2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

A framework of model checking guided test vector generation for the 6DOF manipulator

Yilin Lu[1], Yong Guan[2], Xiaojuan Li[3], Rui Wang[4], Jie Zhang[5]

***Abstract*—In designing robot control systems, simulation is still the primary approach to verifying the functions of circuit descriptions written in hardware design language. The validity of the verification depends on the coverage metric. But not all state spaces can be specified in a simulation. Model checking can overcome the shortcomings of simulation, because all of the state spaces can be traversed. Robot control system is an important part of the robot, used in the control of the manipulator to perform specific tasks. Therefore, in this paper a verification framework is presented for design correctness of robotic manipulator, which combines simulation and model checking. The framework is also applied to other control system or control hardware. Model checking guides the generating of test vectors, and makes the functional coverage reach 100% quickly. In this paper, a manipulator designed with six degrees of freedom is verified. The results of the verification of the manipulator show that the verification framework is effective for checking the design correctness of robotic manipulator.**

I. INTRODUCTION

The development of robots is the way of the future, and the safety and security of control system is a prerequisite for the development of robotic applications. For example, the manipulator is used in the application of the surgical systems. It must make sure that the manipulator with flexibility and safety. Therefore, validation work remains crucial in the design of robot control systems. Simulation-based verification is the primary approach in the industry due to the tractability and usability of the simulation process [1]. The process of simulation involves several steps, such as test generation and response evaluation, but functional coverage is central to all of the steps in the simulation process [2]. The size of the functional coverage determines the quality of the verification. As functional verification occupies most of the time in the verification cycle [3], many test vectors are required to simulate during in verification. With the commonly used method, it is difficult for functional coverage to achieve 100% in a short time. Therefore, how to write a high -quality test vector to verify all of the defined functions in the shortest amount of time possible, shorten the development cycle, and improve the efficiency of verification presents a challenge to verification engineer.

The basic idea of mode checking [4] is to describe the system state transition structure using the Finite State Machine (FSM) and indicate the nature of the design with temporal logic formulas. Mathematical methods are used to verify designs, in order to ensure all possible cases for authentication and make up the shortcomings that the simulation test vectors cannot all address.

The current literature on coverage metrics focuses on formal verification [5], [6].These approaches consider the coverage property that is generated for each important signal. There are also various studies on function coverage [7], [8] in simulation, where the coverage is increased by using a fault insertion method. Recently, [9] proposed a technique that integrates the characteristics of both simulation and formal verification for complex hardware designs, but the calculation process of the technique is complex, and the definition of a functional simulator is not completely. In software development, there are also methods to find counterexample using model checker, [10] proposed a method to derive test sequences based on requirements specification, also [11] described a method about how to formalize the common structural coverage criteria from the formal framework and how to use the criteria to provide test sequences to achieve the coverage. Current research on robot, many of them is about planning problem [12]-[15]. Of course, there are also some research on robot control system, [16] proposed a method for designing controllers that provide guarantee on the stability and safety, and [17] proposed a method to continuum robotic manipulator design and actuation.

In this paper, we focus on verification based on the register transfer level designing. We propose a verification framework that combines simulation and model checking for the 6DOF manipulator, the framework is also applied to other control system or control hardware. We can find a set of test stimuli by model checking that generate counterexample, so that the functional coverage can quickly reach 100%, with the coverpoints including any signal that one wishes to verify. We accomplished the verification, which refers to all aspects of the validation process and shows that the verification framework is effective for checking the design correctness of 6DOF manipulator through an analysis of the results of the verification.

This research was supported by the BJNSF (4122017), ISTCP (2011DFG1-3000), International Science and technology cooperation project (2010DFB10930), and NSF (61373034), and NSF (61303014).

[1](e-mail:luyilinxd@sina.com),[2](e-mail:guanyxxxy@263.net),[3](e-mail:lixjxxxy@263.net), and [4](e-mail:rwang04@163.com) are with the Beijing Engineering Research Center of High Reliable Embedded System, College of Information Engineering, Capital Normal University,Beijing 100048, China.

[5]College of Information Science & Technology, Beijing University of Chemical Technology, Beijing 100029, China. She is now a visiting scholar at the University of Tennessee, Knoxville (e-mail: jzhang62@utk.edu, jzhang@mail.buct.edu.cn).

The rest of this paper is organized as follows: In Section Ⅱ we introduce our framework, which combines simulation and model checking. Verification process is given and the results of our experiments are presented in detail in Section Ⅲ. We conclude with final remarks in Section Ⅳ.

II. THE FRAMEWORK COMBINING SIMULATION AND MODEL CHECKING

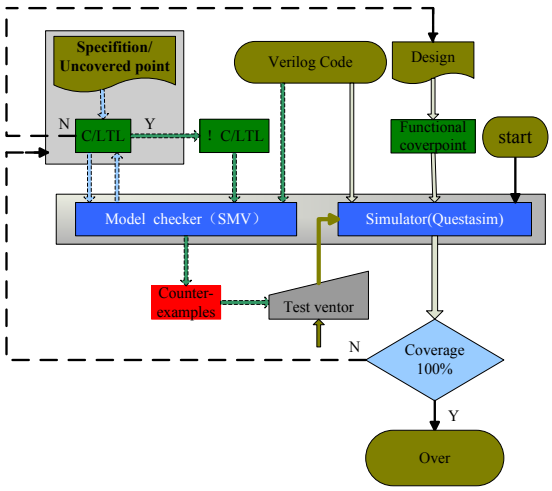

Fig. 1. The work flowchart

The framework integrates two kinds of verification tools: the simulation tool Questasim (Mentor Graphics Corporation) [18] and the model checking tool SMV [19]. The work flowchart is shown in Fig. 1. Not only can one carry out simulation and formal verification simultaneously for DUT (Design Under Test) and give validation results, but the main function is to find the test vector that enables the functional coverage to reach 100% in the shortest possible time. First, a random test vector, verilog code and functional coverpoint for simulation are added to the simulation tool. If all of the defined functional coverpoints are covered during in the initial simulation, the verification is at an end. If the functional coverage does not reach 100%, we should translate the uncovered functional coverpoint into Computation Tree Logic [2], [20], [21]. Second, the CTL formula is put into SMV for formal verification in order to get one counterexample. If the formal verification result is false, this means that the design does not satisfy the attribute in specification. Meanwhile, SMV will give one counterexample, but this counterexample is not what we need to achieve 100% coverage. What's more, we should modify the design because of the false result; If the formal verification result is true, this means that the design satisfies the attribute, in other word, we can find a path that the design satisfies the formula by formal verification, and then illustrates the coverpoint can be covered in theory. In order to obtain the path from SMV, the CTL formula should be negated. Then formal verification should be carried out again with the negated formula. This time, we will get a counterexample that does not satisfy the negated attribute formula. Finally, the counterexample should be added to the test vector set and the simulation should be run again. Repeating steps 3-6, we will find all of the corresponding test vectors to enable functional coverage to quickly reach 100%.

Step_1: Define the functional coverpoints.

Step_2: Generate test vectors.

Step_3: Carry out the simulation. If the functional coverage reaches 100%, then execute step 7. If does not reach 100%, go on to the next step.

Step_4: Formulate the functional coverpoint that is not covered in initial simulation into CTL formula p for formal verification. If the result is not correct, this means that an error is found, and you should modify the design. If the result is correct, go on to the next step.

Step_5: Negate property formula p, and carry out formal verification again, go on to the next step.

Step_6: Add the counterexample into test vector set, executive step 3.

Step_7: Validation is complete.

In the Simulator, we should add to Verilog code, Test vector and Functional coverpoint. The Functional coverpoint comes from the Design. The simulation tool Questasim statistical coverage and gives out the coverage report.

In the Model checker, we should add to the SMV model and the attribute formula. The SMV model comes from Verilog code. And the CTL formula refers to Specification and uncovered functional point. We should carry out formal verification for twice. We add to CTL formula in Model checker for verification for the first time, the Model checker will give out a result. And we should add to the negated CTL in Model checker again, but before we negate the CTL formula, we should ensure that the CTL formula is correct in the verification for the first time. This time the Model checker will give out a counterexample. We add the counterexample to the test vector set for a simulation.

A. Definition of Functional Coverpoint

The functional coverage model to measure simulation tests is the central part of the validation process. Only a functional coverage model that accurately reflects the behavior of the design can be considered fully verified. In [2], one method is presented that can automatically generate coverage models, a property that is described using the CTL formula. A functional coverage group contains one or more coverage points. Coverage points specify what integer expression is to be covered. Each coverpoint includes a set of bins on the sampled value of the expression or hopping cover positions. Coverage bins can be either explicitly specified by the user or generated automatically by systemverilog [22]. The coverage point expression is calculated to occur in the coverage group sampling time. The expression can be calculated in the process of providing the context. The expression can be accessed via a legitimate virtual interface. Functional coverage is particularly important in random tests. In this paper, functional coverage is defined as follows:

$$FC = \frac{\texttt{all passed (TEST) function points}}{\texttt{all tested function points}}$$

Generally, whether in direct testing or random testing, a verification plan will specify which function point needs to be

tested. In direct testing, a test case may correspond to a particular function point, so in many cases, regression testing requires all test vectors to reach the validation requirements. In random testing, a test case may be tested to different function points or one part of these points. Therefore, a specific measure is required. Generally, in accordance with the safety characteristics [17] and protocol requirements for robot verification or other manipulator, the definition of a functional coverpoint can refer to the following aspects [23]: Functional requirements, Interface Requirements, System Specification, and Protocol Specification. A specific verification plan may appear as: whether the FIFO overflows or gives an empty reading, the value of the output, state transition relations, as well as other regulatory requirements. The definition of coverpoint is shown in the following example:

```
bit [1:0] s0;
covergroup g4;
cover1: coverpoint s0 iff (!reset);
endgroup
g4 g4_1 = new();
```

The covergroup shows that if and only if the value of reset is low, and the value of the (!reset) holds true, will the coverpoint cover1 produce a statistic for s0.

B. The Generating of Test Vectors

Generating test vectors is a very important component of verification [24]. Test vectors can usually be generated through several ways: 1) a directed test; 2) a directed random test; and 3) a random test. For each stimulus generator strategy, the test space covered range is different, as shown in Fig. 2. A directed test is more suitable for a simple design. Because of its relatively small test space, we can cover the space with a certain number of tests. A directed test can also be used to test simple function points in a complex design, particularly some of the possible boundary conditions for potential defects. A random test can be applied to any type of design, especially in designs with a relatively large test space, or to a large number of interactive case properties. Because verification engineers cannot list all function points, it is important to hit an undesired functional coverpoint by using a random test. Some unusual random test stimuli can be created to test the design of unique and complex behavior for concurrent or asynchronous events. Computer systems introduce random numbers with a pseudo-random test generator, but these are pseudo-random numbers and not true random numbers [25]. Therefore, for complex chip designs, even though the stochastic method can usually verify most of the function points, it is necessary to manually add a direct stimulus to supplement the test vector set in order to verify all functional coverpoints and reduce the development of test cases.

C. Formulation of Functional Coverpoints

The Calculate Logic Tree formula is used to describe the nature of the system. In [26] and [27], the syntax and semantic definitions of the logic formula CTL are described in detail. The formulation of functional coverpoint in this paper is mainly aimed at finding a counterexample. But unfortunately, not all coverpoints can be formulated into CTL, like coverpoint which describes the changing of dataflow, it is not easy to describe it and even if we get a counterexample, it may not be able to make the functional coverage reached 100%. In order to get a counterexample we need, we negate the CTL formula, which mainly comes from equivalent formula and the principle of model checking [28]. Before we negate the CTL, the CTL formula must be validated in the SMV and get the correct result. In [29], there are several commonly used CTL formulas for digital IC verification.

In accordance with the requirements of the definition of a functional coverpoint, the expression of coverpoint can often be defined as the state transition and the change in the values of the variable. We can find a path that meets both the formula for the attributes and the formal formula for the property. Therefore, the problem of solving the functional point corresponds to the problem of finding the path. In the design of the 6DOF manipulator, we formulate part of the coverpoints as follows:

a) When the status register satisfy a certain value, it can make the start signal start_flag a high level. So, the design requirements of the signal start_flag must be true at some point in the future, that is the expression (start_flag = 1) is true. The problem can be described in all paths and all states, there must exist a path (start_flag = 1) that will be true in a future time, with the specific formula being expressed as: AG EF (start_flag=1);

b) The functional point start_flag_1 iff ((start_reg [1] = 1) & rst_n) indicates that if and only if ((start_reg [1] = 1) & rst_n) is true will there be coverage statistics for start_flag_1. The formula will be formulated to the attribute AG ((start_reg [1] = 1) & rst_n -> AF start_flag_1).

III. VERIFICATION PROCESS

The function of the design is that the manipulator grabs an object and move from the starting point to the ending point. The whole process can be divided into two parts, involving the control model and manipulator movement model. The function of control module is shown in Fig. 3, and the data flowchart between steering engines is shown in Fig. 4. We should power on the manipulator to make it into the state of initialization and wait for the input commands (keyboard, wireless remote control, network, etc.). Use the input command of keyboard in which key1 achieves the movement of gripping objects and key2 realizes the reversal, acceleration and deceleration of a few simple steering engines. After receiving the work order, the manipulators start moving. The manipulator get initiated and move to the designated starting point A. Decline the manipulator to grab the objects then control the steering engine to lift manipulator and move to the ending point B, put down the objects, lift again. Repeat the above process.

The steering engine6 conduct base rotary and after reaching the position of target object, it gives the next operation command answer_output_6; After receiving the command of steering engine6, steering engine5 conducts the rotation of arm. Then it gives the operation command answer_output_5; After receiving the command of steering engine5, steering engine4 conducts the rotation of forearm.

Then it gives the operation command answer_output_4; After receiving the operation command answer_output_4, steering engine3 conducts the wrist rotation. Then it gives the operation command answer_output_3; After receiving the command of steering engine3, steering engine2 conducts the back of the wrist. Then it gives the operation command answer_output_2; After receiving the command of steering engine2, steering engine1 grab the objects and close. Then it gives the operation command answer_output_1; After receiving the command answer_output_1, steering engine5, steering engine4 and steering engine3 conduct three-arm linkage simultaneously to lift the manipulator and give the operation command answer_output_3_1, answer_output_4_1 and answer_output_5_1. At last, in order to output the signal pwm_answer_output_1_345, we should make the signal answer_output_3_1, answer_output_4_1 and answer_output_5_1 to conduct the operation of logical AND on the top floor. After receiving the command of pwm_answer_output_1_345, steering engine6 conducts the movement of base rotary. Then it gives the command pwm_answer_output_6_1. It will rotate when the ADD module is triggered now, it will control the exercise time by changing the value of the counter (HL+/-20'd250), in order to achieve acceleration and deceleration. We can take steering engine6 for example, and the mechanism is shown in Fig. 5.

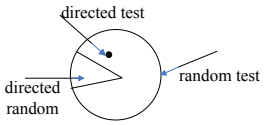

Fig. 2. Incentive to produce and test space coverage

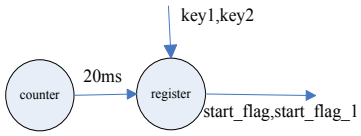

Fig. 3. The function of control module

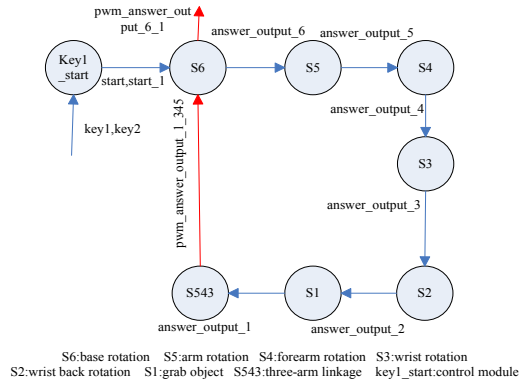

Fig. 4. The data flowchart between steering engines

A. *Definition of Functional Coverpoint*

For 6DOF manipulator, the main functional coverpoints that are defined include the following aspects:

1. The relationship between I/O ports: The direct input signals are key1 and key2 in the top module, while the output signals pwm_pulse_1, pwm_pulse_2, pwm_pulse_3, pwm_pulse_4, and pwm_pulse_5, represent the results under the input excitation signal. The functional coverpoints that need to be defined include four combinations of the input signal and the true and false values of the five output signals. For example: coverpoint pwm_pulse_5 {bins t = {0,1}}.

2. The changing of the register value: In accordance with the timing requirements, the counter reads the value of the key once every 20ms. The value to be inputted into the register, the register maximum overflow value cannot exceed 2^{20}. The reversible flag register and the flip angle are limited to the maximum value of 2^{20} and 2^{31}, respectively. For example: coverpoint pwm_6 iff (cnt2<=20'dFFFFF).

3. The data flow between the steers: These steers transmit signals with each other by the input and output of adjacent modules. These main functional coverpoints monitor the interfaces between six modules so that they will be able to complete the whole movement. For example: coverpoint pwm_answer_output_1_345 iff (pwm_answer_output_3_1 & pwm_answer_output_4_1 & pwm_answer_output_5_1). This mean that, when completed the wrist rotation, the back of the wrist, grab the object and close, it must make sure the signal pwm_answer_output_1_345 is true, so that we can judge whether the grab success. This is the function of the design must satisfy.

4. Design Requirements: The base rotation can only act directly on the arm, but cannot act on other joints because of the relationship between the joints and the 6DOF manipulator. Similarly, the arm can only drive the forearm, the forearm drives the wrist rotation, the wrist rotation drives the back of the wrist, and the wrist rotation has directly affects the grab. Once the object is grabbed, the arm, forearm and wrist should be rotated simultaneously. Therefore, those function points need to be monitored for independence between non-adjacent modules. For example: coverpoint ((cnt2 = 31'd0) && (HL = 20'd40000) && (answer_output_1 = 0) && (answer_output_1_1 = 0)).

B. *The Generating of Test Vectors*

In this paper, we define the test vectors by first taking them at random in the testbench. A random test vector is generated by a random function, but in this case we cannot get enough test vectors by the random function alone to make the functional coverage reach 100%. We find some uncovered functional points, like the functional point (!start_flag_1), that is still uncovered even though the simulation time lasts to 2^{40} (ns/ps) in the random simulation. Therefore, in order to cover these uncovered points, we need find tests with our verification framework.

C. *Formulation of Functional Coverpoints*

We need some kind of special test vectors to test the boundary value. But, first, we need to prove that the function point is true, that is the design has completed the realization of the function. And the function points are not been covered in the simulation, because of the lack of test vectors. Yet, in this

case, there are three points uncovered during in the simulation, see Table 2. The first function point means when the reset signal is valid or shut off the power, even if register values ((cnt2 == 31'd0) && (HL =20'd40000)) reached the execution conditions, steering engine 1 was unable to complete grab objects and closed movement ((answer_output_1 =0)&&(answer_output_1_1 = 0)). The second point means the steering engine cannot be started (!start_flag_1). And the third point means that when the reset signal is invalid and has a button to press (start_reg [1]==1), steering engine can be started(start_flag_1).

We should prove that these paths to which the points correspond are the exit. Practice has shown that such paths exist in SMV. Therefore, we can formulate the coverpoints to the CTL, such as AG EF !start_flag_1.Then, the CTL should be modified in such a way as to negate the CTL in order to get counterexample. Therefore, we modify the CTL to AG AX start_flag_1. That the functional point start_flag_1 is the exit is proved in SMV. Thus, we can get the counterexamples if only we put this property into SMV.

D. *The Results of Verification*

According to the requirements of Ⅲ.A, we basically completed the definition of all the functional coverpoints. In the process of random simulation, functional coverage change over time is shown in Table 1. From Table 1, we find that it is difficult to hit some functional coverpoints with purely random excitation. Therefore, in this paper we combine simulation and model checking, and transform the functional coverpoints that are not covered into CTL. We formulate these uncovered coverpoints into CTL, and negate them, and get some counterexamples in model checking, add these counterexamples into test vector set and make the coverage reached 100% at last. All of the processes are shown in Table 2 and Table 3. From Table 2 and Table 3 we find that when the simulation time is 2^{34} (ns/ps) in the initial stage, it is difficult to get a higher level of coverage. With our verification framework, we can find valid test vectors to make the functional coverage reach 100% in a very short time.

Module / Time (ns/ps)	key1_ start	servo_ 1	servo_2	servo_ 3	servo_ 4	servo_ 5	servo_ 6
2^{12}	0%	0%	0%	0%	0%	0%	0%
2^{13}	55%	92.8%	100%	100%	100%	100%	100%
2^{20}	55%	92.8%	100%	100%	100%	100%	100%
2^{25}	55%	92.8%	100%	100%	100%	100%	100%
2^{30}	65%	92.8%	100%	100%	100%	100%	100%
2^{33}	65%	92.8%	100%	100%	100%	100%	100%
2^{34}	90%	92.8%	100%	100%	100%	100%	100%
2^{35}	90%	92.8%	100%	100%	100%	100%	100%
2^{36}	90%	92.8%	100%	100%	100%	100%	100%
2^{40}	90%	92.8%	100%	100%	100%	100%	100%

Table 1: The change in the functional coverage with the change in time

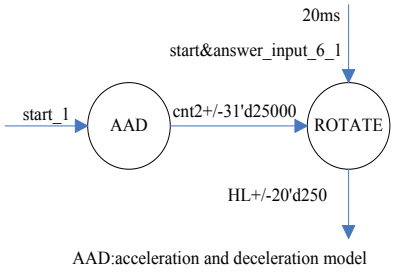

Fig. 5. The operation of acceleration and deceleration

Functional coverpoint	Property formula	Negated formula	Add test vectors	Original coverage	New coverage
(cnt2 == 31'd0) && (HL =20'd40000) && (answer_output_1 = 0) &&(answer_output_1_1 = 0)	AGEF(cnt2 = 31'd0) && (HL = 20'd40000) && (answer_output_1 = 0) && (answer_output_1_1 = 0)	AGAX!((cnt2 = 31'd0) && (HL =20'd40000) && (answer_output_1 = 0) &&(answer_output_1_1 = 0))	HL[19:0]=0;answer_input_1=0;answer_input_1_1=0; cnt2[30:0]=0;flag=0;rst_n=0; start_1=0;	50%	100%
!start_flag_1	AGEF!start_flag_1	AGAXstart_flag_1	cnt[19:0]=0;key1_input=0; key2_input=0;low_key1_input[1]=0;low_key1_input[0]=0;rst_n=0;start_flag_1=0;start_reg[1:0]=00;	50%	100%
start_flag_1 iff (start_reg [1]==1)&& rst_n	AG((start_reg [1]=1) & rst_n -> AF start_flag_1)	EF ((start_reg [1]=1) & rst_n ^ (EG !start_flag_1))	cnt=1111 0100 0010 0100 0000; rst_n = 1;key2_input=1;	50%	100%

Table 2: The process of achieving 100% coverage with the 6DOF manipulator verification

Ⅳ. CONCLUSION

In this paper, we presented a framework of model checking guided test vector generation that combines simulation and model checking. The framework can enable functional coverage to rapidly reach 100%. The 6DOF manipulator grab object is verified in the verification framework, and constitutes the entire verification process. In this process, the formulation of functional coverpoint was difficult, not much work had yet been done. Meanwhile, the definition of functional coverpoints is the key problem. In the application of the verification framework, only by ensuring that the definition of functional coverpoints and the formulation of functional point are correct, can the verification framework be given full play. In future research work, we will strive to achieve automated verification. For now, it is more important to note that the verification framework requires more complex examples to fully explain its effectiveness and efficiency.

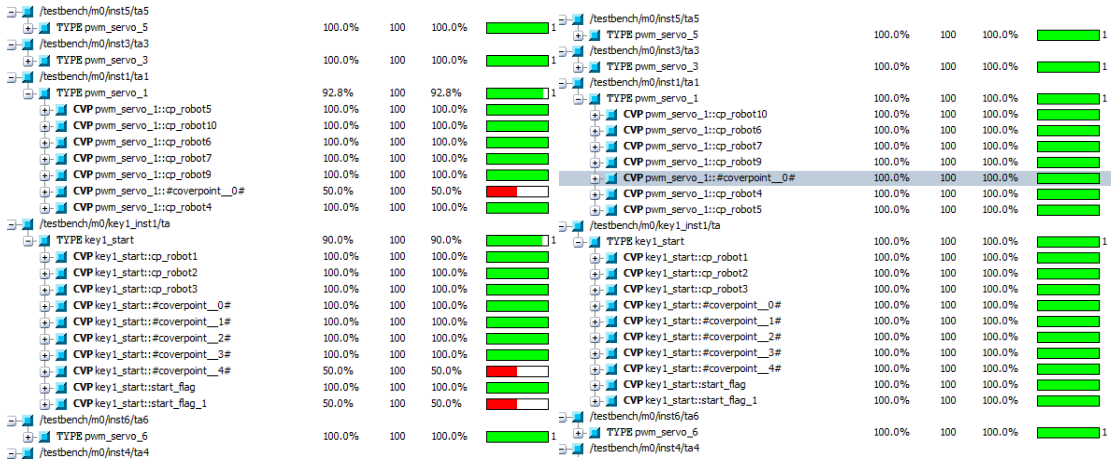

Table 3: The final result in the simulation tool

REFERENCES

[1] Vigyan Singhal and Prashant Aggarwal,Oski Technology, Inc, G. Using Coverage to Deploy Formal Verification in a Simulation World. Gopalakrishnan and S. Qadeer (Eds.): CAV 2011, LNCS 6806, pp. 44–49, 2011.c_Springer-Verlag Berlin Heidelberg 2011.

[2] Shireesh Verma, Ian G. Harris, Kiran Ramineni, Center for Embedded Computer Systems, Department of Computer Science,University of California, Irvine, USA. Automatic Generation of Functional Coverage Models from CTL. IEEE, 2007.

[3] LUO Dengfu and ZHAO Jianming , University of Electronic Science and Technology of China ,Integrated Circuit Verification Based on Coverage[A]. 1004-373X(2009)14-115-05.

[4] YU Dong-mei, College of Electrical and Information Engineering, Gansu Univ. of Tech., Lanzhou, China; Formal Verification Technique for Security Protocols. 1005-3751(2003)11-0112-03.

[5] Yatin Hoskote, Timothy Kam*, Pei-Hsin Ho**, Xudong Zhao, Strategic CAD Labs Design Technology, Intel Corp, Advanced Technology Group Synopsys, Inc. Coverage Estimation for Symbolic Model Checking. DAC 99, New Orleans, Louisiana(c) ACM 1-58113-109-7/99/06, 1999.

[6] Hana Chockler, Orna Kupferman, Hebrew University, School of Engineering and Computer Science, Jerusalem, Israel, Coverage Metrics for Formal Verification. 2003.

[7] Wang Zhong-hai, Microelectronics Center Harbin Institute of Technology. The Improvement for Transaction Level Verification Functional Coverage. 0-7803-8834-8/05/. IEEE, 2005.

[8] Alexander Krupp and Wolfgang Mueller Paderborn University/C-LAB, Paderborn, Germany, Classification Trees for Random Tests and Functional Coverage. 3-9810801-0-6/DATE06, EDAA, 2006.

[9] Lun Li, Stephen A. Szygenda and Mitchell A. Thornton Department of Computer Science and Engineering ,Southern Methodis University, Dallas, Texas，USA, Combining Simulation and Formal Verification for Integrated Circuit Design Validation.

[10] Gargantini, A.; Heitmeyer, C., "Using Model Checking to Generate Tests from Requirements Specifications", Proceedings of the ESEC/FSE'99 doi:10.1007/3-540-48166-4_10.

[11] Rayadurgam, S.; Heimdahl, M.P.E., "Coverage based test-case generation using model checkers," ECBS 2001. Proceedings. doi: 10.1109/ECBS.2001.922409.

[12] Xu Chu Ding, Alessandro Pinto and Amit Surana, Strategic Planning under Uncertainties via Constrained Markov Decision Process. International Conference on Robotics and Automation, IEEE, 2013.

[13] Austin Jones, Mac Schwager, and Calin Belta, A Receding Horizon Algorithm for Informative Path Planning with Temporal Logic Constraints. International Conference on Robotics and Automation, IEEE, 2013.

[14] Chanyeol Yoo, Robert Fitch and Salah Sukkarieh, Provably-Correct Stochastic Motion Planning with Safety Constraints. International Conference on Robotics and Automation, IEEE, 2013.

[15] Scott C. Livingston, Richard M. Murray, Just-in-time synthesis for reactive motion planning with temporal logic. International Conference on Robotics and Automation, IEEE, 2013.

[16] Anirudha Majumdar, Amir Ali Ahmadi, and Russ Tedrake, Control Design along Trajectories with Sums of Squares Programming. International Conference on Robotics and Automation, IEEE, 2013.

[17] Benjamin L. Conrad, Jinwoo Jung, Ryan S. Penning, and Michael R. Zinn, Interleaved Continuum-Rigid Manipulation: An Augmented Approach For Robotic Minimally-Invasive Flexible Catheter-based Procedures*.International Conference on Robotics and Automation, IEEE, 2013.

[18] http://www.mentor.com

[19] Kenneth L. McMillan, “Symsbolic Model Checking” An approach to the state explosion problem, CMU-CS-92-131, May, 1992.

[20] A. Ziv. Cross-product functional coverage measurement with temporal properties-based assertions. In Design, Automation and Test in Europe Conference and Exhibition. IEEE, 2003.

[21] K. Shimizu. Deriving a simulation input generator and a coverage metric from a formal specification. In Design Automation Conference. ACM Press. 2002.

[22] Mikhail Noumerov and Lyubov Zhivova, Using SystemVerilog for IC Verification. SNUG Europe, 2005.

[23] Nicola Bombieri, Franco Fummi, and Graziano Pravadelli, Dipartimento di Informatica - Universit`a di Verona, On the Evaluation of Transactor-based Verification for Reusing TLM Assertions and Testbenches at RTL*. 3-9810801-0-6/DATE06, EDAA, 2006.

[24] Osman Balci. Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA. Validation, Verification, and Testing Techniques throughout the Life Cycle of a Simulation Study. Winter Simulation Conference, 1994.

[25] Zhong Wenfeng, comp. Systemverilog and functional verification. Beijing: Mechanical Industry Press, 2010.

[26] SU Kai-Le, LUO Xiang-Yu, and LU Guan-Feng, Department of Computer Science , S un Yat-Sen University , Guangzhou, Symbolic Model Checking for CTL*. Chinese journal of computers, 2005.

[27] Yang Jun, GE Haitong, Zheng Feijun, Yan Xiaolang, Institute of VLSI Design, Zhejiang University. A formal verification method: Model Checking. Journal of Zhejiang University, 2006.

[28] E. M. Clarke, O. Grumberg, and A. Peled. *Model Checking*. MIT Press, 1999.

[29] Camille Constant, Thierry Je´ ron, Herve´ Marchand, and Vlad Rusu, INRIA, Rennes, Campus universitaire de Beaulieu,35042 Rennes, FranceIntegrating Formal Verification and Conformance Testing for Reactive Systems.IEEE,2007.