HiLL PUBLISHING

Advances in Computer and Communications, 2023, 4(1), 9-20
https://www.hillpublisher.com/journals/acc/
ISSN Online: 2767-2875

Counter Based Hardware Resource Reduction for FIMA System Verilog Assertion Checker

Maoyu Mao

Guangdong University of Science & Technology, Dongguan, Guangdong, China.

How to cite this paper: Maoyu Mao. (2023) Counter Based Hardware Resource Reduction for FIMA System Verilog Assertion Checker. *Advances in Computer and Communication*, 4(1), 9-20.
DOI: 10.26855/acc.2023.02.002

Received: January 8, 2023
Accepted: February 6, 2023
Published: March 3, 2023

[*]**Corresponding author:** Maoyu Mao, Guangdong University of Science & Technology, Dongguan, Guangdong, China.

Abstract

Nowadays, checker synthesis for assertion based verification becomes popular because of the recent progress on the FPGA prototyping environment. Several works have been proposed to synthesize assertion checkers on FPGA based emulation and FIMA-based (Finite Input Memory Automaton based) method is one of such works. FIMA-based method uses a finite input-memory automaton using finite input queue (shift-register chain) to transform SVAs to hardware checkers for FPGA prototyping. FIMA-based method keeps one queue for each input, so it is effective to share the queue on several assertions. However, if an assertion includes a long sequence of some input, then the queue becomes long and a lot of hardware resources are necessary. In the research, a method to cope with assertions including such long sequences has been devised, and counter based method is proposed. A binary counter can represent the length N sequence with log(N) bits and the number of registers can be reduced a lot for long sequences. The proposed method can also reduce the power consumption of the sift-registers. Registers in counter module and registers for each variables can be recycled, thus the sharing within time window and sharing between assertions can be achieved. By using embedded RAM modules, we can further save logic element. The counter based method did reduce the hardware resource for FIMA-based method, and it works extremely well for assertions with long sequence of input and less variables.

Keywords

Assertion, SystemVerilog, Hardware Source Reduction

1. Introduction

In hardware design process, functional verification is important and time consuming. Many works have been done on functional verification not only using the simulation but also using the formal verification methods.

In recent large and complex hardware designs, formal methods and the simulation based methods are both important.

Assertion-based verification (ABV) is one of such combinations where properties described by designers have been checked at simulation/emulation.

The assertions are usually written in RTL description to specify the relation between signals, and are converted to checker circuits when applying emulation as shown in Figure 1. Design under verification corresponds the current design, and Assertion checker is a hardware module to check the assertions. In the simulation, the assertions can be checked directly by the simulator, but the performance can be improved in some case by using a hardware checker. Note that the flexibility to check and notifying the violation of assertions becomes lower by using a hard-

ware checker.

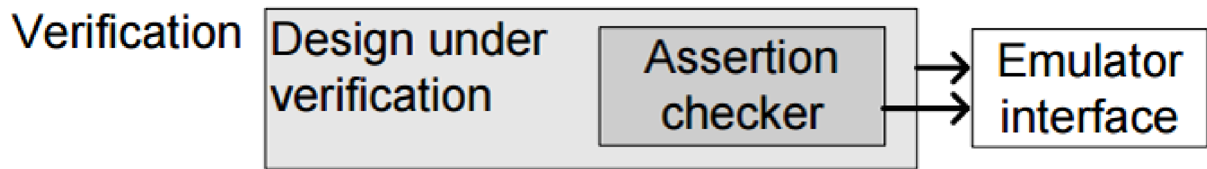


Figure 1. Usage scenarios for hardware assertion checkers.

Although we can find most bugs at RTL level, some bugs still should be checked at gate level. It is not feasible to apply a lot of test vectors at gate level simulation, so checker circuit generation for FPGA prototyping is a challenging problem.

On the SystemVerilog checker synthesis, Finite Input Memory Automaton (FIMA) method has been proposed [1]. FIMA is a finite automaton whose state is finite sequences of input variables and is suitable to represent the sequence in SystemVerilog assertions. The original FIMA method uses a queue for each input variable but the method is inefficient for long sequences.

In this thesis, a counter based synthesis method based on FIMA method is proposed to transform SystemVerilog Assertions to hardware checkers. To reduce the hardware resource, we introduce several ways: (1) compress input string using counters; (2) simplify time window; (3) sharing hardware resource of input memory automata and (4) constructing input memory automata by using embedded RAM modules.

2. System Verilog Assertion and Previous Works

2.1 System Verilog Assertion

SystemVerilog Assertion (SVA) is a description method of properties of a hardware design. A property can be described using a time sequence of input, internal, and output signals. A sequence can include Boolean expressions.

System Verilog Assertions are briefly introduced based on [2] and [3].

A sequence can be defined as follows: at first, a signal is a sequence. Next sequences concatenated with "##N" or "##[M:N] are sequences, where N and M are integer values. "##N" represents N clock delay from the end of the former sequence. "##[M:N]" describes a time window between M clock delay to N clock delay, and the latter sequence will happen during the time window.

For example:

req ##N gnt;

specifies a sequence that if req is true on the current clock tick, and the signal of gnt shall be true on the Nth clock tick from now.

Sequences can be connected by some operators such as "and," "or," "intersect," "within" and "throughout," as well as repetition operators such as "[∗]," "[=]" and "[->]."

Property of hardware design can be described, the relation between sequences and the detailed description of hardware properties is found in [2].

2.2 Previous Works

2.2.1 Formal Checker

Formal Checkers (FoCs) transforms PSL/Sugar [4] properties into hardware checkers.

2.2.2 MBAC Method

MBAC [5], [6] is another tool which generates assertion checkers from PSL/Sugar properties using automata based method.

2.2.3 Finite Input Memory Automation Based Synthesis Method

The finite input-memory automaton based synthesis method has been proposed as a method to transform System Verilog Assertions to hardware checkers for FPGA prototyping.

The basic idea of FIMA method is using shift register chain for detecting a sequence. Register chain keeps finite sequences of necessary input variables and a sequence detection circuits is constructed on such time sequences for matching the time difference between signals in an assertion. A structure of a detection circuit using FIMA is

shown in Figure 2-1.

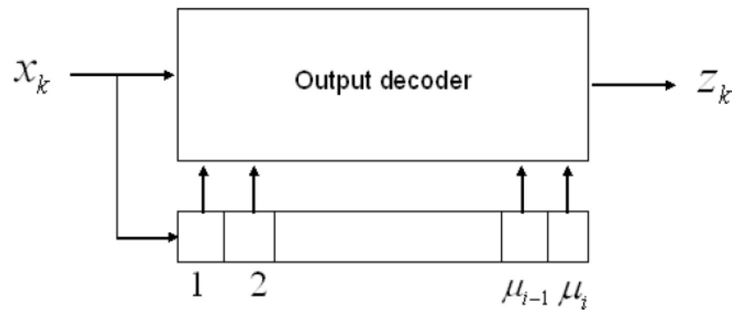


Figure 2-1. A finite input-memory machine.

In a sequence checker, one queue (shift-register) is dedicated for one variable, and (N–1) queues are necessary to construct a checker for a sequence with N variables.

Figure 2-2 is an example of a linear sequence with 4 variables. There are 3 queues are included.

Sequence s1: a##1b##2c##1d

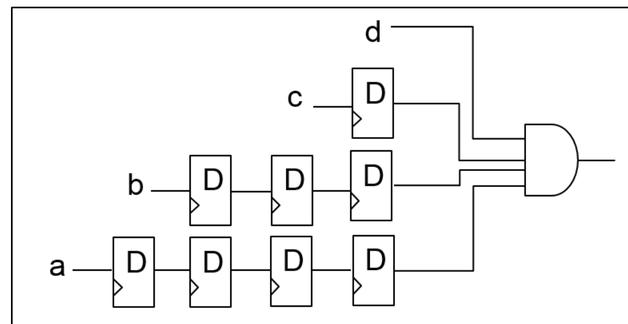


Figure 2-2. Synthesized sequence s1.

In this method, one shift-register chain is used for each variable, so 3 shift-register chains are needed and the depths of them are decided by the amount of delay of each variable (delay means the delay between signal and output in here). For example, signal ‘a’ has delay 4 with respect to the current signal “d” so the depth of the chain is 4

For the SVA with implication operator, the left hand side (LHS) of the implication is called “antecedent” and the right hand side (RHS) is called “consequent.” In each clock, if the antecedent succeeds, then the consequent is evaluated. If the evaluation result succeeds, then the property holds and the property fails otherwise. If the antecedent does not succeed, the property is assumed to be “vacuous success.”

Related to timing, there are two kinds of implications, overlapped implication and non-overlapped implication. The followings are examples of the description methods.

1) Overlapped implication

```
propertyPoverlapped;
  @(posedgeclk) S1 |-> S2;
endproperty
```

If sequence S1 evaluates to true on a given positive clock edge, then the sequence S2 should match on the same positive clock edge.

2) Non-overlapped implication

```
propertyPnonoverlapped;
  @(posedgeclk) S1 |=> S2;
endproperty
```

If sequence S1 evaluates to true on a given positive clock edge, then the sequence S2 should match on the next positive clock edge.

Figure 2-3 shows a checker for the following property with non-overlapping implication.

```
property implication p1;
```

@(posedgeclk) a ##1 b ##1 c |=> d ##1 e;
endproperty

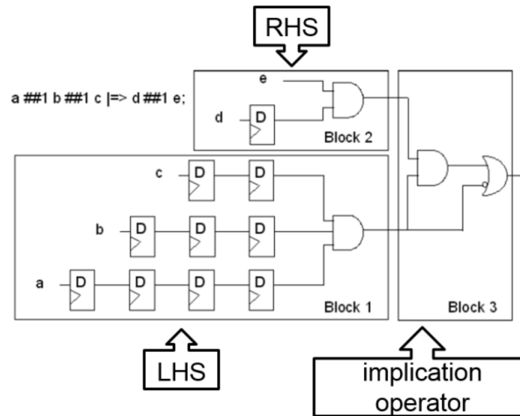


Figure 2-3. Implication example.

For the property implication p1, start from a given clock point, the antecedent sequence can have only one successful match. The property implication p1 evaluates to true only if ‘e’ becomes true in clock N, ‘d’ is true in clock N-1, ‘c’ is true in clock N-2, ‘b’ is true in clock N-3, and ‘a’ is true in clock N-4. The implication property can be transformed into three parts in circuit, LHS, RHS and implication operator. We transform the concatenation sequences of LHS ##1 RHS. These three parts correspond to block 1, block 2 and block 3 in Figure 2-3.

There is another way. At first LHS are considered and construct a checker for the sequence. After that, the manipulation of |=> is considered.

3. Resource Reduction Using Counter

Based on the FIMA method, a counter-based hardware resource reduction method to transform SVAs to hardware checkers has been proposed. In this chapter, an example sequence:

a1: asserusedt property (@(posedgeclk) a |=> b[*2:4] ##1 c);

is used to show the basic algorithm to transform SVAs to hardware checkers.

3.1 Flow of Synthesis Algorithm

To transfer an assertion sequence into hardware checker, we proposed an algorithm which is made up of three main parts: counter module, compare module and output module. We will introduce them part by part. To making it easier to be understood, we keep using the sequence

a1: assert property (@(posedgeclk) a |=> b[*2:4] ##1 c) as example, and the flow path is shown as Figure 3-1.

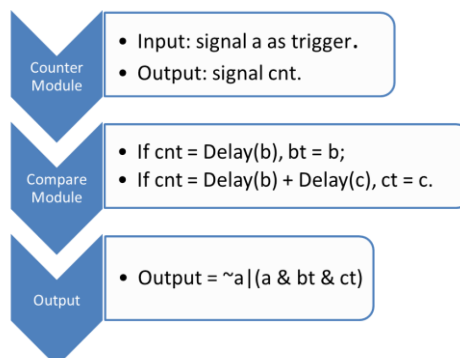


Figure 3-1. Flow path for synthesize sequence a1.

(1) Counter module

A counter is used to reduce the hardware resource especially Flip-flops in a shift register chain. At first, a trigger signal in order to start the counter is discussed. The most suitable signal for triggering is the first signal in that se-

quence. As an example, in sequence a1, the signal "a" is used as a trigger signal for a counter. After receiving the trigger signal, the counter starts to counting. Let the output of counter be "cnt". The bit width for the counter in this module is decided by the largest delay in input sequence. In the example sequence, the bit width for counter is 3 bits because the largest delay in this sequence is 5 (). In the counter module, input is enable signal and the output is cnt signal, and the cnt signal can be used as input signal for compare module as shown in Figure 3-2.

(2) Compare module

In the compare module, we firstly set registers for each signal but the first trigger signal in input sequence. In the example sequence a1, let registers bt and ct be for a signal b and a signal c (there is no register for signal a because we can use the value of signal a directly in the output module).

After received the output of the counter module, cnt signal and the delay signal are compared in the compare module, then the module judges whether assign signal b and c to register bt and ct or not.

For example, in the sequence s1:

a ##1 b ##2 c ##1 d; the delay of {a,b,c,d} is {0,1,2,1};

in sequence a1, the delay of {a,b,c} is {0,1,4}, that is because we use the largest delay in a time window of a signal as delay for that signal.

As the value of signal cnt is increasing because of the character of counter, we should firstly compare it with the delay of the second signal, which in sequence a1 is signal b, if cnt = delay (b), bt = b; after these, the value in register bt is settled. Then we compare the signal cnt with the sum of delay (second signal) and delay (third signal), if these two values are equal, we assign the value of the third signal to the register corresponded. In sequence a1, if cnt = delay (b) + delay (c), ct = c. The rest can be done in the same manner as shown in Figure 3-2.

After the sum of delay for all signals has been compared with the output of counter, no registers stores empty value, then we can move to the next part, which is the output part.

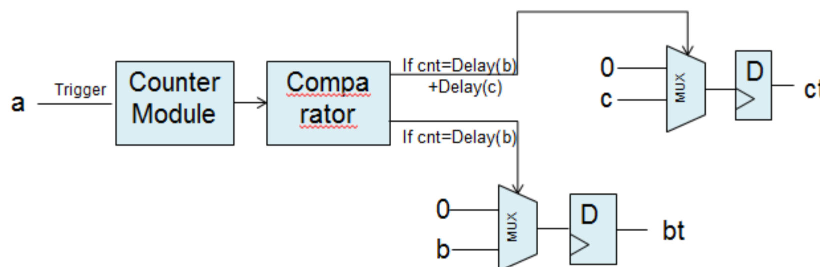


Figure 3-2. Counter module and compare module.

(3) Output module

From the definition of System Verilog assertion, an assertion is true only if all events in it occur, or become true, according to the delay of each signals.

So the output for linear sequence is simple, it equals to the logic and result for all signal registers. For example, the output for sequence s1 is (a &bt&ct&dt).

For the sequence with implication operator, it's more complex to get the final output. The definition of implication operator claims that only if the left hand side part evaluate to true, we can continue to consider the value of right hand side, otherwise, the output is said to be vacuous success. In the example sequence a1, only if 'a' evaluate to true, then we consider signal 'b' and 'c'. So the output of a1 is ~a | (a &bt&ct), shown as Figure3-3 below. Finally we get output for sequence a1 at clock [delay(b) + delay(c) + 1].

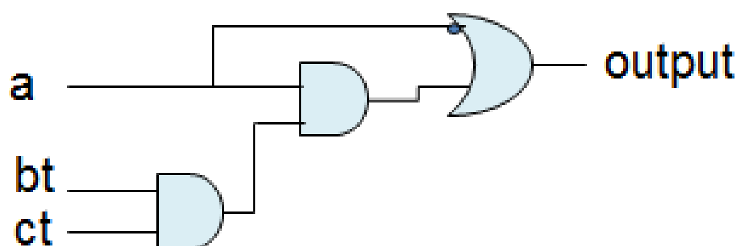


Figure 3-3. Output module.

3.2 Synthesis Result for Sequence a1

In this part, we synthesized sequence a1 based on the input string compressing. The result shows that there are four main blocks in this structure: counter block and three assertion operand blocks. Three assertion blocks correspond to three operands in time window, which will be introduced in the next chapter. Each assertion blocks has four inputs: a, b, c, cnt and one output. The Figure 3-4 below shows the structure of sequence a1.

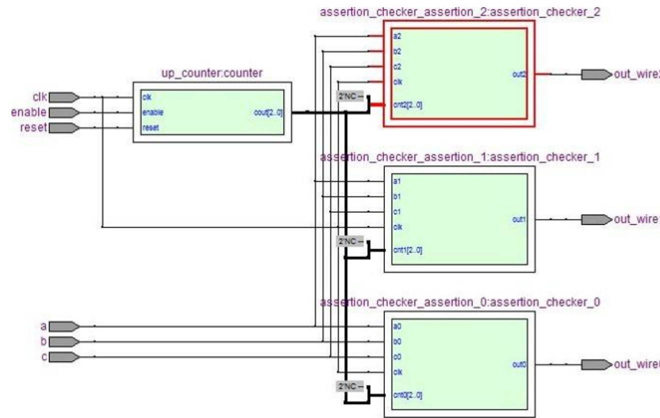


Figure 3-4. Synthesis result for a1 with input string compressing.

Then we consider the counter module, from Figure 3-5, we can know there are three registers for this 3-bits counter.

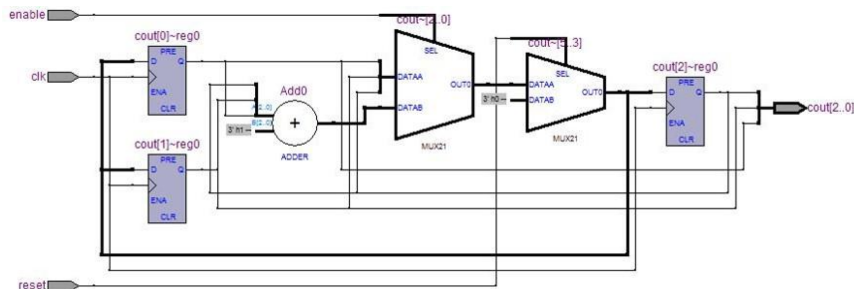


Figure 3-5. RTL Structure of Counter.

Finally, we focus in one assertion module, Figure 3-6 below shows the inner RTL structure of assertion block0. From Figure 3-6 we can know that there are two FFs for signal b and c in one assertion block, so there are in total 9 FFs in this structure, 3 for counter block and 2 for each operand assertion block.

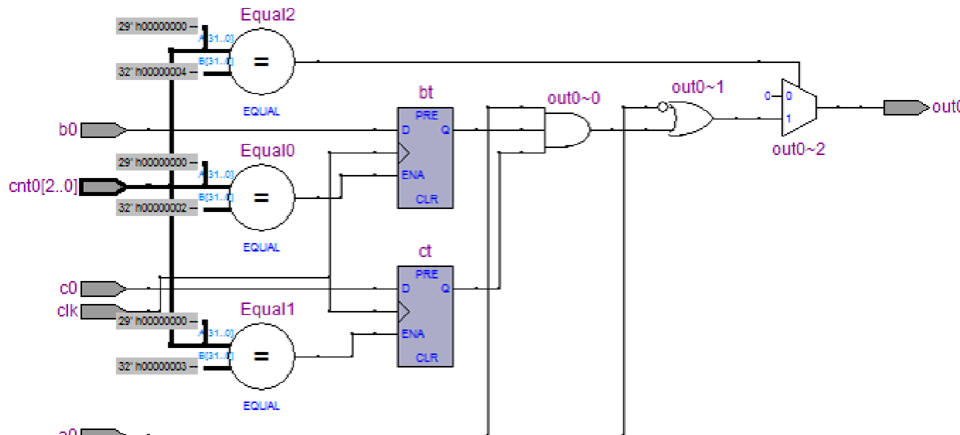


Figure 3-6. RTL Structure of Assertion_checker_0.

Compared with the FIMA method, the number of FFs is the same. It seems that there is no progress by using this counter based method, but we can further reduce the hardware resource by steps two and three. In the next chapter, we will introduce the algorithm to simplify the time window, which will extremely reduce the number of FFs used.

4. Time Window Simplification

There are three parts in Section 3 to introduce our proposal of tome window simplification method. At first, what is time window and why time window simplification is significant will be introduced in section 3.1. Secondly, we give time window simplification algorithm in section 3.2. After that, we will show the result for example sequence a1 after time window simplification in section 3.3.

4.1 Time window in System Verilog Assertion

The time window of a sequence can be considered as logic 'OR' of operand sequence threads in parallel, as shown in Figure 4-1. In general, the time window can be decomposed into several subsequences.

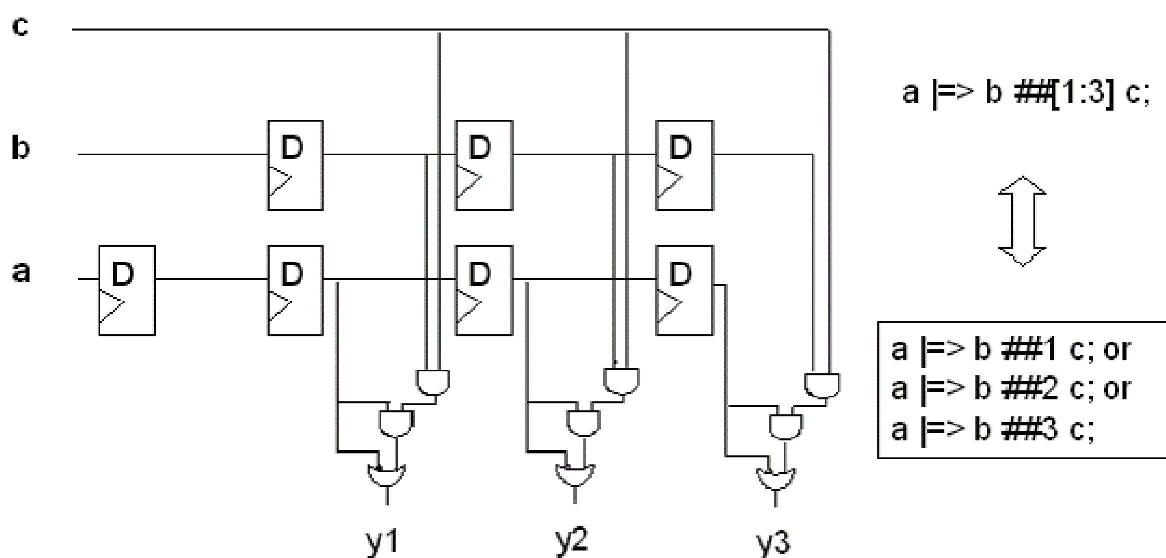


Figure 4-1. Implication example with time window.

Why time window simplification is significant? Obviously, it is because time window allows several possibilities to make an assertion be true. However, it will be too resource consuming to synthesis each parallel operands separately.

By time window simplification, we can extremely reduce the number of FFs or memory resource.

4.2 Time Window Simplification Algorithm

The basic idea to simplify time window in an assertion is to share memory resource between parallel operand blocks. For each operand, result can be obtained at different time clocks because of the time window.

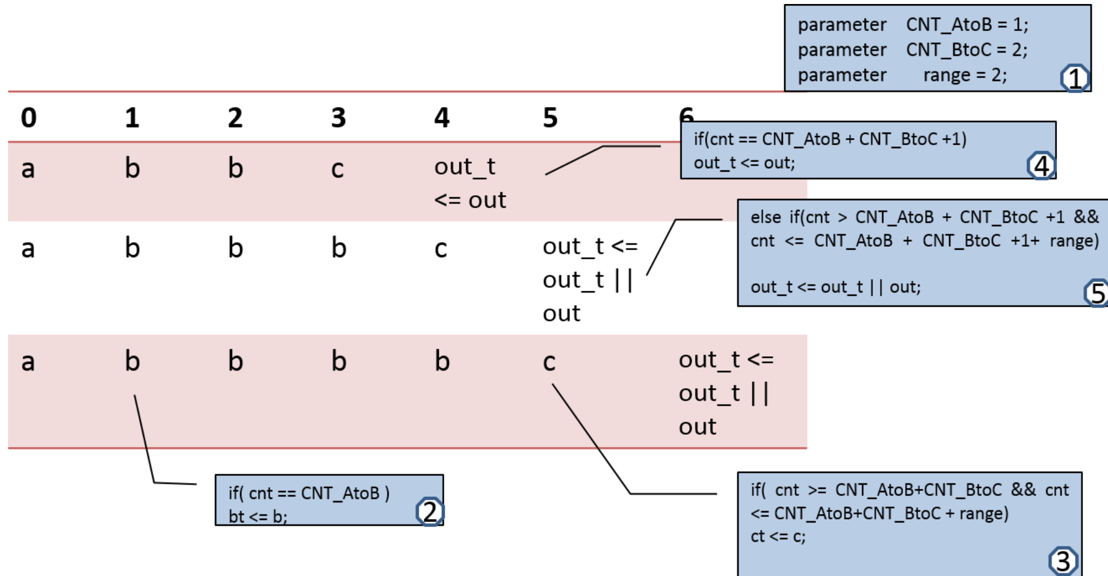
A temporary register out_t is set to save the result for operands and some parameters also should be defined. We use sequence a1 as example to show how to simplify time window.

a1: assert property (@(posedgeclk) a |=> b[*2:4] ##1 c);

Shows in Table 1. The first row in this table means time clock, strat from 0; the second to fourth rows are three operand decomposed from time window [2:4], we always get signal a at clock '0', and signal b at clock '1'; for signal c, it can be got at different time clock in different operands; and the same with signal c, the output for these three operands are determined at different time clock. The basic idea to simplify the time window is to share memory resource between different operands, in other word, for example sequence a1, we define a register ct for signal c, and all these three operand use this register ct to store the value of signal c. That is possible because we need signal c at different time clock through these three operands. Same argument, we define a temporary register out_t to store the output result at different clock. What we have to say is that this so called register out_t can be

achieved by wire in the coding work, so there is no extra memory resource used.

Table 1. Time delay for each situation in time window



0	1	2	3	4	5	6
a	b	b	c	out_t <= out		
a	b	b	b	c	out_t <= out_t || out	
a	b	b	b	b	c	out_t <= out_t || out

There are in total five important steps in this algorithm, we will introduce them one by one.

First, define three parameters for assertion a1: CNT_AtoB, CNT_BtoC and range. CNT_AtoB equals 1, means the delay between signal a and b; CNT_BtoC equals 2, means the shortest delay between signal b and c; range equals 2, means the range in time window, which in example sequence a1 is [2:4].

For second step, we compare the output cnt with CNT_AtoB, if cnt equals to CNT_AtoB. That means at clock 1, signal b is inputted and we should store it's value in register bt.

Thirdly, we compare the output cnt with CNT_BtoC, from Table 1 we can see that at clock 3, 4 and 5 signal c should be identified. To shorter the coding sequence, we use a clock range to describe CNT_BtoC: if the value of cnt is bigger than (CNT_AtoB + CNT_BtoC) and at the same time smaller than (CNT_AtoB + CNT_BtoC + range), then we should assign signal c to register ct.

The fourth step computes the output for the first, shortest operand. If cnt equals to (CNT_AtoB + CNT_BtoC + 1), then assign out into temporary register out_t.

The last step is repeating work, which computes the output result for the rest operands and do the logic or with output_t, then save the result in output_t at different time clock. If the value of cnt is bigger than (CNT_AtoB + CNT_BtoC + 1) and at the same time smaller than (CNT_AtoB + CNT_BtoC + range + 1), assign out_t<= out_t || out.

After these five steps, we get the final result with using only about one third number of memory resources. That is memory resource friendly.

4.3 Synthesis Result for Sequence a1

After time window simplification, we get the synthesis result for example sequence a1. As shown in Figure 4-2.

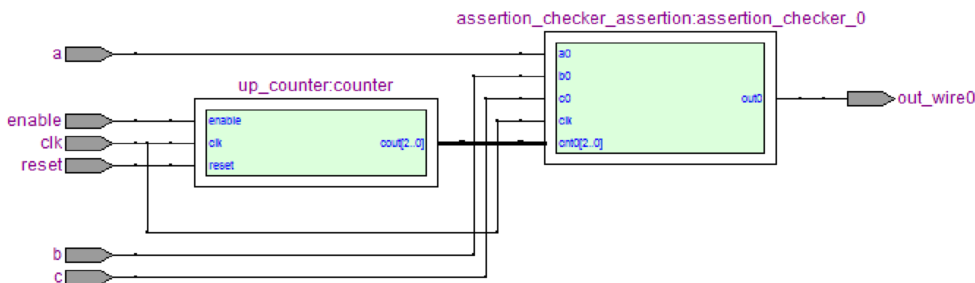


Figure 4-2. RTL construction after time_window simplification.

We can see from the Figure 4-2 that three operand assertion checker blocks have been combined together as one module. Whether it saved hardware resource? We can get the answer in Figure 4-3.

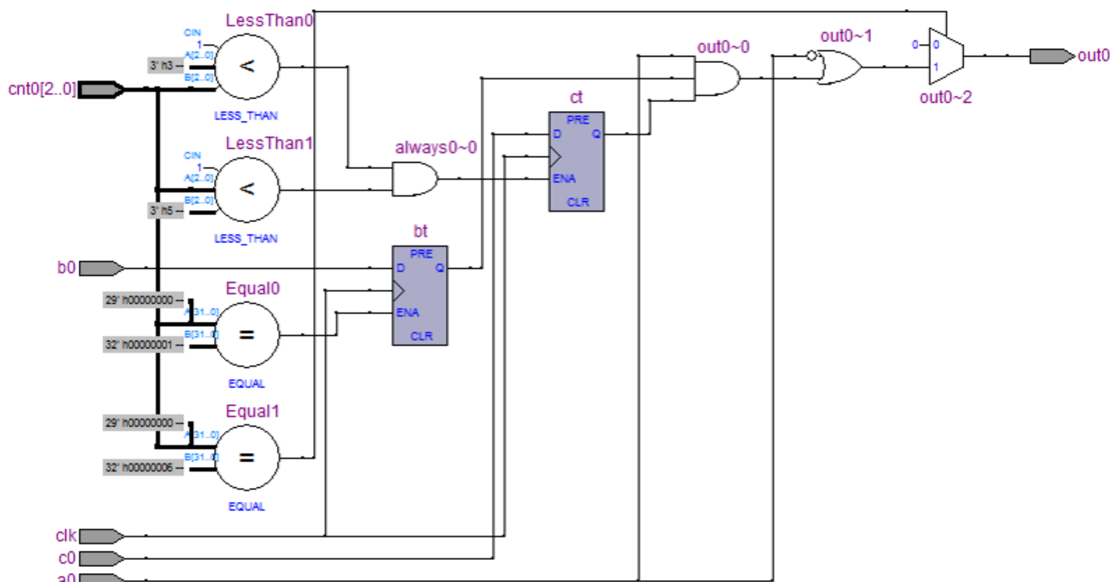


Figure 4-3. RTL construction for assertion checker.

There are only two registers in this structure, one for signal b and another for signal c.

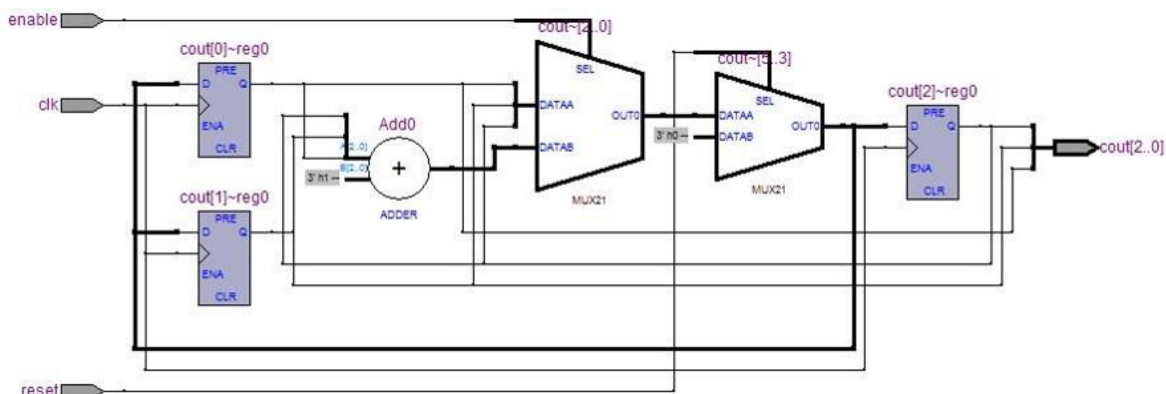


Figure 4-4. RTL Structure of Counter.

From Figure 4-4, we can see that the counter module is the same as before, so there are three FFs in this structure.

The result shows that after time window simplification, the memory resource used in assertion checker module decreased from 6 FFS to 2 FFs, and the total memory resource decreased from 9 FFs to 5 FFs. So after this work, we do saved the memory resource used in synthesis System Verilog assertion checker.

5. Register Sharing between Assertions

The Counter based method takes counter and clock delay between signals to recognize boolean expressions in sequences. For mapping hardware design on FPGA board, adding assertions takes resources of wine interconnection. To reduce the resource usage of FPGA prototyping much more, the sharing of registers between assertions is indispensible.

In section 4.1, the way to divide memory resource into two parts is introduced, and the basic idea of sharing between assertions is shown.

In section 4.2, we use some examples to explain how to share register between assertions.

5.1 Basic Idea of Sharing Memory Resource Between Assertions

The Counter based synthesis method needs counter and clock delay to trace the previous values of signals.

There are two major parts of register resource, one part for counter module and another for variables.

We can separately share these two parts between assertions:

For counter module: use the largest bit number above all assertions as the bit wide for counter. If the largest delay for assertion set is n, then the bit number for counter module is ||.

For variables: provide a register for each variable that exist, at most one for a variable even if it appears at several assertions. That means if two example assertions assertion 1 and assertion 2 both have signal b, we only provide one register for variable b.

5.2 How to Share Register between Assertions.

To make it clear to understand, we use too examples to show:

assertion 1: assert property(@(posedgeclk) a ##1 b ##1 c);

assertion 2: assert property(@(posedgeclk) a[*26]##1c);

Assertion 1 is a positive clock edge active sequence with three variables a, b and c. The longest delay in assertion 1 is 2 clocks. So the counter used in this design is 2-bit counter (< 3 <) and two signals b and c need exclusive register.

Assertion 2 is a positive clock edge active sequence with two variables a and c. The longest delay in assertion 1 is 26 clocks. So the counter used in this design is 5-bit counter (< 27 <) and only one signals c needs exclusive register.

Without resource sharing:

It takes 4 FFs to prototyping assertion 1 (2 for 2-bits counter, 2 for variable b and c).

It takes 6 FFs to prototyping assertion 2 (5 for 5-bits counter, 1 for variable c). So in total 10 FFs.

After register sharing:

It takes only 7 FFs to prototyping these two assertions (5 for 5-bits counter, 2 for variable b and c).

For memory resource sharing between assertions, we always choose the widest bit- wise and the most number of variables, so we can always reduce the memory resource used prototyping assertion set.

6. Experimental Result

To evaluate the Counter-based method, we compare the synthesis results generated by four methods: Counter-based method, FIMA based method, MBAC method and the tool of FoCs.

6.1 Experimental Environment and Benchmarks

The synthesis results using Counter based method are compared with those using FIMA based method, MBAC method and the tool of FoCs. The same assertions are described.

Table 2. Benchmark assertions

```
a1: assert property (@(posedgeclk) a ##1 b##1 c | => d ##1 e);
a2: assert property (@(posedgeclk) a | => d[*1:3] ##1 e);
a3: assert property (@(posedgeclk) a ##1 b | => d[*2:4] ##1 e ##1 c);
a4: assert property (@(posedgeclk) a | => b[*0:2] ##1 c);
a5: assert property (@(posedgeclk) a | => b[*25] ##1 c);
```

Table 3. Practical assertion sets

Practical assertion sets (AS)	Description
AS1: Arbiter, including 4 assertions, Chart 2 [14]	Used to verify the round-robin algorithm of an arbiter.
AS2: SDRAM controller, including 12 assertions, Chart 7 [15]	An ABV IP to verify the transactions between SDRAM and processor.
AS3: SDRAM, including 15 assertions , Chart 5 [14]	Used to verify the control signal of SDRAM and memory controller.

The benchmark assertions are shown in Table 2. Benchmark a1-a5 are simple expressions and all signals are boolean variables. Other three sets of assertion benchmarks are practical assertions for assertion based verification and selected from textbooks [6], which are shown in Table 3.

Altera EP1S80F1508C6 FPGA is used to map the assertion checkers. The VerilogHDL assertion checkers are synthesized by Quartus II 9.0 [7].

6.2 Experimental Result

Table 4 shows the result. We check the proposed Counter based method by two ways: logic element (LE) based checker and embedded RAM module based checker. Compared with FIMA-based method, counter-based method is more memory resource friendly, but the number of LUT becomes larger. So there is a tradeoff between the number of FFs and number of LUTs. Compared with MBAC method, both FIMA method and counter-based method behave worse, although they both behave better than FoCs method. That is because both FIMA method and counter-based method can further share resource between assertions, so it is much efficient to use FIMA method and counter-based method in assertion set, which is practical in real application. Table 5 shows the result after shared resource between assertions, we can compare FIMA method and counter-based method through Table 5.

Table 4. Comparisons of different methods

Asser.	MBAC		FoCs		FIMA			Counter Based		
	FF	LUT	FF	LUT	FF	LUT	RAM	FF	LUT	RAM
a1	4	4	37	47	10	2	28	7	13	24
a2	6	7	37	48	7	3	32	5	10	24
a3	12	14	41	56	19	3	54	8	15	64
a4	4	4	36	47	5	3	30	5	10	24
a5	48	41	59	51	51	9	136	7	11	160

For one assertion, the efficiency of prototyping is related to the number of signals, as well as the length of longest delay. Counter-based method works extremely well in assertion which has few variables and large delay, for example assertion a5, it has only three variables and the largest delay is 26, so compared with FIMA method, counter-based method is much better in this case. For assertion a1, the longest delay is only 5 clocks and there are five variables in this assertion, from the result we can see, the payment to reduce the number of FFs is heavy, because the number of LUT increased more than the decreasing FFs.

Table 5. Resource sharing application

Asser	MBAC		FoCs		FIMA			Cnt Based		
	FF	LUT	FF	LUT	FF	LUT	RAM	FF	LUT	RAM
a1~a5	74	70	210	249	59	20	280	9	59	160
Arbiter	6	16	68	90	12	13	134	4	18	24
Sdram Control	19	40	612	869	15	35	236	4	113	8
Sdram	51	112	241	366	57	65	1188	12	76	384

Table 5 shows the result after sharing, we can compare FIMA method and counter-based method through Table 5

The number of FFs of both FIMA and Counter-based (LE) becomes smaller after applying register sharing technology. For example, in the sets of "a1–a5," the number of FFs for FIMA (LE) decreased from 92 to 59, the number of FFs for Counter-based (LE) decreased from 32 to 9, and both smaller than that of MBAC.

Through Table 5 we can see that no matter assertion set a1~a5, Arbiter set or Sdram and Sdram Control set, the number of FFs is smaller than FIMA method by using counter-based method.

To reduce the logic elements usage in FPGA, the result of RAM based counter implementation is also shown.

For most of AS, after sharing between assertions, the total number of hardware resource is reduced compared with FIMA-based method and both are better than MBAC method. For Sdram-controller AS, counter based method

behave not as well as expected, that is because in Sdram-controller AS, the number of assertions are huge, and there is no large delay between signals in each assertion.

7. Conclusion

In this thesis, a counter-based hardware resource reduction method based on FIMA System Verilog assertion checker is proposed. We proposed three steps to realize a memory resource reduce algorithm.

Firstly we compress input string by using counter to count the delay between each signal in assertion. We analyze the result after synthesis and compare the resource used with FIMA method. It seems that there is no progress after this step. Secondly, based on the first step, we consider the influence of time window, and further reduce the resource used by simplifying time window, which in another word is sharing memory resource between operands in time window. After that, the resource used is extremely reduced. Thirdly, we proposed a method to share registers between assertions, which further reduce the hardware resource used in assertion sets.

It has to be noted that our proposal, counter based hardware resource reduction for FIMA System Verilog assertion checker, works in most of assertions, especially one with few variables and long delay, but it doesn't work well in situations with large number of variables and short delay. Compared with FIMA-based method, in most cases counter-based method is more resource friendly.

References

[1] Chengjie, Z. A. N. G., and Shinji Kimura. "Finite Input-Memory Automaton Based Checker Synthesis of System Verilog Assertions for FPGA Prototyping." IEICE transactions on fundamentals of electronics, communications and computer sciences 92.6 (2009): 1454-1463.

[2] System Verilog 3.1 Language Reference Manual, ver.3.1, http://www.eda.org/sv /System Verilog_3.1_final.pdf.

[3] Sohofi, H., & Navabi, Z. (2014, March). Assertion-based verification for system-level designs. In Quality Electronic Design (ISQED), 2014 15th International Symposium on (pp. 582-588). IEEE.

[4] Property Specification Language Reference Manual, ver. 3.1, http://www.eda.org/vfv/docs/PSL-v1.1.pdf.

[5] M. Boule and Z. Zilic, “Incorporating efficient assertion checkers into hardware emulation,” Proc. 23rd IEEE International Conference on Computer Design, pp. 221-228, 2005.

[6] M. Boule and Z. Zilic, “Efficient automata-based assertion-checker synthesis of SEREs for hardware emulation,” Proc. 12th Asia and South Pacific Design Automation Conference, pp.324-329, 2007.

[7] Altera Corporation, Quartus II Development Software Handbook, 9.1 ed., Mar. 2009.