Logic Optimization by An Improved Sequential Redundancy Addition and Removal Technique

Uwe Gläser*

The German National Research Center for Computer Science (GMD)
System Design Technology Institute
Schloß Birlinghoven
D53757 St. Augustin

Kwang-Ting Cheng

Department of Electrical and Computer Engineering,
University of California,
Santa Barbara, CA 93106

Abstract

Logic optimization methods using Automatic Test Pattern Generation (ATPG) techniques such as *redundancy addition and removal* have recently been proposed. In this paper we generalize this approach for synchronous sequential circuits. We proposed several new sequential transformations which can be efficiently identified and used for optimizing large designs. One of the new transformations involves adding redundancies *across time frames* in a sequential circuit. We also suggest a new transformation which involves adding redundancies to *block initialization* of other wires. We use efficient sequential ATPG techniques to identify more sequential redundancies for either addition or removal. We have implemented a sequential logic optimization system based upon this approach. We show experimental results to demonstrate that this approach is both CPU-time efficient and memory efficient and can optimize large sequential designs significantly.

1. Introduction

Some ATPG-based logic optimization approaches have been proposed recently ([ChEn93], [EnCh93], [ChSa94], [KuMe94]). These approaches optimize networks through iterative addition and removal of redundant connections. Adding redundant wires to a network may cause one or many existing irredundant wires and/or gates to become redundant. If the amount of added redundancies is less than the amount of created redundancies, the transformation of adding followed by removing redundancies will result in a smaller network. The basic idea of these approaches is introduced in Figure 1.

The example circuit was taken from [ChEn93]. In this circuit

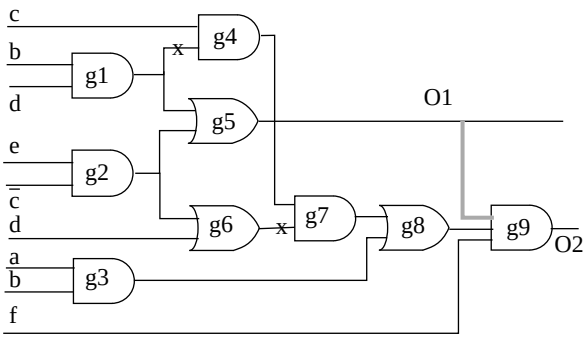

Figure 1: Example for the basic approach

a redundant connection from O1 to g9 is added to the circuit. The addition causes the irredundant connections g1->g4 and g6->g7 to become redundant. Removing these created redundancies results in a much smaller circuit. Identification of such wires to be added has been formulated as a redundancy identification problem for stuck-at faults [EnCh93].

In [ChSa94] several new transformations for combinational circuits based on redundancy addition and removal are shown. They introduced a pertubation method to avoid the solution being stuck at a local minimum. They also presented several *multiple wire addition* techniques. In [KuMe94], recursive learning [KuPr92] is used to derive good Boolean divisors for Boolean optimization of combinational circuits.

This redundancy addition and removal technique can be viewed as a generalization of redundancy removal and boolean resubstitution. In this paper we apply this method for optimizing sequential circuits. We generalize this technique by introducing several new and useful transformations that can be identified efficiently by ATPG-based techniques. Our method allows boolean minimization for synchronous sequential circuits with feedbacks. In addition to transformations used in previous ATPG-based approaches,

*: The work was done when the author was with University of California at Santa Barbara.

we allow *the addition of flip flops* in order to create more redundancies for better optimization. We also propose a new technique to create desired redundancies by blocking the justification of the wires or gates. We use state-of-the-art sequential ATPG techniques as the base for sequential redundancy identification (for either addition or removal).

For combinational circuits, there is no difference between redundancy and untestability of wires. That is, if a wire is redundant, the corresponding fault at the wire is untestable. But for sequential circuits, there may exist wires which are untestable but not redundant [Chen93]. The main reason for this is that a wire in a sequential circuit may be partially testable. That is, it is testable if the power-up state (initial state) of the circuit is in a particular set of states and untestable if the initial state is not one of those states. These faults will be classified as untestable by an ATPG tool. Addition of such an untestable and irredundant fault will change the functionality of the circuit and could not be allowed. Some issues on identification of sequential redundant faults were discussed in [Chen93].

To determine whether a wire added to a circuit is redundant and which wires become redundant after the addition of a redundant connection is the key process of the technique. Efficient techniques for answering these questions for combinational circuits have been proposed in [EnCh93], [ChSa94] and [KuMe94]. It is too computationally expensive to run a complete test generation to determine redundancy for a large sequential circuit. We use only *mandatory assignments* to identify a redundancy. The set of mandatory assignments is a subset of all necessary assignments for detecting a fault. If the set of mandatory assignments is inconsistent, the corresponding fault is redundant. We compute the mandatory assignments by using the implication and the unique sensitization functions of the FAN-algorithm [FuSh83]. Since we use only mandatory assignments for redundancy identification and it is possible that a redundant fault could have a set of consistent mandatory assignments, the circuit resulting from our optimization approach is not necessarily 100% irredundant.

We use the well-known time frame model to model the sequential circuits. In order to identify sequential redundancies, it is necessary to compute mandatory assignments across time frames in sequential circuits, i.e. whenever a flipflop is reached, the computation of mandatory assignment is continued in a corresponding later or earlier time frame. Our implementation can handle a large number of time frames and can therefore identify some hard-to-identify sequential redundancies. In addition to using mandatory assignments in sequential circuits, we also use a data flow analysis to detect wires which are not observable to further improve the efficiency as well as the effectiveness of our sequential redundancy identification process.

The rest of the paper is organized as follows. Section 2 presents the ideas of redundancy addition and removal across time frames and redundancy addition for causing inconsistency in justification for the target fault. In section 3 we discuss the algorithm and the implementation. Experimental results are given in section 4 followed by conclusions.

2. Adding sequential redundancies

In this section the method of adding and removing redundancies is described in detail. There are basically two ways to make an irredundant fault to become redundant and thus removable:

- Adding redundant logic (wires and/or gates) to block the propagation of the fault [EnCh93].
- Adding redundant logic to block the initialization of the fault.

If a redundancy is added in order to block the propagation of a fault, the wire is added to one of the dominators of the target fault. Dominators of a fault are the gates that the fault must be propagated through to reach a primary output for detection. As stated in the introduction, this is the main idea used in [EnCh93] and [ChSa94]. Since the ATPG process depends on both propagatability and justifiability of the fault, blocking the justification is a second possibility to make a fault redundant. If blocking the justification of the fault is desired a redundancy has to be added to the input cone of the to-be-removed target wire. This method will be illustrated in the next section.

For sequential circuits, redundancy addition and removal is performed across time frames. The time frame model and a method of adding redundant connections across time frames is described in section 2.2.

2.1. Adding redundancy to block justification

We have generalized the redundancy addition technique to allow adding redundant wires to the transitive fanin of a target fault. The addition will make justification of the target fault impossible and thus make the target fault redundant and removable. The idea is illustrated in the example in Figure 2.

In order to test the s-a-0 fault in Figure 2, signals a, b and d have to be assigned to “1” value and c has to be assigned to “0” value. As there is a mandatory assignment “1” at d, we can use this value for blocking the initialization of the fault. The candidate connection for addition $\bar{d}$->e as shown in a dashed line is in the transitive fanin of the target fault. The addition will make it impossible to justify a “1” at wire e and in the meantime propagate the fault effect to the output. Therefore, this addition will make the target wire redundant. It can be verified easily that the added wire $\bar{d}$->e in Figure 2 is redundant itself since the propagation of the

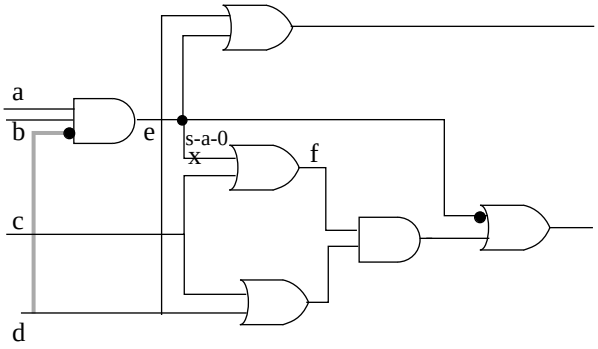

Figure 2: Blocking the initialization

corresponding fault is prohibited. Therefore, we can add the dashed wire without changing the circuit functionality and in turn remove the target wire (e->f stuck-at 0).

For sequential circuits, adding of such redundancies will either (1) make the state required to test the target fault unreachable or (2) invalidate the initialization sequence for the target fault. For both cases, the target fault will become sequentially redundant.

2.2. Adding redundancies across time frames

We can *add flip-flops* to create sequential redundancies, if it results in a better design. To analyze a sequential circuit, the time frame model (or called the iterative array model) is commonly used. Consider the iterative array model shown in Figure 3.

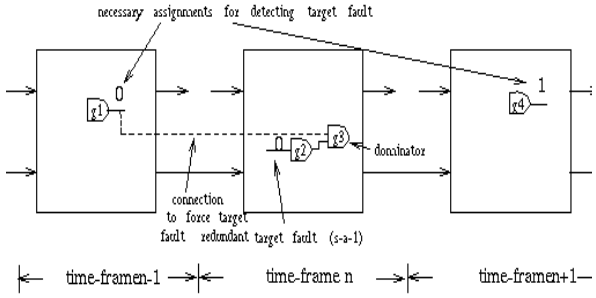

Figure 3: The iterative array model

Suppose that we would like to find a set of candidate connections that their addition would cause the target fault (input of g2 stuck-at-1) to become redundant. To detect this fault, the target wire must have a value 0 for the fault activation and the side inputs of the dominators (including g3) must be in their non-controlling values for fault propagation. The implication of these initial assignments is then performed. The implication will be performed across time frames. After the implication is completed, candidate connections to be added to make the target fault redundant will be collected and examined one at a time to check whether it is redundant.

The candidate connection list includes the following connections (with a proper polarity):

- from one of the signals that has an implied value to a dominator of the target fault or
- from one of the signals that has an implied value to a gate in the transitive fanin of the target fault and its addition will block the initialization.

Current techniques [EnCh93] only allow the source and destination of the added connection to be within the same time frame. In addition to that, we provide a technique for redundancy addition across time frames. In Figure 3, the connection from g1 at time frame n-1 (with implied value 0) to g3 in time frame n (dominator of the target fault) could block the propagation path and thus make the target fault redundant. In a real implementation, this transformation will require the addition of a flip-flop. Signal g1 has to be delayed by one cycle to block the propagation path at g3 as shown in Figure 4. This transformation is very powerful because it basically allows us to freely add and remove flip-flops (or equivalently to change the state assignment dramatically) during sequential logic optimization and thus could potentially exploit more circuit configurations and eventually lead to a better design.

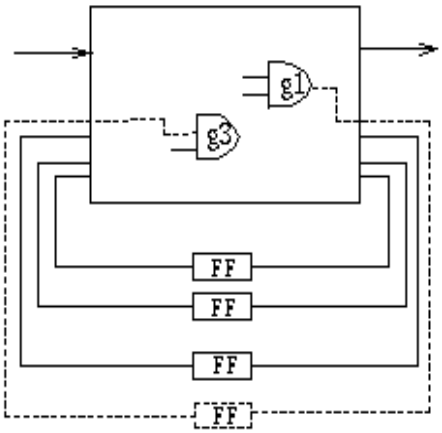

Figure 4: Adding a redundancy across time frames

Checking redundancy for such cross-time-frame candidate connections is similar to those of same-time-frame candidate connections. Suppose a candidate connection from time frame n-i to time frame n has been verified to be redundant and can be added, i flipflops need to be added between the source and destination of the added connection in the final netlist to guarantee to correct functionality of the circuit.

In Figure 5, a simple example circuit is shown to illustrate the usefulness of this cross-time-frame transformation. Assume the target wire to be removed is s1. The mandatory assignment for the stuck-at-1 fault at

s1 at time n consists of a 0 assignment at wire O1 at time n-1 (at time n-1, g6, d, g1, g2, g5 and in turn O1 have to be 0). Therefore, O1 at time n-1 to g9 at time n (a dominator of the target wire) is a candidate connection. This transformation requires adding a new flipflop to the register as shown in shaded lines. This added connection and flip-flop can be identified as redundant since the set of mandatory assignments for the corresponding fault is not consistent. The added logic blocks the propagation of the stuck-at-1 fault at s1. Removal of this fault will result in the removal of one flipflop in the register and gates g7 and g6 and thus result in a much smaller design.

We like to point out that it is not allowed to add a

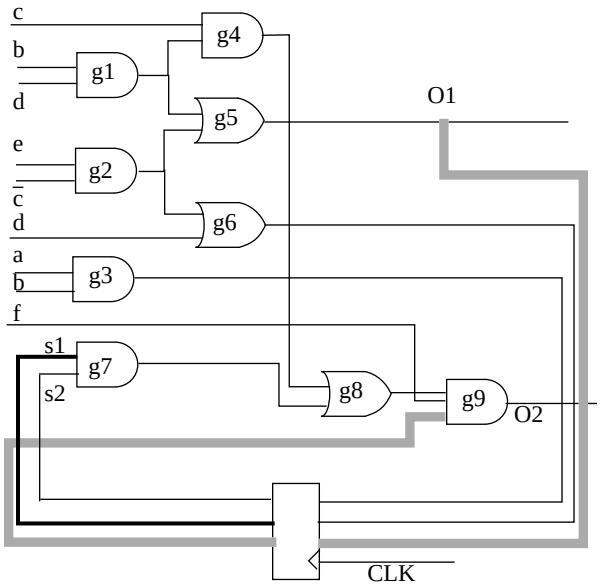

Figure 5: Example for redundancy addition and removal in a sequential circuit

connection from a gate in time-frame n to another gate in time-frame n-i for any positive integer i because a *negative* delay cannot be represented.

2.3. Issues on sequential redundancy identification

In contrary to the approach for combinational circuits we need a multi fault model for sequential circuits that every time-frame has a fault. As in test generation the fault appears in every time frame. Because we use only mandatory assignments for redundancy identification, it may be intuitive that we only need to use a single fault model - only one time-frame has a fault. In the single fault model, only the time frame that the target fault is first activated has a fault. The following example illustrates that the single fault model will cause inaccuracy in redundancy identification and could not be used. Figure 6 shows that an untestable fault could be mistakenly claimed as redundant if the single fault model is used.

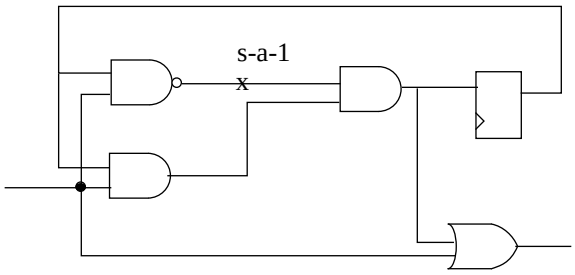

Figure 6: The multi fault model

The circuit in Figure 6 consists of one cycle including the fault location. If we use a single fault model, the propagation of the fault is blocked in a later time frame at the fault location. On the other hand, if we use the multiple fault model, the set of mandatory assignments is consistent and thus the fault is not redundant. This proves the necessity of using a multiple fault model in our approach, where the fault appears in every time frame.

3. The overall algorithm

The overall algorithm of the sequential redundancy addition and removal is basically an extension of the algorithm in former approaches [EnCh93] [ChSa94]. The main difference is that our approach has an enhanced set of sequential transformations and we have efficient techniques to identify sequential redundancies. The overall algorithm is as follows:

```
seq_optimization(){
      redundancy_removal();   (1)
      for i=1 to number_iterations {
          fanout_pertubation();   (2)
          fanin_pertubation();   (3)
          greedy_optimization();   (4)
      }
}
```

In (1) redundancies are removed from the circuit. For the redundancy identification, mandatory assignments are used. If the set of mandatory assignments is inconsistent, the corresponding fault is identified as redundant. Sequential redundancies can possibly be removed since the mandatory assignments are computed across time frames whenever they exist. In (2) fanout pertubation is performed followed by fanin pertubation (3). These processes are similar to the ones proposed in [ChSa94] and are generalized for sequential circuit such that they can be executed across time frames. The main optimization is performed in the greedy optimization step (4). The pertubation functions in (2) and (3) usually don't optimize the circuit themselves, they are mainly

used for getting a network out of a local minima. The greedy optimization function is described as follows:

```
greedy_optimization() {
forall circuit wires w1 {
  compute mandatory assignments(start_time);   (5)
  forall wires w2 with mandatory assign. at time t {
     add wire s from w2 to dominator or controller of w;
     delay s by adding (t - start_time) flip-flops   (6)
     if s is redundant {
        delete w1 from circuit   (7)
        detect further redundancies   (8)
        detect cycles w/o path to a PO (9)
     } else
     delete s from circuit   (10)
  } /* for w2 */
} /* for w1 */
}
```

The greedy optimization function tries to remove a selected wire from the circuit. For this purpose mandatory assignments for the corresponding fault are computed first (5) in order to collect candidate connections which might cause the target wire to become redundant. All those candidate connections have a destination gate which is either a fanin-controller or a dominator of the fault in the circuit. The fanin-controller of a fault is a gate that is in the transitive fanin of the target fault and has a mandatory value in order to activate the target fault. Since a candidate connection might be across several time frames, the connection has to be properly delayed then (6). We then examine the candidate connections one at a time to check whether they are redundant. If the candidate connection s is irredundant s is deleted from the circuit (10). If s is redundant, we may delete w1 immediately from the circuit, since addition of s will cause w1 to be redundant (7), because s was added such that initialization or propagation of the fault at w1 is prohibited. As addition of s might cause more than one redundancy, the circuit is checked for further redundancies (8). The last step is to check whether there exists a cycle in the circuit which does not lead to a primary output. For a combinational circuit, if there are some logic not reachable to primary outputs, there must exist a gate which does not fanout to any other gates or primary outputs. However, it is not necessarily the case for sequential circuits. During optimization, it may have isolated logic that forms a cycle. All gates in the isolated logic have fanouts. Therefore, we need to perform a special check for removing such logic. For this purpose, the fanout stems of all removed branches are checked by a reachability analysis (9). If no primary output can be reached from the stem, a prime fanout branch was removed and thus the whole cycle can also be removed since it does not take part in the circuit function any more.

3.1. Implementation

We have implemented the algorithm in C++ language and has about 11500 lines of source code. The system is named RADAR_S (an enhanced Redundancy ADdition And Removal system for sequential logic optimization). For comparison purpose, we also implement a simpler version for combinational circuits, named RADAR_C, based on the algorithms in [EnCh93] and [ChMa93].

Although theoretically the number of necessary time frames used for mandatory assignments might be large (probably several thousand), the number of time frames used by our software is limited for practical reasons. One reason for the limitation is the main memory requirement for storing each time frame which is calculated below for our approach. As stated below, keeping all time frames in the main memory at the same time is affordable for our approach. In our experiments with the ISCAS'89 benchmarks [BrBr89] we found out that limitation of the number of time frames to less than 100 is senseful. For our experiments shown in Section 4 we used 100 time frames as an upper limit. This limit is user provided such that, if desired, a smaller or larger limit could be used.

The main memory requirement in the software is very low due to an efficient implementation of our approach. In addition to the memory requirement of storing the circuit graph we need only 12 bytes for every node and every time frame. For example a circuit with 100k graph nodes would require 120 megabytes of main memory by using 100 time frames which is not much for a state of the art workstation.

4. Experimental results

In our experiments we used the ISCAS'89 sequential benchmark circuits [BrBr89] and circuit Am2910. Results and CPU-times were computed on a Sparc10 workstation. As a measure for the size of the circuit we used the number of two-input gates with the assumption that an n-input gate is equivalent to n-1 two-input gates.

For the experimental results shown in Table 1, as one experiment we used the benchmarks and first optimized the combinational logic using SIS from Berkeley (using script.rugged). Then we used RADAR_C to optimize the combinational logic further and remove combinational redundancy. We can consider that these results are what we can get using existing combinational methods and tools. For the circuit s13207, SIS cannot complete the job and the results are produced by only RADAR_C.

As a second experiment we applied RADAR_C and RADAR_S (including sequential redundancy removal) on the original ISCAS benchmark circuits. The results

are shown in the last two columns of Table 1. We found that the circuits resulting from the optimization are most of the times smaller than the optimized circuits running through SIS and RADAR_C. For two circuits only (s526, s820) the results of running RADAR_C+RADAR_S are a little worse compared with results obtained from SIS + RADAR_C. The CPU times for RADAR_C+RADAR_S are given in the last column. It is worth mentioning that ATPG-based methods such as RAMBO and RADAR use gate count or connection count as the cost function for optimization while SIS uses literal count as the cost function.

Table 1: Results for the sequential ISCAS benchmark circuits running SIS and RADAR_S

Circuit	initial circuit size # 2- input gates / # FF	SIS + RADAR_C.	RADAR_S with integrated RADAR_C	
		# 2- input gates / # FF	# 2- input gates / # FF	CPU [s]
s208	70 / 8	70 / 8	36 / 5	26
s298	125 / 14	98 / 14	91 / 14	57
s344	109 / 15	105 / 15	102 / 15	68
s382	148 / 21	133 / 21	116 / 21	153
s400	148 / 21	128 / 21	116 / 21	156
s420	140 / 16	139 / 16	38 / 5	36
s444	156 / 21	127 / 21	121 / 21	168
s510	213 / 6	205 / 6	205 / 6	3855
s526	251 / 21	156 / 21	179 / 21	337
s713	160 / 19	158 / 19	132 / 16	320
s820	468 / 5	282 / 5	285 / 5	4277
s832	468 / 5	273 / 5	265 / 5	4302
s1423	491 / 74	442 / 74	440 / 74	1227
s1488	734 / 6	582 / 6	567 / 6	31933
s1494	733 / 6	587 / 6	568 / 6	27776
s5378	1389 / 176	917 / 176	819 / 124	2027
s13207	2573 / 667	2013 / 667	1347 / 443	45908
AN2910	883 / 99	Not applied	827 / 99	7955

5. Conclusion

Redundancy addition and removal has been shown recently by several research groups to be an efficient method for Boolean optimization of combinational and synchronous sequential circuits. In this paper we further extend this method by incorporating several new powerful transformations. The generalization includes addition of redundant connections across time frames and addition of redundancies to invalidate justification. Using such powerful structure transformations will allow exploration of different circuit configurations with completely different state assignments and thus lead to a better optimized design. We use the time-frame model and state-of-the-art sequential test generation techniques to efficiently identify sequential redundancies for either addition or removal. Experimental results are promising even for large sequential circuits.

6. References

[FuSh83] H. Fujiwara, T. Shimono: On the Accelaration of Test Generation Algorithms, IEEE Trans. on Computers (C-32), pp. 1137-1144, 1983

[ChCh89] W. Cheng, T. J. Chakraborty: GENTEST An Automatic Test-Generation System for Sequential Circuits, IEEE Computer'89, pp.43-49

[ChMa93] K. T. Cheng, T. Ma: On the over-specification problem in sequential ATPG algorithms, IEEE Trans. Computer aided design, october 1993, pp. 1599-1604

[BrBr89] F. Brglez, F. Bryant, D. Kozminski: Combinational Profiles of Sequential Benchmark Circuits, Proc. 1989 Int. Symp. Circ. and Systems

[ChEn93] K.-T. Cheng and L. Entrena: Multi-Level Logic Optimization by Redundancy Addition and Removal, Proc. European Conf. on Design Automation (EDAC-93), Feb. 1993.

[EnCh93] L. Entrena, and K. T. Cheng: Sequential Logic Optimization by Redundancy Addition and Removal, Proc. ICCAD-93, pp. 310 - 315

[Kunz93] W. Kunz: HANNIBAL: An Efficient Tool for Logic Verification Based on Recursive Learning, Proc. ICCAD-93, pp. 538 - 543

[ChCh93] S. C. Chang, K. T. Cheng, N.-S. Woo and M. Marek-Sadowska: Layout-Driven Logic Synthesis for FPGAs, Proc. DAC-93, pp. 308 - 313.

[KuPr92] W. Kunz and D. K. Pradhan: Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation for Digital Circuits, Proc. ITC-92, pp. 816 - 825

[ChLe91] J. E. Chen, C. L. Lee and W. Z. Shen: Single-Fault Fault-Collapsing Analysis in Sequential Logic Circuits, IEEE Trans. on CAD Dec. 1991

[Chen89] W. Cheng: The BACK-Algorithm for Sequential Test Generation, Proc. ICCD '88, pp. 66-69

[ChSa94] S. C. Chang and M. Marek-Sadowska: Perturb and Simplify, Multi-level Boolean Network Optimizer, Proc. ICCAD-94, pp. 2 - 5

[KuMe94] W. Kunz and P. R. Menon: Multi-level Logic Optimization by Implication Analysis, Proc. ICCAD-94, pp. 6 - 13

[Chen93] K.T. Cheng: Redundancy Removal for Sequential Circuits Without Reset States, IEEE Trans. on Computer-Aided Design, Vol. 12, pp. 13-24, Jan. 1993.