Test Vectors Considered Harmful

Gordon D Robinson
Fremont, CA
g.robinson@ieee.org

Introduction

The concept of a test vector has been a central feature in digital test hardware and software since the earliest digital test systems of almost forty years ago. It has always been a tricky concept to explain and defend, and newer devices and their test requirements are brutally exposing the weaknesses in the test vector concept.

This paper, in the tradition of other "considered harmful" papers in engineering[1,2], identifies and justifies problems, and suggests alternatives without endorsing a single alternative.

The Test Vector Concept

A test system based on vectors has several defining characteristics:

- There is a "vector sequencer" that defines which "vector" will be obeyed next.
- The vector sequencer broadcasts a small amount of information (such as vector address, period numbers and timeset numbers) to all of the pin channels.
- The pin channels access a small pin command, usually of between 1 and 8 bits.
- The pin command, timeset and period define the set of drive and strobe events that the channel will perform.
- There are limited facilities for the DUT to change the sequence of vectors. The pipelining involved in a reasonable performance test system means that pass or fail on a vector cannot affect the next *n* vectors to be obeyed, where *n* can be anywhere from 1 to 20 or more.

The test vector concept can be seen clearly in the STIL language, where the WaveformTable to be used corresponds directly to a tester's global timeset, and the WaveformCharacter corresponds to the pin command from vector memory.

Some test systems have more than one vector sequencer, usually based on a convenient physical partition that exists in that system. As vector sequencers have moved from a mainframe into the pin channel card, a reasonably large number of vector sequencers are potentially available.

What's Wrong With Test Vectors?

Ignoring for now the systems with multiple vector sequencers, and the limited capability for conditional vector execution, we find that at the moment that the tester's software starts a pattern burst, all of the drive events and all of the strobe events that will occur are predetermined.

The predetermined nature of event sequences, and the independence of one channel's events from any others, has encouraged the tester design style where each channel is independent of the others, even though they all obey the same sequence of vectors.

Unfortunately, modern source synchronous interfaces need more flexibility than that in their strobe timing. It is timing signals from the DUT that specify when some important times are for strobes to take place on other channels [3].

The test vector concept fits very well into a world where devices have only a single important interface whose timing can define the vector boundaries. Modern devices, particularly SOC devices, have numerous interfaces, and each interface has timing independent of the others. As an example a South Bridge from a modern motherboard is likely to have one interface to communicate with the North Bridge, and also have PCI, Ethernet, two IDE interfaces, two serial ATA interfaces, four to eight USB 2 interfaces, Ethernet and more. There is no global division of the activity in multiple interfaces that allows all of them to fit naturally into a vector system's constraints.

Now there are tests that do fit the vector style all the way from their creation to deployment. In particular the tests for using scan mechanisms to check for the standard stuck-at and delay faults completely ignore the device's natural operational modes, and use just the testability timing rhythms. If these were the only tests that ever needed to be run, we'd have no need to examine alternatives to vectors.

As a further example of flaws in the vector concept, a paper on IP for embedded diagnosis[4] uses the idea of a "streaming diagnosis port" to take failure information from embedded BIST engines and transmit it over a reasonable speed interface to the test equipment. Most ATE vector systems have great difficulty handling that style of interface. The "good device" behaves very simply and predictably (nothing happens on the streaming interface), but "bad devices" mean that the tester needs to be able to react to messages on that interface that can start at any time.

Some Alternatives to Vector Systems

The obvious places to look for alternatives to the vector systems are the worlds of design verification (simulation) and of instrumentation.

The verification world never seems to mention vectors in any form that a test system user would recognize. Effective testbenches for device verification are usually based on transactions on the various interfaces, and the simulation models have independent generators and monitors operating on each of those interfaces [7].

The testbenches will permit considerable variation in the actual sequences of activity provided that the activity does not violate what the testbench is actively checking. For example, there could be an expected transaction on an interface that will occur after a flexible number of idle transactions.

When the simulation is run our “design to test” tools will then ignore all of those helpful, user-meaningful high-level constructs in the verification testbench and reduce the full simulation to a set of events to be converted into vectors by “cyclization” tools.

Event-Based Testers

Our first alternative to vector systems are the testers that obey arbitrary sets of events on each channel[5,6]. These trade off the use of large amounts of memory and eliminate the need to cyclize the simulation results. Unfortunately the event-based testers don’t address issues of device-controlled timing or unpredictable transaction sequences and timing.

Natural Environment Testers

A natural environment tester places the DUT in a natural target environment for testing. A simple example is to place a microprocessor on a motherboard and allow the memory and chipsets on that motherboard to operate in a natural mode. The “enthusiast” PC motherboards have numerous settings that can tweak clock rates and supply voltages.

Natural environment test systems do have a fundamental limitation. Each component that surrounds the DUT is likely to use only a subset of the possible transactions across the interface between the DUT and itself. So the DUT’s ability to handle transactions not used by the environment is never tried at all, and cannot be in that style of testing.

The natural environment components cannot always (or usually) produce incorrect transactions on an interface, and may report protocol errors in a fairly cumbersome manner.

Natural environments can also be very tolerant of faulty devices. For example a processor with an inoperable instruction cache can still obey programs and produce the correct answers when connected to real memory.

Some recent high-speed interfaces have calibration features where they adjust some settings either in a startup phase or on request during operation. A DFT facility that allows the settings to be forced to specific values produces a natural environment that can stress the other component, or can check that the other component is operating in a particular area of the specification.

Instrumentation on Interfaces

Each interface that is used is likely to have some specific instrumentation available that can create predetermined transaction sequences and can check the transactions across an interface using the concepts of that interface. If these instruments can be programmed in a uniform manner, and can communicate enough about when transactions should be started, and when they should have taken place, then the instruments can get the benefits of natural environment testing without some of the constraints.

This style of testing raises the abstraction level

Synthesized Interface Instrumentation

Today's universal design medium is the FPGA. Modern FPGAs are capable of handling many of the most aggressive interfaces used in designs. A library of FPGA interface instruments that can be configured into a device-specific test instrument may solve all of the problems of vector-based systems, without our having to have different physical testers for each collection of interfaces on a device.

Our earlier example of the streaming diagnostics port is simple for such a system. The diagnostics port has an instrument on its signals that listens for the start of a message, and then accepts and stores that message along with any timestamps to identify when the message arose. This streaming interface instrument operates independently and asynchronously from the rest of the instruments.

An interface instrument will not usually be a direct use of the normal IP core for the interface that is being accessed. This is a test system and its instrumentation will usually need to stress some details on the interface, and not just use simple settings at all times.

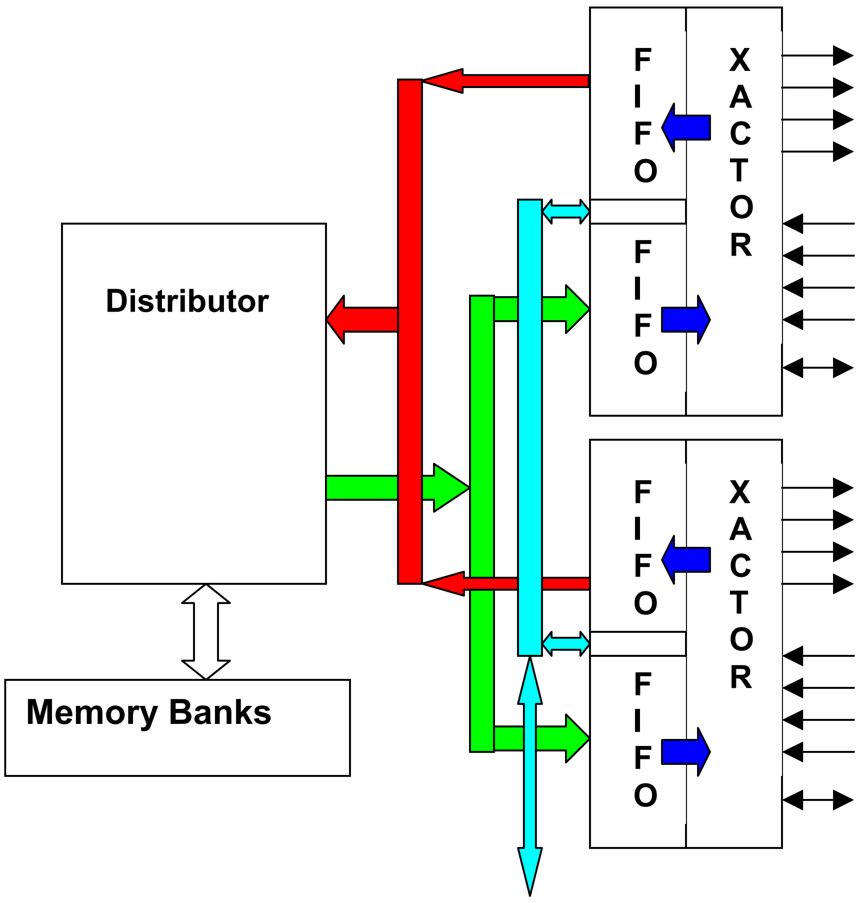

Fig 1: A Transaction-Based Architecture

Figure 1 shows a general idea for the architecture of a tester based on these ideas. Each target interface has a transactor that can handle the repertoire of that interface. A memory is used to provide the stream of requests to each transactor, with a FIFO to hold a small number of requests and decouple the memory interface from the transactions. When the transactors detect problems, they can log data about them via another FIFO.

Examples of the features that interface transactors will need to provide include:

- Starting a transaction when instructed by other instruments
- Monitoring an interface and ignoring some transactions, looking for an expected transaction
- Generating protocol errors in data or timing
- Provide data-controlled or algorithmic transaction sequences
- Signaling important events for other instruments or transactors
- Logging erroneous or unexpected transactions

The amount of memory needed for a transaction is significantly less than the amount of memory a conventional vector-based instrument uses, because much of the detail of handling a transaction is encoded into the transactor itself, rather than being data on every cycle.

Modified Vector Systems as Interface Instrumentation

When the granularity of a vector sequencer is a single card, we can consider using a card as an interface instrument. The pipeline for reacting to the devices can be dramatically shorter when only one card needs to react. It is not unreasonable to consider two or more pipeline lengths for such response, depending on whether a single card, a section of the several nearby cards or the whole test system is involved in the decisions.

Acceptable Non-Determinism

Once we move on from the concept that a digital test instrument performs a fully deterministic sequence of events, we need to decide how much variation is acceptable.

The simplest form of relaxing determinism is to expect a deterministic sequence of transactions on an interface, but permit flexibility about when a transaction starts.

A further flexibility would allow an interface to have any number of idle transactions between the ones being monitored.

An interface could even have several transactions being anticipated, and would accept any of them, retiring those that occur from the collection that are expected.

At an extreme we move away from the idea of actually checking transactions as they occur, and instead look for results at the end of a major sequence.

Software and Tools

Any of the mechanisms for moving from the current vector-based concepts to using natural transactions on interfaces will need significant changes in the test software and the flow of tools. Fortunately some of the needed tools are already in place. Modern waveform viewing tools allow interface transactions to be viewed in terms of transactions and their parameters, not just as waveforms.

Maybe getting rid of test vectors is a necessary step to building good Design and Test links.

References

[1] Dijkstra, E.W. Go To Statement Considered Harmful. CACM Vol 11, No. 3, March 1968. Also available at http://www.acm.org/classics/oct95/
[2] Google search for "Considered Harmful" will turn up hundreds of such articles, including [1].
[3] Cheng, J. When Zero Picoseconds Edge Placement Accuracy is Not Enough. Proceeedings ITC 2001.
[4] Pateras, S. IP for Embedded Diagnosis, IEEE Design and Test, May 2002.
[5] Lesmeister, G. A Tester for Design. ITC '95.
[6] Katz, J and Rajsuman, R, A New Paradigm in Test for the Next Millennium, ITC 2000.
[7] Bergeron, J. Writing Testbenches, Kluwer.