Diagnosing SSD Failures during Testing

Traffic Capture and Debug Tools

Linden Hsu
SSD Testing Background

- **Problem Statement**
  - Identify root cause of SSD failures in a large volume testing – from test equipment perspective

- **Multiple components (memory, controller, PHY, etc.)**
  - System level test requires more than just pass/fail
    - System test reveals interaction issues not found during component test
    - Vendors want to know which part or parts are broken
    - Ability to reproduce the issue is vital: often takes a sequence of events with certain delays that sometimes takes days of testing with many devices

- **Large volumes**
  - Cost and space limits the use of “bench testing” diagnostic equipment
  - Scale in units: Engineering (one to a few, maybe a dozen), Qualification (hundreds), Production (thousands)
Bench Testing Equipment

- “High end bench testing equipment”
  - Protocol Analyzer
  - Serial Data Real-time Oscilloscope
  - Bit Error Ratio Generator
  - Vector Analyzer

- Pros
  - Large memory, relatively objective, sophisticated decode and protocol software

- Cons
  - Cost, space, power, signal alteration
Failure Categories

- **Initial Questions**
  - What failed? When did it fail? Why did it fail? Was the test valid? Was it the test equipment? Is it reproducible?

- **Sources of Failure**
  - Drive, operator, test program, test equipment

- **Kinds of Failure**
  - Drive: hardware, firmware
  - Operator: improper/loose connection
  - Test program: illegal or unsupported operation
  - Test equipment: hardware, firmware, software
Fault Isolation: Hardware

- Offline
  - Go/no-go self-test
  - Channel diagnostics with loopback card
    - Eye measurements to quantify channel quality
    - Different frequencies to catch different kinds of faults
    - Various patterns such as PRBS to emulate traffic, high/low frequency patterns or to exercise the channel

- Run-time
  - Temperature logging and alarms
  - Voltage/current limits
Physical/Link Layer

- **Link Negotiation**
  - Logging **link states** and **equalization negotiation** can be very helpful in debugging
    - Timestamped states can be correlated to test failures
    - Link state history can show progression into failure

- **Example**
  - Inconsistent final equalization requests for adjacent lanes to same drive
    - Lane 1 requests P4 which has zero equalization while Lane 2 requests P7 which has the highest equalization for a preset (6 dB de-emphasis, 3.5 dB pre-shoot)

```
20180723_17:59:32.020516050 [ 96]: delta= 325.244 us [ Request Preset 8 ] (Lane3, 8G, EC2)
20180723_17:59:32.020516078 [ 97]: delta= 0.028 us [ Request Preset 6 ] (Lane0, 8G, EC2)
20180723_17:59:32.020516083 [ 98]: delta= 0.005 us [ Request Preset 4 ] (Lane1, 8G, EC2)
20180723_17:59:32.020516089 [ 99]: delta= 0.006 us [ Request Preset 7 ] (Lane2, 8G, EC2)
```
Transaction Layer

- Transaction Layer capture
  - Sometimes it is helpful to see the transaction layer packets (TLP) between the tester and the drive.
    - Memory read request and corresponding completion response (or lack thereof)
    - Configuration, control and transfer commands

- Resource limitations – bandwidth and storage

- Solutions
  - Provide relatively small capture buffers for ingress/egress TLPs (buffers are small compared to those of a protocol analyzer)
    - For directed debug, enable packet filtering to extend the effective capture duration
  - Alternatively, log command/response summaries of transfer commands
## Data Collection

<table>
<thead>
<tr>
<th>Layer</th>
<th>Feature</th>
<th>Tool Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical</td>
<td>Loopback, Eye scan</td>
<td>Diagnostics</td>
</tr>
<tr>
<td>Link</td>
<td>Link state, speed, power states, lane activity, equalization, flow control</td>
<td>Capture</td>
</tr>
<tr>
<td>Transaction</td>
<td>Transaction layer packet capture/filter</td>
<td>Capture</td>
</tr>
<tr>
<td>Data transfer</td>
<td>Command, data, control registers</td>
<td>Capture</td>
</tr>
<tr>
<td>Global</td>
<td>Universal Timestamp</td>
<td>Capture</td>
</tr>
<tr>
<td>Software</td>
<td>Driver debug, test program, system health</td>
<td>Logs</td>
</tr>
</tbody>
</table>
Data Sorting

- **Log Sifter**
  Tool to pick out the relevant logs/captures that pertain to a failing DUT
## Data Analysis

<table>
<thead>
<tr>
<th>Layer</th>
<th>Feature</th>
<th>Tool Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>Self-test, loopback</td>
<td>Diagnostic</td>
</tr>
<tr>
<td>Physical</td>
<td>Equalization checker</td>
<td>Diagnostic</td>
</tr>
<tr>
<td>Link</td>
<td>Link-State checker</td>
<td>Diagnostic</td>
</tr>
<tr>
<td>Transaction</td>
<td>TLP decoder</td>
<td>Diagnostic</td>
</tr>
<tr>
<td>-</td>
<td>Comprehensive analysis</td>
<td>Diagnostic</td>
</tr>
<tr>
<td>-</td>
<td>Knowledge database</td>
<td>Documentation</td>
</tr>
</tbody>
</table>
Fault Isolation: Software

- **Software installation verification**
  - Loads missing components, reports/replaces incorrect versions of tools

- **Software utilities**
  - As a complement to GUI-based software, command line tools can allow direct access for focused debug
    - Snapshot current system state including versions and logs
    - Logging of commands, power control, link status, …
    - Running of performance tests, power cycling, …
    - Report bus enumeration and hierarchy

- **Human versus software**
  - With all the information gathered, who/what will make sense of it all?
    - Algorithm development for data collection, sorting, and analysis
Conclusions

- **Problem**
  - Drives can fail for many reasons
  - Need to differentiate between real drive issues from other issues
  - The right information is needed to isolate root cause of drive failures
  - The tools must operate within constraints

- **Solution**
  - System diagnostics which include self-test including loopback
  - Traffic monitoring and capture at multiple level to collect information
  - Post-processing to filter and sort the information
  - Analysis software to make sense of all the data