MULTI-VALUED NETWORK COMPACTION USING REDUNDANCY REMOVAL

by

Sunil P. Khatri, Robert K. Brayton
and Alberto Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M98/44

1 June 1998

COVER

MULTI-VALUED NETWORK COMPACTION USING REDUNDANCY REMOVAL

by

Sunil P. Khatri, Robert K. Brayton
and Alberto Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M98/44

1 June 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Multi-Valued Network Compaction using Redundancy Removal

Sunil P. Khatri (linus@ic.eecs.berkeley.edu) *

Robert K. Brayton (brayton@ic.eecs.berkeley.edu) *

Alberto Sangiovanni-Vincentelli (alberto@ic.eecs.berkeley.edu) *

1 June 1998

Abstract

We introduce a scheme to simplify a multi-valued network using redundancy removal techniques. Recent methods [1], [2] for binary redundancy removal avoid the use of state traversal. Additionally, [2] finds multiple compatible redundancies simultaneously. We extend these powerful advances in the field of binary redundancy removal to perform redundancy removal for multi-valued networks.

First we perform a one-hot encoding of all the multi-valued variables of the design. Multi-valued variables are written out as binary variables, using this one-hot encoding. At the end of this step, we have a binary network which is equivalent to the multi-valued network modulo encoding.

Next, binary redundancy removal is invoked on the resulting network. In case a binary signal s_i is determined to be stuck-at-0 redundant, this means that the multi-valued signal s can never take on a value i. Further, if the binary signal s_i is determined to be stuck-at-1 redundant, this means that the multi-valued signal s takes on a constant value i. All redundant binary signals are recorded in a file. The original multi-valued network is modified based on the binary redundancies thus computed.

*CAD Research Group, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720. This research was funded under the Semiconductor Research Corporation Grant SRC-324-040.

Initial experiments using this technique show a 10-20% reduction in the size of the multi-valued description.

1 Introduction

Until recently, most logic synthesis and verification was performed on binary-valued networks. Recent work [3], [4] allows for input in the form of a multi-level network of multi-valued variables. This has some important consequences. For one, it allows the user to design circuits at a higher level of abstraction, making the design task more intuitive. This is especially important for larger designs. Also, the user can experiment with various binary encodings of the multi-valued design, and choose the one that yields the most efficient (binary) implementation. Thus a better exploration of the design space becomes possible.

Techniques that simplify the multi-valued network result in a more efficient design process. In this paper, we report on one such simplification technique, which involves removal of multi-valued redundancies in the network. Our scheme removes the redundant values of a multi-valued combinational node (or latch) as well as the redundant multi-valued nodes (or latches) themselves. The efficiencies that result from our redundancy removal technique include:

- The modified multi-valued design requires less storage.
- Simulation of the modified design is sped up, since the modified design is more compact.
- The designer can perform exploratory encoding and synthesis experiments more efficiently
- The scheme handles sequential designs as well. As a result, multi-valued register values or multi-valued registers themselves can be removed from a design. This should help speed up the task of formal verification.

Recently, highly efficient schemes for binary redundancy removal have been reported [1], [2]. These schemes perform sequential binary redundancy removal without utilizing state space search. We base our multi-valued redundancy removal techniques on this recent work.

Our multi-valued redundancy removal scheme works as follows. First, the multi-valued network (N^M) is transformed into an equivalent binary network (N^B), using a one-hot encoding of all multi-valued variables. So each multi-valued variable s of N^M with n possible values gives rise to n variables in N^B , labeled $s_1, s_2, \cdots, s_n$. Now, binary redundancy removal is performed on N^B, using the algorithm of [2]. If binary signal s_i is found to be stuck-at-0 redundant, then the value i of the multi-valued variable s in N^M cannot occur, and N^M is simplified accordingly. If binary signal s_i is stuck-at-1 redundant, then the value the variable s of N^M can be replaced by a constant i value. Once again, N^M is simplified to reflect this.

The code for this work is written in SIS [5] and VIS [3] [4]. The encoding code is written in VIS. It outputs a file in the *blif* format, readable by SIS. The redundancy removal code is a modified version of the code used in [2], and is implemented in SIS. Finally the network simplification code is written in VIS.

To the best of our knowledge, there has been no prior work in the area of redundancy removal of multi-valued circuits. In this sense our scheme is unique in its scope.

The remainder of this paper is organized as follows. Section 2 describes our redundancy removal scheme, after describing the binary scheme [2] that it is based upon. Section 3 describes preliminary results that we have obtained using our scheme. Finally, in Section 4, we make concluding comments and discuss further work that needs to be done in this area.

2 Our Approach

In the rest of this section, we describe our work in detail, and prove the correctness of the scheme. In Section 2.1 we give a short summary of the binary redundancy removal technique of [1] and [2]. This is followed by a detailed description of our procedure in Section 2.2. The proof of correctness of our scheme is given in Section 2.3

In the sequel, we refer to the combinational elements of a multi-level design as *nodes*, and the sequential elements of a multi-level design as *latches*. Both are collectively referred to as *variables*.

2.1 Efficient Binary Redundancy Removal

Binary Redundancy Removal (BRR) is motivated by ideas from Automatic Test Pattern Generation (ATPG).

Let Z(X) be the logic function of the binary network N^B with primary inputs X, and let f be a stuck-at fault in N^B.

Definition 1 *A test vector t detects a fault f iff $Z_f(t) \neq Z_f(t)$, where $Z_f(t)$ is the logic function of the network N^B in the presence of the fault f.*

Definition 2 *A fault f is detectable if there exists a test t that detects f. Otherwise, f is undetectable.*

Definition 3 *If the network N^B contains an undetectable stuck-at-0 (stuck-at- 1) fault f at node p, then the node p is said to be stuck-at-0 (stuck-at-1) redundant. Under this condition, node p can be replaced by a constant 0 (constant 1) value.*

Traditionally, circuit redundancies were detected by testing for stuck-at faults one at a time using ATPG. This is an expensive proposition, since each ATPG instance is NP-Complete.

Recently, efficient techniques for BRR were developed in [1] and [2]. These methods use implications instead of ATPG to detect redundancies. [1] initially reported the concept, which was improved and extended in [2]. Essentially, for each node p in N^B, we first set a 0 (1) value on that node. The implications that are derived from this assignment form a set S_0 (S_1). Now, all the nodes in $S_0 \cap S_1$ are redundant nodes, since node p cannot attain both 0 and 1 values simultaneously. In [2], the implication process is modified so as to guarantee that all nodes in $S_0 \cap S_1$ can be simultaneously removed.

In [1] and [2], this idea was applied to sequential circuits as well, by using a (combinational) iterative array model [6] of the sequential circuit. Implications are propagated across time-frames (i.e. across latch boundaries) to determine sequential redundancies. Hence state space search is avoided. A state space search would be required if ATPG was used to do BRR for sequential circuits. It was found that an iterative array model with 15 forward and 15 backward time-steps was a good tradeoff, since implications usually died out before this limit was reached.

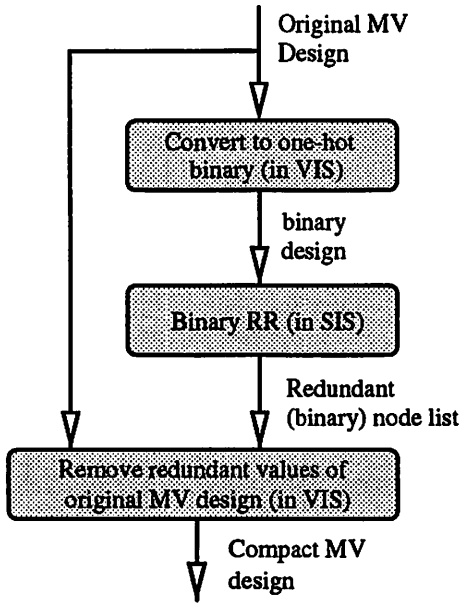

Figure 1: Multi-Valued Redundancy Removal

2.2.1 Creating the Binary Description

The first step of the SMVRR procedure involves encoding the multi-valued design into a one-hot binary design. The multi-valued design under consideration is hierarchical. We first need to flatten the hierarchy since the BRR program cannot handle hierarchical designs. Now each multi-valued node is translated into a series of binary nodes using a one-hot encoding scheme. Latches are also encoded in the same fashion. So if a multi-valued node s has n values, the binary netlist will have n corresponding nodes $s_1, s_2, \cdots, s_n$. Similarly, if a multi-valued latch l has m values, the binary netlist will have m corresponding latches $l_1, l_2, \cdots, l_m$. Since a one-hot encoding is used, $s = j$ iff $s_j = 1$ and all other $s_i = 0$. As a result, the multi-valued design and the binary design are *equivalent modulo encoding*. An example is shown in figure 2.

Since the multi-valued design is hierarchical, but the binary design is constrained not to be hierarchical, the binary variable names are prepended with the full hierarchical path name of the corresponding multi-valued variable. This gives us the ability, given a binary variable name, to find the corresponding multi-valued variable in the hierarchy. This special naming convention is used for all signals, including latch inputs and outputs.

In [1] and [2], the modified circuit is a *c-cycle* replacement for the original one. This means that the behaviors of the two circuits are identical assuming that circuit outputs are observed only after c cycles have elapsed following power-up. The value of c was found to be upto a few thousand. This is a reasonable restriction, since modern sequential designs are often clocked for millions of cycles before any useful work is done.

There are several advantages of this new approach:

- One advantage of the method [2] of computing redundancies is that multiple redundancies are found at once.
- Further, the implication procedure has a more controllable cost than ATPG.
- Finally, these methods are also applicable to sequential circuits, providing us with a scheme to do BRR for sequential circuits without state space search.

2.2 Sequential Multi-valued Redundancy Removal

Our approach for Sequential Multi-Valued Redundancy Removal (SMVRR) is derived from [2], due to its proven efficiencies. We use these breakthroughs in BRR and cast the SMVRR problem as an equivalent BRR problem.

Figure 1 describes the outline of our procedure.

- First, the multi-valued description is encoded into a one-hot binary description.
- Next, BRR is run on the resulting binary description. The redundant one-hot nodes are recorded.
- Finally, the original multi-valued description is modified to reflect the redundancies found in the previous step.

The original network is represented in *blifmv*, the input format to VIS [3], [4]. *blifmv* allows the use of hierarchy in the design. The BRR code is implemented in SIS [5], whose input format *blif* does not handle hierarchical designs. Details of the input formats to VIS and SIS can be found in the corresponding references. Both *blifmv* and *blif* represent the nodes of the design as a *table*.

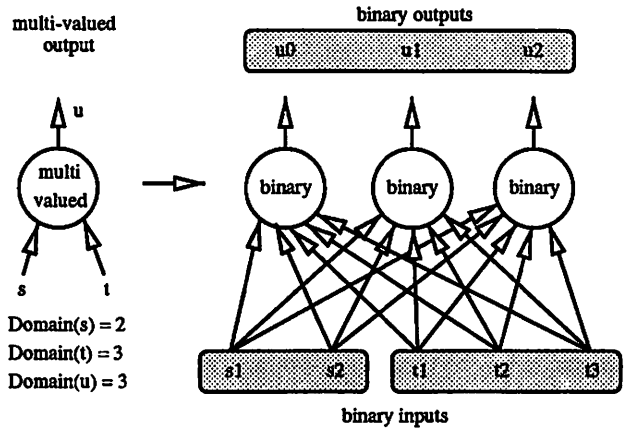

Figure 2: One-hot Encoding of Multi-Valued Node

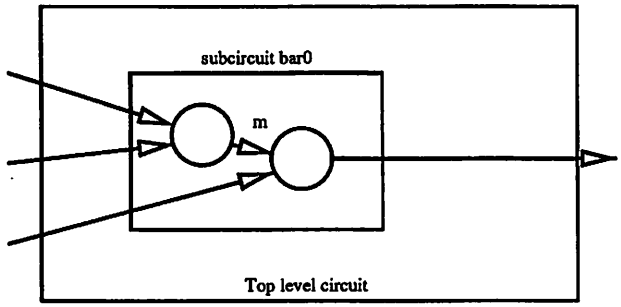

Figure 3: Handling Hierarchy during Encoding

Figure 3 shows an example of the naming of the binary nodes. Assume that the multi-valued node m has k values. The k variables in binary design will thus have names, $bar0.m_1$, $bar0.m_2$, $\cdots$, $bar0.m_k$. To find the multi-valued node corresponding to $bar0.m_i$, we simply descend into the instance bar0, and search for the variable m.

2.2.2 Binary Redundancy Removal

The BRR algorithm we use is reported in [2]. We modify the implementation of [2] to record the discovered redundancies in a separate file. These redundancies are recorded in the order in which they are discovered.

2.2.3 Modifying the Multi-Valued Design

This step involves reading the binary redundancies produced by the preceding step, in the order of their discovery, and modifying the original multi-valued design to reflect the discovered redundancies.

First, the sub-circuit and multi-valued variable corresponding to the redundancy are determined. Since the binary variable names are prepended by the full hierarchical path name of the corresponding multi-valued variable, the correct sub-circuit can be easily found. The affected multi-valued variable is also easily found, since the last field of the binary variable is the same as the corresponding multi-valued variable, appended with the value of the multi-valued variable. For example, the binary variable $foo1.bar3.k_5$ corresponds to the multi-valued variable k. This variable is in the $bar3$ sub-circuit, which is in the $foo1$ sub-circuit of the top level design. This binary variable corresponds to the 5^{th} value of the multi-valued variable k.

Next the affected multi-valued node is modified. Consider a multi-valued node s having m values. There are two possible cases.

- s_j is stuck-at-1 redundant: This means that all other values of s cannot occur (and are therefore stuck-at-0 redundant). In this case we replace the node s with a constant j value. Also, we propagate the constant j value through the circuit, simplifying the network further. This involves:
  - Updating all nodes in the fanout of s to remove their multi-valued minterms which do not contain the j value of s.
  - Removing latches in the fanout of s and replacing them with a constant value j. Further, all nodes in the fanout of such latches are modified so as to remove their multi-valued minterms which do not contain the j value of the corresponding latch variable.
- One or more s_k is stuck-at-0 redundant: This means that the corresponding values k are redundant in the multi-valued network. So we:
  - Update the corresponding multi-valued nodes in the fanout of s. We alter them to reflect the fact that s cannot take on the value k, by removing those multi-valued minterms from the fanout nodes which contain the k value of s.

- Find all latches l in the fanout of s. For all multi-valued nodes in the fanout of l, we remove those multi-valued minterms which contain the k value of l.

If a latch value l_j is found to be redundant, then the procedure is very similar.

- If l_j is stuck-at-1 redundant, then it is removed, and the its immediate fanin node is replaced by a constant j value. Next, all nodes in its fanout are modified to remove their multi-valued minterms which do not contain the j value of l. Similarly, latches l^* in the fanout of l are removed and replaced with constant j values, and all nodes in the fanout of such latches l^* are modified so as to remove their multi-valued minterms which do not contain the j value of the latch variable l^*.

- In case l_j is stuck-at-0 redundant, then all multi-valued nodes in the fanout of l are modified by removing those multi-valued minterms from the fanout nodes which contain the j value of l. We also find all latches l^* in the fanout of l. For all multi-valued nodes in the fanout of l^*, we remove those multi-valued minterms which contain the j value of l^*.

2.3 Correctness of the Scheme

Lemma 2.1 *The binary network obtained after BRR is functionally equivalent to the corresponding multi-valued network after multi-valued redundancy removal.*

Proof: We prove this by induction.

- Base case: The original multi-valued and binary networks are equivalent modulo encoding.

- Induction hypothesis: Assume that the networks are equivalent at step k. Each multi-valued network update is consistent with the corresponding binary network update. Hence the networks are equivalent at step $k+1$.

■

Circuit	# Binary Redundancies	% reduction
ex1	9	13.1
t2	6	9.1
t3	8	9.5
t4	9	5.3
foo	29	13.3
next	3	8.0
random	23	16.7
bar	35	17.0

Table 1: Preliminary Experimental Results - Combinational Designs

3 Experimental Results

In order to prove the efficacy of the SMVRR scheme, we implemented it in VIS [3], [4]. The existing BRR scheme of [2] was modified to dovetail into the SMVRR code. The overall scheme was implemented in the C programming language, and consists of about 3500 lines code. The experiments were performed on a 200 MHz Pentium MMX machine running the Linux operating system.

Preliminary results for combinational circuits are reported in Table 1. The first column represents the circuit name. The second column reports the number of binary redundancies found, while the final column reports the percentage reduction in the size of the multi-valued design after SMVRR. On an average, a reduction of about 12% is obtained. All resulting designs were combinationally verified against the original design, and verified correctly.

The results for sequential circuits are reported in Table 2. The first column represents the circuit name. The second column reports the number of binary redundancies found, while the final table reports the percentage reduction in the size of the multi-valued design after SMVRR. On an average, a reduction of about 18% is obtained. Note that this is much larger than the reduction obtained in the combinational case. In the sequential case, the opportunity for reduction of additional nodes (and latches) is allowed, resulting in this improvement. The resulting designs were not sequentially verified against the original design, since the designs are *c-cycle* equivalent, and hence are not guaranteed to verify in the traditional sequential sense. None of the circuits reported in Table 2 resulted in the removal of a latch

Circuit	# Binary Redundancies	% reduction
srand	21	14.1
readone	25	11.3
start	32	15.2
st	5	16.1
st10	4	28.6
sx	3	20.7
sx1	5	19.5

Table 2: Preliminary Experimental Results - Sequential Designs

altogether. The values of c in all these examples was less than 100.

4 Conclusions and Future Work

Redundancy removal for multi-valued designs has many uses, and can prove to be very helpful in reducing the size of designs, speeding up simulation, and allowing the user to efficiently explore different synthesis options before choosing a final binary implementation for the circuit. Sequential multi-valued redundancy removal can increase the efficiency of formal verification as well.

We have presented a novel method to perform redundancy removal for multi-valued designs. First we showed the analogy between binary and multi-valued redundancy removal. Then, we cast the multi-valued redundancy removal problem as an instance of the binary problem, and solved it using a very efficient binary redundancy removal scheme as the engine.

We implemented the prototype system in VIS and SIS, and preliminary results show an average 12% reduction in the size of combinational multi-valued designs, and an 18% reduction in the size of sequential multi-valued designs.

In the future, we plan to refine the code. The current implementation was meant to prove the concept, and has many areas for possible improvement.

References

[1] M. Iyer, D. Long, M. Abramovici, "Identifying Sequential Redundancies without Search," in *Proceedings of the 33rd Design Automation Conference (DAC)*, 1996.

[2] A. Mehrotra, S. Qadeer, V. Singhal, R. Brayton, A. Aziz, A. Sangiovanni-Vincentelli, "Sequential Optimization without State Space Search," in *Digest of Technical Papers, International Conference on Computer Aided Design (ICCAD)*, 1997.

[3] R. Brayton, G. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, S. Sarwary, T. Shiple, G. Swamy, T. Villa, "VIS : A System for Verification and Synthesis," in *International Conference, Computer Aided Verification*, 1996.

[4] R. Brayton, G. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, S. Sarwary, T. Shiple, G. Swamy, T. Villa, "VIS," in *International Conference, Formal Methods in Computer-Aided Design*, 1996.

[5] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Tech. Rep. UCB/ERL M92/41, Electronics Research Laboratory, Univ. of California, Berkeley, CA 94720, May 1992.

[6] M. Abramovici, M. Breuer, A. Friedman, "*Digital Systems Testing and Testable Design.* Computer Science Press, 1990.