Design-to-Test Approach for Black-Box Testing of Programmable Controllers

Canlong Ma and Julien Provost

***Abstract*— This paper focuses on a global consideration of design and testing phases of a safety-critical automation system with programmable controllers. A design-to-test approach is proposed to improve the testability of programmable controllers and reduce overhead during the testing phase, under the premise that the nominal behavior of a system is guaranteed to remain unchanged during normal execution. This approach is elaborated and illustrated on a medium scale case study throughout the paper, and then applied to a larger case study in order to illustrate its scalability.**

I. Introduction

Nowadays, automation systems involve knowledge and technology from several fields such as mechanical and electrical engineering as well as computer science and thus become more difficult to design and test. Usually, the testing phase is not executed until the design and implementation phases are finished. This kind of approach is advantageous for simple and small scale systems because of its clear structure and easy operability. However, it faces challenges from high requirements of modern automation systems, especially for safety-critical systems.

Programmable controllers are widely employed even for safety-critical systems. Since their functionality requirements are getting more and more complex, testing of these controllers is becoming a crucial and challenging topic.

Functional safety standards of many industrial applications such as automotive (ISO 26262), process industry (IEC 61511) and railway (IEC 62279) strongly recommend the use of model-based design, as well as testing in complement to formal verification to validate the final implementation. When testing a safety-critical system, complete testing, i.e. exhaustive input combinations testing, is often required. However, complete testing of large scale systems is intractable due to the state-space and inputs' values combinations explosion. To cope this issue, safety-critical systems are often distributed over independent architecture, with different safety integrity levels (SIL). According to IEC 61508-2, if sufficient independence of implementation is established, each subpart of a whole system can be tested independently w.r.t. its SIL's testing requirements.

[1] shows that feasibility of functional test is currently limited by the cost for test program development. For safety-critical systems, verification of the software code is not sufficient to validate the correctness of the hardware controllers. In [2], a new technique of runtime verification for both software and hardware levels for ultra-critical systems is proposed. In [3], a fault injection framework for the evaluation of software-based self-tests for safety-critical systems according to the safety standard IEC 61508 is presented. However, this method needs a comprehensive fault library, which requires expert knowledge and is not always easy to be obtained in industrial practice.

Canlong Ma and Julien Provost are with Assistant Professorship for Safe Embedded Systems, Faculty of Mechanical Engineering, Technische Universität München, Garching bei München, Germany ma@ses.mw.tum.de; provost@ses.mw.tum.de

The design-to-test (DTT) method proposed in this paper does not require any expert knowledge. This method permits to automatically update the initial specification models in order to improve the testability of the final implementation executed by a programmable controller. This method also guarantees that the initial behavior of the specification models remains unchanged w.r.t. their initial sets of inputs and outputs. To the best of our knowledge, no such approach has been applied in field of programmable controllers yet.

Black-box testing is an important testing strategy focusing on functional behaviors of software products, where the internal structures are ignored [4]. For programmable controllers where the specification and the implementation are modeled as finite state machines (FSMs), the complete workflow of a black-box testing is carried out in 3 steps (Fig. 1). The prefix of a test case is to determine the current state; this is realized by homing or synchronizing sequences. The postfix is to identify the current state after a test case; this is realized by state identification or verification sequences [5]. These two steps constitute the *testing overhead*, which can be quite high for large scale systems with complex state-space structure.

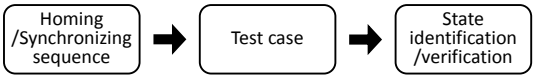


Fig. 1. Black-box testing for FSMs

To reduce the testing overhead, the issues of controllability by means of C-guards and observability by means of O-actions are investigated later in this paper in section VI and V respectively. These are the first two aspects of the proposed DTT approach. The third aspect is the single-input-change (SIC) testability problem [6], which will be solved by means of T-guards in section IV.

On the other hand, these additional guards and actions will add a *design-to-test overhead* to the system. Thus, a basic principle of the proposed DTT approach is to achieve a good balance between design-to-test overhead, system testability and testing overhead.

The remainder of the paper is organized as follows: Section II presents a reminder of TDD and DTT approaches,

Moore machines with signals guards, the cyclic execution mode of a programmable controller as well as definitions of SIC-testability, observability and controllability problems. The basic idea of the proposed DTT method on programmable controllers and a case study are explained in section III. In sections IV, V, VI, which correspond to the contributions of this paper, the DTT methods specifically for SIC-testability, observability and controllability problems are explained with algorithms and applied on the case study. To further evaluate the performance of the proposed method, a larger application is presented in section VII. Conclusions and prospects for further works are given in the last section.

II. Background

A. Test-Driven Development & Design-To-Test

Test-Driven Development (TDD) and Design-To-Test (DTT) methods both consider test at early phase of design and implementation. TDD aims at first defining the test cases and then writing and updating the code until all test cases pass. DTT aims at improving the testability of an implementation by design.

TDD method is far less applied on industry compared to the expectation [7]. Several researches have been undertaken to investigate the reasons and obstacles that restrict the wide usage of TDD. In [7], a summary of published results from research projects and practical experiments conclude that although the TDD approach provides better code coverage, it cannot be proven to be generally superior to other traditional approaches in terms of development time, change and maintenance cost. In [8], seven essential factors that limit the industrial adoption of TDD have been identified: "increased development time, insufficient TDD experience/knowledge, lack of upfront design, domain and tool specific issues, lack of developer skill in writing test cases, insufficient adherence to TDD protocol, and legacy code". However, worthwhile attempts have been made to adapt TDD on specific industrial applications. For instance, in [9], TDD is adapted to embedded C programming by applying a dual-targeting approach. In [10], a TDD process introduces adaption of UML models to enable effective test case derivation of automation systems.

DTT (or "design-for-test") method is initially conceived for testing of integrated circuits, aiming at achieving a high fault coverage by inserting test points into circuits at the cost of increasing the circuit area and the number of pins of the board [11]. In this domain, researchers mainly consider gaining a lower energy consumption [12], lowering the test time and silicon area cost [13], and also reducing overheads to fulfill fault coverage requirements [14].

The two latter ones, which focus on fault coverage and testing overhead, are however also of interest to the testing approach proposed in this paper for programmable controllers, while power consumption is of less interest for programmable controllers.

B. Moore Machine with Signal Guards

In this paper, SIC-testability, observability and controllability will be investigated for programmable controllers, assuming that the specification models are Moore machines, due to their simple structures and good compliance with common industrial applications. Also, many other FSMs can be easily transformed into Moore machines.

A Moore machine with Signal Guards is defined by a 6-tuple $(S, s_{init}, \Omega, \Lambda, \delta, \lambda)$ where:

- S is a set of states
- s_{init} is the initial state, $s_{init} \in S$
- Ω is a set of input signals
- Λ is a set of output actions
- $\delta : S \times 2^{\Omega} \rightarrow S$ is the transition function
- $\lambda : S \rightarrow 2^{\Lambda}$ is the output function

C. Cyclic Execution Mode

Cyclic execution is widely adopted by many programmable controllers. In each cycle, a controller runs successively: inputs reading, program execution, waiting until end of period and outputs updating. Inputs reading is the phase when the SIC-testability problem can occur [6].

Programmable logic controller (PLC) is picked as an example of implementation hardware for the application in this paper because of its high standardization, reliable performance and wide usage in industrial environments. However, the same approach could be applied to other event- or signal-driven cyclic programmable controllers as well.

D. SIC-testability

Single input change (SIC) test is initially a method used for electronics circuits testing, taking advantage of higher fault coverage [15] and lower power consumption [16] compared to multiple input change (MIC) test.

In the domain of PLCs, experiments have been conducted, and proved that biased or non-valid test results may be obtained when using a non-SIC test sequence because the I/O scanning cycle of the controller provokes asynchronism between input signals that are assumed to be synchronous [17]. These error-prone results can be completely excluded if all the input changes during one test step are guaranteed to be synchronous. One effective method is to construct complete SIC-test sequences. The coverage rate of SIC-testable parts of a system is defined as SIC-testability and a method to automatically obtain such parts is proposed in [6]. [18] gives a discussion to clarify the conformance relation between specification and implementation for PLCs, and raises several possibilities to maximize SIC parts, which lowers the risk of errors in total testing. It is, however, not always possible to obtain a complete SIC-test sequence from any specification. Thus, this paper consider another way to achieve full SIC-testability.

E. Observability

In several references upon test approaches for programmable controllers, e.g. recent ones [6] and [18], researches were carried out under a strong assumption: all states are identifiable by observing the emitted outputs at time t. If this is not satisfied, problems of state identification arise during the test execution [5].

For Moore machines, output actions are linked to respective states. As in the domain of black-box testing all internal structures (states and transitions) are invisible to testers. If some states have the same output actions, they can not directly be distinguished from each other. This state identification problem might be solved by searching for a distinguishing sequence. However, the existence of such sequences is not always guaranteed and if they exist they may be of exponential length [5].

F. Controllability

A system is said to be controllable at time t_0, if it is possible by means of a control vector to transfer the system from any initial state $x(t_0)$ to any other state in a finite interval of time [19](Page 675).

For FSMs, the controllability problem can be solved in two steps: identify the final state after some operations and then move to the desired state. The first step is solved by searching for a homing or synchronizing sequence. Both methods determine the final state of a machine after applying the sequence [5]. After that, the machine can be brought to any specific state by applying the corresponding input sequences derived from the specification model in the design phase. Of course, this process is conditioned on fulfillment of the previously mentioned observability requirement.

III. Design to test (DTT)

A. DTT Method On Programmable Controllers

The DTT method proposed in this paper is slightly different from the researches conducted in the domains introduced in subsection II-A.

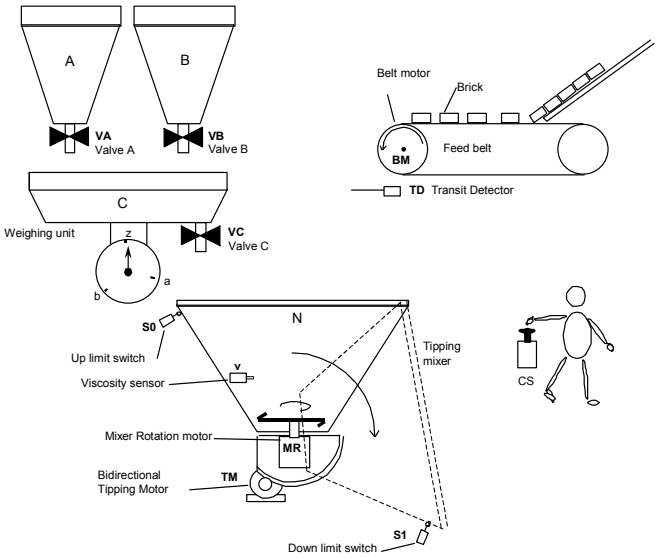
For the test of programmable controllers modeled as FSMs, as discussed in this paper, the basic idea is to insert some points on transition guards or output actions of a system during the design phase. Thus, in the testing phase, testers can use them to achieve a full fault coverage and full SIC-testability, and to optimize the length of the whole test sequence thanks to the improved controllability and observability features.

On the other hand, during the normal execution those points will be plugged respectively to the logic *True* or *False* signals and guarantee that the initial behavior remains unchanged w.r.t. the initial sets of inputs and outputs.

B. Illustration on a Case Study

In this paper, the DTT methods for SIC-testability (section IV), observability (section V) and controllability (section VI) will be illustrated through a case study, adapted from [20], with 8 logic inputs and 7 logic outputs (Fig 2).

In the case study, an automatic weighing-mixing system containing 4 units is presented. In the weighing unit, product *A* is weighed while the mark of scale *C* goes from *z* to *a*, then product *B* is dosed until mark *b* is reached. After that, the two products are poured into a mixer. Meanwhile, in the brick unit, two bricks are brought one by one through a belt into the mixer. The mixer holds its position up until a preset viscosity *v* of the mixture is detected. Then, a tipping



Inputs	**CS**	**TD**	**a**	**b**
	Cycle Start	Transit Detector	Fluid weight A reached	Fluid weight A+B reached
	z	**S0**	**S1**	**v**
	Empty weighing unit	Mixer up	Mixer down	Viscosity Sensor
Outputs	**BM**	**MR**	**TMp**	**TMm**
	Belt Motor	Mixer Rotation motor	Tipping Motor down	Tipping Motor up
	VA	**VB**	**VC**	
	Opening Valve A	Opening Valve B	Opening Valve C	

Fig. 2. Case study

motor tilts the mixer down to tip off the mixture, while the rotation continues. Finally, the mixer finishes emptying, stops rotating and turns back to the upright position again. To ensure the proper composition of the 4 units, a control unit is established.

The 5 units are modeled with Moore machines as presented in Fig. 3, where boolean signals are used as input guards. After applying synchronization (parallel composition of FSMs with stability research) by Teloco [6], a composed FSM is obtained with 19 stable locations and 50 evolutions (Fig. 4).

There are several issues to investigate in this case study. In the brick unit, states *B2*, *B3* and *B4* share the same output action *BM*, which consequently leads to the fact that several sets of locations in the composed FSM may also share the same output actions such as locations 2, 3 and 6 which emit *BM* and *VA*, and locations 4, 5 and 7 which emit *BM* and *VB* and so on. Besides, the composed FSM is not fully SIC-testable, this will be detailed in the next section.

IV. DTT for SIC-Testability

A. T-Guard Method

For the SIC-testability problem, DTT provides a fundamental solution by setting up a T-guard on every transition of a FSM. For every transition, its guard is updated as follows: $g' := g \cdot T$. For example in the case study, $g'_{W2 \to W3} := a \cdot \neg b \cdot \neg z \cdot T$.

During the normal execution of the implementation, the variable T is assigned the value *True* (by linking the input signal to 24V for example). Thus, the initial behavior of the

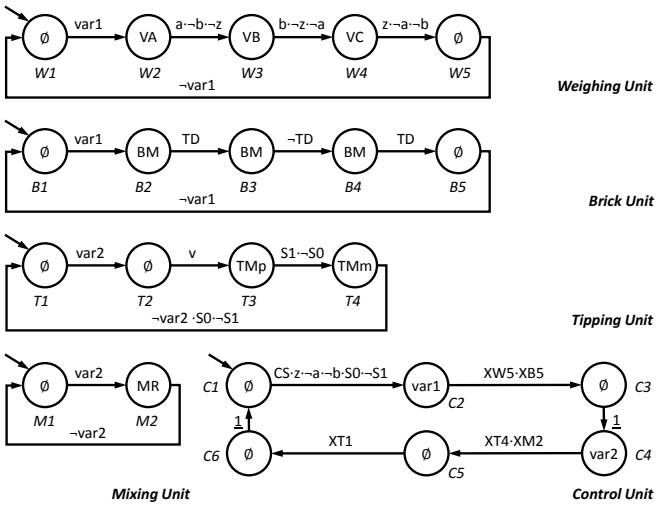


Fig. 3. 5 FSMs modeling the behavior of the case study ("X"-signals such as XT4 and XW5 indicate if the states T4 and W5 are activated at the moment in the corresponding FSMs.)

specification remains unchanged ($g \cdot T_{|T=True} = g$). Then, even if the number of inputs is increased by one, this doesn't increase the size of the state-space to be tested because only the part where T is *True* needs to be tested.

Fig. 5 presents how full SIC-testability can obtained by the T-guard method. A controller runs periodically input reading, program execution and outputs updating (waiting phase is not drawn in the figure). During the execution of the tests, the variable T will be assigned the value *True* as long as a transition is SIC-testable. In the case of the test of a non-SIC-testable part of a location, the variable T will be assigned the value *False* at least one period before the MIC occurs, which will freeze the firing of transitions from the current state. Since outgoing transitions are frozen, the MIC input signals will have time to be stabilized; then after at least one period, the variable T will be assigned to the value *True* again, and a transition could be fired. Thus, any outgoing transition requiring a MIC test step to be fired will be temporary frozen and thus the system can be completely tested without errors due to asynchronism between input signals.

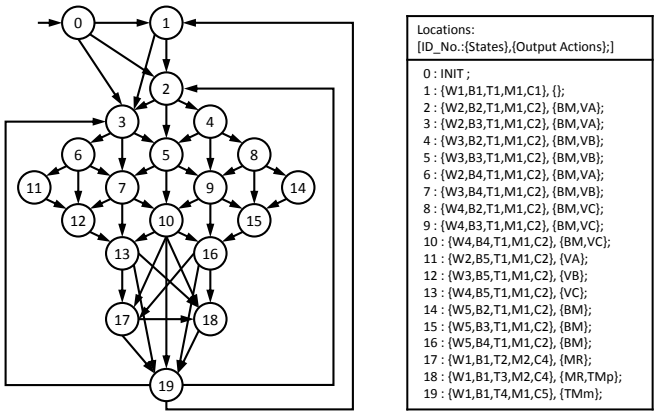
B. T-Guard Method on Case Study

With the help of Teloco, the composed model is checked for SIC-testability. The results show that 9 out of the 19 locations are not fully SIC-testable. For example, the SIC-testable part of location 4 is "$SIC_4 = \neg TD + a + \neg b + z$", so the non-SIC-testable part is $N_4 = TD \cdot \neg a \cdot b \cdot \neg z$, which is a combination of 4 input signals. Taken the total 8 input signals of the system into account, consequently the SIC-testability coverage amounts to $1 - 2^4/2^8 = 15/16$.

After adding T-guards, the model is guaranteed to be fully SIC-testable by construction.

C. Advanced T-Guard Method

However, for large scale systems, adding a T-guard on every transition guard can bring a large amount of overhead in the implementation code. Thus, in order to reduce overheads



Locations:
[ID_No.:{States},{Output Actions};]

0 : INIT ;
1 : {W1,B1,T1,M1,C1}, {};
2 : {W2,B2,T1,M1,C2}, {BM,VA};
3 : {W2,B3,T1,M1,C2}, {BM,VA};
4 : {W3,B2,T1,M1,C2}, {BM,VB};
5 : {W3,B3,T1,M1,C2}, {BM,VB};
6 : {W2,B4,T1,M1,C2}, {BM,VA};
7 : {W3,B4,T1,M1,C2}, {BM,VB};
8 : {W4,B2,T1,M1,C2}, {BM,VC};
9 : {W4,B3,T1,M1,C2}, {BM,VC};
10 : {W4,B4,T1,M1,C2}, {BM,VC};
11 : {W2,B5,T1,M1,C2}, {VA};
12 : {W3,B5,T1,M1,C2}, {VB};
13 : {W4,B5,T1,M1,C2}, {VC};
14 : {W5,B2,T1,M1,C2}, {BM};
15 : {W5,B3,T1,M1,C2}, {BM};
16 : {W5,B4,T1,M1,C2}, {BM};
17 : {W1,B1,T2,M2,C4}, {MR};
18 : {W1,B1,T3,M2,C4}, {MR,TMp};
19 : {W1,B1,T4,M1,C5}, {TMm};

Evolutions: [From(Location) : To(Location) : Input Guards;]

0 : 1 : ¬CS+a+b+¬z+¬S0+S1;	6 : 11 : TD·¬a+TD·b+TD·z;	12 : 13 : ¬a·b·¬z;
0 : 2 : CS·¬TD·¬a·¬b·z·S0·¬S1;	6 : 12 : TD·a·¬b·¬z;	13 : 17 : ¬a·¬b·z·¬v;
0 : 3 : CS·TD·¬a·¬b·z·S0·¬S1;	7 : 10 : ¬TD·¬a·b·¬z;	13 : 18 : ¬a·¬b·z·S0·v+¬a·¬b·z·¬S1·v;
1 : 2 : CS·¬TD·¬a·¬b·z·S0·¬S1 ;	7 : 12 : TD·a+TD·¬b+TD·z;	13 : 19 : ¬a·¬b·z·¬S0·S1·v;
1 : 3 : CS·TD·¬a·¬b·z·S0·¬S1;	7 : 13 : TD·¬a·b·¬z;	14 : 15 : TD;
2 : 4 : ¬TD·a·¬b·¬z;	8 : 14 : ¬TD·¬a·¬b·z;	15 : 16 : ¬TD;
2 : 3 : TD·¬a+TD·b+TD·z;	8 : 9 : TD·a+TD·b+TD·¬z;	16 : 17 : TD·¬v;
2 : 5 : TD·a·¬b·¬z;	8 : 15 : TD·¬a·¬b·z;	16 : 18 : TD·S0·v+TD·¬S1·v;
3 : 5 : TD·a·¬b·¬z;	9 : 15 : TD·¬a·¬b·z;	16 : 19 : TD·¬S0·S1·v;
3 : 6 : ¬TD·¬a+¬TD·b+¬TD·z;	9 : 10 : ¬TD·a+¬TD·b+¬TD·¬z;	17 : 18 : S0·v+¬S1·v;
3 : 7 : ¬TD·a·¬b·¬z;	9 : 16 : ¬TD·¬a·¬b·z;	17 : 19 : ¬S0·S1·v;
4 : 8 : ¬TD·¬a·b·¬z;	10 : 16 : ¬TD·¬a·¬b·z ;	18 : 19 : ¬S0·S1;
4 : 5 : TD·a+TD·¬b+TD·z;	10 : 13 : TD·a+TD·b+TD·¬z;	19 : 1 : ¬CS·S0·¬S1+a·S0·¬S1+b·S0·¬S1+¬z·S0·¬S1;
4 : 9 : TD·¬a·b·¬z;	10 : 17 : TD·¬a·¬b·z·¬v;	19 : 2 : CS·¬TD·¬a·¬b·z·S0·¬S1;
5 : 9 : TD·¬a·b·¬z;	10 : 18 : TD·¬a·¬b·z·S0·v+TD·¬a·¬b·z·¬S1·v;	19 : 3 : CS·TD·¬a·¬b·z·S0·¬S1;
5 : 7 : ¬TD·a+¬TD·¬b+¬TD·z;	10 : 19 : TD·¬a·¬b·z·¬S0·S1·v;	
5 : 10 : ¬TD·¬a·b·¬z;	11 : 12 : a·¬b·¬z;	
6 : 7 : ¬TD·a·¬b·¬z;		

Fig. 4. Composed FSM of the case study

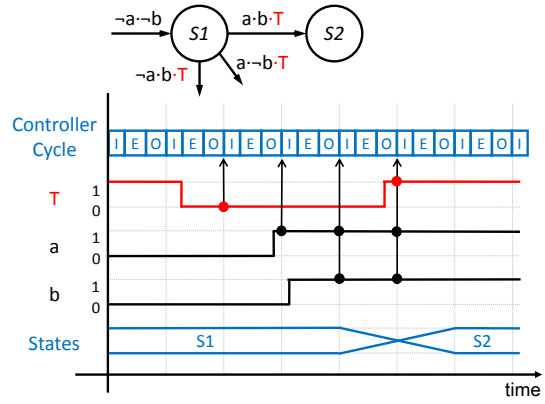


Fig. 5. Synchronization of a MIC test step through the T-guard method. For clarity reasons, self-loops are not presented: for all combinations of inputs, if none of the outgoing transitions is enable, then the system remains in the same active state (especially when T is *False*).

to the initial specification, a potential improvement is raised: instead of adding T-guards on every transition guard of FSM models, it could be enough to have T-guards only on some "key transition guards" in the models.

For small scale or simple structure systems, it may be possible to select these "key transition guards" intuitively. However, it is nearly impossible for large scale systems. The easiest automatic method is to list all the combinations of transitions, in which T-guards will be added, then recalculate the SIC-testability for every combination, and select the best solution. Since a system with t transitions has 2^t possible combinations of transitions, this algorithm would have a complexity of $O(2^t)$ with t the number of transitions, which

could hardly be acceptable for large scale systems.

Hence an algorithm with lower complexity is pursued. One suggestion to automatically find an optimized number of "key transition guards", namely the advanced T-guard method, is proposed in Alg. 1. The complexity is $O(mn)$, with m the number of non-fully SIC-testable locations, which is bounded by the number of all locations, and n the number of input variables. For practical application, the composed FSM should be strongly connected (or trim), i.e. all locations should be reachable and coreachable. Thus, since every location is reachable from any other location, the number of locations is smaller or equal to the number of transitions. In addition, for common industrial cases, a system usually has much more transitions than input variables. Since for large numbers $O(mn) < O(t^2) < O(2^t)$, this algorithm has a lower complexity than the previous one. However, this algorithm is sub-optimal and may be further optimized.

Given, a system with n boolean input variables denoted $I_1, I_2, ..., I_n$. In practice, the use of the T variable can only force a guard to *False*. Thus, in order to be able to freeze a transition, whatever the values (*True* or *False*) of the input variables are, the initial input variables set is extended as follows: $\forall 1 \leq j \leq n, I_{n+j} = \neg I_j$.

Then, the non-SIC-testable part N_i of a location can be obtained by calculating the complement of the SIC-testable part and formulated in canonical conjunctive normal form, e.g. "$N_i = (a + b) \cdot (c + d + e) \cdot (f + g + h + i)$". Thus, m locations, denoted $N_1, N_2, ..., N_m$, with non-SIC-testable parts are obtained. The $2n$ boolean variables together with m non-SIC-testable parts compose the inputs for the algorithm. The output is aimed to be an optimal set with a minimum number from the $2n$ variables, that covers all the m non-SIC-testable parts.

Data: $\Omega_0 = \{I_1, I_2, ..., I_{2n}\}, \Sigma = \{N_1, N_2, ..., N_m\}$
Result: Ω_{adv}
$\Omega := \Omega_0$;
for $i = 1$ **to** $2n$ **do**
  $I_i := 0$;
end
for $j = 1$ **to** $2n$ **do**
  $\Omega := \Omega \backslash I_j$;
  $I_j := 1$;
  for $k = 1$ **to** m **do**
    if $N_k \neq 0$ **then**
      $\Omega := \Omega \cup I_j$;
      $I_j := 0$;
      break
    end
  end
end
$\Omega_{adv} := \Omega$;

Algorithm 1: Pseudo-code of the advanced T-Guard algorithm

D. Advanced T-Guard Method on Case Study

In the case study, the non-SIC-testable parts for each location are as follows:

- $N_2 = TD \cdot a \cdot \neg b \cdot \neg z$
- $N_3 = \neg TD \cdot a \cdot \neg b \cdot \neg z$
- $N_4 = TD \cdot \neg a \cdot b \cdot \neg z$
- $N_5 = \neg TD \cdot \neg a \cdot b \cdot \neg z$
- $N_6 = TD \cdot a \cdot \neg b \cdot \neg z$
- $N_7 = TD \cdot \neg a \cdot b \cdot \neg z$
- $N_8 = TD \cdot \neg a \cdot \neg b \cdot z$
- $N_9 = \neg TD \cdot \neg a \cdot \neg b \cdot z$
- $N_{10} = \neg TD \cdot \neg a \cdot \neg b \cdot z$
- for the rest, $N_i = 0$

Applying this method on the case study, one feasible solution is obtained: $\Omega_{adv} = \{\neg b, \neg z\}$. That means, T-guards should be added on all guards of transitions containing the variable's value $\neg b$ or $\neg z$. In the individual models of the case study, T-guards are added on guards of transitions (*W2* $\rightarrow$ *W3*), (*W3* $\rightarrow$ *W4*) and (*W4* $\rightarrow$ *W5*) in the weighing unit. The composed FSM is then proved by Teloco to be fully SIC-testable.

V. DTT for Observability

A. O-Action Method

Unlike the method of searching distinguishing sequences, DTT solves the observability problem by inserting additional output actions into the internal structure of the system.

If two states S_i and S_j have the same output action: $A_{Si} = A_{Sj} \in \Lambda$, the output actions will be modified by adding an O-Action (*True/False*) as follows: $A'_{Si} = A_{Si} \cdot O_k$, $A'_{Sj} = A_{Sj} \cdot \neg O_k$

For a more general case, if n states share a same output action, it is easily deduced that $\lceil \log_2 n \rceil$ output actions are enough to differentiate each of them.

B. O-Action Method on Case Study

This problem occurs in the case study. In the brick unit, the 3 states *B2*, *B3* and *B4* share the same output action *BM*. Thus the number of necessary O-Actions is 2. After insertion, one feasible set of new output actions for these states can be:

- $A'_{B2} = BM \cdot O_1 \cdot \neg O_2$
- $A'_{B3} = BM \cdot O_1 \cdot O_2$
- $A'_{B4} = BM \cdot \neg O_1 \cdot O_2$

VI. DTT for Controllability

A. Controllability Performance

Controllability and observability play two key roles in the design of control systems. The principle also applies to the domain of testing. Once the problem of observability has been solved in the previous section, all states/locations are distinguishable from each other. In this section, the performance of how rapid and accurate each location of a FSM can be reached, is quantitatively evaluated and optimized.

In the context of FSM test, controllability is defined by the distance (or cost) from an initial location to a target location. The composed FSM of a system provides a set of direct paths. For pairs of locations which are not directly connected, Floyd–Warshall algorithm will be employed to

calculate the shortest path between them. The complexity is $O(l^3)$ with l the number of locations [21]. Thus, a path cost matrix of all location pairs is obtained, in which columns are initial locations and rows are final locations for all transitions (Fig. 6, left part).

B. C-Guard Method

In practical applications, e.g. an efficient test of a whole system, a higher controllability than the initial one is usually demanded. A C-guard method applying the DTT idea for improvement is proposed in Alg. 2. The basic idea is to add several C-guards between some pairs of locations, to make a direct path between them through this C-guard transition. Consequently, some other long paths also benefit from the new direct paths. Thus, the paths in between all transitions are globally shortened.

Inputs of the algorithm are the initial path cost matrix of a system P_{init}, and the limit of maximal path length $limit$ defined by the designer according to test requirements. This method is aimed at obtaining a minimum set C of C-guards added to the initial system, so that an optimized path cost matrix P_{opt} is reached, which fulfills the controllability requirement for a testing, i.e. all path costs of the system are within the given $limit$.

C. C-Guard Method on the Case Study

Initially, the maximum distance between two locations is equal to 7, e.g. from location 3 to location 14 (Fig. 6, upper part). To pursue a better controllability, the limit of step lengths from one location to any other is set to be 4. Then, in Fig. 6, the paths longer than the limit are marked with blue color, longer paths have darker colors.

Applying the proposed method, 6 C-guards are added to the composed system on following pairs of locations: $(4 \rightarrow 6)$, $(11 \rightarrow 4)$, $(12 \rightarrow 8)$, $(14 \rightarrow 2)$, $(15 \rightarrow 11)$, $(18 \rightarrow 14)$. In the single units, these C-Guards build respectively 8 new transitions: (*W1* $\rightarrow$ *W5*), (*W3* $\rightarrow$ *W2*), (*W5* $\rightarrow$ *W2*), (*B2* $\rightarrow$ *B4*), (*B3* $\rightarrow$ *B5*), (*B5* $\rightarrow$ *B2*), (*T3* $\rightarrow$ *T1*), (*C4* $\rightarrow$ *C2*).

The right part of Fig. 6 shows that the C-guard method successfully achieves a better controllability requested by testing requirements.

Moreover, attempts with a $limit$ of 6, 5, 3 and 2 for the case study have also been executed by the algorithm, returning 1, 5, 18 and 32 C-guards on the composed FSM, which correspond to adding 2, 6, 24 and 24 new transitions on the individual models, respectively. Compared to all the other choices the $limit$ of 4 is a good trade-off between testing and design overheads.

VII. Scalability Evaluation

To further evaluate the performance of the proposed DTT method, it has also been applied on a larger scale system (12 inputs, 12 outputs, 4 distributed plant models and 2 supervisory controllers) adapted from [22]. After using the same modeling and composition methods as for the case study, 6 individual models are composed into one FSM with 206 stable locations and 3029 evolutions.

Data: $P_{init}, limit$
Result: P_{opt}, C
$P := P_{init}$; $C := \varnothing$; $improvement := True$;
while *improvement* **do**
  $\forall [i,j] \in P$: **if** $P[i,j] > limit$ **then** $P_{new}[i,j] := 1$;
    else $P_{new}[i,j] := 0$;
  if $P_{new} = \varnothing$ **then**
    $improvement := False$;
  else
    $r_{max} := 0$; $c_{max} := 0$;
    $\forall i$: $cnt := \sum_i P_{new}[i,:]$;
      if $cnt > r_{max}$ **then**
        $(r_{max}, i_{max}) := (cnt, i)$;
    $\forall j$: $cnt := \sum_j P_{new}[:,j]$;
      if $cnt > c_{max}$ **then**
        $(c_{max}, j_{max}) := (cnt, j)$;
    if $r_{max} > c_{max}$ **then**
      $min := \infty$;
      $\forall i \neq i_{max}$: $sum \mathrel{+}= P_{new}[i,j] * P[i,j]$;
        if $sum < min$ **then**
          $(min, i_{min}) := (sum, i)$;
      $(C_{from}, C_{to}) := (i_{min}, i_{max})$;
    else
      $min := \infty$;
      $\forall j \neq j_{max}$: $sum \mathrel{+}= P_{new}[i,j] * P[i,j]$;
        if $sum < min$ **then**
          $(min, j_{min}) := (sum, j)$;
      $(C_{from}, C_{to}) := (j_{max}, j_{min})$;
    end
    $C := C \cup (C_{from}, C_{to})$;
    $P := Floyd{-}Warshall(P)$;
  end
end
$P_{opt} := P$;

Algorithm 2: Pseudo-code of the C-Guard algorithm

None of these locations are fully SIC-testable, and most of them are even fully non-SIC-testable. To obtain a full SIC-testability, all the transitions of individual models that depend on at least one input are updated with T-guards, i.e. 12 in total.

One pair of 2 locations, four pairs of 3 locations and 1 pair of 4 locations in the individual models share same output actions, thus 11 O-Actions need to be added to the individual models to satisfy observability requirement.

The maximum distance between two locations is 7. In order to limit the maximum distance between any pair of locations to 6, 5, 4, 3 and 2, the composed FSM is updated with 1, 4, 17, 157 and 343 C-guards, which correspond to adding 4, 15, 30, 52 and 31 new transitions to the individual models, respectively.

The state space of this second system is more than 10 times larger and the number of evolutions is more than 60 times larger than the first case study. The DTT method applied on this second example only requires 4 times more T-guards, and 5.5 times more O-actions and 3.75 times more

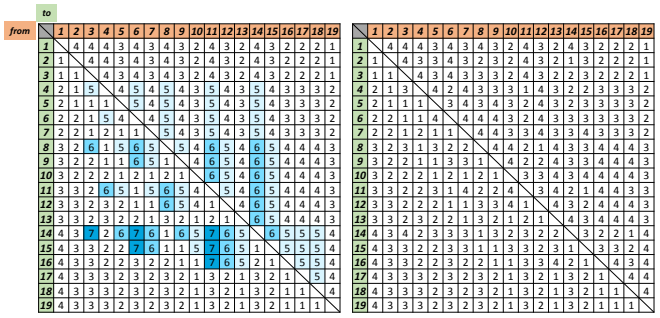


Fig. 6. Initial and optimized path cost matrix

C-guards on the individual models to obtain the same SIC-testability, observability and controllability results.

It is also interesting to note that for this example, the number of additional C-guard transitions on the individual models needed to obtain a maximum distance between two locations of 2 is almost identical to the number needed for a maximum distance of 4, i.e. 31 and 30 respectively. Also, the number of C-guard transitions that need to be added on the individual models for a maximum distance of 2 is smaller than the number needed for a maximum distance of 3. This is due to the fact that the algorithm is executed on the composed FSM and the relation between the number of evolutions in the composed FSM does not grow up linearly with the number of transitions in the individual models. This implies that the proposed DTT method generates slightly more C-guards transitions than needed on the individual models. However, that permits to significantly limit the computational complexity for the proposed algorithm: the individual models are updated only once at the end of the algorithm.

VIII. Conclusion

This paper has presented a design-to-test approach for programmable controllers. It proposes a fundamental solution to solve the SIC-testability problem and also provides a systematical scheme to obtain a good balance between the classic testing overhead (long synchronization and identification sequences) and new design-to-test overhead (modified or additional transitions and output functions). It is important to note that during the normal execution of the implementation, the T-guards are set to *True* and C-guards are set to *False*. Thus, the proposed DTT method guarantees that the initial behavior of the specification remains unchanged.

Following works are aiming at further improve and optimize the proposed DTT methods. For instance, concerning observability problem for large scale systems, there could be a large number of states that hold the same output actions. Solely using DTT method could lead to a lot of insertions to the system, which is also not optimal for the encapsulation. In such cases, a combination of the classic way of distinguishing sequences and DTT method would be a promising approach. An executable software tool and examples are available online (www.ses.mw.tum.de).

The next step would be the extension of the DTT approach to industrial application languages such as MATLAB Stateflow or SysML State machine. Then, a systematic and comprehensive comparison of the DTT and other testing approaches would be proposed in future works.

References

[1] M. Sonza Reorda, "In-field test of safety-critical systems : is functional test a feasible solution ?" in *Test Symposium (LATS), 2015 16th Latin-America*. Puerto Vallarta: IEEE, 2015, pp. 1–2.

[2] L. Pike, S. Niller, and N. Wegmann, "Runtime Verification for Ultra-Critical Systems," in *2nd Int. Conf. on Runtime Verification (RV 2011).*, 2012, pp. 310–324.

[3] A. Holler, G. Schonfelder, N. Kajtazovic, T. Rauter, and C. Kreiner, "FIES: A Fault Injection Framework for the Evaluation of Self-Tests for COTS-Based Safety-Critical Systems," in *15th Int. Microprocessor Test and Verification Workshop*, 2014, pp. 105–110.

[4] R. S. Pressman, *Software Engineering A Practitioner's Approach Seventh Edition*. MC Graw Hill, 2010.

[5] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines – a survey," *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1090–1123, 1996.

[6] J. Provost, J.-M. Roussel, and J.-M. Faure, "Generation of Single Input Change Test Sequences for Conformance Test of Programmable Logic Controllers," *IEEE Trans.on Ind. Inform.*, vol. 10, pp. 1696–1704, 2014.

[7] A. Bulajic, S. Sambasivam, and R. Stojic, "Overview of the Test Driven Development Research Projects and Experiments," in *Proceedings.Informingscience.Org*, no. 2011, 2012. [Online]. Available: http://proceedings.informingscience.org/InSITE2012/InSITE12p165-187Bulajic0052.pdf

[8] A. Causevic, D. Sundmark, and S. Punnekkat, "Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review," in *2011 4th IEEE Int. Conf. on Software Testing, Verification and Validation*, 2011, pp. 337–346.

[9] J. Grenning, *Test-Driven Development for Embedded C*. The Pragmatic Programmers, 2011. [Online]. Available: http://shop.oreilly.com/product/9781934356623.do

[10] R. Hametner, D. Winkler, T. Östreicher, S. Biffl, and A. Zoitl, "The Adaptation of Test-Driven Software Processes to Industrial Automation Engineering," in *8th IEEE Int. Conf. on Ind. Inform.*, 2010, pp. 921–927.

[11] C. Schotten and H. Meyr, "Test point insertion for an area efficient BIST," in *IEEE Int. Test Conf.*, 1995, pp. 515 – 523.

[12] P. Girard, L. Guiller, C. Landrault, and H.-J. Pravossoudovitch, S. Wunderlich, "A Modified Clock Scheme for a Low Power BIST Test Pattern Generator," in *19st IEEE VLSI Test Symp.*, 2001, pp. 306–311.

[13] Y. J. Huang, J. F. Li, J. J. Chen, D. M. Kwai, Y. F. Chou, and C. W. Wu, "A built-in self-test scheme for the post-bond test of TSVs in 3D ICs," in *29th IEEE VLSI Test Symp.*, 2011, pp. 20–25.

[14] F. Liang, L. Zhang, S. Lei, G. Zhang, K. Gao, and B. Liang, "Test patterns of multiple SIC vectors: Theory and application in BIST schemes," *IEEE Trans. on VLSI Syst.*, vol. 21, no. 4, pp. 614–623, 2013.

[15] R. David, P. Girard, C. Landrault, S. Pravossoudovitch, and A. Virazel, "Hardware generation of random single input change test sequences," *Journal of Electron. Testing: Theory and Applications*, vol. 18, pp. 145–157, 2002.

[16] W. Yi, F. Xing-hua, and W. Dai-qiang, "An Implementation of Random Single Input Change Technique for Low-Power Test," in *2nd Int. Conf. on Anti-counterfeiting, Security and Identification*, 2008, pp. 352–355.

[17] J. Provost, J.-M. Roussel, and J.-M. Faure, "Technical report on Conformance Test of Programmable Logic Controllers – Execution of Minimum-Length Test Sequences," LURPA, ENS Cachan, France, Cachan, Tech. Rep., 2014.

[18] A. Guignard and J.-M. Faure, "A Conformance Relation for Model-Based Testing of PLC," in *12th Int. Workshop on Discrete Event Systems*, Cachan, 2014, pp. 412–419.

[19] K. Ogata, *Modern Control Engineering*. Prentice Hall, 2012.

[20] IEC 60848, *GRAFCET specification language for sequential function charts*, 3rd ed. International Electrotechnical Commission, 2011.

[21] R. Floyd, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.

[22] F. Basile, P. Chiacchio, and D. Gerbasio, "On the Implementation of Industrial Automation Systems Based on PLC," *Automation Science and Engineering, IEEE Trans. on*, vol. 10, no. 4, pp. 990–1003, 2013.