Center for Reliable Computing

TECHNICAL REPORT

ELF18 Test Environment and Defect Classification

Intaik Park, Erik Chmelar and Edward J. McCluskey

07-1 (CSL TN # 07-1) July 2007	**Center for Reliable Computing** Gates Bldg. 2A, Room #236 Stanford University Stanford, California 94305-9020

Abstract:

This report provides test environment and defect classification information for the ELF18 test chip. Due to the test chip's sequential design, special test environments were developed to aid in defect classification

Funding:

This work was supported in part by the LSI Logic Corporation

ELF18 Test Environment and Defect Classification

Intaik Park, Erik Chmelar and Edward J. McCluskey

CRC Technical Report No. 07-1
(CSL TR No. 07-1)
July 2007

CENTER FOR RELIABLE COMPUTING
Gates Bldg. 2A, Room #236
Computer System Laboratory
Department of Electrical Engineering
Stanford University
Stanford, California 94305-9020

Abstract

This report provides test environment and defect classification information for the ELF18 test chip. Due to the test chip's sequential design, special test environments were developed to aid in defect classification.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

In previous experiments at the Center for Reliable Computing (CRC), two different test chips, Murphy and ELF35, were designed and tested thoroughly to learn about the defects and their behaviors [Franco 95, Ma 95, Li 99, McCluskey 04]. New experiments are being conducted on a new test chip, which has been designed and fabricated by Philips. This new chip, the ELF18 test chip, is fabricated using 0.18μm technology, which is a more advanced technology than the 0.7μm technology used for Murphy and the 0.35μm technology used for ELF35.

The goal of the ELF18 experiments was to understand how defect behavior changes as fabrication technology scales down. Therefore, the experiments conducted on Murphy and ELF35 were applied to ELF18. However, because ELF18 contains sequential circuits, on contrary to combinational circuits of Murphy and ELF35, special tester setups were developed. This report presents the test setups and the tester environments for the ELF18 experiments. Additionally, the defect classification methodology and defect classification results for ELF18 are presented.

2. Test Chip

The ELF18 test chip was designed and fabricated by Philips using the 0.18μm Corelib technology. Each chip contains the total of 26 cores, including RAM, ROM, and the sequential logic cores. The cores of interest in the ELF18 experiments are ctl, DieID and RDMR logic cores.

2.1. ctl Core (Controller)

The ctl core directs which of the 26 cores is being tested at a given time. It directs inputs signals from the tester to a specific core being tested and directs outputs signals from that core back to the tester. The ctl core is controlled by the first 26 vectors (main vector) of each test set.

2.2. DieID Core (Die identification)

The DieID core contains a 16-bit register that is written to by blowing a series of fuses. This single register is used to record the four sets of unique numbers that represent lot, wafer, wafer x-position, and wafer y-position of the packaged chips. The readings of the die id are used to identify chips in the ELF18 experiments. Although there should be no duplication of die id numbers, there are a few chips that have same die id number. One such example is the chips with an all-zeros die ID.

2.3. RDMR Core

The RDMR core is an implementation of the R.E.A.L. Digital Signal Processor [Keivitis 98] but with all the memory cells (SRAMs) replaced by scan flip-flops and all the primary inputs and outputs replaced by scan flip-flops for boundary scan. The RDMR core also contains ring oscillators that are used as process monitors [Mitra 04].

Table 1 shows the general specifications of the RDMR core. Table 2 lists input pins and Table 3 lists output pins. The mapping of the RDMR input and output pins to tester channel and tester pin names is shown in Appendix I.

Table 1: RDMR DSP Core Specification

Equivalent gate count	53,732
Library Cells	21,796
Number of inputs	22
Number of outputs	19
Number of scan flip-flops	2,289
Number of scan chains	13
Shorted scan chain length	111
Longest scan chain length	184

Table 2: RDMR Core Input Pin List

Pin name	Operation
clk	Clock
sc	Shift control
cs	Chip select
gb	Output enable bar
td [11:0]	Test data (scan input)
btd	Boundary test data (scan input)
timein_rdmr	Delayline data in
col_rdmr [2:0]	Selection between 3 deliri's
row_rdmr	Selection between delay or ringo

Table 3: RDMR Core Pin List

Pin name	Operation
tq[11:0]	Test data (scan output)
btq	Boundary test data (scan output)
timeout_rdmr [2:0]	Output delay lines, only rdm1r
freqout_rdmr [2:0]	Output ringo's, only rdm1r

Each ELF18 test chip contains 6 copies of RDMR core, each with a slightly different implementation (such as number of metal layers, with or without antenna protected cells, and with or without wire spreading). The 6 different RDMR cores are named Rdm2R0, Rdm2R1, Rdm2R2, Rdm3R0, Rdm3R1 and Rdm3R2. All the structural and parametric tests are applied to these cores in the ELF18 experiments.

2.4. Packaging

The ELF18 test chips are packaged using a QFP80 package with 80 pins. There are a higher number of pads than the number of pins due to double power supply (21 pads) and EMC (4 pads).

3. Test Environment

The ELF18 experiments were performed using an Agilent 93000 SOC tester. This section describes the tester setups including test flow, timing, voltage levels, pattern conversion, test sets application, and failing cycle logging.

3.1. Test Flow

The basic test flow for the ELF18 experiments is depicted in Figure 1. The contact test is the first test applied. It insures proper insertion of the test chip into the socket and verifies that all the pins communicate with the tester channels properly. The next test is the controller test that checks the proper operation of the ctl core. If a chip passes these two tests, the die ID of the chip is read and all the test sets for the RDMR cores are applied.

The test sets for the RDMR core are applied 3 times, each with a different test application speed: fast speed, rated speed and slow speed. However, all the 6 cores (Rdm2R0, Rdm2R1, Rdm2R2, Rdm3R0, Rdm3R1 and Rdm3R2) are tested individually. Before any other test to the RDMR core is applied, the scan chain test is applied first to confirm the correct operations of the scan chain circuitry. If the scan chain test fails, the flow skips to the next core. Only when the scan chain test is passed, are the core test sets applied.

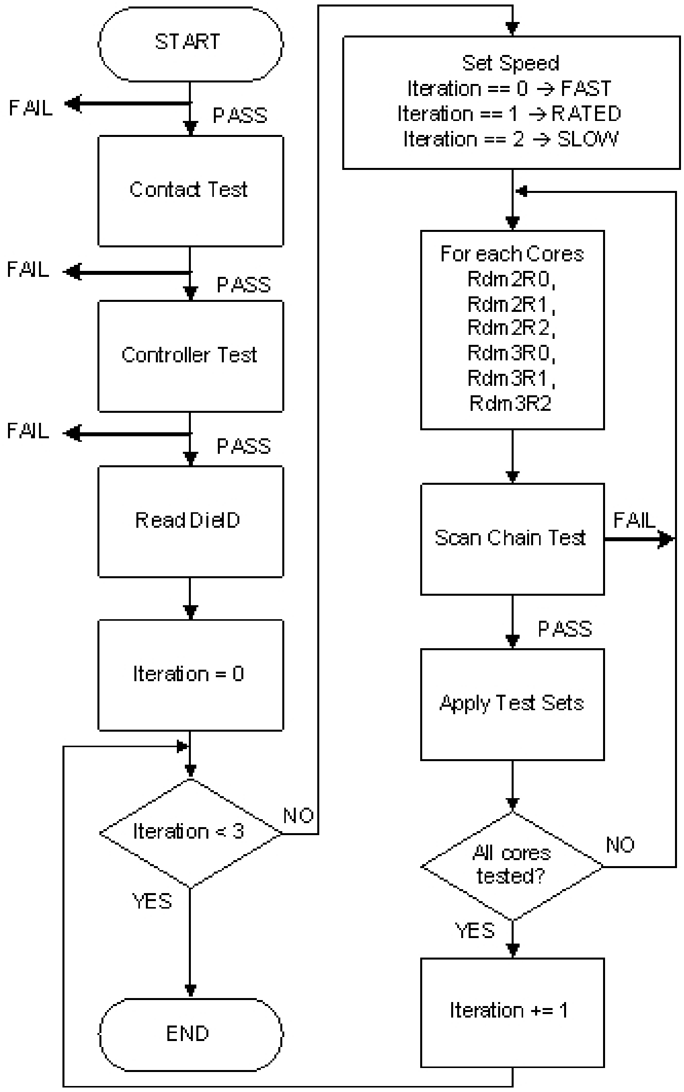

Figure 1: ELF18 Test Flow Flowchart

The file that describes the test flow in the Agilent SmarTest environment is a plain text file with specific syntax. Hence, a PERL script is written to automate the generation of the test flow (gen_testflow.pl).

3.2. Timing Setup

The RDMR core test timing is a set of complex timing equations since the RDMR core is tested with many different test sets that require various timing setups. This subsection describes the different timing setups used in the ELF18 experiments.

3.2.1. Scan Shift Timing

The scan shift for the RDMR core is performed at the frequency of 20MHz (50ns periods for a tester cycle). This frequency is determined by performing shmoo tests with scan chain test patterns on many good chips (test chips without known defects) and finding the highest frequency where scan chain tests passed.

The logic values (0 or 1) for each of the input pin (scan enable and scan-in pins) is set at the beginning of the tester cycle (at 0ns). The clock pulse rising edge is at 50% of the tester cycle (at 25ns) and the falling edge is at 82.5% of the tester cycle (at 41.25ns). The measurement of the output pins (scan-out pins strobe) is at 45% of the tester cycle (at 22.5ns). The following figure describes an entire period of one scan shift cycle.

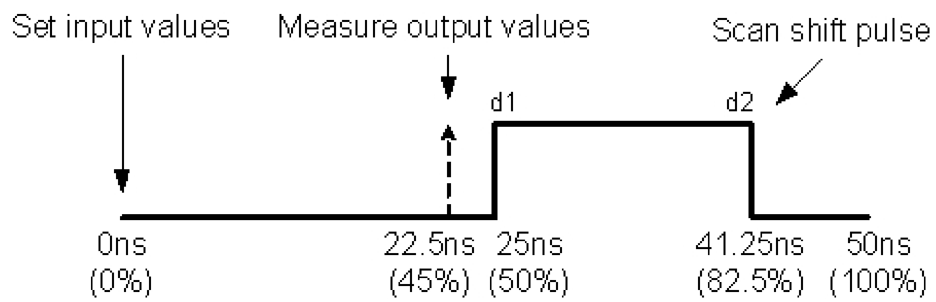

Figure 2: Scan shift timing (cycle name: cyc)

In the SmarTest timing setup, the scan shift clock pulse is implemented using two clock edges.

Cycle name: cyc
d1 = 0.500 * Period (rising edge)
d2 = 0.825 * Period (falling edge)

3.2.2. Stuck-at Test Timing

The application of a stuck-at test set requires one capture pulse of the clock signal. Therefore, the position of the rising edge of the capture clock pulse is moved back and forth to implement various test application speeds of stuck-at test sets. The fastest stuck-at test application speed is determined by shmoo tests with stuck-at test patterns on many good chips. The distance between the rising edges of the last shift and the capture pulse is the test application period, and the inverse of this period is the stuck-at test application frequency. For example, if the time difference between the rising edges of the last scan shift and the capture clock pulses is 27.77ns, the application frequency is 36MHz.

The 'Fast' speed is the maximum test application frequency determined by the shmoo test. The 'Rated' speed is 1.3 times slower than the fast speed and the 'Slow' speed is 3 times slower than the fast speed. By changing test application frequency, it essentially changes the distance between the scan enable signal change (high to low) and the rising edge of capture clock pulse. The following Table and Figure summarize the stuck-at test application speeds.

Table 4: Stuck-at Test Application Speeds

Speed	Frequency	Period	Time between SE and capture clk
Fast	36Mhz	27.77ns	2.77ns
Rated	27.97Mhz	35.75ns	10.75ns
Slow	15Mhz	66.67ns	41.67ns

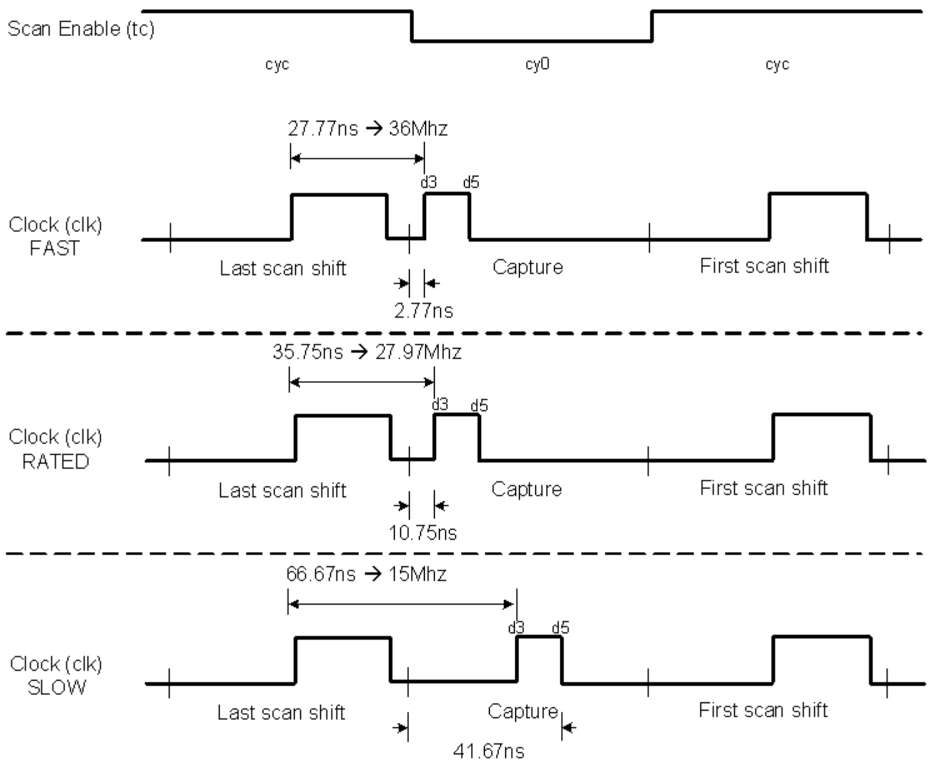

Figure 3: Stuck-at Test Timing (cycle name: cy0)

In the SmarTest environment, 2 drive edges are used to implement the capture clock (d3 for and d5 for falling edge of cycle name 'cy0').

LPer: stuck-at test launch period (in nsec).
halfPerS: half of scan shift period (50ns / 2 = 25nsec)

Cycle name: cy0
d3= LPer – halfPerS (rising edge)
d5= LPer - halfPerS + 10 (falling edge)

3.2.3. Transition Test Timing

The transition test requires two patterns to induce a transition of the logic value at a fault site. These two patterns are implemented either by the launch-on-capture scheme or the launch-on-shift scheme. However, the ELF18 experiment utilizes only the launch-on-capture scheme. The launch-on-shift scheme cannot be used in the ELF18 test experiments because it requires a fast scan enable signal to switch between the launch vector and the capture vector, but the scan enable signal tree in ELF18 test chips is not fast enough to propagate the transition of scan enable signal in time. Hence, all the transition test sets in the ELF18 experiments are applied using the launch-on-capture scheme.

The launch-on-capture scheme requires two consecutive clock pulses to implement two test patterns. The test application frequency of the transition test is determined by the distance between these two pulses. In the ELF18 setup, the position and the width of both pulses are adjusted to implement various test application frequencies.

The 'Fast' speed for the transition test is determined by shmoo tests on good chips. The 'Rated' speed and the 'Slow' speed are slower than the fast speed by factor of 1.3 and 3 respectively. For core Rdm2R0, Rdm2R1, Rdm2R2 and Rdm3R0, the fastest transition test speed was determined to be 100MHz. However, Rdm3R1 and Rdm3R2 cores cannot run as fast as other cores because of a design issue in the global clock network. The fast speed for these two cores is found to be 90MHz. The transition test application speeds are summarized in Table 5.

Table 5: Transition Test Application Speeds

Cores	Speed	Frequency	Period
Rdm2R0,Rdm2R1, Rdm2R2,Rdm3R0	Fast	100MHz	10ns
	Rated	76.92MHz	13ns
	Slow	33.33MHz	30ns
Rdm3R1, Rdm3R2	Fast	90MHz	11.11ns
	Rated	69.23MHz	14.44ns
	Slow	30MHz	33.33ns

The transition test timing uses two separate clock pulses in two tester cycles. The launch clock pulse is located at the end of the first tester cycle and the capture clock pulse is at the beginning of the second tester cycle. The test application period is determined by the distance between these two pulses, with the test application frequency being the inverse of this period. The following figure depicts the transition test timing of the ELF18 experiments for Rdm2R0, Rdm2R1, Rdm2R2 and Rdm3R0 cores. The timing for Rdm3R1 and Rdm3R2 are similar but with different frequencies.

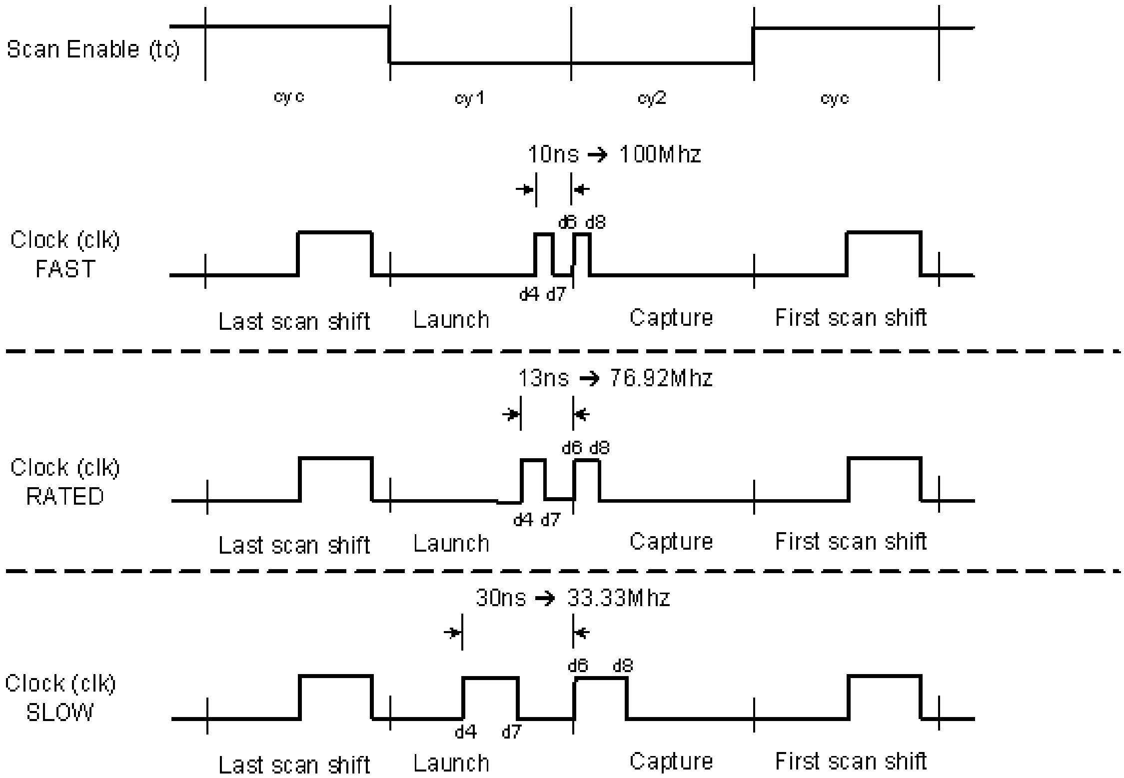

Figure 4: Transition Test Timing for Core Rdm2R0, Rdm2R1, Rdm2R2 and Rdm3R0 (cycle name: cy1 and cy2)

The implementation of the transition test timing requires 2 clock pulses and hence 4 edges are used to implement these two pulses.

Period: tester cycle period (in nsec).
halfPer2: half of transition test launch period (in nsec).

Cycle name: cy1 (launch pulse)
d4= Period - halfPer2 * 2 (rising edge)
d7= Period - halfPer2 (falling edge)

Cycle name: cy2 (capture pulse)
d6= 0 (rising edge)
d8= halfPer2 (falling edge)

3.2.4. Multiple Capture Test Timing

A multiple capture test set contains three clock pulses: one launch pulse and two capture pulses. The implementation of the multiple capture test set is the same as the transition test except that there is one additional capture clock pulse. The timing diagram for a multiple capture test is shown below.

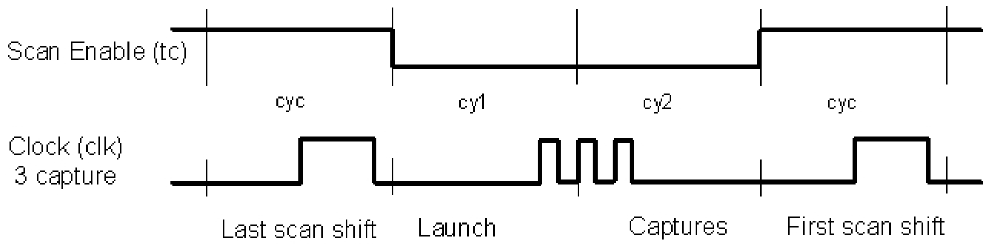

Figure 5: Multiple Capture Test Timing

The fast speed for the multiple capture test set is the same as the transition test sets (90MHz for Rdm3R1 and Rdm3R2 cores and 100MHz for the rest of the cores). However, the rated speed and slow speed for these test sets are not the same as in the transition test. Because there are 3 clock pulses defined in one tester cycle, the widths and the spacing of the pulses limits how slow the test can be applied. Hence, the tester cycle period was changed from 50ns (20MHz scan shift) to 62.5ns (16MHz scan shift). The 62.5ns tester cycle period allowed the slowest test frequency to be 70MHz. The following table lists the test application frequencies of the multiple capture tests on different cores, while the Figure 6 presents timing diagrams for the first 4 cores.

Table 6: Multiple Capture Application Speeds

Cores	Speed	Frequency	Period
Rdm2R0,Rdm2R1, Rdm2R2,Rdm3R0	Fast	100MHz	10ns
	Rated	85MHz	11.76ns
	Slow	70MHz	14.28ns
Rdm3R1, Rdm3R2	Fast	90MHz	11.11ns
	Rated	80MHz	12.5ns
	Slow	70MHz	14.28ns

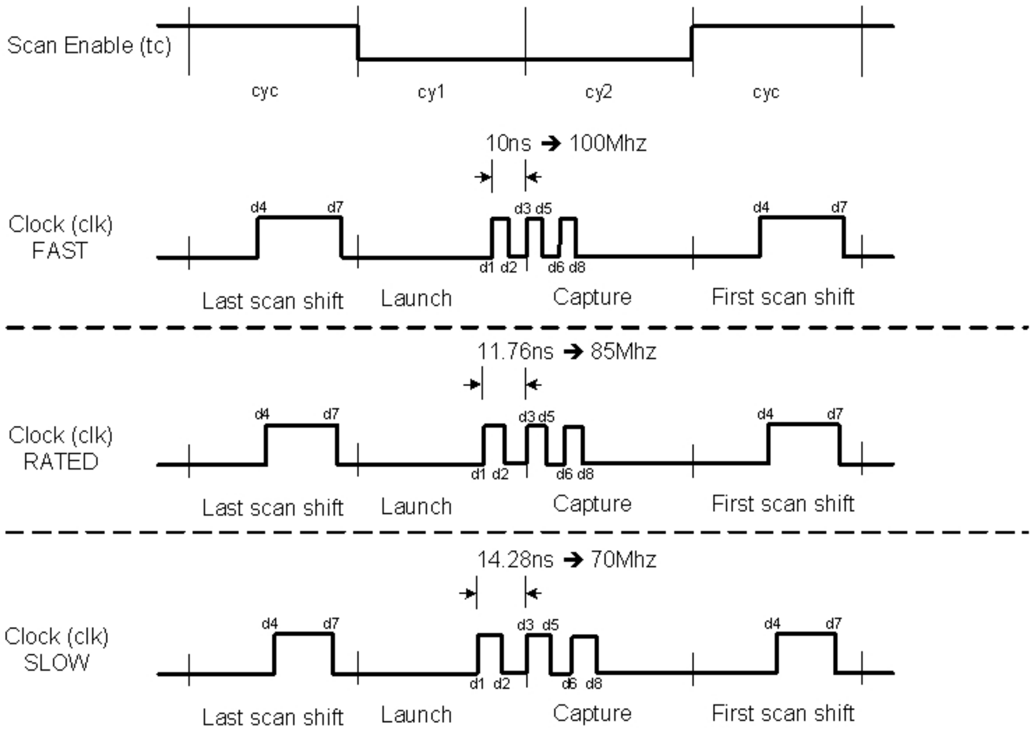

Figure 6: Multiple Capture Test Timing for Core Rdm2R0, Rdm2R1, Rdm2R2 and Rdm3R0 (cycle name: cy1 and cy2)

The Implementation of the timing of the multiple capture test sets is limited due to the tester constraints. The multiple capture test sets require multiple pulses in one clock signal, but the Agilent 93000 SOC tester only supplies 8 signal edges for one signal pin. Hence 3 rising edges (d1, d3 and d6) and 3 falling edges (d2, d5 and d8) are used to implement 3 clock pulses for 3 capture test sets and 1 rising edge (d4) and 1 falling edge (d7) are used to generate scan shift pulse.

```
Period: tester cycle period (in nsec).
halfPer2: half of multiple capture test launch period (in nsec).

Cycle name: cyc            (scan shift pulse)
d4= 0.5 * Period                     (rising edge)
d7 = 0.825 * Period                  (falling edge)

Cycle name: cy1            (launch pulse)
d1 = Period - halfPer2 * 2           (rising edge)
d2 = Period - halfPer2               (falling edge)

Cycle name: cy2            (1st and 2nd capture pulse)
d3= 0                                (rising edge)
d5= halfPer2                         (falling edge)
d6= halfPer2 * 2                     (rising edge)
d8= halfPer2 * 3                     (falling edge)
```

3.2.5. Very-Low-Voltage Test Timing

To screen weak suspect chips, the Very-Low-Voltage (VLV) test was performed on ELF18 test chips. However, since the voltage level used in this test is lower than the specified voltage level, the test application speed is also slowed down to compensate for the effects of the low voltage level [Chang 96].

In the ELF18 experiments, the timing of the VLV test was exactly same as the stuck-at test timing except that it was performed with 150ns of tester cycle (6.66MHz of scan shift frequency) and 7MHz of capture frequency.

3.3. Voltage Level Setup

The nominal voltage for the ELF18 test chips was specified as 1800mV, hence all the tests except the VLV test were applied with this voltage. Conventionally, the voltage level for the VLV test is 2~3 times of the threshold voltage [Chang 96]. However, in ELF18, voltage level of 650mV is used for VLV testing even though the typical threshold voltage for NMOS is 630mV and PMOS is 570mV in the RDMR core. This value is determined from shmoo tests and also suggested by the manufacturer. The following Table summarizes the voltage level setups for the ELF18 experiments.

Table 7: ELF18 voltage levels setups

Setup	Voltage level	Max. Current
Nominal	1800mV	500mA
Very-Low-Voltage (VLV)	650mV	500mA

3.4. Pattern Conversion

The test patterns for ELF18 are generated using 3 different commercial automatic test generation programs (ATPG) and they produced test patterns in different formats. Hence, conversion scripts for each ATPG tool were written in PERL script to generate tester-compatible test vectors.

The ATPG generated test patterns are first written in text format pattern files (e.g. VHDL or ASCII) and converted to another text format (HP format) by the scripts. The HP format test patterns are again converted to binary format test pattern files (BINL format) using a program called v2b before being loaded to the tester. The following figure describes the ELF18 test pattern conversion process.

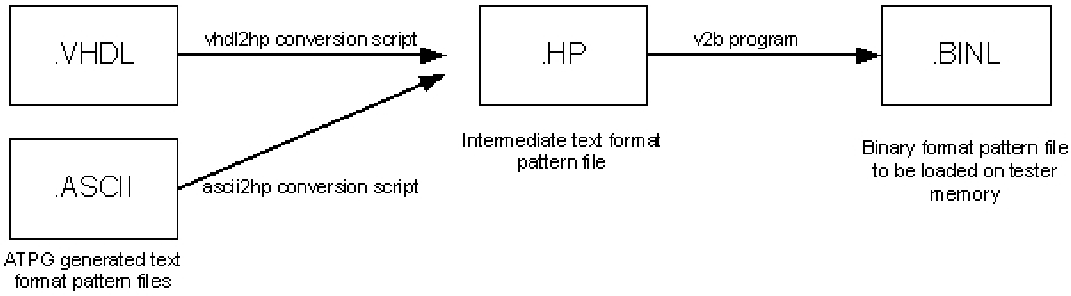

Figure 7: ELF18 Test Pattern Conversion Process

The HP format is a serialized test pattern format that lists the input and output values of each pin at each tester cycle. Therefore it carries information about tester cycle names. The example of HP format is as below.

```
FORMAT CLK1 CTL_EN CTL_D DIG_D24 DIG_D11 DIG_D10 DIG_D9 DIG_D8 DIG_D7 DIG_D6
DIG_D5 DIG_D4 DIG_D3 DIG_D2 DIG_D1 DIG_D0 DIG_D12 DIG_Q11 DIG_Q10 DIG_Q9
DIG_Q8 DIG_Q7 DIG_Q6 DIG_Q5 DIG_Q4 DIG_Q3 DIG_Q2 DIG_Q1 DIG_Q0 DIG_Q12
TIMEOUT FREQOUT TIMEIN ACLK CLK2 ADATAIN CTL_TC DIG_D23 DIG_D22 DIG_D21
DIG_D20 DIG_D19 DIG_D18 DIG_D17 DIG_D16 DIG_D15 DIG_D14 DIG_D13 TRIGGER ADATAOUT
BGVOUT CTL_Q DIG_Q16 DIG_Q15 DIG_Q14 DIG_Q13 VVCO ;
R1 cyc 11010000000001000XXXXXXXXXXXXXXX00000000000000000XXXXXXXX;
R1 cyc 11011101011101100XXXXXXXXXXXXXXX00000000000000000XXXXXXXX;
…
R1 cyc 11011001110001011XXXXXXXXXHXXXXX00000000000000000XXXXXXXX;
R1 cy1 11000000000000000XXXXXXXXXXXXXXX00000000000000000XXXXXXXX;
R1 cy2 11000000000000000XXXXXXXXXXXXXXX00000000000000000XXXXXXXX;
R1 cyc 11010000000000000LHHHLLLHLLHHHXX00000000000000000XXXXXXXX;
…
```

The format starts with a header line that lists all the pins in order. For example, system clock (CLK1) is the 1st pin listed in HP and DIG_D24 (scan enable) is listed as 4th pin. The pin data is presented in the following format.

R1 <cycle name> <pin data…>

The cycle name field tells how the tester recognizes each line (what cycle name in timing setup to use. For example, cyc, cy0, cy1 or cy2) and the pin data field represents the input (0 or 1) or output values (L or H or X) of all the pins. Since the ATPG generated test pattern files does not have any cycle information, the conversion script has to insert appropriate cycle names depending on the type of tests being converted.

The generated HP format pattern files are converted to BINL binary test pattern files, which can be loaded directly to the tester memory. This conversion is performed by a conversion program v2b. The command for the conversion is as following.

v2b -d <pin mapping file> -p <configuration file> -v <HP file> -l <BINL file> -ALPCUE

The pin mapping file can be extracted from the timing setup file and the configuration file is the device configuration file that is used in the SmarTest environment.

3.5. Test Application

A test set can be applied directly in a testflow. However, since RDMR core shows some number of inconsistent results due to inconsistent failing bits (section 4.1), each test are applied multiple times to get more reliable test results. The test application of ELF18 experiments are implemented in a testmethod program, which is a C++ program that is invoked in testflow of SmarTest environment and utilizes firmware commands supplied by Agilent to manipulate the tester operations. The procedure for test application is depicted in Figure 8.

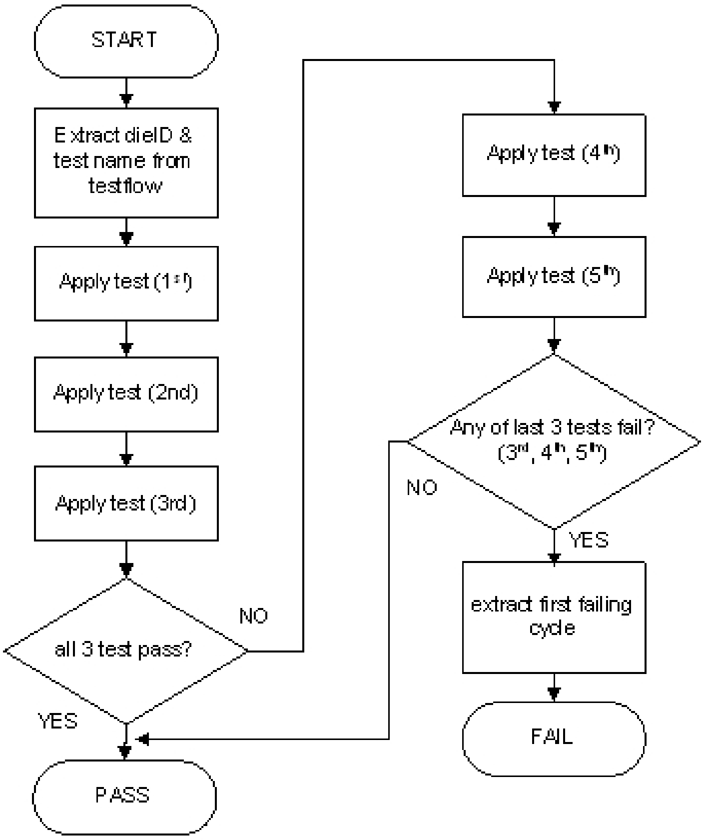

Figure 8: Test Application Procedure

There were few cores that show inconsistent results when same test set is applied multiple times. To avoid unreliable results due to this inconsistency, each test set is applied 3 times. If all three test applications pass, the core is accepted as a good core. However, if any of the first three tests fails, the same test is applied two more times. If at least one test application from the last three test application fails, the core is rejected. Relevant firmware commands used in test application testmethod are listed below.

FUNCTIONAL_TEST() : apply functional test
GET_FUNCTIONAL_TEST_RESULT() : get pass/fail test result
CHER? : get list of failing pins
ERCT? : get total number of failing cycles
ERCC? : get number of failing cycles per pin
ERCY? : get failing cycles for a given pin(s)

3.6. *Failing Cycle Logging*

To complete the defect classifications (section 5), recording all the failing cycles of all the pins is required. However, only 1,024 failing cycles can be logged in a testflow under SmarTest environment. To overcome this problem, a new testmethod is written to implement the failing cycle logging. The procedure implemented in the testmethod is described in Figure 9.

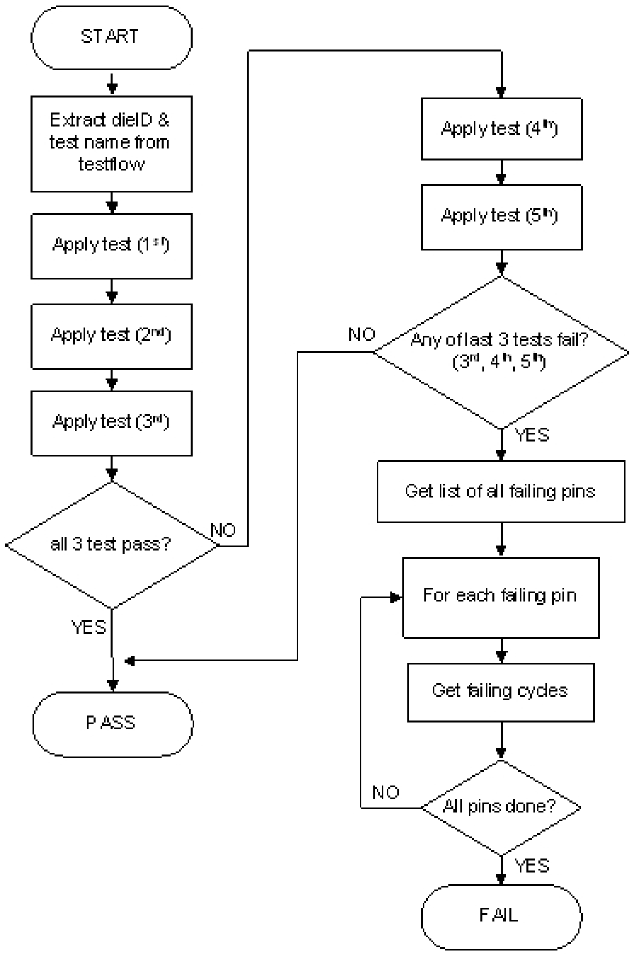

Figure 9: Failing Cycle Logging Procedure

In this testmethod, the same set of firmware commands are used as in test application testmethod. However, the tester memory can hold only finite number of failing cycles. If the number of failing cycles exceeds the maximum number the tester memory can hold, the test is applied again to obtain the rest of the failing cycles. Firmware command 'SQGB ACQF' is used to change the range of cycles the tester memory logs.

4. Failure Modes

In the ELF18 experiments, some unusual failure traces were obtained and this section discusses how these failures are found and how they behave.

4.1. Inconsistent Bits

The inconsistent bits (a.k.a. flakey bits) sometimes fail and sometimes pass the test when it is applied multiple times with the same test condition (same test set, speed and temperature) on contrary to the consistent failing bits that fail the test all the time. They are discovered in the course of test set debug when a test set is applied multiple times.

To investigate the cause of the inconsistent bits, an experiment was performed that varies the test application speed of the same test sets. A transition test set is applied at 100MHz application frequency ('Fast' speed for transition tests in the ELF18 experiment) and then applied at 101MHz and 99MHz. The results for scan flip-flops with inconsistent bits are as following.

Test @ 99MHz: Passing
Test @ 100MHz: Inconsistent failing
Test @ 101MHz: Consistent failing

The experiment is depicted in Figure 10. When the propagation delay of the signal is about 10ns, 101MHz test (period: 9.9ns) fails consistently and 99MHz test (period: 10.1ns) passes consistently. However, 100MHz test (period: 10ns) shows an inconsistent result. The inconsistent bits are believed to be caused by the inaccuracy of the timing edge placement of the tester (jitter on signal transition timing). Inconsistency happens when the transition of the clock signal is expected to be placed very close to the D-input signal transition of a flip-flop (100MHz, test in the experiment). In this case, the clock transition occurs sometimes before the input signal transition and sometimes after the input signal transition. Hence, the flip-flop captures sometimes incorrect value and sometimes correct value. Further experiments and discussions on inconsistent bits will be presented in other document.

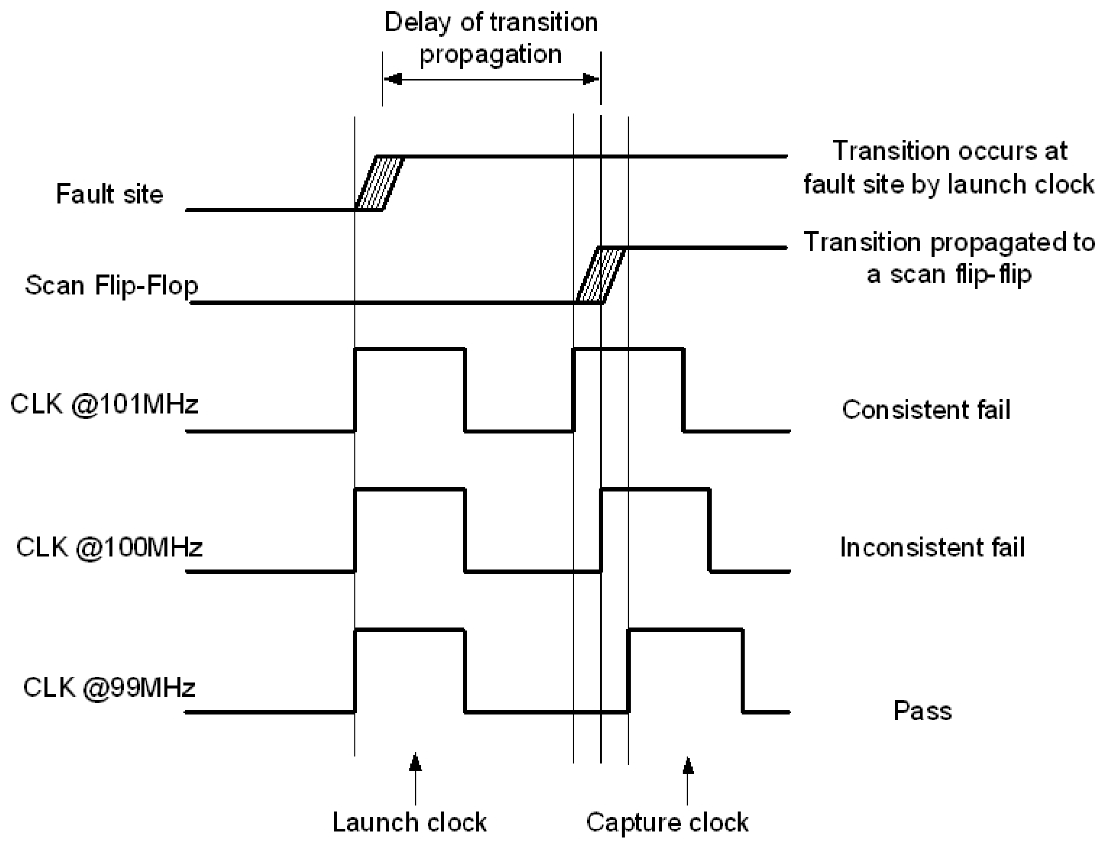

Figure 10: Experiments to Detect Inconsistent Failing Bits

The inconsistent bits co-exist with consistent failing bits in some cores and there are cores containing only inconsistent bits at the specified test frequency. The cores with only inconsistent bits can cause inconsistent test results; the core sometimes accepted and sometimes rejected. To resolve this problem, the test application of ELF18 is modified. In the ELF18 experiment, each test set is applied many times and a core is declared as defective only if it fails the test three times in a row (section 3.5).

5. Defect Classifications

To understand how the defects are behaving, failing cores are classified into categories according to their defects' behavior in previous ELF-Murphy experiments at Center for Reliable Computing [McCluskey 04]. The same classification is performed on ELF18 test chips. This section presents the categories of defects and the techniques to classify the defects.

5.1. Timing and Sequence Dependent Defects

The timing and sequence dependent defects are defects that change their behavior when the timing of test application and/or sequence of test vectors are varied. In ELF18 experiments, all the test sets are applied at three different speeds (fast, rated and slow). Therefore, to find timing and sequence dependent defects, output responses of different application speeds of the same test set are compared. If there is at least one failing bit presented in a response of one test application speed and does not show up in the other test speed, the defect is classified as timing and sequence dependent defect.

5.2. Sequence Dependent Only Defects

The sequence dependent only defects are defects that change their behavior depending on the order in which the test vectors are applied, independent of test application speed. In previous experiments of Murphy and ELF35 test chips, sequence dependent only defects are identified by applying the same stuck-at test set in varied order (normal order, reverse order, 0-bit pattern inserted between patterns, 1-bit pattern inserted between patterns, bit-complemented pattern inserted between patterns and one-bit shifted pattern inserted between patterns) and comparing their output responses [McCluskey 04]. This experiment essentially changes the previous circuit states by changing the previous test vectors. The experiment could be performed on Murphy and ELF35 test chips because the test chips include combinational cores where previous circuit state is same as the previous test vector.

However, the same technique cannot be used in ELF18 because the RDMR core is a fully-scanned sequential core. The Application of a test vector to a full-scanned sequential core requires shift operations, which alter the circuit states at each shift. For example, the previous state of a fully-scanned circuit when a stuck-at test vector is applied is one-bit shifted version of the same test vector.

In order to overcome this limitation, a new method of detecting sequence dependent defect is developed. It uses two transition test sets that have identical capture vectors but with different launch methods. One test uses the launch-on-capture (LOC test) scheme while the other uses the launch-on-shift (LOS test) scheme.

This method exploits the fact that the capture vectors of launch-on-capture and the launch-on-shift have different previous circuit states. The LOC test pattern includes two test vectors (launch vector and capture vector) and the previous circuit state of the capture vector (2nd vector) is the same as the launch vector (1st vector). On the other hand, the previous state of the capture vector of the LOS test is the one-bit shifted version of itself. When the both tests are applied to a core and they produce different output responses, the core is classified as sequence dependent core because the defect in the core changed the output responses based on the previous states of the identical capture vectors.

However, since the LOC test applies vectors by applying two capture clock pulses; it is possible for the 1st vector (launch vector) to capture some defects. In this case, the capture vector of the LOC test would not be the same vector as the original LOC test. Therefore another one-capture test (reference test, REF test) is generated that applies the launch vector (1st vector of the LOC test) only. If this test fails, it indicates that the capture vector in LOC is not the same as the capture vector in LOS and therefore it invalidates the comparison of LOC and LOS tests. The following figure introduces the conceptual view of the experiment.

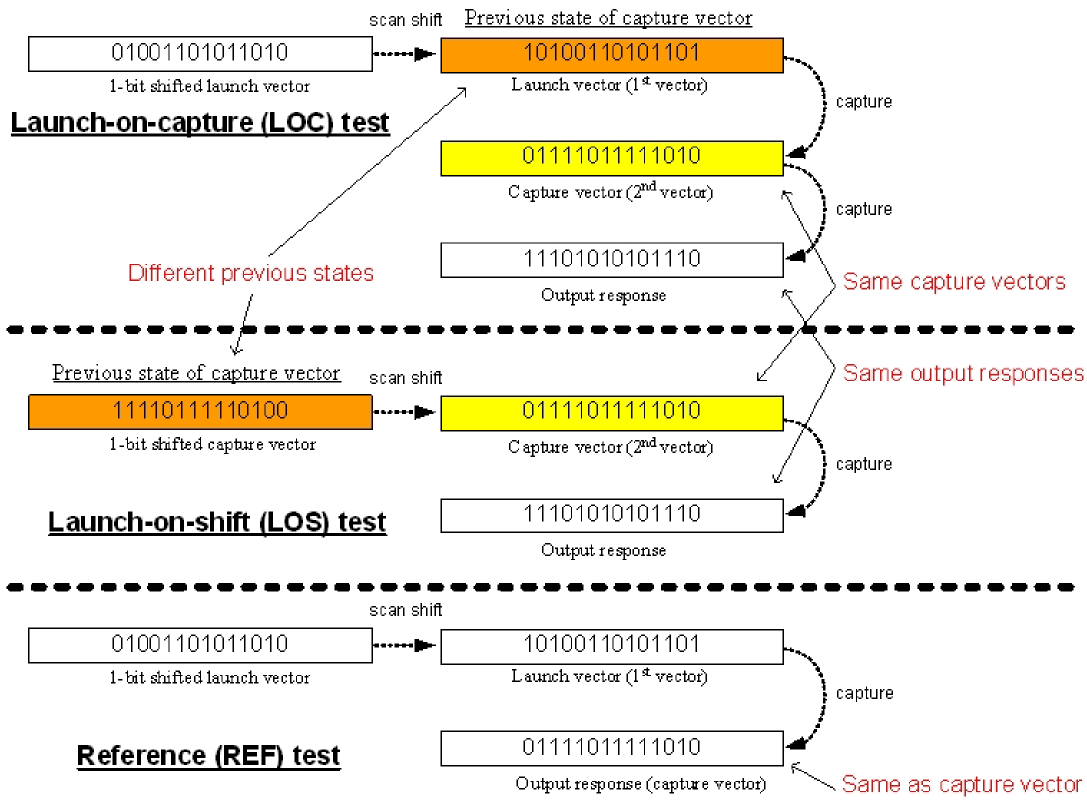

Figure 11: Sequence Dependent Defect Detection

Unlike the sequence dependent experiments in Murphy and ELF35 where 6 different previous state variations are applied, the variations of previous states in this scheme are limited to 2. Hence, to increase the chance of detecting sequence dependent defect, transition test set that contains the most number of test vectors (transition 15 Ndetect test: 18,726 vectors) are used in ELF18 experiment.

To make the LOC test, the original test set (transition 15 Ndetect) is used without any modification and the capture vector of this original test is extracted to generate LOS test. Also, one capture clock is removed from the original test set to make REF test.

In ELF18 RDMR core, launch-on-shift test cannot be applied at high speed because the scan enable signal could not propagate fast enough. Hence, all the three tests (LOC, LOS, and REF) are applied at single stuck-at test application speed.

Output responses (failing pins and failing cycles) of all three tests are recorded and compared per vector bases if REF test vector is passing. If REF test is failing, the test results of this vector are discarded.

5.3. Timing Independent Combinational (TIC) Defects

Timing independent combinational defects are defects that do not change output responses to a test even when the test application speeds or sequences of tests are varied. In ELF18 experiments, all the defective cores that are not recognized as timing dependent or sequence dependent are timing independent combinational cores.

5.4. Defect Classification Results

Total of 2,777 ELF18 test chips are tested and 2,176 chips passed all the tests that are applied. Table 8 lists the bin names and their description as well as their chip counts.

Table 8: ELF18 Bins and Chip Counts

	Binning	Description	Chip count
Good Parts	Bin1	Good Parts	2,176
Defective Parts	Bin2	ctl core fail	58
	Bin3a	**RDMR core fail (logic fail)**	**259**
	Bin3b	RDMR core fail (scan chain fail)	149
	Bin4	Weak suspects (Iddq/VLV fail)	71
	Bin8	Contact fail	64
Total chip count			2,777

259 chips (bin3a) failed at least one of the structural test sets and in these 259 chips, 483 cores are identified as defective cores. These 483 cores are classified according to their defective behaviors. The 137 cores out of 483 cores (28%) did not have any inconsistent failing bits, but 346 cores (71%) had at least one inconsistent failing bit identified by one of the test sets applied. The defect classifications results are shown in following table and figure.

Table 9: Defect Classification Results

Defect class	Consistency	# of cores	Percentage
Timing and sequence dependent	Consistent	87	18.01%
	Inconsistent	342	70.81%
	Total	429	88.82%
Sequence dependent only	Consistent	3	0.62%
	Inconsistent	1	0.21%
	Total	4	0.83%
Timing independent combinational	Consistent	47	9.73%
	Inconsistent	3	0.62%
	Total	50	10.35%

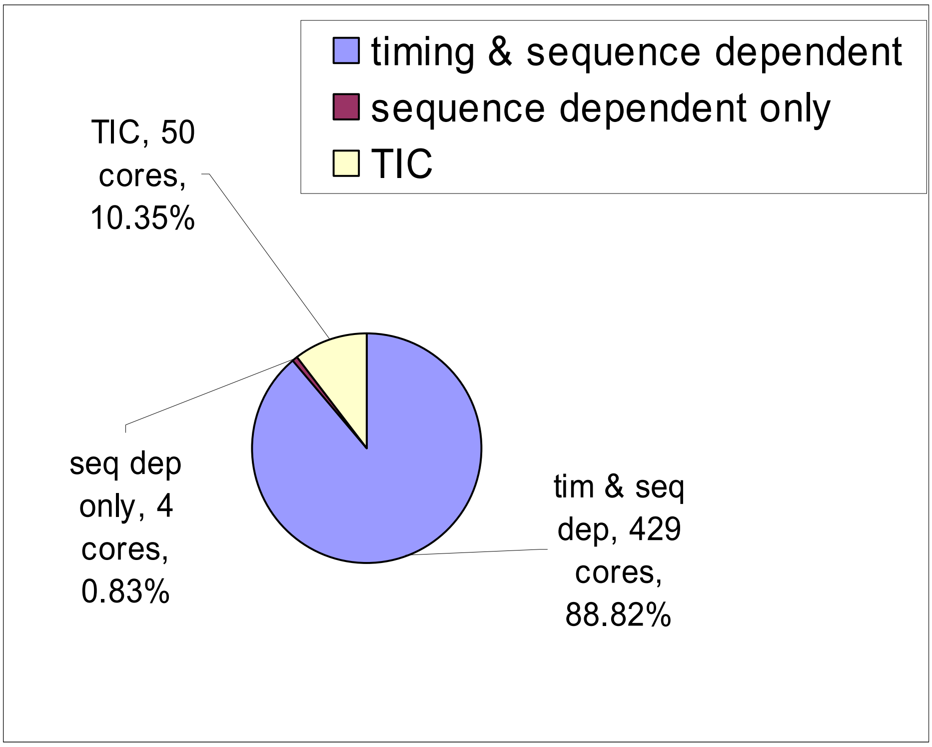

Figure 12: ELF18 Defect Classification

6. Summary

In this report, ELF18 test setup and classification results are presented. The ELF18 test chips and cores being tested are introduced in section 2. The tester setup for ELF18 test chips is presented in section 3; especially on how and why test program is designed the way it is. There is some failure modes in ELF18 not mentioned in previous ELF-Murphy experiments and these failure modes are described in section 4. Finally, the classification methodology and results are presented in section 5.

Reference

[Chang 96]Chang, J., and E.J. McCluskey, "Quantitative Analysis of Very-Low-Voltage Testing," VLSI Test Symposium, pp.332-337, 1996.

[Chang 98] Chang, J. and et. al., "Analysis of Pattern-dependent and Timing-dependent Failures in an Experimental Test Chip," Proc. ITC, 1998.

[Franco 95] Franco, P. and et. al., "An Experimental Chip to Evaluate Test Techniques Chip and Experiment Design," Proc. ITC, pp.653-662, 1995.

[Kievits 98] Keivits, P., E. Lambers, C. Moerman and R. Woudsma, "R.E.A.L. DSP Technology for Telecom Broadband Processing," Proc. Intl. Conf. Signal Processing Applications and Technology, 1998.

[Li 99] Li, J.C.M., J.T.-Y. Chang, C.W. Tseng, and E.J. McCluskey, "ELF35 Experiment - Chip and Experiment Design," CRC TR 99-3, Oct. 1999.

[Ma 95] Ma, S.C., P. Franco, and E.J. McCluskey, "An Experimental Chip To evaluate Test Techniques Experimental Results," Proc. ITC, pp.663-672, 1995.

[McCluskey 04] McCluskey, Edward, Al-Yamani, A., Li, J.C.-M.,Chao-Wen Tseng, Volkerink, E., Ferhani, F.-F., Li, E., and Mitra, S., "ELF-Murphy data on defects and test sets," Proc. ITC, pp.16-22, 2004.

[Mitra 04] Mitra, S., Volkerink, E., McCluskey, E.J., Eichenberger, S., "Delay defect screening using process monitor structures," VLSI Test Symposium, pp. 43-48, 2004.

Appendix

Table 10: RDMR Core Pin and Tester Channel Mapping

wgl_name	swav_name	swav_pin	spec_name	spec_pin	connect	93k_ch	pin_type
Clk	CLK1	1	clk1_p	47	47	10315	input
Cs	CTL_EN	2	ctlen	66	**70**	10506	input
Gb	CTL_D	3	ctld	67	**71**	10507	input
Tc	DIG_D24	4	dig_d24	80	80	10516	input
td[11]	DIG_D11	5	dig_d11	14	14	10114	input
td[10]	DIG_D10	6	dig_d10	13	13	10113	input
td[9]	DIG_D9	7	dig_d9	12	12	10112	input
td[8]	DIG_D8	8	dig_d8	11	11	10111	input
td[7]	DIG_D7	9	dig_d7	8	8	10108	input
td[6]	DIG_D6	10	dig_d6	7	7	10107	input
td[5]	DIG_D5	11	dig_d5	6	6	10106	input
td[4]	DIG_D4	12	dig_d4	5	5	10105	input
td[3]	DIG_D3	13	dig_d3	4	4	10104	input
td[2]	DIG_D2	14	dig_d2	3	3	10103	input
td[1]	DIG_D1	15	dig_d1	2	2	10102	input
td[0]	DIG_D0	16	dig_d0	1	1	10101	input
Btd	DIG_D12	17	dig_d12	15	15	10115	input
tq[11]	DIG_Q11	18	dig_q11	39	39	10307	output
tq[10]	DIG_Q10	19	dig_q10	33	33	10301	output

tq[9]	DIG_Q9	20	dig_q9	32	32	10216	output
tq[8]	DIG_Q8	21	dig_q8	31	31	10215	output
tq[7]	DIG_Q7	22	dig_q7	30	30	10214	output
tq[6]	DIG_Q6	23	dig_q6	27	27	10211	output
tq[5]	DIG_Q5	24	dig_q5	26	26	10210	output
tq[4]	DIG_Q4	25	dig_q4	25	25	10209	output
tq[3]	DIG_Q3	26	dig_q3	24	24	10208	output
tq[2]	DIG_Q2	27	dig_q2	23	23	10207	output
tq[1]	DIG_Q1	28	dig_q1	20	20	10204	output
tq[0]	DIG_Q0	29	dig_q0	18	18	10202	output
btq	DIG_Q12	30	dig_q12	40	40	10308	output
timeout_rdmr[1]	TIMEOUT	31	time_out	58	58	10410	output
freq_rdmr[1]	**FREQOUT**	32	freqout	62	62	10414	output
timein_rdmr	TIMEIN	33	timein	71	**73**	10509	input
	ACLK	34	a_clk	73	**75**	10511	input
	CLK2	35	clk2_p	41	41	10309	input
	ADATAIN	36	adatain	72	**74**	10510	input
	CTL_TC	37	ctltc	77	77	10513	input
	DIG_D23	38	dig_d23	65	**67**	10503	input
	DIG_D22	39	dig_d22	64	**66**	10502	input
	DIG_D21	40	dig_d21	45	**65**	10501	input
	DIG_D20	41	dig_d20	44	**64**	10416	input
	DIG_D19	42	dig_d19	37	37	10305	input
	DIG_D18	43	dig_d18	36	36	10304	input
	DIG_D17	44	dig_d17	35	35	10303	input
	DIG_D16	45	dig_d16	34	34	10302	input
	DIG_D15	46	dig_d15	19	19	10203	input
	DIG_D14	47	dig_d14	17	17	10201	input
	DIG_D13	48	dig_d13	16	16	10116	input
	TRIGGER	49			81	10601	(input)
	ADATAOUT	50	adataout	63	63	10415	output
	BGVOUT	51	bgv_out	59	59	10411	output
	CTL_Q	52	ctl_qout	53	53	10405	output
	DIG_Q16	53	dig_q16	52	52	10404	output
	DIG_Q15	54	dig_q15	49	49	10401	output
	DIG_Q14	55	dig_q14	48	48	10316	output
	DIG_Q13	56	dig_q13	46	46	10314	output
	VVCO	57	vvco	70	**72**	10508	output
col_rdmr[2]	COL2						input
col_rdmr[1]	COL1						input
col_rdmr[0]	COL0						input
row_rdmr	ROW						input
timeout_rdmr[2]	TIMEOUT2						output
timeout_rdmr[1]	TIMEOUT1						output
timeout_rdmr[0]	TIMEOUT0						output
freq_rdmr[2]	FREQ2						output
freq_rdmr[1]	FREQ1						output
freq_rdmr[0]	FREQ0						output

RDMR Core Timing Setup File

```
hp93000,timing,0.1
PCLK 1,1,20,(@)
CLKR 1,20,10
BWDF
1,N,N,N,N,N,N,(DIG_D0,DIG_D1,DIG_D2,DIG_D
3,DIG_D4,DIG_D5,DIG_D6,DIG_D7,DIG_D8,DI
G_D9,DIG_D10,DIG_D11,DIG_D12,DIG_D13,DI
G_D14,DIG_D15,DIG_D16,DIG_D17,DIG_D18,D
IG_D19,CLK2,CLK1,DIG_D20,DIG_D21,DIG_D2
2,DIG_D23,CTL_EN,CTL_D,TIMEIN,ADATAIN,A
CLK,CTL_TC,DIG_D24,TRIGGER)
ETIM
1,1,0,0,0,0,0,0,(DIG_D0,DIG_D1,DIG_D2,DIG_D
3,DIG_D4,DIG_D5,DIG_D6,DIG_D7,DIG_D8,DI
G_D9,DIG_D10,DIG_D11,DIG_D12,DIG_D13,DI
G_D14,DIG_Q0,DIG_D15,DIG_Q1,DIG_Q2,DIG
_Q3,DIG_Q4,DIG_Q5,DIG_Q6,DIG_Q7,DIG_Q8,
DIG_Q9,DIG_Q10,DIG_D16,DIG_D17,DIG_D18,
DIG_D19,DIG_Q11,DIG_Q12,CLK2,DIG_Q13,C
LK1,DIG_Q14,DIG_Q15,DIG_Q16,CTL_Q,TIME
OUT,BGVOUT,FREQOUT,ADATAOUT,DIG_D20
,DIG_D21,DIG_D22,DIG_D23,CTL_EN,CTL_D,V
VCO,TIMEIN,ADATAIN,ACLK,CTL_TC,DIG_D24
,TRIGGER)
DCDT 1,""
TSUX 1,1
EQSP TIM,WVT,#9000007958WAVETBL
"Formats"

DISPLAY multi

PINS CLK1
WFDF 0 D11 F0N . . . .
      0   0 . . .
      1   1 . . .
BWDF . . . . . .
DCDF cyc  0

PINS CLK2
WFDF 0 D11 F0N . . . .
0   0 . . .
1   1 . . .
BWDF . . . . . .
DCDF cyc 0

PINS ACLK
WFDF 0 D11 F0N . . . .
0   0 . . .
1   1 . . .
BWDF . . . . . .
DCDF cyc 0

PINS TRIGGER
WFDF 0 D11 . . . . .
0   0 . . .
1   1 . . .
BWDF . . . . . .
DCDF cyc 0

PINS all_dnrz
WFDF 0 D11 . . . . .
0   0 . . .
1   1 . . .
BWDF . . . . . .
DCDF cyc 0

PINS DIG_D21
WFDF 0 D11 . . . . .
0   0 . . .
1   1 . . .
BWDF . . . . . .
DCDF cyc 0

PINS all_out
WFDF 0 EE1 . . . . .
0   . . L .
1   . . H .
2   . . X .
3   . . M .
DCDF cyc 0


WAVETBL  "RdmC"

DISPLAY multi

PINS CLK1
WFDF 0 D11 F0N . . . .
      0   0 . . .
      1   1 . . .
BWDF . . . . . .
DCDF cyc  0

PINS CLK2
WFDF 0 D11 F0N . . . .
0   0 . . .
1   1 . . .
BWDF . . . . . .
DCDF cyc 0

PINS DIG_D21
WFDF 0 D11 F0N . . . .
      0   0 . . .
      1   1 . . .
BWDF . . . . . .
DCDF cyc  0

PINS ACLK
WFDF 0 D11 F0N . . . .
0   0 . . .
1   1 . . .
BWDF . . . . . .
```

```
DCDF cyc 0

PINS TRIGGER
WFDF 0 D11 . . . . .
0   0 . . .
1   1 . . .
BWDF . . . . . .
DCDF cyc 0


PINS all_dnrz
WFDF 0 D11 . . . . .
0   0 . . .
1   1 . . .
BWDF . . . . . .
DCDF cyc 0


PINS all_out
WFDF 0 EE1 . . . . .
0   . . L .
1   . . H .
2   . . X .
3   . . M .
DCDF cyc 0


##########################################
################
# RDMR wave table
#
# edige d1, d3, d4, d6 tristate (D11 ACTION)
#

WAVETBL  "RDMR_wavetable"

DISPLAY multi

PINS CLK1
WFDS  0 "d1:D11 d2:F0N"
0   0       .  .        .
1   1       .  .        .
WFDS  1 "d3:D11 d5:F0N"
2   0       .  .        .
3   1       .  .        .
WFDS  2 "d4:D11 d7:F0N"
4   0       .  .        .
5   1       .  .        .
WFDS  3 "d6:D11 d8:F0N"
6   0       .  .        .
7   1       .  .        .
BWDS ""
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3


PINS CLK2
WFDS  0 "d4:D11 d7:F0N"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11 d7:F0N"
2   0       .  .        .
3   1       .  .        .
WFDS  2 "d4:D11 d7:F0N"
4   0       .  .        .
5   1       .  .        .
WFDS  3 "d4:D11 d7:F0N"
6   0       .  .        .
7   1       .  .        .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS ACLK
WFDS  0 "d4:D11 d7:F0N"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11 d7:F0N"
2   0       .  .        .
3   1       .  .        .
WFDS  2 "d4:D11 d7:F0N"
4   0       .  .        .
5   1       .  .        .
WFDS  3 "d4:D11 d7:F0N"
6   0       .  .        .
7   1       .  .        .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS TRIGGER
WFDF 0 "d4:D11"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11"
2   0       .  .        .
3   1       .  .        .
WFDS  2 "d4:D11"
4   0       .  .        .
5   1       .  .        .
WFDS  3 "d4:D11"
6   0       .  .        .
7   1       .  .        .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS all_dnrz
WFDF 0 "d4:D11"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11"
2   0       .  .        .
3   1       .  .        .
WFDS  2 "d4:D11"
4   0       .  .        .
```

```
5   1        .  .        .
WFDS  3 "d4:D11"
6   0        .  .        .
7   1        .  .        .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS DIG_D21
WFDF 0 "d4:D11"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11"
2   0        .  .        .
3   1        .  .        .
WFDS  2 "d4:D11"
4   0        .  .        .
5   1        .  .        .
WFDS  3 "d4:D11"
6   0        .  .        .
7   1        .  .        .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS all_out
WFDF 0 EE1 . . . . .
0   . . L .
1   . . H .
2   . . X .
3   . . M .
WFDF  1 . EE1 . . . .
4   . . L .
5   . . H .
6   . . X .
7   . . M .
WFDF  2 . EE1 . . . .
8   . . L .
9   . . H .
A   . . X .
B   . . M .
WFDF  3 . EE1 . . . .
C   . . L .
D   . . H .
E   . . X .
F   . . M .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

#######################################
###############
# RDMR VLV wavetable
#
# exactly same as RDMR wavetable, but, cy0 is
same as cyc
#
# edige d1, d3, d4, d6 tristate (D11 ACTION)
#

WAVETBL  "RDMR_VLV_wavetable"

DISPLAY multi

PINS CLK1
WFDS  0 "d1:D11 d2:F0N"
0   0        .  .        .
1   1        .  .        .
WFDS  1 "d1:D11 d2:F0N"
2   0        .  .        .
3   1        .  .        .
WFDS  2 "d4:D11 d7:F0N"
4   0        .  .        .
5   1        .  .        .
WFDS  3 "d6:D11 d8:F0N"
6   0        .  .        .
7   1        .  .        .
BWDS ""
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3


PINS CLK2
WFDS  0 "d4:D11 d7:F0N"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11 d7:F0N"
2   0        .  .        .
3   1        .  .        .
WFDS  2 "d4:D11 d7:F0N"
4   0        .  .        .
5   1        .  .        .
WFDS  3 "d4:D11 d7:F0N"
6   0        .  .        .
7   1        .  .        .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS ACLK
WFDS  0 "d4:D11 d7:F0N"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11 d7:F0N"
2   0        .  .        .
3   1        .  .        .
WFDS  2 "d4:D11 d7:F0N"
4   0        .  .        .
5   1        .  .        .
WFDS  3 "d4:D11 d7:F0N"
6   0        .  .        .
7   1        .  .        .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
```

```
DCDF cy1    2
DCDF cy2    3

PINS TRIGGER
WFDF 0 "d4:D11"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11"
2   0        .  .         .
3   1        .  .         .
WFDS  2 "d4:D11"
4   0        .  .         .
5   1        .  .         .
WFDS  3 "d4:D11"
6   0        .  .         .
7   1        .  .         .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS all_dnrz
WFDF 0 "d4:D11"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11"
2   0        .  .         .
3   1        .  .         .
WFDS  2 "d4:D11"
4   0        .  .         .
5   1        .  .         .
WFDS  3 "d4:D11"
6   0        .  .         .
7   1        .  .         .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS DIG_D21
WFDF 0 "d4:D11"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11"
2   0        .  .         .
3   1        .  .         .
WFDS  2 "d4:D11"
4   0        .  .         .
5   1        .  .         .
WFDS  3 "d4:D11"
6   0        .  .         .
7   1        .  .         .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS all_out
WFDF 0 EE1 . . . . .
0   . . L .
1   . . H .
2   . . X .
3   . . M .
WFDF  1 . EE1 . . . .
4   . . L .
5   . . H .
6   . . X .
7   . . M .
WFDF  2 . EE1 . . . .
8   . . L .
9   . . H .
A   . . X .
B   . . M .
WFDF  3 . EE1 . . . .
C   . . L .
D   . . H .
E   . . X .
F   . . M .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

#####################################################
###############
# RDMR wave table
#
# edige d1, d3, d4, d6 tristate (D11 ACTION)
#

WAVETBL  "RDMR_multicapture"

DISPLAY multi

PINS CLK1
WFDS  0 "d4:D11 d7:F0N"
0   0        .  .         .
1   1        .  .         .
WFDS  1 "d4:D11 d4:F0N"
2   0        .  .         .
3   1        .  .         .
WFDS  2 "d1:D11 d2:F0N"
4   0        .  .         .
5   1        .  .         .
WFDS  3 "d3:F1N d5:F0N d6:D11 d8:F0N"
6   0        .  .         .
7   1        .  .         .
BWDS ""
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3


PINS CLK2
WFDS  0 "d4:D11 d7:F0N"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11 d7:F0N"
2   0        .  .         .
3   1        .  .         .
```

```
WFDS  2 "d4:D11 d7:F0N"
4   0       .  .        .
5   1       .  .        .
WFDS  3 "d4:D11 d7:F0N"
6   0       .  .        .
7   1       .  .        .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS ACLK
WFDS  0 "d4:D11 d7:F0N"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11 d7:F0N"
2   0       .  .        .
3   1       .  .        .
WFDS  2 "d4:D11 d7:F0N"
4   0       .  .        .
5   1       .  .        .
WFDS  3 "d4:D11 d7:F0N"
6   0       .  .        .
7   1       .  .        .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS TRIGGER
WFDF 0 "d4:D11"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11"
2   0       .  .        .
3   1       .  .        .
WFDS  2 "d4:D11"
4   0       .  .        .
5   1       .  .        .
WFDS  3 "d4:D11"
6   0       .  .        .
7   1       .  .        .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS all_dnrz
WFDF 0 "d4:D11"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11"
2   0       .  .        .
3   1       .  .        .
WFDS  2 "d4:D11"
4   0       .  .        .
5   1       .  .        .
WFDS  3 "d4:D11"
6   0       .  .        .
7   1       .  .        .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS DIG_D21
WFDF 0 "d4:D11"
0   0 . . .
1   1 . . .
WFDS  1 "d4:D11"
2   0       .  .        .
3   1       .  .        .
WFDS  2 "d4:D11"
4   0       .  .        .
5   1       .  .        .
WFDS  3 "d4:D11"
6   0       .  .        .
7   1       .  .        .
BWDF . . . . . .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3

PINS all_out
WFDF 0 EE1 . . . . .
0   . . L .
1   . . H .
2   . . X .
3   . . M .
WFDF  1 EE1 . . . . .
4   . . L .
5   . . H .
6   . . X .
7   . . M .
WFDF  2 EE1 . . . . .
8   . . L .
9   . . H .
A   . . X .
B   . . M .
WFDF  3 EE1 . . . . .
C   . . L .
D   . . H .
E   . . X .
F   . . M .
DCDF cyc    0
DCDF cy0    1
DCDF cy1    2
DCDF cy2    3
@
EQSP TIM,EQN,#9000002925EQNSET  1
"Default"


SPECS
Period       [ns]
RiseClk1     [%]
FallClk1     [%]
RiseClk2     [%]
FallClk2     [%]
```

```
RiseDIG_D21  [%]
FallDIG_D21  [%]
RiseAclk     [%]
FallAclk     [%]
DnrzDelay    [%]
Strobe       [%]
Trigger      [%]
Launch_freq        [MHZ]
Launch_freq2       [MHZ]

EQUATIONS

##############################
# RDMR timing
# scam_half_period
halfPerS = Period / 2

# for stuck-at
# launch period
LPer = 1000/Launch_freq
halfPer = LPer / 2

# for transition
# launch clock width
LPer2 = 1000/Launch_freq2
halfPer2 = LPer2 / 2


TIMINGSET 1 "General"
period = Period

PINS CLK1
e1= (RiseClk1/100) * Period
e2= (FallClk1/100) * Period

PINS CLK2
e1= (RiseClk2/100) * Period
e2= (FallClk2/100) * Period

PINS ACLK
e1 = (RiseAclk/100) * Period
e2 = (FallAclk/100) * Period

PINS DIG_D21
e1 = (RiseDIG_D21/100) * Period

PINS all_dnrz
e1 = (DnrzDelay/100) * Period

PINS TRIGGER
e1= (Trigger/100) * Period

PINS all_out
r1 = (Strobe/100) * Period


TIMINGSET 2 "RDMC"
period = Period

PINS CLK1
d1= 0.000 * Period
d2= 0.125 * Period
d3= 0.250 * Period
d4= 0.500 * Period
d5= 0.375 * Period
d6= 0.625 * Period
d7= 0.750 * Period
d8= 0.875 * Period

PINS CLK2
e1= (RiseClk2/100) * Period
e2= (FallClk2/100) * Period

PINS ACLK
e1 = (RiseAclk/100) * Period
e2 = (FallAclk/100) * Period

PINS DIG_D21
e1 = (RiseDIG_D21/100) * Period
e2 = (FallDIG_D21/100) * Period

PINS all_dnrz
e1 = (DnrzDelay/100) * Period

PINS TRIGGER
e1= (Trigger/100) * Period

PINS all_out
e1 = (Strobe/100) * Period

########################################
#######
# RDMR timing
TIMINGSET 3 "RDMR"

period = Period

PINS CLK1
# for scan
d1 = 0.500 * Period
d2 = 0.825 * Period
# for ssf
d3= LPer - halfPerS
#d5= LPer - halfPerS + halfPer
d5= LPer - halfPerS + 10
# for tr, launch clkPINS CLK2
d4= Period - halfPer2 * 2
d7= Period - halfPer2
# for tr, capture clk
d6= 0
d8= halfPer2

PINS CLK2
e1= (RiseClk2/100) * Period
e2= (FallClk2/100) * Period

PINS ACLK
e1 = (RiseAclk/100) * Period
e2 = (FallAclk/100) * Period

PINS DIG_D21
e1 = (RiseDIG_D21/100) * Period

PINS all_dnrz
```

```
e1 = (DnrzDelay/100) * Period

PINS TRIGGER
e1= (Trigger/100) * Period

PINS all_out
#r1 = (Strobe/100) * Period
e1 = (Strobe/100) * Period
e2 = 0.825 * Period
e3 = 0.1 * Period


##############################################
#######
# RDMR timing
TIMINGSET 4 "RDMR_multicapture"

period = Period

PINS CLK1
# shift cycle (cyc)
d4 = 0.5 * Period
d7 = 0.825 * Period
# 1st clk in cy1
d1 = Period – 2 * halfPer2
d2 = Period – halfPer2
# 2nd clk in cy2
d3= 0
d5= halfPer2
# 3rd clk in cy2
d6= halfPer2 * 2
d8= halfPer2 * 3

PINS CLK2
e1= (RiseClk2/100) * Period
e2= (FallClk2/100) * Period

PINS ACLK
e1 = (RiseAclk/100) * Period
e2 = (FallAclk/100) * Period

PINS DIG_D21
e1 = (RiseDIG_D21/100) * Period

PINS all_dnrz
e1 = (DnrzDelay/100) * Period

PINS TRIGGER
e1= (Trigger/100) * Period

PINS all_out
#r1 = (Strobe/100) * Period
e1 = halfPer2 * 3
e2 = 0.825 * Period
e3 = 0.1 * Period
@
EQSP TIM,SPS,#9000031415


EQNSET 1 "Default"

WAVETBL "Formats"

CHECK 1period_drv 1period_rcv edgeorder_tri
edgeorder_win

SPECSET 1 "Controller timing"


# SPECNAME       *****ACTUAL*****
*****MINIMUM**** *****MAXIMUM**** UNITS
COMMENT
Launch_freq2     5
[MHZ]
Launch_freq      5
[MHZ]
RiseDIG_D21      0
[  %]
FallDIG_D21      0
[  %]
Period           200
[ ns]
RiseClk1         50
[  %]
FallClk1         75
[  %]
RiseClk2         0
[  %]
FallClk2         0
[  %]
RiseAclk         0
[  %]
FallAclk         0
[  %]
DnrzDelay        0
[  %]
Strobe           48
[  %]
Trigger          0
[  %]


WAVETBL "RDMR_wavetable"

CHECK 1period_drv 1period_rcv edgeorder_tri
edgeorder_win

SPECSET 2 "RDMR timing"


# SPECNAME       *****ACTUAL*****
*****MINIMUM**** *****MAXIMUM**** UNITS
COMMENT
Launch_freq2     90
[MHZ]
Launch_freq      30
[MHZ]
RiseDIG_D21      0
[  %]
FallDIG_D21      0
[  %]
Period           50
[ ns]
```

RiseClk1 50
[%]
FallClk1 100
[%]
RiseClk2 0
[%]
FallClk2 0
[%]
RiseAclk 0
[%]
FallAclk 0
[%]
DnrzDelay 0
[%]
Strobe 45
[%]
Trigger 0
[%]

WAVETBL "RdmC"

CHECK 1period_drv 1period_rcv edgeorder_tri edgeorder_win

SPECSET 3 "RDMC timing"

SPECNAME *****ACTUAL***** *****MINIMUM**** *****MAXIMUM**** UNITS COMMENT
Launch_freq2 5
[MHZ]
Launch_freq 5
[MHZ]
RiseDIG_D21 0
[%]
FallDIG_D21 25
[%]
Period 200
[ns]
RiseClk1 50
[%]
FallClk1 100
[%]
RiseClk2 25
[%]
FallClk2 50
[%]
RiseAclk 0
[%]
FallAclk 0
[%]
DnrzDelay 0
[%]
Strobe 48
[%]
Trigger 0
[%]

WAVETBL "Formats"

CHECK 1period_drv 1period_rcv edgeorder_tri edgeorder_win

SPECSET 4 "RAM timing"

SPECNAME *****ACTUAL***** *****MINIMUM**** *****MAXIMUM**** UNITS COMMENT
Launch_freq2 5
[MHZ]
Launch_freq 5
[MHZ]
RiseDIG_D21 0
[%]
FallDIG_D21 0
[%]
Period 200
[ns]
RiseClk1 50
[%]
FallClk1 75
[%]
RiseClk2 0
[%]
FallClk2 0
[%]
RiseAclk 0
[%]
FallAclk 0
[%]
DnrzDelay 0
[%]
Strobe 95
[%]
Trigger 0
[%]

WAVETBL "Formats"

CHECK 1period_drv 1period_rcv edgeorder_tri edgeorder_win

SPECSET 5 "ROM timing"

SPECNAME *****ACTUAL***** *****MINIMUM**** *****MAXIMUM**** UNITS COMMENT
Launch_freq2 5
[MHZ]
Launch_freq 5
[MHZ]
RiseDIG_D21 0
[%]
FallDIG_D21 0
[%]
Period 200 10 500
[ns]

RiseClk1 50
[%]
FallClk1 75
[%]
RiseClk2 0
[%]
FallClk2 0
[%]
RiseAclk 0
[%]
FallAclk 0
[%]
DnrzDelay 0
[%]
Strobe 95 50 100
[%]
Trigger 0
[%]

WAVETBL "Formats"

CHECK 1period_drv 1period_rcv edgeorder_tri edgeorder_win

SPECSET 6 "Analog timing"

SPECNAME *****ACTUAL***** *****MINIMUM**** *****MAXIMUM**** UNITS COMMENT
Launch_freq2 5
[MHZ]
Launch_freq 5
[MHZ]
RiseDIG_D21 0
[%]
FallDIG_D21 0
[%]
Period 200
[ns]
RiseClk1 0
[%]
FallClk1 0
[%]
RiseClk2 0
[%]
FallClk2 0
[%]
RiseAclk 25
[%]
FallAclk 75
[%]
DnrzDelay 0
[%]
Strobe 20
[%]
Trigger 0
[%]

WAVETBL "RDMR_VLV_wavetable"

CHECK all

SPECSET 7 "Slow Speed 150ns"

SPECNAME *****ACTUAL***** *****MINIMUM**** *****MAXIMUM**** UNITS COMMENT
Period 150
[ns]
RiseClk1 50
[%]
FallClk1 100
[%]
RiseClk2 0
[%]
FallClk2 0
[%]
RiseDIG_D21 0
[%]
FallDIG_D21 0
[%]
RiseAclk 0
[%]
FallAclk 0
[%]
DnrzDelay 0
[%]
Strobe 45
[%]
Trigger 0
[%]
Launch_freq 7
[MHZ]
Launch_freq2 20
[MHZ]

WAVETBL "RDMR_multicapture"

CHECK 1period_drv 1period_rcv

SPECSET 20 "RDMR_multiCapture"

SPECNAME *****ACTUAL***** *****MINIMUM**** *****MAXIMUM**** UNITS COMMENT
Period 50
[ns]
RiseClk1 50
[%]
FallClk1 100
[%]
RiseClk2 0
[%]
FallClk2 0
[%]
RiseDIG_D21 0
[%]
FallDIG_D21 0
[%]

RiseAclk 0
[%]
FallAclk 0
[%]
DnrzDelay 0
[%]
Strobe 20
[%]

Trigger 0
[%]
Launch_freq 36
[MHZ]
Launch_freq2 90
[MHZ]
@