+p1)**
So1	C(p1)	**C[p3+C(p0+p1)]**

* **Bolded font** denotes the multiplier output
** C( ) denotes the "carry of "
(b)

Figure 2 (a) Ripple Carry Adder (RCA) Partial-Product Reduction Architecture (b) Timing Table



(a)

	Cycle 0	Cycle 2	Cycle 2	Cycle 3
Pi0	P0	p2	0	0
Pi1	P1	p3	0	0
Si0	0	0	C(p1+p2)	C[p3+C(p1+p2)]
Si1	0	p1	p3	0
Po0	**P0**	**p1+p2**	**p3+C(p1+p2)**	**C[p3+C(p1+p2)]**
So0	C(p0)=0	C(p1+p2)	C[p3+C(p1+p2)]	0
So1	P1	p3	0	0

(b)

Figure 3 (a) Carry Save Adder (CSA) (b) Timing Table.

Conventionally, detailed timing tables, as shown in Fig 2(b) and Fig 3(b), are constructed to manually verify the correctness of a folded architecture. The tedious symbolic manipulation on variables with poor interface to HDL-based design environments makes the verification very time-consuming and error-prone. Traditional sequential EC based on state-traversal on the product machine does not work because the two architectures do not have an identical synchronous behavior – neither the timing nor the I/O sequencing is the same. Combinational verification techniques can verify the equivalence of the two folded architectures by unfolding transformation with proper retiming to completely remove all registers as shown in Fig 4. The proposed folding verifier implicitly performs the register removal via iterative ROBDD construction based on the fundamental BDD shown in Fig 5. Because of the limited pages, we only demonstrate the construction process of the partial ROBDD rooted by the most significant bit of the multiplication result. Fig 6 and Fig 7 show the incremental ROBDD construction of RCA and CSA architectures respectively.

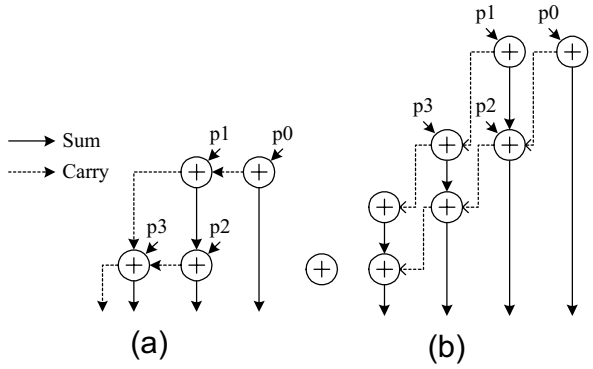


Figure 4 Unfolded Representation of (a) RCA (b) CSA Product Reduction Architecture

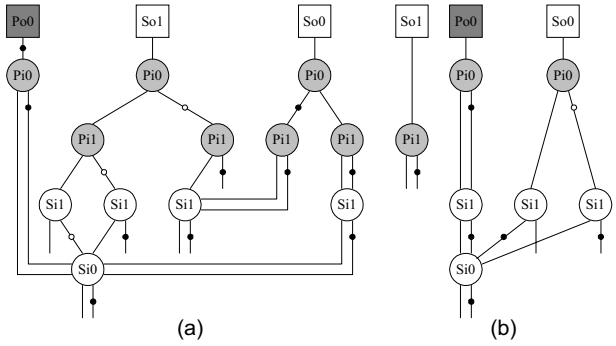


Figure 5 Fundamental BDD of (a) RCA (b) CSA Product Reduction Architecture

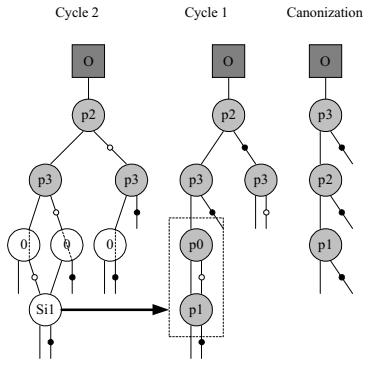


Figure 6 Incremental ROBDD Construction for RCA Architecture

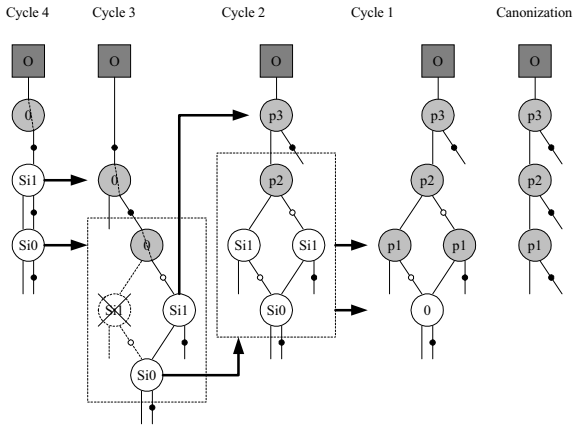


Figure 7 Incremental ROBDD Construction for CSA Architecture

4 Experimental Results

We have implemented all described algorithms in C based on the CUDD package [15]. Table 1 depicts the comparison between our equivalence checker for the folded multipliers and a conventional ROBDD-based EC with explicit unfolding and retiming to completely remove the pipelining registers but with no structural heuristics. The first two columns show the required pipelining registers and execution cycles for the two folded (bit-serial) multipliers with distinct timing and I/O sequencing. The simulation keeps track of the maximum number of active BDD nodes constructed during the verification process to represent the maximum memory usage.

Table 1. Performance Comparison*

Bits	Registers		Cycles		BDD Nodes (MAX)	
	RCA	CSA	RCA	CSA	Combi. **	Proposed
8	8	15	8	16	4,959	1,021
16	16	31	16	32	26,843,238	262,141
24	24	47	24	48	N.A.	67,108,861

* No result is available for sequential EC because the partial product reduction architectures do not have identical synchronous behaviors.
** An ROBDD-based combinational EC is built for simulation, with explicit unfolding & retiming but no structural heuristic.

Our folding verifier explores the structural similarity in DFG at the very early stage to prune the matching blocks (not used in this comparison). The proposed algorithm requires the BDD construction only for the fundamental DFG that is usually much smaller than the completely unfolded representation used in the traditional formal combinational EC. The recursive substitution process is much simpler than the saved BDD construction efforts and the fundamental BDD functions as a specific computation cache used in general BDD packages. It also simplifies the depth-first BDD construction and node traversal in EC that reduces the memory usage dramatically in our experiment. The extra overhead is the canonization process with redundant node removal and variable reordering. No explicit unfolding or retiming is needed. Conventional sequential EC based on state-traversal on the product machine is not included in this comparison because the synchronous behaviors of these two multipliers under verification are different. As the word-length of the multipliers under verification increases, the growth of memory usage represented in the maximum number of active BDD nodes during verification is much slower with our proposed method than that with a conventional EC.

5 Conclusion

This paper presents an efficient methodology to verify the functional equivalence of two folded architectures with distinct timing and I/O sequencing, which are very problematic in the conventional sequential EC. Experimental results show that the proposed method requires much less memory and computation than the combinational EC with explicit unfolding and retiming transformations. Our future research is to simplify the automatic isolation process of the feedback paths in non-terminating folded architectures (e.g. IIR filters) with finite unfolding factors to recover the distinct scheduling and allocation and novel robust register mapping.

References:

[1] R. E. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, August 1986

[2] R. E. Bryant, Y. A. Chen, "Verification of Arithmetic Circuits with Binary Moment Diagrams," *Design Automation Conference (DAC)*, June 1995

[3] A. Ghosh, S. Devadas, A. R. Newton, *Sequential Logic Testing and Verification*, Kluwer Academic Publisher, 1992

[4] G. D. Hachtel, F. Somenzi, "FSM Equivalence Checking," *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publisher, 1996

[5] K. L. McMillan, *Symbolic Model Checking*, Kluwer Academic Publisher, 1994

[6] G. P. Bischoff, K. S. Brace, S. Jain, R. Razdan, "Formal Implementation Verification of the Bus Interface Unit for the Alpha 21264 Micro- processor," *International Conference on Computer Design (ICCD)*, 1997

[7] C. A. J. van Eijk, "Sequential Equivalence Checking without State Space Traversal," *Design, Automation and Test in Europe (DATE)*, March 1998

[8] J. R. Burch, V. Singhal, "Robust Latch Mapping for Combinational Equivalence Checking," *International Conference on Computer-Aided Design (ICCAD)*, November 1998

[9] K. K. Parhi, *VLSI Digital Signal Processing Systems – Design and Implementation*, John Wiley & Sons, 1999
[10] D. D. Gajski, N. D. Dutt, C. H. Wu, Y. L. Lin, *High Level Synthesis – Introduction to Chip and System Design*, Kluwer Academic Publisher, 1992
[11] T. J. Lin, C. W. Jen, "Data Stream Generation for Concurrent Computation in VLSI Signal Processors," *International Conference on Signal Processing (ICSP)*, August 2000
[12] R. K. Ranjan, V. Singhal, F. Somenzi, R. K. Brayton, "Using Combinational Verification for Sequential Circuits," *Design Automation and Test in Europe (DATE)*, March 1999
[13] P. F. Williams, et al, "Equivalence Checking of Hierarchical Combinational Circuits," *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 1999
[14] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," *International Conference on Computer-Aided Design (ICCAD)*, 1993
[15] CUDD Package, http://vlsi.colorado.edu