Increasing Software Testability with Standard Access and Control Interfaces

Allen P. Nikora, Raphael R. Some
Jet Propulsion Laboratory,
California Institute of Technology
Pasadena, CA 91109-8099
anikora{rsome}@jpl.nasa.gov

Yuval Tamir
Concurrent Systems Laboratory
Computer Science Department
UCLA, Los Angeles, CA 90095-1596
tamir@cs.ucla.edu

1. Introduction

Testing is the most common method of determining whether a software system satisfies its requirements. Traditionally, testing starts with the detailed examination of individual functions or methods, progresses through the integration of functions or methods into subsystems, and ends with testing the functionality and behavior of the completely integrated system. At each stage of testing, the amount of functionality and behavior of the artifact being tested is increasingly limited. One reason for this is that it becomes impossible to test all paths through the system within a reasonable amount of time. However, another reason for this progressive decrease of test coverage has to do with increasingly limited control of and visibility into the state of the artifact being tested. During unit test, it is rather simple to control the inputs of individual functions or methods or view their internal state – modern development environments provide adequate facilities for doing so. However, these facilities do not scale up to the testing of partially or completely integrated systems. Control of and visibility into the system's state is then limited to the input and output facilities provided by the software itself as well as the hardware on which the software is hosted during the test. These facilities are usually insufficient to precisely control the state of individual components or sets of components of the system; they are also inadequate to the task of displaying on demand the state of specific components. We describe an approach to improving the testability of complex software systems with software constructs modeled after the hardware JTAG bus, used to provide visibility and controllability in testing digital circuits.

2. Controlling and Examining Software State

In the following paragraphs we outline an approach to implementing a Software JTAG bus in which we directly map the hardware constructs of the JTAG specification [IEEE01] to software equivalent structures. We then briefly explore some of the issues associated with this approach. It should be noted, however, that this is not the only implementation approach that can be considered. A one-to-one mapping between the hardware and software versions is not necessarily optimal. However, what is necessary from any approach is that:

1. All relevant application software modules must be organized with their control structured as a localized state machine rather than being distributed throughout the module. Our experience is that in many cases such a transformation is relatively straightforward, as the control logic does not change, just its structural implementation. Note that not all modules need be organized in this way. Low-level modules with minimum control structure and minimum complexity may not require a software JTAG bus interface. Further, note that the level of abstraction encoded in the state machine is a design choice that must be made by the software engineer. The level of abstraction represented by the software "state" can be at the bit level, the mode level, or anywhere in between, and is determined by the degree of visibility and control judged to be necessary or useful for test and validation of the system. The level of abstraction defines the granularity of state definition and the resulting visibility and control available through the software JTAG interface. Too high a level of granularity will not allow sufficient visibility and control for proper testing. Too low a level of granularity will result in excessive complexity and an explosion of state space. An appropriate level of granularity provides a practical level of testability, i.e., one allowing full testing of the system with sufficient visibility to localize errors to a region, but not necessarily to pinpoint the faulty instruction(s).
2. Data must be organized as well-defined data structures. For the most part this is not an issue and will require minimal changes to the code
3. The software JTAG interface must provide at least the following functions: state machine - read, write, halt, start; data – read, write; Module I/O (if present) read, write, halt, start; module /system synchronize; system – halt, start, synchronize. Beyond these basic functions, other built in test functions may be provided, such as shadowing of test variables. For initial software test and

Copyright 2003 Chillarege Press

validation, these built in tests are not required as this code may be incorporated into an external test program. They may, however, be useful in testing a deployed system either for regression purposes or for ongoing system test and validation activities.

4. A standardized middleware and driver level segment of the software JTAG facility must be defined to provide overall system level control and visibility and to provide accessibility to external test programs and/or operator. Definition of this component is key to software JTAG operation, as it will define the mechanisms by which the software JTAG interacts with the OS as well as timing, synchronization mechanisms, and granularity.

The diagram in Figure 1 below shows the way in which JTAG-like constructs might fit into a software system, in this case the controlling modules of a hypothetical fault protection component for a space mission flight control software system. Within each module, the **SW JTAG Interface** corresponds to the JTAG Test Access Port and the Test Access Port Controller, the **Test Data Structures** correspond to JTAG test data registers, and the instructions are labeled on the arcs within each module. The **state machine** represents the module's functionality and behavior.

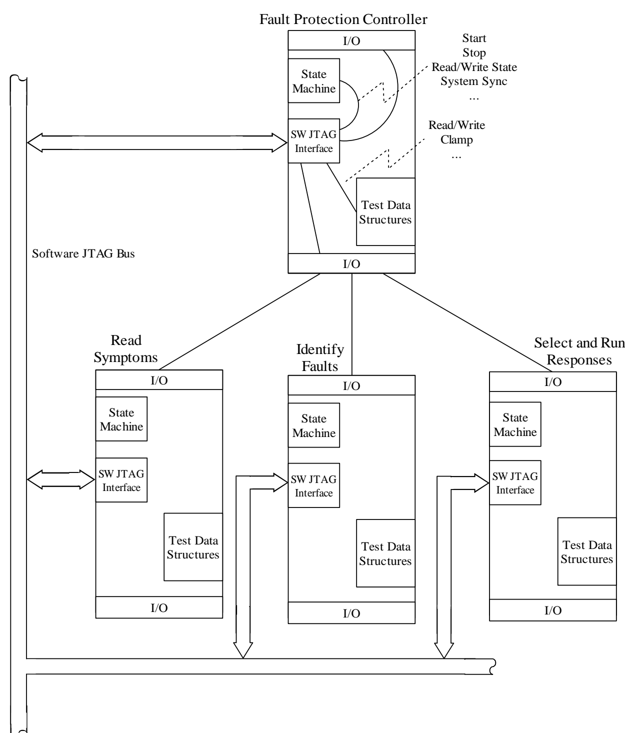

Figure 1 – Software JTAG Conceptual Diagram

Numerous issues must be addressed in developing these constructs, including:

- The additional space required by the constructs could substantially increase the size of the flight system. Our experience is that memory is always a scarce resource for space mission software; the constructs must be specified and implemented to minimize memory usage.
- The addition of constructs such as shadow variables may affect system performance. The time to write to a primary variable will effectively be doubled. Guidelines for specifying shadow variables so as to minimize the impact on system performance must be developed.
- Scalability is a significant issue in terms of the amount of test data to be monitored. For large systems, the amount of test data to be monitored could easily be large enough to make these techniques unusable. It will be necessary to develop techniques to handle the sheer volume of test data that will be generated.
- Determining the appropriate level of abstraction at which to specify and implement the constructs will be a significant issue for large, complex systems. Assuming that these types of systems can be represented as collections of communicating finite state machines, guidelines regarding the level of abstraction appropriate for different levels of a hierarchy must be established. For example, guidelines establishing the appropriate number of states to be specified and implemented in a module must be established, and the amount and type of test data appropriate to different types of modules must also be specified.
- The system must behave the same way during operation as during test. For example, timing must be considered for real-time systems – the ordering of events and the satisfaction of deadlines must be the same during test as during mission operations. This becomes especially important as we move from single-processor single-threaded systems to multi-processor, multi-threaded systems.

Acknowledgments

The work described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[IEEE01] "IEEE Standard Test Access Port and Boundary-Scan Architecture", IEEE Std 1149.1-2001, Institute of Electrical and Electronic Engineers, 2001.

Copyright 2003 Chillarege Press