ISSN 1392 – 1215 ELECTRONICS AND ELECTRICAL ENGINEERING 2011. No. 6(112)
ELEKTRONIKA IR ELEKTROTECHNIKA

T 170 ELECTRONICS
ELEKTRONIKA

Verification of Initialization Sequences for Sequential Circuits

K. Morkunas, R. Seinauskas
Software Engineering Department, Kaunas University of Technology,
Studentu str. 50-406, LT-51368 Kaunas, Lithuania, phone: +370 670 75907, e-mail: kestutis.morkunas@ktu.lt
crossref http://dx.doi.org/10.5755/j01.eee.112.6.446

Introduction

An initializing sequence is a set of inputs that switches a system from any unknown state into a single fixed state. Such sequences may not exist, or be hard to find.

There are two ways to set flip-flop values in sequential circuits. One is inputting patterns which switch flip-flop values from unfixed to fixed ones. It is also possible to use scan, pushing required values directly into flip-flops. Both methods have their advantages and disadvantages.[1–12] An initializing sequence may not exist, be too long, or set a circuit into an undesired state [1–4]. Using scan allows easy flip-flop setting, but also increases chip size due to direct access requirement. This increase the chip price, heat output and power consumptions. Breaks in these extra connection lines may result in failed quality tests.

Finding an initializing sequence is not an easy task. It is simple if states and transitions vectors sets are fully known. Forming such a transition tree is difficult for large circuits with many flip-flops present [2]. I.e. 3 inputs and 3 triggers would result in a state space size of 8 and 8 inputs. It will take 64 computation cycles to test transitions from state to state for all possible input/state combinations. 14 inputs and 6 triggers result in 1’048’576 computation cycles. s35932 circuit from ISCAS’89 would require 5,20e+530 computation cycles for a full test.

Previous work and experiment pre-conditions

This article is based on random search algorithm. It was used to discover initializing sequences for circuits. The circuits’ operations were emulated using software prototypes and not actual hardware. These emulating prototypes were translated from ISCAS’89 verilog source files.

Such method operates under more difficult conditions, as there is no knowledge of the inner circuit wiring and elements [6–10]. Therefore, separate elements, gates, flip-flops may not be analysed, joined into groups or removed as suggested by other researchers. In experiments the results proved to be as good as or better to those in similar research papers [6–8].

Verification of found initializing sequences is based on heuristics. We state that, if input patterns managed to bring a system from a large number of starting states (50,000) into a fixed state, there is a good chance it will do so with larger numbers as well. Yet there is no 100% guarantee until transitions from all possible starting states are checked.

Initializing sequences were revalidated against some of the ISCAS’89 benchmark circuits using Verilog, part of Cadence collection. Verilog is a hardware description language that allows creation of tests using four values logic (0,1,X,Z). In our experiments, only three (0,1,X) were used. X stands for an unknown trigger value, which may be 0 or 1.

An initializing sequences were found and validated using two different methods with two sets of results as an outcome. First set came from using and initializing sequence with an algorithm described. Second set of results was produced by emulating the operation of circuit using Verilog. S386 circuit (it’s emulated software prototype) was tested.

Table 1. AND(&) gates calculation difference

AND, &	**0**	**1**	**X**
0	0	0	0
1	0	1	X
X	0	X	X

The results from two methods differed. Our random search algorithm uses two value logic. Verilog’s decisions are explained in Table 1.

Result difference between ternary and bi-valued logics

Upon noticing differences in results, additional inspection and checks were used. To do this, an entire trigger state space was generated. S386 circuit has 6 triggers, which results in 64 possible states. An initializing sequence was used, which consisted of two different input patterns. Change of state space was inspected after using each of input patterns.

Table 2 displays some of the changes in the state space using an algorithm described in this article. The process goes like this: a system exists in an unknown state.

Each input pattern from the initializing sequence is used. Each input transfers the system from a previous state to a new one. Output is also produced

Table 2. Changes in state space using binary logic

Description/Action	State space
Starting space set	000000, 000001 …111111
First input pattern is applied	
Resulting state space (unique states only)	001100, 000100, 000000
Second input pattern is applied using resulting states space	
Resulting state space (unique states only)	001100

Table 2 clearly displays that using and initializing sequence of two input patterns and a full state space, all circuit's triggers are switched into a fixed state (001100). After using first input pattern on full trigger state space, three unique states appear in output. It means that the first input signal is able to switch circuit's triggers into one of three possible states, no matter what was the starting state of circuit's memory and trigger values. So, a full initialization of all 6 of 6 triggers is achieved.

Using the same initializing sequence on Verilog provided different results. The number of initialized triggers was 5 out of 6. Upon closer inspection, it appeared that this difference in results might be caused by Verilog's use of 3-valued logic. It appears that after step 1 Verilog discovered that triggers 1,2,5,6 were fully initialized. For triggers 3 and 4 it generated and used all possible states (00;01;10;11), although Table 2 clearly displays, that 10 state can never be reached during operation of the circuit. If all 4 states are used (Verilog's 3-value logic example) instead of 3 states (proposed algorithm based on 2-value logic), then one of the triggers stays in an unset state (X).

Additional experiments were made to test the proposed algorithm in such conditions. The proposed algorithm was altered to simulate basic 3-value logic. In such case, the results changed and became the same as those produced by Verilog's emulation.

This proves, that using 2-value logic does have it's benefits. Such a method allows removing illegal states from the state space, thus reducing it's size and finding more accurate initializing sequence. It also allows more triggers to be switched into fixed states.

The search for an initializing sequence

Search for an initializing candidate consists of: generating a small set of starting system states (20 in our case) and a larger number of input patterns (100 in this experiment). A state-to-state transition is checked for each input pattern to starting state combination. After all resulting states for each input pattern are calculated, an analyzer starts. It checks how many flip flops are switched from random into fixed states. After all input patterns are tried, best setting pattern is added as a part of full circuit initializing sequence. Such sequence may consist of 1 or more input patterns. The length was limited to maximum of 50 input patterns in this research. After this step, a new set of input patterns is generated, yet the old, partly fixed flip flop state set is used. The search is stopped if initializing sequence length goes beyond the length limit or the number of set flip-flops does not increase with newly generated sets of input patterns.

Upon discovering an initializing candidate, it is taken for verification. An initializing sequence made of one or more input patterns is used, and a large number of starting system states is randomly generated. 50,000 of starting states were used for verification of solution in this experiment.

There is no guarantee, that found initializing sequence is 100% correct, because the starting states set is only a small fraction of the whole set of possible states.

The presumption is based on an idea that if initializing sequence is able to switch a system into a fixed state from a small number of states, and also a large number of states, it might be true for all starting states. Small number of starting states is used to minimize the computer calculation time consumption.

Process of initializing sequences verification

Verification normally approves the found solution, yet sometimes the solution does not pass through [11]. In this case, the validation process outputs the number of fixed flip-flops. The number is always smaller, yet a smaller result might still be an acceptable solution.

The pseudo-code of initialization sequence search algorithm used is:

```
CALL generateInputs(100)
CALL generateStates(10)
FOR each input signal
  FOR each state
    CALCULATE CircuitOutput(input, state)
  END FOR
   IF resetFound(circuitOutputsArr) THEN
     SET initSeqCandidate
  ELSE
     INCREMENT initSeqPointer
     SET initSeqCandidatePart(initSeqPointer)
  END IF
END FOR
```

The validation of solution pseudo-code is:

```
CALL generateStates(50000)
FOR each state
   CALCULATE CircuitOutput(initSeqCandidate, state)
END FOR
IF resetFound(circuitOutputsArr) THEN
 DISPLAY validationSuccessful
ELSE
  DISPLAY actualValidationResults;
END IF
```

The number of set flip-flops during validation is more reliable, as it uses a much large number of starting system states. Imitation of ternary logic is implemented by adding an additional randomization factor into validation process. Pseudocode:

```
CALL generateStates(50000)
FOR each input pattern from initializing sequence
  FOR each state
     CALCULATE CircuitOutput(input, state)
     CALL randomizeUnsetTriggers();
  END FOR
END FOR
IF  resetFound(circuitOutputsArr) THEN
 DISPLAY  validationSuccessful
ELSE
  DISPLAY actualValidationResults;
END IF
```

An additional randomization function was added. This should create conditions more similar to Verilogs' results. Comparing results on bivalue and ternary testing for circuit s386 (described earlier in this article), this method allows to imitate using X for unset values. In previous example system could only gain values 00;01;11 and never 10 for two of it's flip-flops. And randomization after each input pattern could possibly introduce the missing 10 into the state space used in calculations.

Results of both methods are displayed in Table 3. Binary method is described in depth in [12].

Table 3. Verification results based on two approaches

Circui t no.	**Trigger s**	**Set triggers using binary**	**Set triggers, terniary imit.**	**Seq. length**
s13207	638	477 (74,7%)	432	15
s1423	74	74 (100%)	74	3
s15850	534	458 (86%)	458	18
s298	14	14 (100%)	14	2
s38584	1426	1423 (99,78%)	1423	36
s386	6	6 (100%)	6	2
s526	21	21 (100%)	21	2
s5378	179	167 (93%)	163	9
s9234	211	154 (73%)	154	6
s953	29	25 (86%)	10	1

Only those circuits with an initialization sequence length of 2 or more input patterns were analysed.

Tests using ternary logic turned out to decrease the number of set flip-flops in three cases. Knowledge of circuits inner structure was not used, therefore it is hard to explain the large difference between circuit testing results.

Results of initialization sequences verification process (proposed approach)

For some circuits, found initializing sequences set different number of flip-flops while using binary, simulated ternary and hardware ternary logic. Testing the manufactured hardware circuit using ternary logic provides most reliable results displaying unset flip-flops as having X values instead of fixed ones (0 or 1).

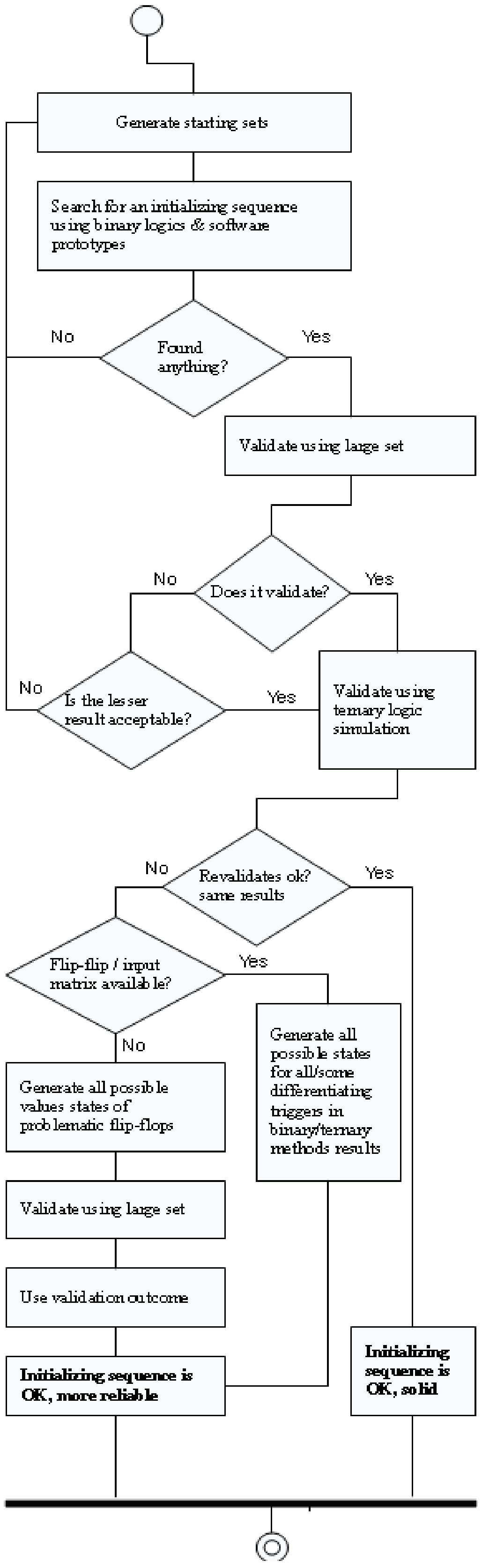
Generate starting sets
Search for an initializing sequence using binary logics & software prototypes
No
Found anything?
Yes
Validate using large set
No
Does it validate?
Yes
No
Is the lesser result acceptable?
Yes
Validate using ternary logic simulation
No
Revalidates ok? same results
Yes
Flip-flip / input matrix available?
Yes
No
Generate all possible values states of problematic flip-flops
Generate all possible states for all/some differentiating triggers in binary/ternary methods results
Validate using large set
Use validation outcome
Initializing sequence is OK, more reliable
Initializing sequence is OK, solid

Fig. 1. Proposed algorithm for verification of initialization sequences

Simulated ternary method described in this article provides similar results, yet is based on heuristics, because full set of state-transition vectors are not available. It is possible to generate this set for small sized circuits, yet it is not possible or impractical for circuits with large number of flip-flops. Search using binary seems to provide best results in terms of the number of set flip-flops and with the minimal length of initializing sequence.

Due to this difference in results, and approach is employed, which might give more reliable results. It is described in Fig 1.

Conclusions

This article suggests and approch to verification of initializing sequences for sequential circuits. Binary method migt be considered as providing optimistic results, as ternary – pesimistic ones. This article also explains the difference between these two methods, the differences in results and why this is happening.

Using ternary approach might prove useful if validation results for both methods differ greatly. Binary method appears to be more accurate at finding initializing sequences, because of reduction to states space after each of input patterns is used. This is not the case when using ternary logic testing under Verilog.

References

1. **Pomeranz I., Reddy S. M.** On the Detection of Reset Faults in Synchronous Sequential Circuits // VLSI Design, 1997. – P. 470–474.
2. **Cheng K., Agrawal V.** Initializability consideration in sequential machine synthesis // IEEE Trans. Comput, 1992. – Vol. 41. – P. 374–379.
3. **Wehbeh J. A., Saab D. G.** On the Initializationof Sequential Circuits // Intl. Test Conf., 1994. – P. 233–239.
4. **Pomeranz I., Reddy S. M.** On Removing Redundancies from Synchronous Sequential Circuits with Synchronizing Sequences // IEEE Trans., 1996. – P. 20–32.
5. **Keim M., Becker B.** On the (Non–)Resetability of Synchronous Sequential Circuits // IEEE VLSI test symposium, 1996. – P. 240–245.
6. **Xiaojing H., Zhengxiang S.** Ant Colony Optimizations for Initalization of synchronous ssequential circuits // IEEE Circuits and Systems International Conf., 2009. – P. 5–18.
7. **Corno F., Prinetto P.** Initializability analysis of synchronous sequential circuits // ACM Trans. on Design Automation of Electronic Systems, 2002. – Vol. 7. – No. 2. – P. 249–264.
8. **Lu Y., Pomeranz I.** Synchronization of Large Sequential Circuits by Partial Reset // IEEE VLSI Test Symp., 1996. – P. 93–98.
9. **Bareiša E., Jusas V., Motiejūnas K., Šeinauskas R.** Functional delay test generation based on software prototipe // Microelectronics Reliability, 2009. – No. 49(12). – P. 1578–1585.
10. **Bareiša E., Jusas V., Motiejūnas K., Šeinauskas R.** On the Enrichment of Functional Delay Fault Tests // Information Technology and Control, 2009. – No. 38(3). – P. 208–216.
11. **Bareisa E., Jusas V., Motiejunas K., Seinauskas R.** On Delay Test Generation for Non–scan Sequential Circuits at Functional Level // Electronics and Electrical Engineering. – Kaunas: Technologija, 2011. – No. 3(109). – P. 67–70.
12. **Morkūnas K., Šeinauskas R.** Circuit Reset Sequences based on Software Prototypes // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 7(103). – P. 71–76.

Received 2011 02 15

K. Morkunas, R. Seinauskas. Verification of Initialization Sequences for Sequential Circuits // Electronics and Electrical Engineering. – Kaunas: Technologija, 2011. – No. 6(112). – P. 61–64.

This article suggests an approach for verification of initializing sequences. Such sequences were discovered using circuit emulating software prototypes. Software prototypes operate using bivalent logics (0 and 1), while hardware testing employs ternary logic (0, 1 and X). Experimental results show, that validation using ternary logic is too strict, labeling good initializing sequences as bad ones. Experimental results are based on ISCAS’89 benchmark. Ill. 1, bibl. 12, tabl. 3 (in English; abstracts in English and Lithuanian).

K. Morkūnas, R. Šeinauskas. Mikroschemų sekų nustatymo verfikavimas // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2011. – Nr. 6(112). – P. 61–64.

Šiame straipsnyje pasiūlytas mikroschemų nustatymo sekų verifikavimo metodas. Tokios sekos buvo gautos naudojant mikroschemas imituojančius programinės įrangos prototipus. Prototipai operuoja naudodami dvireikšmę logiką (0 ir 1), kai, testuojant sintezuotas schemas, galima naudoti trireikšmę logiką (0, 1 ir X). Eksperimentų rezultatai rodo, kad trireikšmė logika yra per griežta ir atmeta teisingas nustatymo sekas kaip klaidingas. Eksperimento rezultatai pagrįsti ISCAS’89 testinių mikroschemų rinkiniu. Il. 1, bibl. 12, lent. 3 (anglų kalba; santraukos anglų ir lietuvių k.).