

Testing the Control Part of Peripheral Interfaces

S. Zielski, J. Sosnowski

Institute of Computer Science, Warsaw University of Technology

ul. Nowowiejska 15/19, Warsaw 00-665, Poland

Email: jss@ii.pw.edu.pl

Abstract

Testing peripheral interfaces of computers is the challenging problem in the context of their complexity and multi layer structure. This paper presents our experience with a new systematic approach to generate tests for interfaces and check testability features. It was verified for real interfaces.

1. Introduction

Various interconnection networks are used in computers to assure communication between internal resources (CPUs, memories, I/O adapters, etc.) with the peripheral devices and external environment. They are optimised for speed, communication scenarios, and traffic profiles. In this process various parameters are considered: number of device ports, peak or average bandwidth, message latency or size, traffic pattern, reliability, availability, etc.

Many papers on interfaces deal with verification and conformance testing e.g. [4-7] and [9], most of them use formal models or coverage measures. The goal of verification is checking the satisfaction of required properties defined in specifications. The conformance testing is the assessment of an implementation with the specification. Due to many components in interfaces an important issue is to test their co-operation (control part of the protocol). The crucial point is finding representative test cases based on voluminous and descriptive specification of real protocols (e.g. [2], [3], [11] and references), which are neglected in the literature.

This paper presents an experience of generating test cases for computer peripheral interfaces. We have adapted and extended conformance testing techniques [4], [7] to the specificity of the considered interfaces. This approach is illustrated with results for SCSI and IEEE 1394 interfaces [2], [11]. For these interfaces we have developed formal specifications and analysed their

testability features. Section 2 gives an outline of modelling interfaces. Section 3 describes the concept of the developed test generator and section 4 presents a new approach based on inter-layer testing.

2. Interface Modelling

Interfaces can be modelled using finite state machines (FSMs), state charts, and ISO standards: Lotos, Estelle or other formalisms [1], [5], [9], [10-12]. Complex interfaces (this is the case of peripheral interfaces), can be decomposed horizontally and vertically. Horizontal decomposition relates to different layers of interface protocols e.g. physical, data link, transactional and application layers (e.g. consistent with OSI 7 layers standard [11]). Vertical decomposition relates to subfunctions, options or versions of implementing specific protocol layers. All interface services and functions can be specified at different levels: physical, structural, functional or abstract [10].

Interfaces can be implemented in hardware and software (middleware, firmware) and located in different system parts e.g. peripheral adapter or device, operating system (device drivers), applications, communication nodes (e.g. hubs in USB), etc. The used circuitry can be quite complex involving CPUs, RAMs, FIFOs, DMA or interrupt controllers, specialised CRC coders/decoders arranged in embedded systems, etc. Hence, detailed modelling can be done in a hierarchical way, which has an impact on generating tests: tests for specific circuits or programs and higher level tests to check co-operation of these elements, etc.

SDL (Specification and Description Language) is a standard for the specification and description of the telecommunication and real time systems [1], [12]. SDL models of various interface functions are good bases for generating test cases. We have developed complete specification in SDL for SCSI and IEEE 1394 interfaces, and found that SDL is sufficiently accurate

tool for this purpose. However it creates some problems in generating test cases due many branches related to parameters (e.g. transmission mode, data format). Hence, to simplify generating test cases, we have decided to map the SDL specification into FSM model and profit the experience of other authors generating conformance tests [4,5]. The algorithm outlined in [4] assumes complete FSM graphs i.e. for all states we should have transitions related to all possible inputs states. In the case of real complex protocols we have many possible inputs, however only a few of them have the impact in specified state. So direct use of the proposed idea in fact is unmanageable. To overcome this problem we have introduced a new algorithm. In the first phase it maps the initial FSM (obtained from SDL) into a complete FSM by adding trivial (artificial) transitions $s_i - s_i$ which are specially marked to distinguish them from the primary transitions.

The FSM models of the analysed interfaces are obtained from SDL specification diagrams according to the following rules:

1. Skip all the input and output parametric blocks from SDL and replace them as edges
2. Concatenate branches from decision blocks
3. SDL output signals are specified in FSM notation, in the case of no output signal in the transition add label NULL

All branches modified according to step 1 and 2 are additionally labelled with links to the related elements in the primary SDL models.

The control part of the communication protocols can be modelled as I/O FSMs. Each FSM can be defined in a classical manner as a tuple $\langle S, s_0, I, O, \delta, \tau \rangle$ with the set of finite states (S, s_0 – initial state), input (I) and output (O) states, output (δ) and transition (τ) functions.

This approach is used in conformance testing where an implementation model I/O FSM_I is checked against I/O FSM_S model related to protocol description or specification [5]. In protocol verification quite often the authors are interested in some general properties e.g. deadlock, livelock, reliability and safety. For this purpose various specifications are used: LTS (Label Transition System), CCS (Calculus of Communication System), CSP (Communication Sequential Processes), LOTOS, etc. [5, 9]. One of the verification techniques for specified properties is model checking. A safety property assures that some bad feature is always precluded e.g. that bad states can never be reached and bad actions never happen. Liveness property assures that some good feature is eventually fulfilled e.g. a good state or action.

The generated FSMs are relatively simple with low number of branches among states. For illustration, tab.1 gives some statistics for SCSI FSMs specifying physical

and protocol layers for the *initiator* and *target* devices: *InitPh*, *TarPh*, *InitPr* and *TarPr*, respectively. These statistics give the number of internal states $|S|$, inputs $|I|$, outputs $|O|$, number of states with branches $|B|$ (i.e. at least 2 transitions), number of loops $|L|$ and the number of parameter blocks $|BP|$ in SDL model. In many used algorithms these parameters characterise their calculation complexity. Also the path length between subsequent branch states is important. For the considered FSM we have got relatively low path lengths:

InitPh – 2.17, *InitPr* – 1.47, *TarPh* – 2.2, *TarPr* – 1.4.

Table 1. Complexity of SCSI interface

FSM	S	I	O	T	B	L	BP
InitPh	33	27	18	43	8	5	15
InitPr	21	20	15	25	6	5	14
TarPh	38	27	25	44	8	6	12
TarPr	21	20	18	24	6	5	14

The complexity of IEEE 1394 FSM, specifying different layers of the interface, was also in the similar range.

3. Generation of Test Cases

We have developed a program which generates test cases for FSM models of analysed protocols. The basic idea is related to the one described in [4]. However it was not sufficiently detailed moreover it assumed complete FSM graphs which resulted in practically unmanageable analysis. Hence, we have developed a special program based on an original algorithm, which has been successfully used for generating test cases for SCSI and IEEE1394 interfaces. The generated test cases are in the form of a set of input-output sequences covering all transitions in the FSM graphs. This gives the possibility of checking systematically the control part of the protocol.

To generate test cases for FSM we have to find all possible state transitions. For each transition, a test case can be described as

$$\text{RES } \textit{shortest path} (s_0 - s_j) T_{ij} \text{UIO}(s_j)$$

where RES is the initialisation sequence setting the tested system to the initial state s_0 , than we have a sequence moving the FSM to state s_j (with the *shortest path*), T_{ij} – is the transition sequence from state s_i to s_j and $\text{UIO}(s_j)$ is the input output sequence, which identifies uniquely state s_j

The most complex is finding distinguishing sequences (UIOs). For this purpose we have developed an original algorithm (based on [4]):

1. Create the root node of the state graph and attribute to it two sets of states P and T, initiated to the set S. Set the graph layer index i to 1.
2. For each input $in \in I$ create a temporary child node and attribute to it two sets: T and O. The set T comprises all the states, which are the next states in transitions from the states comprised in the set P of the parent node (i -th level). The set O comprises output states generated for the above mentioned transitions.
3. For each temporary node from step 2 generate $|O|$ child nodes, each correlated with the specified output states in the set O. For each node attribute P and T sets, which comprise all the initial and final states related to the analysed transitions in step 2, with the same generated output signal.
4. Repeat steps 2 and 3 each time incrementing the graph layer index i , do not reveal terminal nodes i.e. nodes which comprise single element P and T sets.
5. For each terminal node generate the distinguishing sequences defined by the sequences of input and output signals on the paths from the graph root node to the terminal nodes.

In step 2 we take into account only primary transitions of the FSM model. The introduced artificial transitions (to assure FSM completeness) are skipped.

The developed algorithm is effective and for SCSI initiator model (33 states, 27 input signals, 18 output signals) it found all sequences UIO in 5 sec (IBM PC, DURON processor, 1.4 GHz, 512 MB RAM). The algorithm based on [4] even after 10 hours did not generate these sequences.

The generated test sequences are transformed into TTCN notation [13]. This notation specifies correct test responses, also it is enhanced with timeouts, which specify maximal waiting time for output signals in subsequent states.

However there are some simplifications e.g. related to compressing branches with parameters during transformation of SDL graph into FSM. Nevertheless each transition based on such compressing is specified and it is possible to expend it in many transitions covering paths related to different parameters.

Having generated test cases we have found that they are relatively simple. In most cases, the distinguishing sequences UIOs were single input. Hence, the length of tests was also short. For IEEE 1394 on average they were in the range 2.5 – 4.1 (over all FSMs related to various protocol layers, minimal 2 and maximal 6).

It is worth noting that in practice the used interfaces are limited to some specific configurations, subsets of functions, etc. Hence, by generating tests we take into account these restrictions.

4. Interlayer Testing

In complex interfaces we have several layers of protocols. Hence arises the question of testing at higher protocol layers and finding test coverage for lower layers. For this purpose we have developed a special program which analysis the transitions on neighbouring layers. It finds the mapping between the models of the two layers, identifies equivalent states and transitions. We distinguish directly covered and indirectly covered transitions of the lower layer. For directly covered transitions we have 1-to-1 mapping. Not directly covered transitions of lower layers relate to transitions at higher layers, which cover a sequence of lower layer transitions (and states). As the result, we get a set of equivalent states and transitions as well as the set of transitions from the lower layer which have no direct equivalence.

We have developed an original algorithm which evaluates diagnosibility capability of the i -th layer from $i+1$ -th layer. The outline of this algorithm is as follows:

1. For each state of the i -th layer we check non empty transitions which correspond to higher layer. For this purpose we check the prefix of the input signal. If it is \$ (relates to an output signal from the higher layer) then the analysed transition corresponds to the beginning of the sequence of corresponding transitions in the higher $i+1$ -th layer.
2. We find the path in the lower layer composed of subsequent transitions leading to the transition with output signal marked by prefix %, which denotes sending signal (response) to the higher layer.
3. We identify the corresponding states in the i -th and $i+1$ -th layers, by selecting states with the same prefixes. These prefixes are attributed during the specification of the protocols.
4. For the selected transition of the higher layer we find the transition which corresponds to a transition in the lower layer, which initiates the sequence of transitions in this layer (the both transitions start in the corresponding states of both layers).
5. We find a path in the higher layer by joining subsequent transitions till the transition with the prefix % at the input signal (obtained from the lower layer).
6. The obtained sequence of transitions is included into a dynamic list, the checked transitions and states are deleted from the primary graph
7. Repeat 1-6 for the updated graph till finding all corresponding transitions.

The outcomes of this algorithm are pairs of corresponding states (i -th and $i+1$ -th layers) and for each of them sequences of corresponding transitions. Typically the length of the sequence of transitions at the higher layer is lower or equal to the corresponding one

from the lower layer. This reduces the diagnosability accuracy. The highest precision is assured for single transition sequence on both layers.

For the SCSI initiator we found 43 transitions of physical layer. Interlayer analysis with the presented algorithm gave the following distribution of corresponding transition sequences of the protocol and physical layers:

$$(1/2) - 3, (2/2) - 2, (1/3) - 2, (2/3) - 3$$

where (A/B) denotes a sequence of A transitions within the protocol layer corresponding to B transitions in the physical layer. Hence we have 25 transitions from the physical layer directly stimulated from the protocol layer (the diagnosability accuracy is decreased by one transition). These transitions check operations within the considered layers as well as the interaction between layers. All not correlated transitions (18) of the physical layer are local to this layer or involve co-operation of more system elements. This signifies that they are hard to be tested directly from the higher layer. Having found these relations for each pair of protocol layers we can calculate also diagnosability accuracy between any layers.

This analysis is useful in planning test strategies in the system. In particular we can identify transitions which can be covered from higher protocol layers and transitions which can be tested locally (without the need of other devices). In addition we can identify tests involving other devices. These tests can be good candidates for developing DFT and BIST mechanisms. Moreover, to provide appropriate test stimuli and avoid device unsafe operation (e.g. overwriting a file on a disc), we can introduce various supplementary or dummy operations for the testing purposes, which are accessed only in the test mode.

6. Conclusion

Our research proved that computer peripheral interfaces, usually specified in descriptive way in bulky documentation, can be easily mapped into SDL models. These formal models create a good base for generating test cases in a systematic way. We map SDL into FSM models to simplify the test generation algorithm and concentrate on the control parts of interfaces. The generated tests (with the proposed algorithms) give also a good view on interface testability features. The proposed interlayer analysis is helpful in managing test scenarios. In particular, we can decompose testing into tasks related to individual HW or SW components as well as checking their co-operation. Tests can be initiated from different layers of communication protocols, and cover different functionalities. Higher layer testing activates more functional blocks but lowers testability and diagnosability. In practice, it is

required to involve DFT and BIST mechanisms. A large part of these mechanisms can be implemented in software as externally called stubs emulating the behaviour of external devices. Artificial short-range loops can simplify testing of many functions beyond the external world. The next issue is the capability of performing dummy functions, which are ignored by external devices (e.g. to eliminate data destruction by write commands to disc) in the test mode. The effectiveness of generated tests can be referred also to various coverage measures (e.g. protocol code coverage [6], hardware circuitry stress coverage, and device commands coverage).

Further research is targeted at mapping the generated test cases to testing programs in real system environment (extension of [8] approach).

Acknowledgment This work was supported by Ministry of Science and Higher Education grant 4297/B/T02/2007/33.

7. References

- [1] F. Belina, D. Hogrefe, A. Sarma, *Sdl with Applications from Protocol Specification*, Prentice Hall 1991.
- [2] G. Field, P. Ridge, *The Book of SCSI for the New Millennium*, No Starch Press, 2000.
- [3] K. Grimsurd, H. Smith, *Serial ATA Storage Architecture and Applications*, Intel Press, 2007.
- [4] Gerard J. Holtzman, *Design and Validation of Computer Protocols*, Prentice Hall 1991
- [5] Jae-Dong Lee, et al., "Verification and Conformance Test Generation of Communication Protocol for Railway Signalling Systems", *Computer Standards and Interfaces*, 29, 2007, pp. 143-151.
- [6] F. Saglietti, et al., "Interface Coverage Criteria Supporting Model Based Integration Testing", *Proc. of 20th Int. Conference on Architecture, ARCS Workshop*, VDE Verlag GMBH, Berlin, 2007 pp.85-93.
- [7] B. Sarikaya, *Principles of Protocol Engineering and Conformance Testing*, Ellis Horwood 1993
- [8] J. Sosnowski, "Software Based Self-Testing of Microprocessors", *Journal of System Architecture*, 52, 2006, pp. 257-271.
- [9] A. Sung et al., "An Interface Test Model for Hardware-Dependant Software and Embedded OS API of the Embedded System", *Computer Standards and Interfaces*, 29, 2007, pp. 430-443.
- [10] D. Trawczynski, J. Sosnowski, J. Zalewski, "A Tool for Databus Safety Analysis Using Fault Injection", *SAFECOMP, Springer LNCS 4166*, 2006, pp. 261-274.
- [11] R. Zurawski (edit.), *Industrial Communication Technology*, CRC Press, 2005.
- [12] *Specification and Description Language (SDL)*, 2002. <http://www.itu.int/ITU-T/studygroups/com17/languages/Z100.pdf>,
- [13] <http://www.iec.org/online/tutorials/ttcn/>