ELECTRONICS AND ELECTRICAL ENGINEERING
ISSN 1392 – 1215 *2010. No. 7(103)*
ELEKTRONIKA IR ELEKTROTECHNIKA

SYSTEM ENGINEERING, COMPUTER TECHNOLOGY
T 120
SISTEMŲ INŽINERIJA, KOMPIUTERINĖS TECHNOLOGIJOS

Circuit Reset Sequences based on Software Prototypes

K. Morkūnas, R. Šeinauskas
Department of Software Engineering, Kaunas University of Technology,
Studentų 50-404, LT-51368 Kaunas, Lithuania, phone: +370 670 75907, e-mail: kestutis.morkunas@ktu.lt

Introduction

Present-day circuits manufacturing is expected to deliver top-class product quality in the shortest time frame possible. Product might be finished faster by reducing time needed for various stages of crafting process, like design, implementation, test generation, etc. This can be done by using automated design tools, various chip size optimization techniques and more accurate, reliable and speedy tests.

Concurrent design and test generation may come in hand. Normally, a circuit is tested after it is synthesized and burned on a chip. This step is important, and can not be avoided. The manufacturing process is displayed in Fig. 1. Best case scenario is when test cases are ready at the time of chips' completion and the test generation process does not require any more time. Various software prototypes emulating the designed chip can be made before and during the specification generation phase. It helps to find design flaws and detect errors early in the process.[10]

Chip fabrication process is as follows:

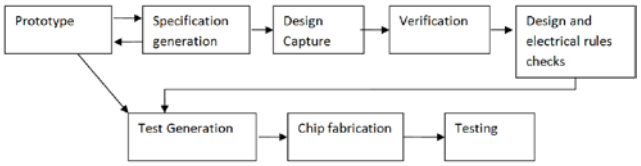

Fig. 1. Stages of chip manufacturing

Test generation can be moved next to specification generation and design capture phases. Test cases are generated replacing not-yet-existing chip with it's software prototype.[9]

To test a chip or it's software prototype, a set of valid input signals must be known. System boots up into unknown state. To start testing, system must be in a fixed state. Only then signal inputs can be sent and results evaluated. If expected and actual results match – the test passes. If not, there is a fault in fabricated chip. To switch system into a fixed state, a reset sequence must be found.

There are two main ways to reset a system into a fixed state: send a reset signal or signals; use a DFT (Design for testability) methods. DFT use additional input line/lines to send required logical value straight into memory elements. Therefore, all required memory elements can be set to a known and needed state. Reset sequence sends a system from any/random booted state into fixed state. Both methods have their advantages and disadvantages. DFT offers fast and convenient solution, but it has it's disadvantages – extra inputs and additional hardware are required for memory elements. By allowing to set the system into required state, DFT allows better chip coverage while searching for faults. Reset sequence does not require extra input lines, but the fixed state it transfers system to might not be enough. Some critical chip sections and paths might get left out and/or unreachable from this fixed state, and untested.

Test case generation for testing a software prototype and test pattern generation for a manufactured system-on-chip differs.

3-value logic (0, 1, x – unknown) used by test generation on gate level for hardware testing is a problem when testing software prototypes. Using this logic in software prototypes explodes the number of if statements required to emulate the chip's logic. This limitation increases the difficulty of the problem.

Test generation using internal chip's structure is not available at the design phase of the chip, as only software prototypes may exist. The chip itself is not manufactured yet, and the exact number of memory elements is not known yet. It will be the fabrication algorithm which decides on how to assemble available logical elements, so that they do the required calculations. [11]

If the chip's internal architecture is available, it is popular to analyze it and determine various factors influencing its results, like logic element placement priorities and such. By removing non essential logical elements or merging them into separate entities it is possible to reduce the chip size and the total state space. This reduction results in a smaller scope.

A reset sequence is needed to generate test cases for the chip emulating software prototype A set of input signals turn system's memory elements into known state, which is unknown at the time of chip's power-up. Therefore, without a known starting state a chip emulating software prototype may not be tested. It is possible that

system-on-chip fires-up into a known state, which eliminates the need of a reset sequence.

To test a software program, test cases are used. It's a set of variables and conditions which help determine if work of the system is correct or not. An oracle is a mechanism used to determine if a system passed or failed a test.

Preceding work

A large portion of the research work is devoted to synchronizing circuits using chip's known internal structure and the connections between logical elements. In this case, irrelevant elements and element groups may be removed, identical states identifies (and removed, thus reducing the overall chip size), relations between memory elements investigated and so on.

Chip's structure is unavailable when using software prototypes which emulate a system-on-chip. Therefore some of the preceding works create solutions that are not applicable whilst using prototypes.[11]

According to [1], a reset sequence can be difficult or impossible to find due to logical and conditional loops present in manufactured chip's structure. Structural self-loop forms when an input of a flip-flop or a trigger is determined by its output alone. In this case, trigger is removed from the main structure using a reset line. The logical self-loops are the structural loops that appear after part of triggers are already synchronized.[1] The authors suggest using partial reset (DFT technique), if a full reset is not found or not existing. Part of all triggers is selected for a forced reset. They are selected after the circuit is analyzed and logical or conditional loops found. According to [1] this technique allows to synchronize largest ISCAS'89 circuits.

A synchronization tree method may be used to generate all the resetting sequences of the shortest length [2]. This method checks if the circuit is synchronizable first. All input combinations are required to form such a tree, therefore this method is not practical for large chips.

Wehbeg and Saab use logical and functional synchronization in their work [3]. Partitioning of the chip is used, considering relationships between the memory elements. Logical method is tried first. If it fails – functional is used.

Reset sequences have their disadvantages; i.e. a set of states that can be reached using reset sequences may be limited [4]; critical states that must be tested may become unreachable. A reset sequence may not exist because of existing chip fault. In these cases, a DFT technique is needed to control the chip state and detect faults whose detection requires specific states.

M. Kein offers a method to quickly check if a given circuit is resetable or not [5]

An approach to finding reset sequences using ant colony optimizations is also presented [6]. It appears, that nature solved the problem of finding the shortest path from point A to point pretty well. This article explains how ants find shortest route to food supplies scattered around in vast territories. It appears that they use specific scent – pheromone, which is left along the trail by the ant scouts. Other ants follow this scent, which marks the shortest/best path found. The scent decays with time, and must be renewed. If a new better path is found, the path is altered. It is called pheromone updating. In this article, such approach is used to find shortest reset sequences in systems-on-chips. System-on-chip's internal architecture is used. Results are provided for two different algorithms [6] and [7].

Another algorithm is presented in [8] for partial reset of large circuits, as well as results for ISCAS'89 circuits. Chip's internal architecture is used.

A chip may not have a reset sequence. Various methods end up offering partial reset as a faster and simpler way to achieve results. A partial reset is a set of signals with some extra input lines to reset "hard to reset" or "impossible to reset" triggers [8].

Calculation scope and data sets reduction

The amount of calculation required for building a binary tree or full scan of large circuits grows exponentially as the number of triggers increase. 5 inputs and 3 triggers would result in a state space size of 8 and 32 inputs. 256 computation cycles are required to test state to state transitions 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 5,19855155862821476939959184714 19e+530 computation cycles for a full test.

In this article we present a basic algorithm based on software prototype emulating operation of a circuit. It is based on random generation of input signals and trigger states. After a reset candidate is found, it is validated using greatly increased state space. Generated sets are only a part of the whole set of possible inputs and states. Therefore, all results are based on heuristics or assumption. The primary reduction in generated sets removes the need for huge amounts of calculations mentioned above. If reset candidate passes validation, one can assume that a reset sequence is found. This may or may not be true, as only part of whole state space has been tried.

Circuit gate level model use 3-value logics (0,1, x-unknown), where software prototype can only use 0 and 1. This increases the number of checks required.

Tests have been made using ISCAS'89 circuits and experimental results are presented. Some circuits may be reset using 1 input signal or word, while others require more. Some circuits may not be fully reset using a reset signal. In this case, a partial reset may be an option [1].

Various ISCAS'89 circuit emulating prototypes were used for experiments. The main goal was to find a full reset sequence, which turns all prototypes' memory elements into fixed state, no matter what the starting values of these memory elements were before reset sequence was applied. Such a sequence was not found for all available prototypes. Some of them were only partially reset. For some circuits (namely s510) it was not possible to switch even a single memory element into a fixed state.

Time is needed for reset sequence algorithm to produce results. The bigger the circuit, more time is needed. An exception might be circuit no s35932, which

resets gracefully. This might be due to one of input pins being a reset line for the entire circuit.

Due to large amount of time required for larger circuits, an idea of set reduction was introduced. The idea is that if a reset signal is able to bring a system into a fixed state from a set of tens of starting states, it might do that with thousands of starting states. I.e. if there are 10 starting states and 1 reset input candidate which switch the system from all 10 starting states into a single fixed state, it might do the same for 100, 1000, 10'000 (and so on) starting states.

It is possible to greatly reduce the calculation scope by applying each input signal from a set of input signals to reduced set of states and analyzing results.

Once a reset candidate is found, it is re-validated against a large number of starting states. This allows reducing the amount of work required, as the number of calculations for state-input-state transitions is reduced.

Proposed algorithm

Full circuit test is a difficult task, as explained in previous section. Therefore, a reduced size approach was used in reset sequence finding algorithm. The main idea behind it is: if the algorithm finds a reset sequence for small set of trigger states, and validates the same reset signal against much larger set of trigger states, then we may say this is a resetting sequence for the circuit used. Once again – this method is based on assumptions. Not all possible input signals and triggers states were tested, so, there is always a possibility, that results may or may not be correct with sets that were not used in calculations.

Algorithm steps are as follows:

1. Algorithm generates a medium sized set of input patters to send into the software prototype emulating the circuit. More input patterns increase the chance to find best-possible reset candidate, but also increases number of calculations required and time required to produce results. In experiments, a set of 300 input patterns were used.

2. Small set of trigger states are generated. Bigger sets produce more reliable results and increase probability that validation will pass successfully. This also increases calculation time. Smaller sets result in faster completion, but provides more false-positive reset candidates. In experiments, a set of 20 states were used.

3. Each input signal is matched against each one of trigger states and the new state is calculated. This is the state into which system transfers to from a previous state after an input pattern is applied.

4. For each input pattern, resulting states are analyzed and number of fixed triggers calculated. If this number equals total number of triggers, then a reset candidate is found. If not, next input pattern is used for calculations.

5. If a better (initializes more triggers than other input patterns tried before) reset-candidate is found, it is saved as a best one.

6. A very large (compared to starting set) set of states are generated. In experiments, 50'000 states were used.

7. Best reset-candidate is used with this large set of trigger sets to calculate new system states. (Steps 3 and 4).

8. If this input pattern validates – an initializing pattern has been found.

```
      Begin;
generate_medium_sized_set_of_inputs
();
generate_small_sized_set_of_trigger_s
tates();
foreach(input_signal){
  foreach(trigger_state){
    calculate_circuit_output();
}
  is_found_reset()
{grab_reset_seq_candidate();}
    else  {  take_best_reseting_input();
increase_reset_sequence_lenght();     }
}
generate_vlarge_sized_set_of_trigger_
states();
foreach(trigger_state){
   calculate_circuit_output(using single
input – reset candidate);
}
if is_found_reset()
validation_successfull();

End;
```

Fig. 2. Proposed algorithm

Experimental results

This algorithm was tested using ISCAS'89 circuits.

Results show, that this algorithm (named PROTO) performs better or at least as-good-as 3 other algorithms provided by other researchers. This article's algorithm operates under increased difficulty, because internal structure of the chip is not known. Therefore, many techniques employed by other researchers are unavailable.

Partial reset algorithm only provides partial results (for those circuits that did not require partial reset and full reset sequence was found).

Result table 1 displays experimental results using proposed algorithm. First column contains names of ISCAS'89 test circuits, second number defines amounts of inputs, outputs and triggers. Both input and output numbers include number of triggers. Third column describes whether a full or partial reset sequence was found. If a full reset was found, a result string "fixed state" is used. This means, that it is possible to switch circuit from any random power-up state into a single fixed state using a set of resetting signals. If a full resetting signal is not found, number of fixed flip-flops / memory elements is displayed. This means, that experiment, using the algorithm provided could not find a full resetting sequence, but managed to set a part of triggers into fixed state. Results vary from 32% (s38417) to 99.7% (s38584) of triggers set to fixed state after a found partial reset sequence is used. Circuit s38584 has 1426 triggers, and only 3 were left unset. Fourth column provides a number of fixed triggers using PROTO (proposed software prototype emulating system-on-chip algorithm) software.

Fifth column displays the length of found reset sequence (full or partial). The search depth was limited to maximum set of 50 input signals. Some circuits may have a full reset sequence which is longer than 50 and show up only as partial resetting ones in this experiment. Other 5 columns are identical to fifth and sixth columns and provide results for algorithms used by other researchers (ACO-Init, GA-Init, Partial). A number next to each name refers to an article describing each of algorithms, a list of articles is provided in literature. The "-" signs used in results table mean that these circuits were not tested by the other articles authors.

Largest of circuits are very hungry for CPU-time to test and has plenty of triggers. Therefore, using increased input/states set sizes; depth of search may take longer, but provide better results.

If a full reset is required, but only partial resetting sequence is available, it is possible to reset some of the triggers using this set, and reset the non-fixed ones using DFT or direct input.

While experimenting, results were analyzed trying to discover how small a set of starting states can be to produce valid results. If a states set is too small, false reset candidates appear in most circuits, which do not validate in the validation step. If the set is increased too much, an excess of calculations is required, resulting in a larger time frame.

The table and graph below summarize the results. It is based on 22 tested ISCAS'89 circuits.

Table 1. Experimental results: reset sequence lengths and numbers of reset flip-flops using software prototypes

Circuit No	**No of Inputs/Outputs /Triggers**	**Compared algorithms**							
		PROTO			**ACO-Init[6]**		**GA-Init[7]**		**Partial[8]**
	In/Out/Triggers	**Full reset or max. no of memory elements set**	INITs	LEN	INITs	LEN	INITs	LEN	LEN
s1196	32/32/18	fixed state	18	1	18	1	18	1	
s1238	32/32/18	fixed state	18	1	-	-	-	-	
s13207	700/790/638	454 (71%)	454	18	314	30	207	20	
s1423	91/79/74	fixed state	74	2	74	3	74	3	
s1488	14/25/6	fixed state	6	1	6	1	6	1	
s1494	14/25/6	fixed state	6	1	-	-	-	-	
s15850	611/684/534	458 (86%)	458	18	306	16	308	18	
s208	19/10/8	fixed state	8	1	-	-	-	-	1
s27	7/4/3	fixed state	3	1	-	-	-	-	1
s298	17/20/14	fixed state	14	2	14	2	14	2	2
s344	24/26/15	fixed state	15	1	-	-	-	-	2
s35932	1763/2048/1728	fixed state	1728	1	1728	2	1728	1	1
s382	24/27/21	fixed state	21	1	21	1	21	1	
s38417	1664/1742/1636	578 (35%)	570	14	579	13	372	9	
s38584	1464/1730/1426	1423 (99,78%)	1423	37	1398	59	1398	36	
s386	13/13/6	fixed state	6	2	-	-	-	-	
s400	24/27/21	fixed state	21	1	21	1	21	1	1
s420	35/18/16	fixed state	16	1	-	-	-	-	
s444	24/27/21	fixed state	21	1	21	2	21	1	
s510	25/13/6	0 (0%)	0	-	0	0	0	0	
s526	24/27/21	fixed state	21	2	21	1	21	2	2
s5378	214/228/179	167 (93%)	167	9	179	16	179	16	14
s641	54/43/19	fixed state	19	1	19	1	19	1	
s713	54/42/19	fixed state	19	1	19	1	19	1	
s820	23/24/5	fixed state	5	1	-				
s832	23/24/5	fixed state	5	1	5	1	5	1	
s838	67/34/32	fixed state	32	1	-				
s9234	247/250/211	154 (73%)	154	4	152	10	53	2	
s953	45/52/29	25 (86%)	25	8	13	2	10	1	25

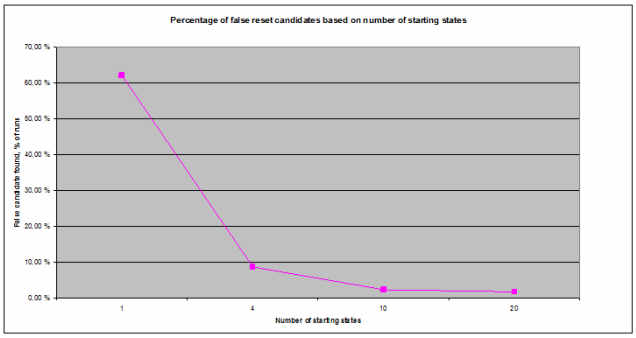

Fig. 3. Percentage of false reset candidates based on number of starting states

Results (Fig. 3) show (50 runs for each circuit), that a set of 20 starting states is enough for proposed reset sequence search algorithm. 20 out of 22 tested circuits (91%) returned 0 false reset candidates when using starting set of 20 states or more. 72% of circuits returned 0 false reset candidates when using starting set of 10 states or more.

Table 2. The decrease of found false positives based on increasing number of starting states

Starting number of states	False positives
1	*62,10 %*
4	*8,65 %*
10	*2,30 %*
20	*1,60 %*

There were two circuits affecting results at the time of research: s208 (started returning 0 false reset candidates once the size of starting states set reached 100) and s344 (kept returning false candidates no matter the starting states set size) . Largest of ISCAS'89 circuits were not included in this attempt due to calculation scope and time required (50 runs were used for each circuit).

These results do not guarantee, that such method of state amount reduction works with any and all circuits, but it worked pretty well with the ones used for testing. Therefore, this approach might be considered if finding a reset sequence is required.

Conclusions

I presented an algorithm for finding shortest length reset sequences using circuit emulating software prototypes. Experimental results show, that this algorithm performs better or as-good-as compared to other methods on ISCAS'89 test circuits. This algorithm operates under conditions of increased difficulty, as the internal structure is not known and can not be examined and used to reduce the problem scope.

Increasing sizes of data sets might have provided better experimental results, but it increases time required for each run, which might be limited.

A reset sequence may not always be found. In such cases, a different approach should be used, or a partial reset solution accepted.

In some cases, a reset candidate, which was found using an algorithm suggested, may not validate. It means, that the solution found is checked against a large number of starting states, and in some cases, the candidate may not be a resetting one. This often happens if a reset candidate sets most of the triggers to a fixed state, but a few triggers remain unstable when testing with an increased set. For a small set, these triggers may appear to be fully set.

This does not necessarily mean that the non-validating reset candidate is bad and should not be used. In my experiment, such non-validating candidates are good reset sequences once the number of fixed triggers is slightly reduced. I.e. an algorithm finds a reset candidate using 20 starting states for circuit s13207 with a number of fixed triggers 460 out of 638 total. Validation process fails, stating, that this candidate was unable to switch all 460 triggers into a fixed state. Instead, only 454 triggers were fixed during the validation (which would pass).

Therefore, 454 set triggers is still quite a good solution. In this case there was a 1,5% drop from the entire "before-validation" solution, and such reduction may still be acceptable.

As for the rest of triggers, partial reset methods might be used to achieve even better resetability.References

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 Clock Fault Models // Information Technology And Control. – Kaunas, Technologija, 2008. – No. 1(37). – P. 12–18
10. **Bareiša E., Jusas V., Motiejūnas K., Šeinauskas R.** The Use of a Software Prototype for Verification Test Generation // Information Technology And Control. – Kaunas, Technologija, 2008. – No. 4 (37). – P. 265–274.
11. **Bareiša E., Jusas V., Motiejūnas K., Šeinauskas R.** On the Enrichment of Functional Delay Fault Tests // Information Technology And Control. – Kaunas, Technologija, 2009. – No. 3(38). – P. 208 – 216.

Received 2010 03 29

K. Morkūnas, R. Šeinauskas. Circuit Reset Sequences based on Software Prototypes // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 7(103). – P. 71–76.

In this article, sequential circuit reset and initialization problem is presented. A method and algorithm is proposed for finding shortest length reset sequences using circuit emulating software prototypes. Using a software prototype gives the benefit and possibility of early test case generation. A reset sequence is able to switch circuit to a know state, regardless of the initial state. In this work, finding a reset sequence consists of using a software prototype which emulates an actual circuit. The proposed method and algorithm use randomly generated sets of circuit states and input signals, finding the best reset candidate and the validation of solution. ISCAS'89 benchmark sequential circuits were used for experiments. The results are provided within the article. It shows, that this method can achieve better, or at least "as good as" results compared to other algorithms, even though this method operates under more difficult conditions. Ill. 3, bibl. 11, tabl. 2 (in English; abstracts in English, Russian and Lithuanian).

К. Моркунас, Р. Шейнаускас. Вычисление установочной последовательности на основе программного прототипа // Электроника и электротехника. – Каунас: Технология, 2010. – № 7(103). – С. 71–76.

Проблема тестирования является критической проблемой всего процесса проектирования цифровых устройств, которая увеличивает время попадания устройств на рынок. С целью уменьшения сложности задачи генерирования тестов надо начать их проектирование на функциональном уровне. Возникает задача вычисления установочной последовательности на основе программного прототипа. Предложенный метод и алгоритм использует вероятностное генерирование входных сигналов, выбор лучших кандидатов и проверку решения. Результаты экспериментального исследования прилагаются. Ил. 3, библ. 11, табл. 2 (на английском языке; рефераты на английском, русском и литовском яз.).

K. Morkūnas, R. Šeinauskas. Schemų nustatymo sekų aptikimas naudojant programinės įrangos prototipus // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2010. – Nr. 7(103). – P. 71–76.

Straipsnyje apžvelgta mikroschemų nustatymo ir inicializacijos problema. Siūlomas metodas ir algoritmas trumpiausiai nustatymo sekai rasti naudojant schemas imituojančius programinės įrangos prototipus. Naudojant šiuos prototipus galima anksčiau pradėti ruoštis testavimui schemų gamybos procese. Nustatančioji seka pakeičia schemos vidinių atminties elementų būseną į žinomą būseną nepriklausomai nuo prieš tai buvusios. Šiame darbe paieškos algoritmas naudoja schemos veikimą imituojantį programinės įrangos prototipą. Siūlomas metodas ir algoritmas naudoja atsitiktinai sugeneruotas pradinių būsenų ir įėjimo signalų aibes, randa geriausią ir patikrina sprendimą. ISCAS'89 schemos buvo naudojamos eksperimentams. Rezultatai pateikiami straipsnio pabaigoje. Šiuo metodu galima gauti geresnius, palyginti su kitais algoritmais, arba tokio pat lygio rezultatus, nors šis algoritmas veikia sunkesnėmis sąlygomis. Il. 3, bibl. 11, lent. 2 (anglų kalba; santraukos anglų, rusų ir lietuvių k.).