



The Electrical Testing Can o' Worms

The MicroSS Components Guide to Electrical Test program construction

Covering the why's and wherefore's of the individual electrical tests and their associated parameters ...

Electrical and Functional Test Program Construction

Electrical and Functional Test Program Construction.....	1
Why do we test components?	3
Semiconductor Test Programs.....	5
Continuity.....	6
Supply current	6
Input leakage and related characteristics.....	6
Functional testing, gross.	7
Output drive and related characteristics.....	7
Output leakage.....	8
Specification VIL/VIH threshold tests at supply limits.....	8
Specification speed tests at supply limits.....	9
Overall.....	9
Test Vector Format	11
a). Test Vector File	11
Sample Test vector file :	12
b). Test Vector Information File.....	14
A Sample file	16

Why do we test components?

We test manufactured components to address different issues. Firstly, components are tested to ensure that they've been manufactured correctly, and that there is nothing wrong with the manufacturing process. Secondly, the components are tested to ensure their compliance with the procurement specification or manufacturers data-sheet. Thirdly, the components are tested to ensure that they will function within the desired application. The first two reasons are normally addressed by the component manufacturer, and he supplies the components with a warranty that matches his data-sheet. The latter reason, of matching a customer's application is generally only done either by the customer himself, a contract test-house acting on behalf of the customer, or by the manufacturer under some specific contractual agreement.

Tests performed by the component manufacturer may well be uniquely tailored to his process, and not all of the possible combinations of operation or function may be covered. This was often the case for memory device manufacturers, as after prolonged investigation, they knew exactly where their component weakness was. Additional tests, often undocumented, may be performed by the component manufacturer, in order to achieve a specific delivered PPM (Parts Per Million). A manufacturer may elect to only perform a subset of the available electrical tests for economic reasons, opting to perform a subsequent rigorous test on a batch by batch sample basis. Some parameters may never be tested, offered as "guaranteed by design", but however the component manufacturer decides to test, it is his warranty that determines the components desirability, suitability and viability.

Whilst DC and AC parameters may be well catered for in a detailed specification, the specification of IC interconnection and functionality is generally inadequately covered in a data-sheet or data-book, the implicit functionality is really only determined by the component test program. A semiconductor manufacturer would normally use test vectors for determining device operation and functionality that are automatically generated from the design CAD simulations. In general, these would often be just a sub-set of the whole simulation data suite, as the complete simulation data set may be both uneconomic and swamp the component tester.

Consider a simple micro-controller: if it has "N" micro-code instructions with a "X" bit data-bus, then the first test approach might be to perform $2^X * N$ tests, covering each instruction. This may not cover sufficient combinations of micro-codes, so we might have to perform a second micro-code operation after the first, leading to $2^X * N^M$, where M is the number of subsequent micro-code operations we require to feel "comfortable" with ... and that's just it! At what point do you stop and draw the line, declaring that the device has been "sufficiently" tested. If this micro-controller was destined for a "tamagochi" production, such as a birthday card "whistler", we might take one view, but if this micro-controller was to be used in a pace-maker or an aircraft navigation system, we're bound to have a differing view on the level of functional testing needed.

Test vectors used by the manufacturer would determine internal function and connectivity, but, unless the part had facile functionality, would be unlikely to test the full application or operation. Additional in-built functionality is often employed to aid testing, such as BIST, BILBO, internal LS scan-

chain paths etc., but details of these test features are seldom made public.

Another classic but simple example of in-built test feature requirement is that of long-chain counters. Simply consider a consumer watch chip. In application, the first function would be to check that the chip counts from 00:00 hours through to 23:59 without missing out a single minute. As the chip was design to run at 32KHz, and assuming that we can run the chip at 1000 times it's normal running speed (this would be highly unlikely, given standard watch-chip technology), the test time would still amount to almost 90 seconds to complete that simple test, clearly given the expected cost of the chip would make such a test unviable. Therefore the long-chain counters are broken down into shorter counter sections, and each one becomes testable and/or pre-loadable, such that the test requirements may only involve setting the time to 23:59 and checking that the counters roll-over to 00:00. Information relating to such test modes may not be made available to any third party ... this is proprietary information unique to that manufacturer and his design methodology.

Other circuit elements and techniques can also cause a problem with testing, such as power-on-reset circuits, phase-locked loops, or multiple redundancy / voting circuitry, where although the end function may be observed, the actual internal function cannot be guaranteed by testing.

One of the keys here is the need for design-for-test, a vast subject in its own right, and probably the subject of another informative booklet from Micross Components!

Any third party generating test vectors for a complex semiconductor part would have to tackle the part

from the application viewpoint, often making the program writing and generation an onerous and expensive task. It certainly helps if the right tools are emplaced, and Micross Components Ltd., have long experience in writing test plans and programs for a whole variety of complex digital, analog and mixed-signal semiconductor devices.

Semiconductor Test Programs

Electrical and functional test programs written for testing semiconductor components on Automatic Test Equipment (ATE) generally follow the same format, sequence and construction from device to device, even though the native tester languages vary from machine to machine and each device's functionality is different. The format and sequence described here has been determined at Micross Components, over the past 25 years of semiconductor component testing, as being the most effective; building up the tests predominantly in the ascending order of likelihood of device failure. The format most generally used at Micross Components for digital and mixed-signal test programs follows the sequence described below :

- Continuity test
- Supply current test
- Input leakage test
- Gross functional test
- Output drive tests
- Output leakage tests
- Specification functional VIL/VIH threshold tests at supply limits
- Specification functional speed tests at supply limits

This list is not exhaustive, and really only covers the major digital device-test headings, so taking them in order, we can expand upon the why's and wherefore's of each test. Much of the data and comments made are, by necessity, of a general nature, and cannot cover all tests for all devices.

Continuity

The general test is a pin-by-pin test that, with the supply and ground pins grounded, forces a small current, in the range of 50uA, into and/or out of each I/O pin. Here we're looking for the parasitic clamp or protection diodes, so we're generally expecting a voltage drop of around 0.6V ~ 0.7V. Some pins that are mainly resistive may require different test conditions and limits, and in general CMOS is tested for both the bottom and top clamp diodes (to V_{SS} and V_{DD}) whereas TTL/bipolar technologies usually only have the V_{EE} (Ground) clamp diode present.

This test is to ensure that correct contact is made to each pin that the ATE is expected to monitor and control, that the device is orientated in the socket correctly (or right way round for the probe card), and that the appropriate diode characteristics are observed. The limits are normally set quite wide, say from 0.3V to 0.9V, to accommodate differences in input and output pins, and it is worthwhile noting that this parasitic diode exhibits the standard $-2\text{mV}/^\circ\text{C}$ temperature characteristic, so the resultant datalog may be used to give an approximate indication of the device's temperature during the test.

Supply current

Once the continuity test has passed, the next stage is to power up the device in some meaningful manner and check the supply current drawn. This is an important test, as it serves to protect both the ATE tester, device under test (DUT) and the socket from damage caused by a bad or misaligned device.

Power supply current trips may also be employed to the same end.

Further measurements of the supply current may be made with greater accuracy, and under specified functional conditions, such as device reset, or the device operating at known speeds. In general the supply current is measured at the maximum supply voltage, as this is usually the condition of maximum supply current draw.

Input leakage and related characteristics.

Before we commence performing functional tests, it is common to check the input DC characteristics, generally that of leakage, under worse case conditions. This again is to ensure that we can, at least, drive input stimuli into the device inputs without sustaining tester or device damage.

Again at maximum supply voltage, the inputs in turn are driven to both V_{ILmin} (the lowest input voltage, normally V_{SS} or ground) and V_{IHmax} (the highest input voltage, normally V_{DDmax}) and the leakage current drawn is measured. It is normal to put all other inputs into an opposing state during this measurement, if possible. The limits may vary dependant upon the input type, and whether pull-up or pull-down resistors have been employed in the design. Additional leakage tests at other input voltages may be used to determine specific characteristics. It is important to remember that when attempting to measure and resolve small currents, the ATE's measurement unit will require a specific settling time. To speed up this test, it is sometimes feasible to gang multiple alike inputs together and to

measure the total current, but this is certainly not common practice at Micross Components.

Many specification limits for input leakage on CMOS technology device are, in fact, quite gross, typically $\pm 10\mu\text{A}$, whereas in reality a good part may well exhibit leakage well below 10nA , so pins exhibiting higher than average leakage currents, even though within the specification limits, can be suspected of having minor ESD damage. Caution should be exercised in automatically assuming ESD damage, the general characteristics of each design and batch should be carefully examined before judgement is made, but it remains a valid alert to potential ESD problems.

At Micross Components it is common for us to use dual limits for our own benefit, the lower limit set somewhere between 500nA and $1\mu\text{A}$ to monitor and observe this feature, and the upper limit to the given much higher specification value. It should be noted that our internal lower limit is very design dependant, so a detailed knowledge of the design and technology of the device is important.

TTL devices often specify additional tests, such as the input clamp voltage, V_{IC} , and input gross current V_{IH} , which are also tested at this point.

Functional testing, gross.

Once the inputs are known to be good, and the device power consumption is in the right range, the next stage is to check that the device functions as expected. Gross input levels and crude timing values are used to exercise a functional pattern, often consisting of upto 1 million individual test vectors or more. The main object here is to determine that the part is fully functional, and the outputs correctly

respond to the input stimuli as predicted; this will be needed to be able to drive the outputs into various states or conditions for the further tests.

In general, this test is often referred to as the "wobble" test, and is exercised only at nominal supply voltages and drive/sense conditions, with test rates somewhat below the maximum speed of the device. The test rates employed are often around the 1MHz mark, although for test throughput the faster the test rate, the better. As already stated, we are really only concerned with the devices' functionality, whether it can then perform to tight specification limits is for the later tests to determine.

Output drive and related characteristics.

Once the outputs have been proved to operate correctly, DC parametric tests can now be used to accurately measure and check the output pin drive levels, normally when loaded to their fullest extent. This involves running one or more functional patterns to achieve the desired output condition, then forcing a voltage or current, and measuring the resultant current or voltage, as appropriate, on each output pin in turn. Some devices, often those that are dynamic in nature, cannot be controlled to give a stable output measurement, in which event dynamic measurements, rather than static DC measurement techniques, will need to be used. These tests are normally performed under worst conditions, i.e. lowest operating supply voltage.

Normally, these V_{OL} and V_{OH} values are measured against the V_{OLmax} and V_{OHmin} voltage limits using I_{OLmax} and I_{OHmax} current values (note that I_{OHmax} is given as a negative current), generally whilst using V_{ILmax} and V_{IHmin} levels as the input voltage levels

(these again are the worst case voltages for which the device is specified to function)

TTL and some CMOS parts specify additional drive capability tests, such as output short circuit current tests, where the output, whilst driving high, is deliberately driven to an opposing high-current state, and the maximum drawn current is measured. This particular test has its roots in node-forced board testing, and now is generally not found in modern specifications.

Output leakage

Often referred to as tristate leakage, the device is again exercised using one or more functional patterns to achieve putting the output in a tristate or high-impedance state and thence measure the output pins leakage. Tests similar to those used for input leakage tests are used to determine the output tristate leakage tests, and again these tests are usually performed at the highest operating supply voltage, to enhance or uncover leakage paths. Open drain/source type outputs naturally require both the appropriate drive and high-impedance leakage tests.

Specification VIL/VIH threshold tests at supply limits

Having completed all the static DC tests on the output characteristics, the input DC functional characteristics, which generally cannot be measured, are then tested for. These include such parameters as V_{IHmin} and V_{ILmax} , and any Schmidt trigger

characteristics. These are tested again by the use of functional patterns, and the V_{IHmin} and V_{ILmax} values are used simultaneously as the input levels.

The tests are generally run three times, at nominal, maximum and minimum supply voltages to ensure specification conformance. If numeric values are required for the input levels, then a voltage ramp or binary-chop technique is employed on the grouped VIL and VIH levels in turn, determining the worst V_{IL} and V_{IH} values for the whole device. As these particular methods are notoriously slow (changing the V_{IX} value requires some settling time), they are generally only used during detailed device characterization, and then either on a grouped or pin-by-pin basis. Unfortunately, this is also the only method of extracting Schmidt trigger threshold or hysteresis values, so sometimes this test methodology has to be employed during production testing.

Again where some output pins cannot be sufficiently controlled to permit accurate DC parameter measurement, the binary-chop type search techniques may once again be used to determine the output V_{OL} and V_{OL} levels when driven into specific hardware loads.

Specification speed tests at supply limits

Often referred to as the dynamic tests, these tend to be the most complex of all the digital tests. Again the functional test patterns are used, and sometimes custom test patterns are created to uncover specific critical path timing. In general, the input drive levels and output sense levels are relaxed, values of 0.0V and 3.0V for the V_{IL}/V_{IH} drive levels and 1.5V for the V_{OL}/V_{OH} sense levels into specified loads are common for many 5.0V supply logic parts. The timing set, consisting of a number of timing generators and a timing period generator, is usually set to run as fast as the device will allow, although the majority of ATE testers cannot match the F_{MAX} parameter value of many modern parts.

Typically the dynamic tests are set-up to sense and determine the propagation delay from input to output by accurate placement of input control signals and accurate positioning of monitor strobes on the outputs. The majority of larger ATE equipment has a vast array of possible timing formats, such as RZ/NRZ/RTO/XOR, and edge placement accuracy can, as the result of good auto-calibration routines, be sub-nanosecond. When reading and comparing ATE dynamic specifications, repeatable timing accuracy at the device under test is significantly more important than the more often-quoted timing resolution.

These dynamic functional tests are performed at nominal, maximum and minimum supply voltage levels, and the test result in production is normally a go/no-go result. Where timing values are required, either as specific measurements or for device characterization, timing binary-chop or time-sweep techniques can be used to extract accurate timing

values from the test vectors. Some testers specifically have a timing measurement unit (TMU) whose sole task is to extract, measure and compare these timing values, whereas other testers rely on fast execution of a binary chop or sweep algorithm to produce timing measurements.

One problem that commonly occurs in timing measurements is any measurement that involves tristate timing, such as T_{PXZ} or T_{PZX} , as these measurements are naturally heavily dependant upon external resistive and capacitive loading, so can give highly misleading values.

Overall

Whilst the above details give a fair representation of the tests involved with a simple digital part, mixed-signal and analog test programs also generally follow a similar structure, building up the test complexity during successive tests, but intermixed with the set-up and interconnection of analog resources, such as frequency synthesizers and digitizers. DC measurement techniques can be very similar, although other issues, such as loop closing, frequency measurement, noise floor and overall measurement accuracy can play a more significant role.

At Micross Components, when writing test programs, we try not to “re-invent the wheel”, but usually base any new test program on one of our tried and proven kernel programs and specialist software routines. We also use a range of in-house development and generation software tools, such as PalSolve™, which can automatically produce the test program including test vectors, from, for example, a JEDEC fuse map or

EPROM Hex code. Nonetheless, all test programs require to be fully debugged, and so we tend to build in a certain level of datalogging and characterization features into every test program. This is to assist the engineer during initial debug, and then later to aid in the acceptance sign-off and QA engineering checks, required for every test program that is to be used for testing customer product. In practice, doing this also helps us in explaining to customers various details of particular tests, and again, during production testing, these features are often used to quickly determine and resolve device product issues.

Many test programs require to be run at more than one temperature, so these programs require internal options to select the test temperature, and where required, adjust specific parameters according to that temperature. Test programs used for life testing, for high reliability qualification, or for space delta measurements again have an increased logistic complexity to ensure accurate, repeatable and reliable operation during production tests.

*Alun D Jones
Technical Director
Micross Components Ltd.,
Oriol Court, Omega Park
Alton, Hampshire, UK
GU34 2YT*

*Tel : +44 (0) 1420 594180
Email : alun.jones@micross.com
Web : www.micross.com*

Test Vector Format

To provide Micross Components with sufficient information to create an ASIC test program, two data files are required, (a) the **Test Vector File**, and (b) the **Test Vector Information File**. This vector file format is a subset of the Advantest vector file format, and can be simply generated from JEDEC 3C vector files.

a). Test Vector File

The Test Vector File is an ASCII text file, containing the test vectors as a sequential pattern, each test vector terminated with a <CR/LF> (DOS format) or a <CR> (SUN format).

The Test Vector File is a time-sliced file, not event driven, with each vector specifically relating to a fixed time period. The data within each vector then specifies the action on individual pins during that fixed time period.

Each test vector follows the following format :-

```
<spc>!<spc>Pin-Vector-Data ; Vxxxxx {Remarks}<CR/LF>
```

where :

!	indicates the start of a test vector data line
<spc>	optional white space characters, may be single or multiple TAB <09h> or SPACE <20h> characters.
;	indicates the end of the test vector data
{Vxxxxx}	{optional}, serial vector count, starting from V00000.
{Remarks}	{optional}, Any textual comments/remarks.
<CR/LF>	Line termination characters.
Pin-Vector-Data	Text-line describing pin function during that test vector. This serves as a pin-related table, each pin related to it's character position. <i>Table 1.</i> shows the character/function relationship.

Table 1.

Character	Pin Function	Notes
1	Drive Logic "1"	Input or I/O pin only
0	Drive Logic "0"	Input or I/O pin only
H	Sense Logic "1"	Output or I/O pin only
L	Sense Logic "0"	Output or I/O pin only
Z	Sense Tristate condition	Output or I/O pin only
X	No Care	Output, I/O, or NC pin only

Note that power supply pins, or device pins not of a digital nature, are present in the Pin-vector-data.

Sample Test vector file :

```

! 01000000000000000000000000000000 ; V00000 Reset active
! 00000011110000000000000000000000 ; V00001 Chip enabled
! 10000011110000000000000000000000 ; V00002 Start Clock
! 10000011110000000000000000000000 ; V00003
! 10000011110000000000000000000000 ; V00004
! 10000011110000000000000000000000 ; V00005
! 10000011110000000000000000000000 ; V00006
...
! 10000011110000000000000000000000 ; V00111 Overdrive IOCS1
! 10000011110000000000000000000000 ; V00112 Overdrive IOCS2

```

Notes ...

- **Pattern Size**

The Test Vector File can contain upto a maximum of 256K (262144) test vectors, each to a maximum of 128 pins, but for practicality the pattern length should be kept much shorter (less than 64K).

- **Partitioning and Initialization**

If required, the test vector file may be partitioned into many smaller files, but each file should contain sufficient vectors to enable stand-alone functionality (i.e. should contain all required device initialization). Each test pattern file should be capable of operation from both a power-up condition, and continuous cycling after power-up.

- **Pin Scrambling and pin order**

The position of pin data within the Test Vector must remain consistent, but need not correspond directly with ASIC device pin numbers. If the pin data position does not correspond one-to-one with the device pin numbers, however, then pin scrambling information must be provided in the Test Vector Information file.

- **Data Formatting**

The test vectors may include or assume data formatting, e.g. Clocks, etc., and pin formats such as RZ, RTO and NRZ are permitted, but details of these formats go beyond the scope of this document. Please refer to an Micross Components Test engineer for further details.

- **Pattern loops, pattern matching and repeat vectors.**

As previously stated, the above format is a subset of a more complex pattern format, that can include such capabilities as pattern or vector matching, multiple clocks per vector, multiple repeat vectors, pattern looping, pattern subroutines, etc., which go beyond the intended scope of this document. Should more exotic vector control be required, please refer to a senior Micross Components engineer for further details and advice.

b). Test Vector Information File

The Test Vector Information File contains all the miscellaneous data, preferably in tabular form, required to implement the Test Vector File for the tester. This information should include :-

- 1). Test Vector Data Formatting, i.e.
 - (i) RTO, RZ, NRZ Waveform formatting
 - (ii) Pin I/O formatting

- 2). Test Vector Timing data, i.e.
 - (i) Test Rate (test frequency)
 - (ii) Strobe timing & positioning
 - (iii) Clock & signal timing
 - (iv) Pin timing assignments

- 3). Test Vector to Pin scrambling, i.e.
 - (i) Vector-DUT pin relationship
 - (ii) Power pin connections

- 4). Test Voltage/Current conditions, i.e.
 - (i) Power supply voltages
 - (ii) Output Loads

- 5). Output drive locations
Vector numbers (pattern locations) at which individual output pins may be parametrically tested for High, Low, and Tristate conditions. It is preferable that locations are selected that have a maximum number of pins available for parametric testing, to reduce test time. For the same reason, it is also desirable that all parametric testing be limited to a single Test Vector File.

- 6). Miscellaneous conditions
Any other conditions required to provide or maintain device functionality, i.e.
 - (i) Crystal or resonator circuitry,
 - (ii) Input to Output connections
 - (iii) Power supply sequencing

- 7). IDDQ and IDDQA conditions
Where applicable, the vector numbers (pattern locations) at which the quiescent supply currents should be measured, either for the digital subsection, analog subsection, or both.

i.e. In tabular form

DUT	Pin Data		Type	Pin Mode		Locations		
	Vector	Name		Format	Load	H	L	Z
01	03	Clock	I	RZ	-	-	-	-
02	13	Reset	I	NRZ	-	-	-	-
03	02	Enabl	I	NRZ	-	-	-	-
04	78	UFC0	I	NRZ	-	-	-	-
05	79	UFC1	I	NRZ	-	-	-	-
06	80	UFC2	I	NRZ	-	-	-	-
07	22	Disable	I	NRZ	-	-	-	-
08	01	Clear	I	RTO	-	-	-	-
09	-	VDD	PWR	-	-	-	-	-
10	06	Sense1	I/O	-	A	019	020	191
11	-	VSS	PWR	-	-	-	-	-
12	07	Sense1	I/O	-	A	019	022	191
.
.
84	29	OQ32	O	-	B	334	407	1077

Pin Timing :-

Test Rate	(Tp)	1000nS	
Clock	(Tr)	100nS	
	(Tf)	600nS	
Others	(Tr)	0nS	
	(Tf)	1000nS	
Strobe	(Ts)	900nS	(implies edge strobe)

Loads :-

Type (A) 2K0 2% pull-up to VDD
 Type (B) +/-4.0mA pull to 1.5V

A Sample file

A simple demonstration Verilog-XL file for producing the **Test Vector File**.

```
/* This alias module is for use internal to the netlister only.
   Please do not use the same name for modules or assume the
   existence of this module.
   Eg.
   reg io_D0, io_D1; <<< for IO control
   cds_alias  cds_alias_inst1(D1, io_D1);
   cds_alias  cds_alias_inst0(D0, io_D0);
*/

module cds_alias( cds_alias_sig, cds_alias_sig);

parameter width = 1;
input [width:1] cds_alias_sig;

endmodule

// Verilog DEMO file for producing test vectors for Advantest (ADJ)
integer logdata;
wire x,z,Z;

//=====
//
// Function :    getHL (x);
//
// Operation :    Converts output data into "HLXZ" format for
//                JEDEC 3B vector statements.
//
// Usage : Use in $strobe statements equating to "%c"
//         character print, as getHL(x), where "x" is
//         the output pin in "10xz" format. Do not use
//         with arrays/busses.
//
// Notes : The inclusion of this function inhibits the
//         creation of the ".tms" file, used by the VRA
//         option. 2 warning messages will be created during
//         compilation, but this does not inhibit cWaves usage.
//
// Author :    Alun D. Jones
//
//=====

function [7:0] getHL;
input I;
begin
    case (I)
```

```

                1'b0:          getHL="L";
                1'b1:          getHL="H";
                1'bx:          getHL="X";
                1'bz:          getHL="Z";
                default:       getHL="x";
            endcase
        end
    endfunction

//=====
//
// Function :    getIO (x,y);
//
// Operation :   Converts output data into "HLXZ" format for
//               JEDEC 3B vector statements.
//
// Usage : Use in $strobe statements equating to "%c"
//         character print, as getIO(x,y), where "x" is
//         the output Data pin in "10xz" format, and y
//         is the Data Enable.
//         Do not use with arrays/busses.
//
//
//         x      y   Output Text
//         -----
//         any    x/z   x
//         0      0     0
//         1      0     1
//         x      0     X
//         z      0     Z
//         0      1     L
//         1      1     H
//         x      1     X
//         z      1     Z
//
// Notes : The inclusion of this function inhibits the
//         creation of the ".tms" file, used by the VRA
//         option. 2 warning messages will be created during
//         compilation, but this does not inhibit cWaves usage.
//
// Author :    Alun D. Jones
//
//=====

function [7:0] getIO;
    input I,J;    // I=Data, J=Enable
    begin
        case (J)
            1'b1:case (I)
                1'b0:          getIO="L";
                1'b1:          getIO="H";
                1'bx:          getIO="X";
                1'bz:          getIO="Z";
            endcase
        endcase
    end
endfunction

```

```

                default:      getIO="x";
            endcase
        1'b0:case (I)
            1'b0:  getIO="0";
            1'b1:  getIO="1";
            1'bx:  getIO="X";
            1'bz:  getIO="Z";
            default:      getIO="x";
        endcase
        default:getIO="x";
    endcase
end
endfunction

initial begin

    $sdf_annotate("sdfaa1.sdf", test.top,, "MAXIMUM");
    $log ("SDFAA101.VECMAX");      // Open Datalog File

    logdata=1;

//
//  PUT MAIN Simulation Test Bench Data Here
//

    $stop;
    $finish;
end

always begin
    #50 MCLK=~MCLK;                // Clock Running
end

//=====//
//  Typical $strobe setup for Advantest vectors
//
//  ! ....test.vectors.... ; Vxxxx
//
//      %b for binary data (0/1/X)
//      %c for character data (0/1/X/Z/H/L)
//
//  Limitations :
//  Use of the "$strobe" command in this manner does not permit
//  timing / vector guardbanding, nor does it check for vector
//  level stability.
//  The simplest (and fastest) method within Verilog-XL for
//  vector stability is to issue and output three "$strobe"
//  commands in succession with, say 5nS intervals, and let "awk"
//  do the rest in the post simulation conversion, compiling the
//  aggregate vector and nulling any output level variations on a

```

```

//      pin-by-pin basis
//
//
//=====
//
//      Note ... re-write for specific part, this is a DEMO ONLY !!!
//
//          1. adjust pinout & pinout display
//          2. adjust timing ... DEMO example has
//              Tper      = 100nS
//              Strobe    = 90nS
//              Other     = T0, unless spec'd in main pattern
//                      eg. MCLK RZ, Tstart = 50nS, Tstop = 100nS
//
//
//
initial begin
  if (logdata == 1) begin
    #90 // Initial delay (actual strobe offset)
    forever begin
      $strobe("! %b%b%b%b%b %b%b %b%b%b%b%b%b%b%b ",
        "%c%c%c%c%c%c%c%c %c %c; V%0d",
        MCLK,CE,OE,SCLR,M0,PARIN,SERIN,
        D7,D6,D5,D4,D3,D2,D1,D0,
        getHL(Q7),getHL(Q6),getHL(Q5),getHL(Q4),
        getHL(Q3),getHL(Q2),getHL(Q1),getHL(Q0),
        getHL(PAROUT),getHL(TESTX),
        $time/100);
      #100; // Cycle time
    end
  end
end
end

```