Prelab 2 – Continuity Test

Introduction

In this lab exercise, you will write and test the code to perform the continuity test. The purpose of the continuity test is to determine if there is a complete connection from the ATE instruments to the device. The primary function of the continuity test is to determine if the device has been loaded properly into the DIB. Additionally, it checks for fabrication errors in the device pads. Since most devices failing continuity do so because of improper loading (devices are loaded automatically using a handler and sometimes misalignment causes loading errors), often the devices that fail this test are reloaded and tested again. Therefore, the test is designed to ONLY check for a continuous connection from instrument to device pin. The easiest method of performing this test is to check for the pad protection circuitry that is built into the pad. Fig. 1 shows an example of typical pad protection circuitry. The diodes are reverse biased, and thereby have no impact on the input signal unless the pin goes to a voltage higher than VCC or lower than VEE. Other pad structures do exist and can affect the continuity test.

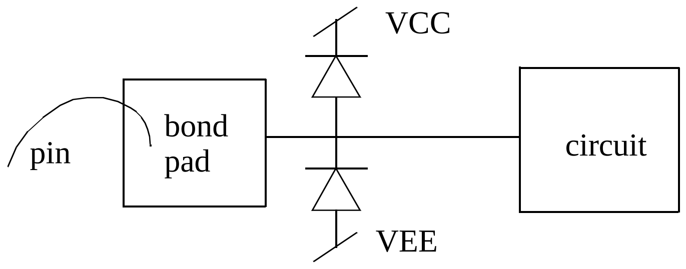

Figure 1: Pad protection circuitry

The simplest and most controlled method of performing this test is presented below and shown in Fig. 2:

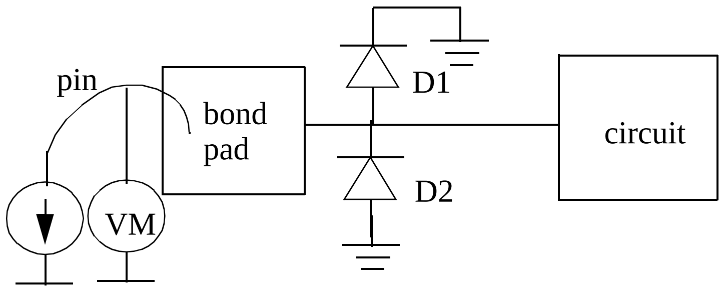

Figure 2: Continuity Test

1. Turn off the rails, (e.g. set $V_{CC} = V_{EE} = 0$)
2. Apply a small current (1mA) to all input and output pins of the chip. Positive current should forward bias D1 and negative current should forward bias D2. Some pads only have D1 or D2.

3. Measure the resulting voltage (Volt Meter, VM). The resulting voltage should equal the diode drop across the forward biased diode (0.2V to 1V). If you are measuring D1, VM should be positive. If you are measuring D2, VM should be negative.
4. Log the results and compare them to the datasheet limits.

Prelab Assignment

a) Be sure and read the introduction so that you understand the test that we will be performing. I will not have time to get to this in class before we do this in lab.
b) Print out a copy of the Comparator Daughter Card schematic for each site. This is the schematic for the devcie interface board (DIB) daughter card.
c) Show all paths necessary to perform this test. In this manner, you will determine which relays must be closed and which resources you will use.
d) Purchase 2 comparators from the instrument room.
e) If you want to start thinking about how you will program this test, that is great. However, please do not actually add any code to your existing project. New functions MUST be added in a particular method or things won't work. I will show you this method in lab.

Lab 2 - Continuity Test

In this lab, you will add the continuity test to your project.

A. Create a new function

1. Create a new function, which will add a function to the program.
2. Give the function a "Test Label" and "Function" name. I usually try to make these names the same and reflective of the test being performed. The "Function" will be the name of the function in the .cpp file and the "Test Label" will be the name of the label in the datasheet file.
3. Select the number of "rows" for the datasheet file. The number of rows is equal to the number of values you want to compare to a set of limits for a given site. The multi-site elements will be handled later when we save the values to the datalog. For the continuity test, you want to test Vplus, Vminus, and Vo for each site. Therefore, you need 3 rows.
4. We will always use the datasheet to determine our test flow (order of tests to be run). Having multiple test flows is useful when you have some tests that aren't run all the time because their intention is to make sure that there is no drift in the process, but they aren't production tests. You can choose the Test Flow and some test flows will have a different subset of tests than another.

B. Fill out the Datasheet File

The datasheet file contains all of the information that you want to save in the datalog. This is the information that you want sent to the screen and saved to the files. It also contains the test limits. The measured value will be compared to the test limits and if it falls within range, the test will pass. If it is out of range, or an alarm goes off such as an instrument current or voltage limit, the test fails. The reason the test limits is stored in a different file is so that you can use the same program to test a family of parts with different datasheet files for the different limits.

1. Start filling out the datasheet file. This is the first test, so the Test Number should be 1 for all 3 rows.
2. We will separate the different measurements with the Subtest Number. Set these to the values 1-3.
3. Provide a Datalog Description with enough detail that you will know what each measurement is for. In this case, ORDER MATTERS! You must set these in same order as your instrument group or datalogging will be harder later.
4. Set the low and high limits. Since you are measuring the diode drop, the resulting value should be between 1V and 0.2V. Note: you will have to be careful of the sign. If you are assuming a positive voltage, you are assuming the you are checking the diode to VCC and thereby are sourcing current from the instrument. The conditions for the test must match the sign of the results. Note: current **out** of the instrument is positive.
5. Set the units equal to the units that will be measured. If you are datalogging a single measurement, it should be in the units measured (e.g. V for voltage and mA for current). If you have performed math on the measurements, you need to account for the resulting units.
6. Set the Low Fail Bin and the High Fail Bin. When i.c.'s are tested, they are typically sorted as to what passed and what failed. Physically, any chip that fails is scrapped, so further separation is unnecessary and impractical since there are only a limited number of physical bins to choose from. However, a product engineer might like to know which tests failed and how often so that he can propose improvements and choose which test to work on improving. This information can easily be tracked in software where there are a large number of resources. Therefore, there are "hardware bins", where the devices are physically sent when they pass or fail, and "software bins", where the information about each chip is stored. Bin 1 is reserved for passing devices. Bin 8 is often continuity failures. Map the many software bins to the few hardware bins.

D. Write Continuity Function Code

The continuity function should have each of the following steps:

(i) Set up the hardware. This involves closing necessary relays and setting the mode and ranging of the source/measurement units, and connecting them to the device.
(ii) Wait for the circuit to settle
(iii) Measure the resulting currents and/or voltages
(iv) Store the results into the datalog
(v) Turn the appropriate equipment off

Step (i): Close the necessary relays and initializing the equipment. First, close your relays to get the proper equipment connected to the chip. You will need to wait for the relays to close and charge to redistribute appropriately. Be sure all instruments are turned off before opening or closing a relay, or the relay will be "hot switched." Hot-switching can cause the voltages to arc between the relay terminals destroying the relay. Next, set up your continuity source-measurement devices (APUs) and your supply source-measurement devices (SPUs) as directed in the prelab schematics. Example: set the Vplus APU to source 1mA of current into the continuity pins. You should expect a diode drop voltage measurement, which determines the range of the instrument. Note: positive current leaves the instrument.

Step (ii): wait for the circuit to settle with the input values set. At this time, estimate a wait time of 1000usec.

Step (iii): Measure the instrument readings and store the results. To begin, take measurements averaging 4 measurements 13usec apart. We can change these settings on the ATE and see the impact on the measurements. The measurements are stored in a datastructure. Each element in the datastructure contains 4 fields (resource or instrument number, site, value, and whether the measurement Passed or Failed), so the data for each site must be stored along the y-axis, as shown in Table 1. Since we have 3 APUs with site 1 and 3 APUs with site 2, the final datastructure should contain 6 elements as shown.

Table 1: Datastructure for continuity

resource	site	value	PassFail	
APU0 (Vplus)	1	0.7	1	site 1
APU1 (Vmin)	1	0.72	1	
APU2 (Vout)	1	.68	1	

Table 1: Datastructure for continuity

resource	site	value	PassFail	
APU8 (Vplus)	2	10	0	site 2
APU9 (Vmin)	2	10	0	
APU10 (Vout)	2	10	0	

Step (iv): log the results. By logging the results, the measurements are compared to the datasheet file and failing tests are flagged. Be careful that the datasheet index corresponds to the correct measurement.

Step (v): Once the results are properly stored, you must turn the equipment off.

E. Modifying the Set-Up Code

Each time you have written a function, you must also go back to the functions FailSite and TestCompletion to make sure that any new instruments that you have used are turned off if a site fails or when the program is done. That way, you will not blow the next device that is placed into the socket.

1. Go to the TestCompletion routine. Make sure that all resources are turned off.
2. Go to the FailSite routine. In this case, you must only turn off the equipment in the failed sites.

F. Test and Debug

1. *Compile and correct errors.*
2. *Build and correct errors.*
3. *Run and correct errors.*

   If you have no run-time errors, the results of the tests for both sites should be displayed.
   All continuity tests should fail without a chip.
   Run-time errors are denoted by the red alarm going off in the upper left-hand corner.
   If you have run-time errors, break points can be set by clicking on the gray area beside the code. They can be removed by clicking again. Set a break point, run with debugging and find which message causes the error.
4. *Backup your code to a memory stick.*
5. *Take your memory stick to the testing room and restore your code onto the ATE computer*
6. Open your code. (Note: be sure to always open your code before running it. This ensures that you are running YOUR code and not someone else's.)
7. Run your test code with no chips inserted in the DIB at first. Make sure you still have no run-time errors. (This protects your chips from bad code).
8. Insert your chips onto the board and run your code with both chips. See if both chips pass.
9. Deactivate site 2 and see if the test passes site 1 and fails site 2.
10. Retest both sites a few times and notice how the numbers change slightly. This is due to quantization noise in the instruments.
11. Exit the program and remove your chips.
12. If you made any necessary changes to your code, make sure you back up the new version and restore it onto your computer for next week.

G. Write-up your results

Describe how your test results change from test to test. Record your test time. What percentage of your test time is in continuity versus userInit? Turn in your continuity function code along with your test results.