

# **Standard Test Data Format (STDF) V4 - 2007 Specification**

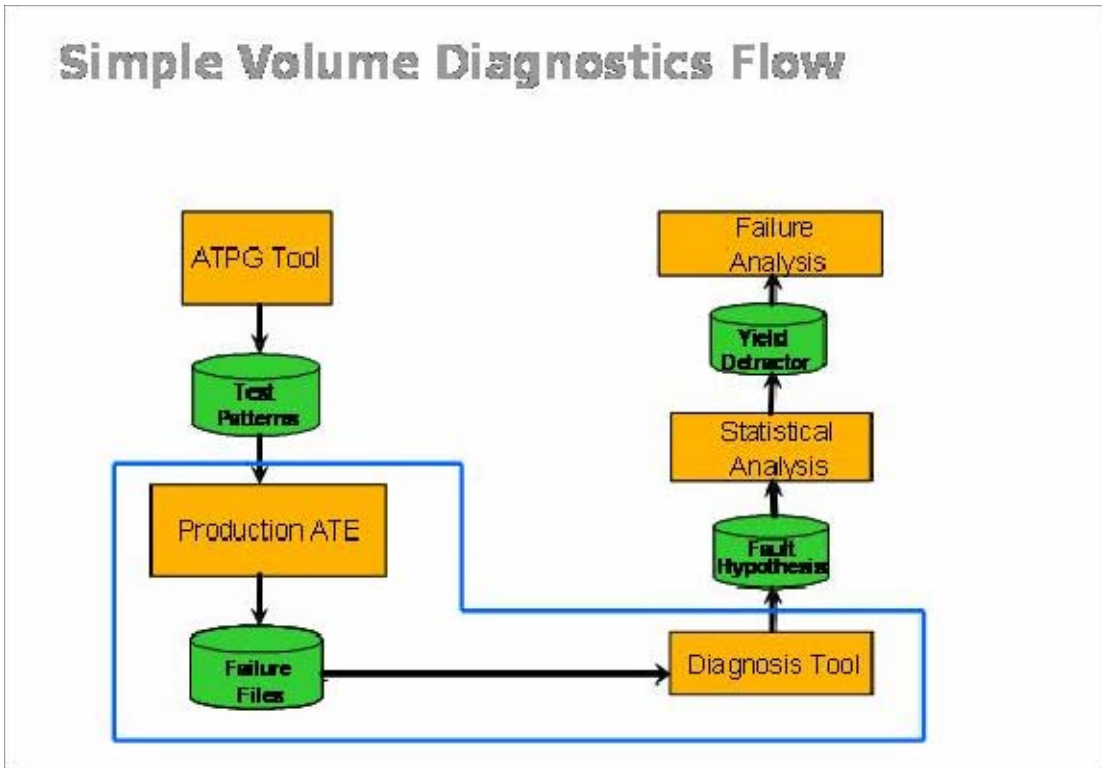
## **Purpose**

The purpose of this standard is to provide a common format for scan fail datalog specification along with necessary synchronization information enabling an efficient dataflow for volume diagnostics applications.

## **Background**

Manufacturing yield is a very important factor in the production of a semiconductor product. It is important in all phases of the production: first silicon, volume ramp and normal production. Historically the yield fallout was mainly caused by random defects and process problems, so traditional yield improvement strategies included clean room improvements and process improvements. Correspondingly the data collection from ATE for yield monitoring was focused on parametric and gross pass/fail information collection. However, advances in technology have created a situation where yield loss is now dominated by the systematic design and process interactions, which are hard to understand before the silicon implementation. To understand these interactions requires fail data collection in volume manufacturing. Moreover the design and pattern dependent nature of these issues requires fail data to be collected on the internal nodes using structural test techniques. Therefore the industry is moving toward Volume Diagnostics flow, where fail data on internal nodes is collected during volume manufacturing and processed by the diagnostic tools from the EDA vendors to identify the failing structures. The information on the failing structures is statistically analyzed to identify the yield improvement opportunities.

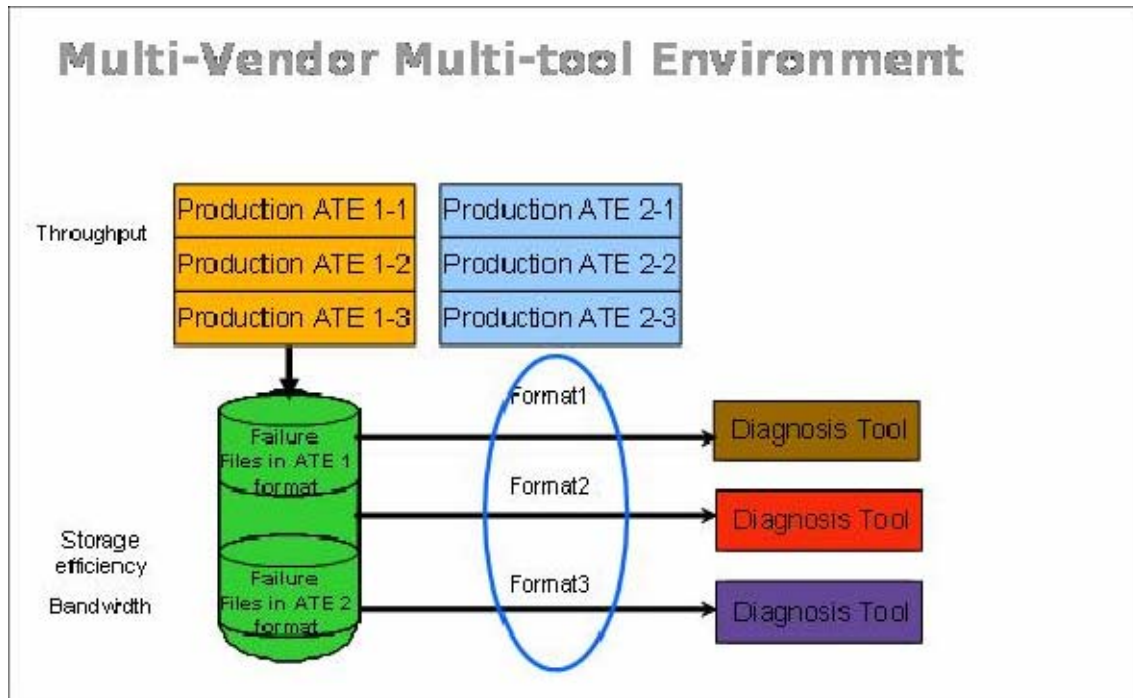
Figure 1 shows a flow for volume diagnostic mentioned above. In this flow the test patterns



**Figure 1**

generated by ATPG tools from EDA vendors are applied to the production device by the ATE of choice. The ATE then collects the fail information in failure files.

The failure information in these failure files is then used by the diagnosis tools to identify the failing structure, e.g., failing gate, failing via, failing net, etc. Currently the format in which these failure files are written depends on the ATE at hand. Each ATE vendor writes this information in a custom format.



**Figure 2**

To complicate the matters further, each diagnosis tool also has its custom input format. So a multi-ATE and multi-diagnosis tools environment for a customer looks like the one shown in Figure 2. A customer in this situation has to create an IT infrastructure to store information in multiple formats one for each ATE and then translate this information into the input format of the diagnosis tool. The responsibility for developing and supporting the translation tools can be with the EDA vendors or with ATE vendors or the end users. The situation in the EDA and the ATE cases for a multi-tool, multi-vendor environment is shown in Figure 3 and Figure 4 respectively. In either case it is an extra

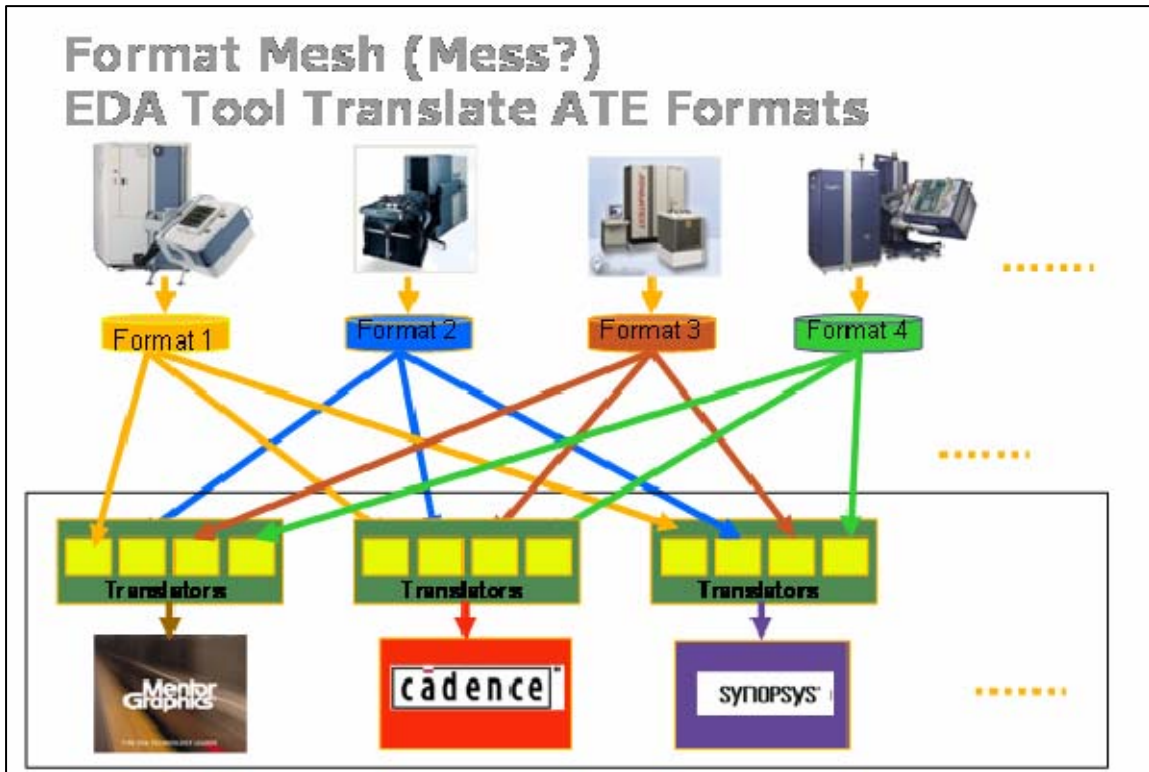


Figure 3

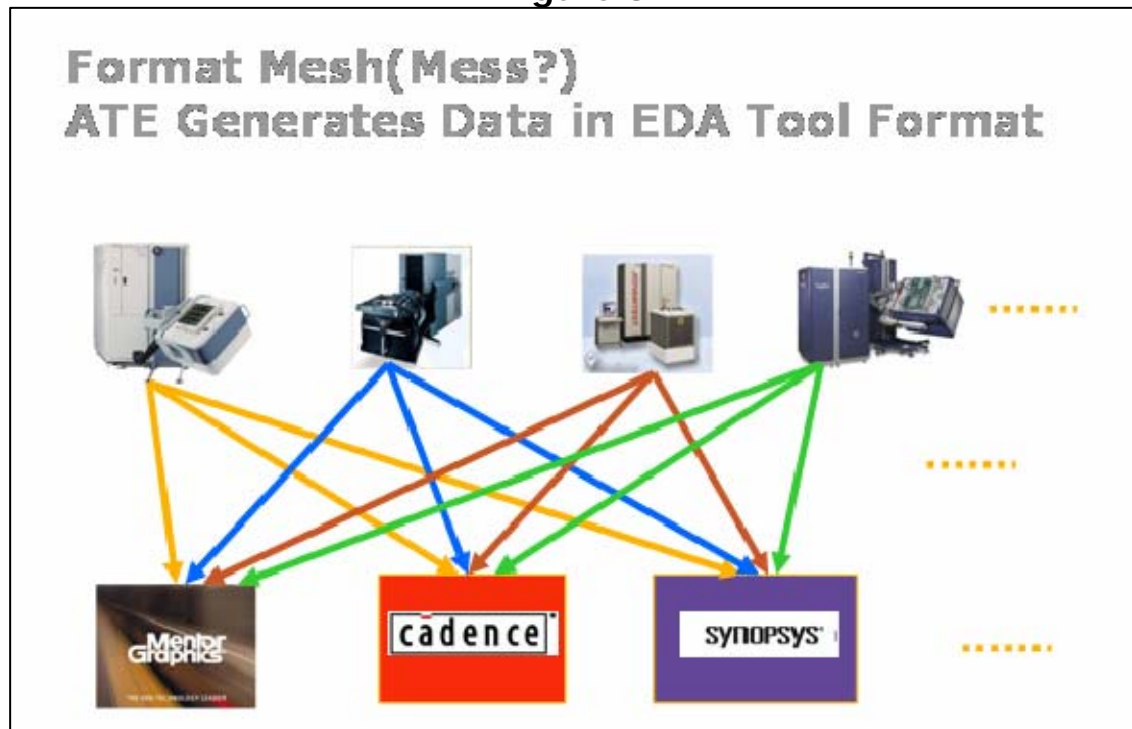


Figure 4

overhead for the party responsible for developing and maintaining the conversion tools as well for the end user who has to deal with the conversion tools mesh/mess.

The rest of the paper is organized as follows:

The following section describes the desired solution for the problems mentioned in the introduction. The STDF standardization section provides information on the ongoing standardization activities in terms of group structure and operation. The Data model section provides the details of the data model that has been developed by the working group to enable Volume Diagnostics flow. The Status and Plan then describes the group's plan for the future activities.

### Desired Solution

The overhead associated with development and support of the conversion tools as well as with IT infrastructures can be eliminated if all the ATE could create failure files in a standard format and all the diagnosis tools can read the same standard format. This document describes the Extension of STDF V4 that has been developed to serve this need as shown in Figure 5.

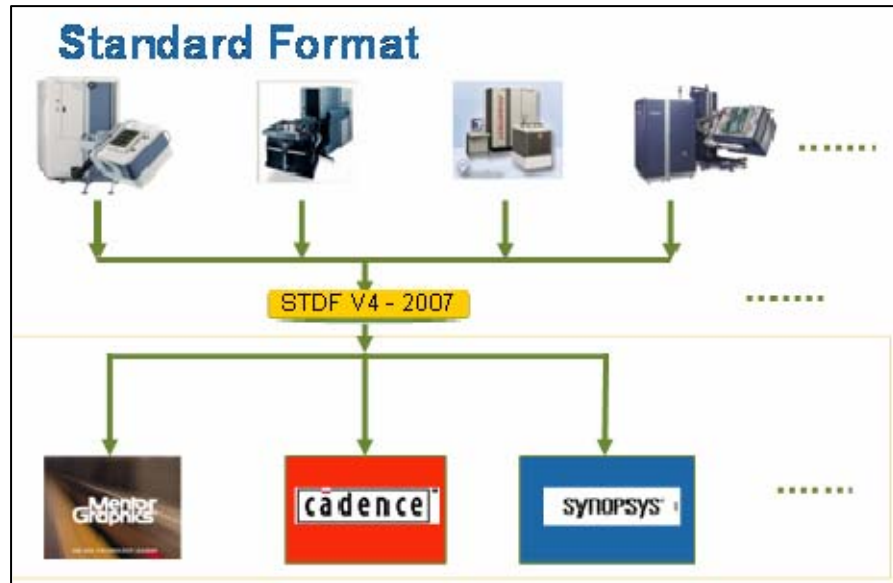


Figure 5

### Scope

- Scan fail datalog for volume diagnosis
- Fail logging at wafer as well as package test
- Format should allow capture of millions of failures (what one would expect in 2010)
- Format should support the design-test-design flow

## Data Model for Scan Fail Datalog

Figure 6 shows a conceptual dataflow diagram for volume diagnostic applications as a base line for this standard. It shows how the test data flows through the design-to-test-to-design as well as transformations that the test data may undergo. As the data flows through the loop, The standard provides mechanisms to keep track of the changes to the extent that the design/diagnosis tools can correctly relate the fail datalog to the original files from which the test patterns were generated to perform correct diagnosis. Any changes to the test data in this loop can be communicated to the analysis tools using this standard. In addition the standard also supports representation of the information on the conditions and equipment used to perform the test when the fail data was collected, to enable volume fail analysis over wafers/lots.

In particular, the standard allows following classes of information must be collected and supported by the format:

- Device Identification
- Test Identification (Test Flow, Test Suite, patterns)
- Test Environment (Temp, freq, volt, etc.)
- Transformation information for synchronization
  - Assignment of patterns to test suites
  - Addition/deletion/truncation of patterns on tester
  - Any Name mappings
  - Buffer full/ datalog truncation
- Format Specification for Fail data
- Fail Data

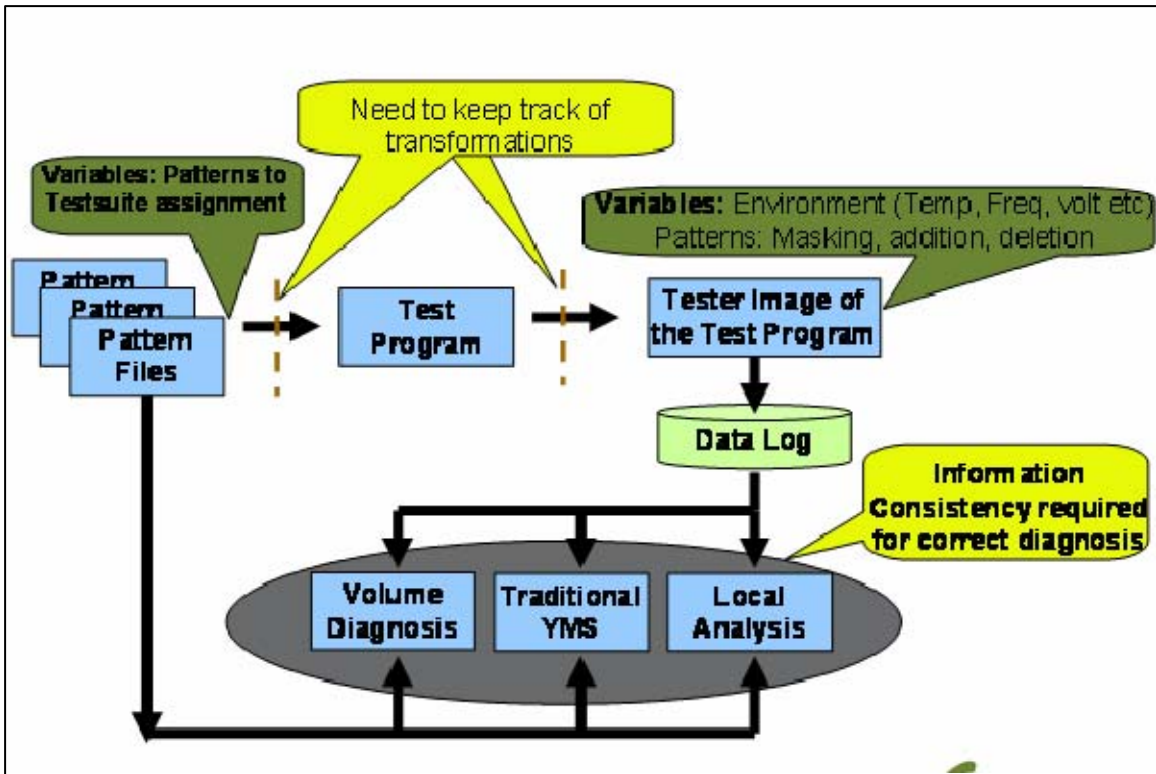


Figure 6

Figure 7 shows an overview of the data model, containing data objects. Each of these objects will be described in the remaining part of this section with their intent and contents.

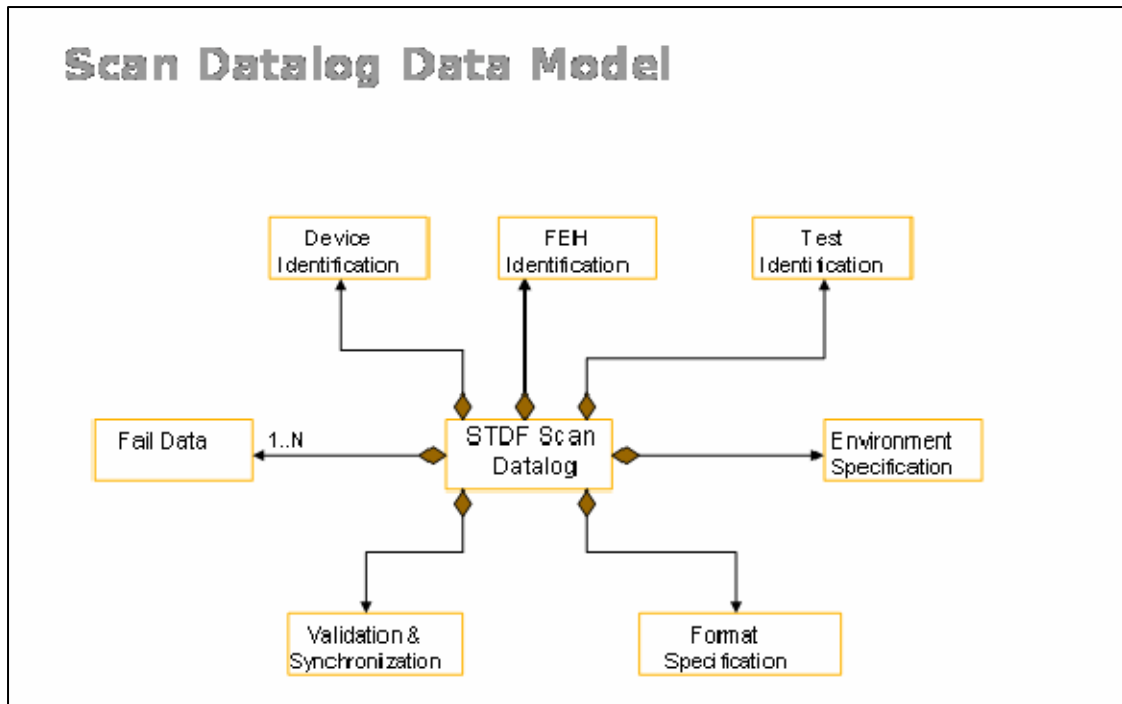


Figure 7

### Device Identification

Device Identification information allows unique identification of the device for which the fail data is collected. Both wafer die and packaged part identification are supported. Device identification is required for statistical analysis of the diagnosis result for identifying any trends or correlation with other measurements. Wafer die identification data consists of:

- Wafer ID
- Lot ID
- XCoord, YCoord
- PartNumber
- Electronic ID

Package Identification data consists of:

- Serial Number
- Lot ID
- Die ID
- Electronic ID

### FEH Identification

This class of data provides the unique identification of Front-End-Hardware (e.g., DUT boards, probes, etc.) that have been used to connect the product to the tester:

- Site ID: Which site the data was collected, e.g., Singapore, test facility
- Cell ID: The Test Cell where the testing was performed
- Prober/Handler ID: Identification of the prober/handler being used during the test
- LoadBoard ID: Identification of the loadboard being used during the test
- Optional field: Other environment-specific information can be added in the optional section.

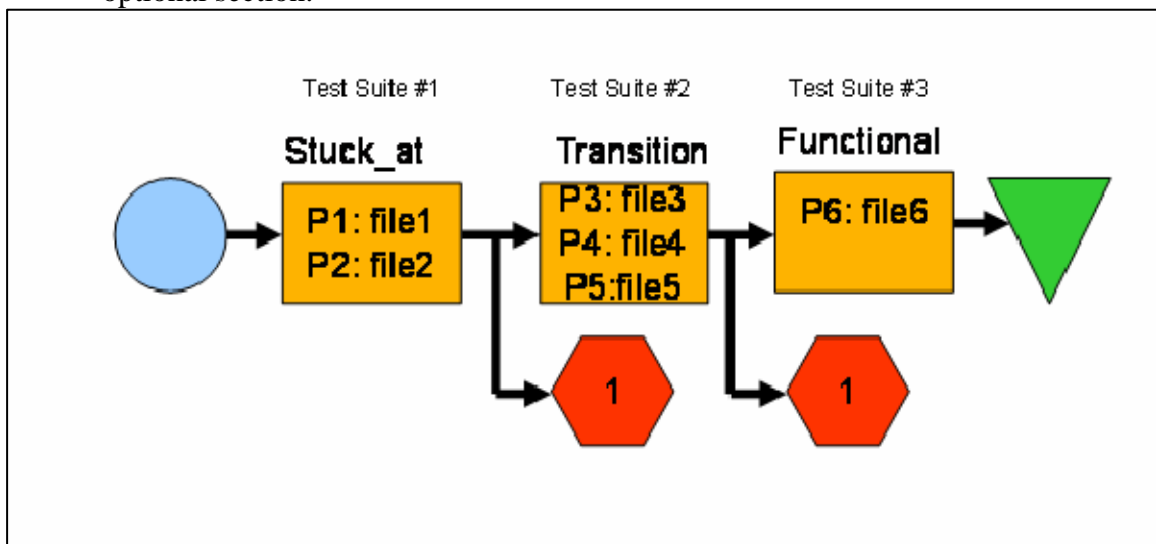


Figure 8

## **Test Identification**

Test Identification data allows unique identification of the test under which the fail datalog was collected. It contains the following information:

- Test Program Identification
- Test Stage Identification – e.g., Wafer Sort1, Package, test number, etc.
- Test Suite Identification: This field identifies the test suite within a test flow as shown in Figure 8.
- Test Pattern Map : This data provides the information on how the patterns for a test suite are assembled. It contains one entry for each pattern that makes up patterns for a test suite. For example for P1 and P2 patterns in Test Suite #1 in Figure 8. Each such entry contains the following information:
  - Unique source file identification
  - Pattern within the source file.
  - Location of the pattern in the list of patterns for the test suite in terms of offset from the last pattern, Start index and optionally the End index.

## **Test Condition Specification**

Test Condition Specification data provides information on the conditions under which the test was performed when the datalog was collected. In particular, conditions that are supported include temperature, timing and voltage. The temperature is a global setting, but the timing and voltage are port-based settings (i.e., one setting per port of the device). The voltage and timing specification is also specific to a test in the test flow.

## **Validation and Synchronization**

Validation and Synchronization data allows the downstream tools to check the fail datalog against any special conditions that are encountered during the execution of the tests during the fail datalog generation. This allows the downstream tools to perform data integrity checks and allows the volume diagnosis flow to remain in sync with respect to the test data, test conditions, fail data collection state and any data transformation/transport. In particular this class of information contains:

- TotalFails: Total number of failures that are logged within the current setup
- BufferLimit[1..N]: This field indicates the fail collection limit per pin.
- FailsAfterBufferFull[1..N]: This is a flag that indicates that failures were observed after the fail buffer became full. One flag per pin is stored. This information allows the diagnosis tools to mark the patterns that were applied after the last pattern/cycle was logged as good/bad, depending on the value of these flags.
- Changelog: If the patterns are modified on the tester after the initial load, then this field will allow communication of those changes to the diagnosis tools. The diagnosis tools can update their image of the patterns to make correct assumptions about the state of those patterns. An example of this is masking of an output, which must be communicated to the downstream tools to not incorrectly assume it to be a passing state
- RunTimeSync: This information states the starting and end states of each pattern during the actual test run. This allows handling of any exception that may happen during a test and not let a pattern run to its normal completion. This information is

meant for the diagnosis tools to avoid making incorrect assumptions in the case of exceptions.

### **Format Specification**

Format specification data indicates the conventions used in the datalog for the fail data. In particular it indicates the type of datalog and how the Z state is handled in the datalog. Both cycle based and pattern based datalogs are supported. In addition, the datalog is also used to communicate the pattern change log as well as measured datalog. An flag in the format specification indicates the log type that follows in the datalog.

The Z-handling information allows one to handle a tester's capabilities in terms of making a dual limit comparison. Table 1 shows the options for the Z-handling.

**Table 1**

<b>Enumeration value</b>	<b>condition</b>
<b>0</b>	<b>No handled</b>
<b>1</b>	<b>Z mapped to L</b>
<b>2</b>	<b>Z mapped to H</b>
<b>3</b>	<b>Z mapped to Z</b>
<b>4</b>	<b>Z mapped to X</b>

### **Fail Data Specification**

Fail data specification contains the actual fail log information. Depending on the format being specified in the Format specification, a series of records for the fail log in the selected format will follow. Depending on the number of failures that are captured, this information will contribute most to the data volume.

## Use Model

The new standard is meant to coexist with the STDF V4 based data flow. In particular following three use models are supported.

- STDF V4 with No Scan Fail (Traditional usage)
- STDF V4 with traditional and Scan fail information
- STDF V4 with Scan fail information only

## Glossary of Terms

### Need to Add

### Data Type Definitions

The STDF specification uses a set of data type codes that are concise and easily recognizable. For example, R\*4 indicates a REAL (float) value stored in four bytes. A byte consists of eight bits of data. For purposes of this document, the low order bit of each byte is designated as bit 0 and the high order bit as bit 7. The following table gives the complete list of STDF data type codes, as well as the equivalent C language type specifier.

**Table 2**

---

<b>Code</b>	<b>Description</b>	<b>C Type Specifier</b>
C*12	Fixed length character string: If a fixed length character string does not fill the entire field,t must be left-justified and padded with spaces.	char[12]
C*n	Variable length character string: first byte = unsigned count of bytes to follow (maximum of 255 bytes)	char[]
S*n	Variable length character string: first two byte = unsigned count of bytes to follow (maximum of 64K bytes)	char[]
C*f	Variable length character string: string length is stored in another field	char[]
U*1	One byte unsigned integer	unsigned char
U*2	Two byte unsigned integer	unsigned short
U*4	Four byte unsigned integer	unsigned long
U*8	Eight byte unsigned integer	
I*1	One byte signed integer	char
I*2	Two byte signed integer	short
I*4	Four byte signed integer	long
R*4	Four byte floating point	float
R*8	Eight byte floating point number	long float (double)
B*6	Fixed length bit-encoded data	char[6]
V*n	Variable data type field:	

	The data type is specified by a code in the first byte, and the data follows (maximum of 255 bytes)	
B*n	Variable length bit-encoded field: First byte = unsigned count of bytes to follow (maximum of 255 bytes). First data item in least significant bit of the second byte of the array (first byte is count.)	char[]
D*n	Variable length bit-encoded field: First two bytes = unsigned count of bits to follow (maximum of 65,535 bits). First data item in least significant bit of the third byte of the array (first two bytes are count). Unused bits at the high order end of the last byte must be zero.	char[]
N*1	Unsigned integer data stored in a nibble. (Nibble = 4 bits of a byte). First item in low 4 bits, second item in high 4 bits. If an odd number of nibbles\ is indicated, the high nibble of the byte will be zero. Only whole bytes can be written to the STDF file.	char
	<i>kxTYPE</i> Array of data of the type specified. The value of 'k' (the number of elements in the array) is defined in an earlier field in the record. For example, an array of short unsigned integers is defined as <i>kxU*2</i> .	TYPE[]

---

## Overview of the Standard

As mentioned above, the new standard leverages the existing STDF V4 specifications. The new standard uses the existing records for the device and test equipment identification and adds new record types for the scan test related information. In particular the scan related records are added as the sub-record type under the existing major record types in STDF. Table 3 shows the existing major record types in STDF V4.

**Table 3**

Major Type	Description
0*	Information about the STDF file
1*	Data collected on a per lot basis
2	Data collected per wafer
5	Data collected on a per part basis
10	Data collected per test in the test program
15*	Data collected per test execution
20	Data collected per program segment
50	Generic Data

New record sub-types are added under the starred Major types in table 3. Table 4 shows a structural overview the standard

**Table 4**

Purpose	Records
Version Identification	<div data-bbox="818 401 1261 443" style="border: 1px solid black; padding: 2px; text-align: center;">Version Update Record</div>
Per Lot Information	<div data-bbox="818 506 1261 548" style="border: 1px solid black; padding: 2px; text-align: center;">Pattern Sequence Record</div> <div data-bbox="818 564 1261 606" style="border: 1px solid black; padding: 2px; text-align: center;">Name Map Records</div> <div data-bbox="818 623 1261 665" style="border: 1px solid black; padding: 2px; text-align: center;">Scan Structure Records</div>
Device and Test Setup Information	<div data-bbox="818 705 1261 747" style="border: 1px solid black; padding: 2px; text-align: center;">Device Identification Records</div> <div data-bbox="818 764 1261 806" style="border: 1px solid black; padding: 2px; text-align: center;">Test Identification Records</div> <div data-bbox="818 823 1261 865" style="border: 1px solid black; padding: 2px; text-align: center;">FEH Identification Records</div>
Per Test Execution Information	<div data-bbox="818 930 1261 1066" style="border: 1px solid black; padding: 2px; text-align: center;">Scan Test Record</div> <div data-bbox="1019 1077 1040 1161" style="text-align: center;">           ■            ■            ■         </div> <div data-bbox="818 1182 1261 1318" style="border: 1px solid black; padding: 2px; text-align: center;">Scan Test Record</div>

| The rest of this document describes the records that are used as part of the new standard.

### **Continuation Records**

| The amount of data required for some of the new record types will occasionally exceed what can be accommodated in a single 65k record limit. To facilitate this expanded data volume the concept of “continuation records” is introduced. Any number of continuation records may be added after an initial record type. A one byte “CONT\_FLG” field is used in these records to specify whether it is the initial record, a continuation of the previous record, and whether another continuation record follows this record. The CONT\_FLG field is included in the PSR, STR, and SSR records: The first two bits of this field specify the record’s continuation status. The remaining bits may be used to specify other flags that are unique to that record type.

**Table 13**

<b>Bit</b>	<b>Name</b>	<b>Description</b>
0	CONT_BIT	0: This is a new record; 1: This is a continuation of the previous record
1	INCMP_BIT	0: The current record is complete and hence is the last one for a scan test; 1: current record is not complete, another continuation record follows this record

## Version Update Record (VUR)

Version update Record is used to indicate that the file contain record defined using the new standard.

This record is added to the major type 0 in the STDF V4. So the new structure of the major type 0 is shown in Table 5 where the text is blue indicates the new record type.

**Table 5**

Major Code	Sub-Type	Purpose
0		Information about the STDF file
	10	File Attribute Record (FAR)
	20	Audit Trail Record (ATR)
	30	Version Update Record (VUR)

Table 6 shows the structure of VUR record.

**Table 6**

<b>Version Update Record (VUR)</b>			
Function: Contains the Version Update Information			
Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYPE	U*1	Record Type (0)	
REC_SUB	U*1	Record sub-type (30)	
UPD_NAME	C*n	Update Version Name	

### Field Description:

UPD\_NAME: This field will contain the version update name. For example the new standard name will be stored as “STDF V4 – 2007” string in the UPD\_NAME field.

## Header Records

STDF V4 -2007 leverages the records from the original STDF V4 specification for the representation of header information related to device identification, Wafer identification and Equipment identification. In particular it uses following records for that purpose:

- Master Information Record (MIR)
- Wafer Information Records (WIR)
- Part Results Record (PRR)
- Test Synopsis Record (TSR)
- Site Description Record (SDR)

These records are described next where the description has been copied from the original specification for completeness sake. In the description, where the usage of certain fields is modified is highlighted in blue.

## Master Information Record (MIR)

This record contains the information on the device identification, test facility identification and any global conditions. From the data model perspective it stores following pieces of information:

- Lot ID
- Test Software version
- Test Program version
- Test Flow ID
- Test Stage
- Test Facility ID
- Test Floor ID
- Tester ID

Table 7

<b>Master Information Record</b>			
<b>Function:</b>		The MIR and the MRR (Master Results Record) contain all the global information that is to be stored for a tested lot of parts. Each data stream must have exactly one MIR, immediately after the FAR (and the ATRs, if they are used). This will allow any data reporting or analysis programs access to this information in the shortest possible amount of time.	
<b>Field Name</b>	<b>Data Type</b>	<b>Field Description</b>	<b>Missing/Invalid Data Flag</b>
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (1)	
REC_SUB	U*1	Record sub-type (10)	
SETUP_T	U*4	Date and time of job setup	
START_T	U*4	Date and time first part tested	
STAT_NUM	U*1	Tester station number	
MODE_COD	C*1	Test mode code (e.g. prod, dev) space	
RTST_COD	C*1	Lot retest code space	
PROT_COD	C*1	Data protection code space	
BURN_TIM	U*2	Burn-in time (in minutes) 65,535	
CMOD_COD	C*1	Command mode code space	
LOT_ID	C*n	Lot ID (customer specified)	
PART_TYP	C*n	Part Type (or product ID)	
NODE_NAM	C*n	Name of node that generated data	
TSTR_TYP	C*n	Tester type	
JOB_NAM	C*n	Job name (test program name)	
JOB_REV	C*n	Job (test program) revision number	length byte = 0
SBLLOT_ID	C*n	Sublot ID	length byte = 0
OPER_NAM	C*n	Operator name or ID (at setup time)	length byte = 0
EXEC_TYP	C*n	Tester executive software type	length byte = 0
EXEC_VER	C*n	Tester exec software version number	length byte = 0

<b>TEST_COD</b>	<b>C*n</b>	<b>Test phase or step code</b>	<b>length byte = 0</b>
<b>TST_TEMP</b>	<b>C*n</b>	<b>Test temperature</b>	<b>length byte = 0</b>
<b>USER_TXT</b>	<b>C*n</b>	<b>Generic user text</b>	<b>length byte = 0</b>
<b>AUX_FILE</b>	<b>C*n</b>	<b>Name of auxiliary data file</b>	<b>length byte = 0</b>
<b>PKG_TYP</b>	<b>C*n</b>	<b>Package type</b>	<b>length byte = 0</b>
<b>FAMILY_ID</b>	<b>C*n</b>	<b>Product family ID</b>	<b>length byte = 0</b>
<b>DATE_COD</b>	<b>C*n</b>	<b>Date code</b>	<b>length byte = 0</b>
<b>FACIL_ID</b>	<b>C*n</b>	<b>Test facility ID</b>	<b>length byte = 0</b>
<b>FLOOR_ID</b>	<b>C*n</b>	<b>Test floor ID</b>	<b>length byte = 0</b>
<b>PROC_ID</b>	<b>C*n</b>	<b>Fabrication process ID</b>	<b>length byte = 0</b>
<b>OPER_FRQ</b>	<b>C*n</b>	<b>Operation frequency or step</b>	<b>length byte = 0</b>
<b>SPEC_NAM</b>	<b>C*n</b>	<b>Test specification name</b>	<b>length byte = 0</b>
<b>SPEC_VER</b>	<b>C*n</b>	<b>Test specification version number</b>	<b>length byte = 0</b>
<b>FLOW_ID</b>	<b>C*n</b>	<b>Test flow ID</b>	<b>length byte = 0</b>
<b>SETUP_ID</b>	<b>C*n</b>	<b>Test setup ID</b>	<b>length byte = 0</b>
<b>DSGN_REV</b>	<b>C*n</b>	<b>Device design revision</b>	<b>length byte = 0</b>
<b>ENG_ID</b>	<b>C*n</b>	<b>Engineering lot ID</b>	<b>length byte = 0</b>
<b>ROM_COD</b>	<b>C*n</b>	<b>ROM code ID</b>	<b>length byte = 0</b>
<b>SERL_NUM</b>	<b>C*n</b>	<b>Tester serial number</b>	<b>length byte = 0</b>
<b>SUPR_NAM</b>	<b>C*n</b>	<b>Supervisor name or ID</b>	<b>length byte = 0</b>

**MODE\_COD:** Indicates the station mode under which the parts were tested. Currently defined values for the MODE\_COD field are:

- A = AEL(AutomaticEdgeLock)mode
- C = Checkermode
- D = Development / Debug test mode
- E = Engineering mode (same as Development mode)
- M = Maintenancemode
- P = Production test mode
- Q = Quality Control

All other alphabetic codes are reserved for future use by Teradyne. The characters 0 - 9 are available for customer use.

**RTST\_COD** Indicates whether the lot of parts has been previously tested under the same test conditions. Suggested values are:

- Y = Lot was previously tested.
- N = Lot has not been previously tested.
- space = Not known if lot has been previously tested.
- 0 - 9 = Number of times lot has previously been tested.

**PROT\_COD** User-defined field indicating the protection desired for the test data being stored. Valid values are the ASCII characters 0 - 9 and A - Z. A space in this field indicates a missing value (default protection).

**CMOD\_COD** Indicates the command mode of the tester during testing of the parts. The user or the tester executive software defines command mode values. Valid values are the ASCII characters 0 - 9 and A - Z. A space indicates a missing value.

**TEST\_COD** A user-defined field specifying the phase or step in the device testing process.

**TST\_TEMP** The test temperature is an ASCII string. Therefore, it can be stored as degrees Celsius, Fahrenheit, Kelvin or whatever. It can also be expressed in terms like HOT, ROOM, and COLD if that is preferred.

## Wafer Information Record (WIR)

This record contains the wafer identification information.

Table 8

Wafer Information Record (WIR)			
<b>Function:</b> Acts mainly as a marker to indicate where testing of a particular wafer begins for each wafer tested by the job plan. The WIR and the Wafer Results Record (WRR) bracket all the stored information pertaining to one tested wafer. This record is used only when testing at wafer probe. A WIR/WRR pair will have the same HEAD_NUM and SITE_GRP values.			
Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (2)	
REC_SUB	U*1	Record sub-type (10)	
HEAD_NUM	U*1	Test head number	
SITE_GRP	U*1	Site group number	255
START_T	U*4	Date and time first part tested	
WAFER_ID	C*n	Wafer ID	length byte = 0

**SITE\_GRP:** Refers to the site group in the SDR. This is a means of relating the wafer information to the configuration of the equipment used to test it. If this information is not known, or the tester does not support the concept of site groups, this field should be set to 255.

**WAFER\_ID:** Is optional, but is strongly recommended in order to make the resultant data files as useful as possible.

## Part Results Record (PRR)

This record is used to identify the part for which datalog is generated. It is used in conjunction with MIR and WIR to uniquely identify the part. In particular following data model objects are stored in this record.

- XCOORD, YCOORD
- Electronic ID (Stored in PART\_TXT)
- Sr. No (Stored in PART\_ID)

**Table 9**

<b>Part Results Record (PRR)</b>			
<b>Function:</b> Contains the result information relating to each part tested by the test program. The PRR and the Part Information Record (PIR) bracket all the stored information pertaining to one tested part.			
Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (5)	
REC_SUB	U*1	Record sub-type (20)	
HEAD_NUM	U*1	Test head number	
SITE_NUM	U*1	Test site number	
PART_FLG	B*1	Part information flag	
NUM_TEST	U*2	Number of tests executed	
HARD_BIN	U*2	Hardware bin number	
SOFT_BIN	U*2	Software bin number	65535
X_COORD	I*2	(Wafer) X coordinate	-32768
Y_COORD	I*2	(Wafer) Y coordinate	-32768
TEST_T	U*4	Elapsed test time in milliseconds	0
PART_ID	C*n	Part identification	length byte = 0
PART_TXT	C*n	Part description text	length byte = 0
PART_FIX	B*n	Part repair information	length byte = 0

**HEAD\_NUM,SITE\_NUM:** If a test system does not support parallel testing, and does not have a standard way to identify its single test site or head, then these fields should be set to 1. When parallel testing, these fields are used to associate individual datalogged results(FTRs and PTRs) with a PIR/PRR pair. An FTR or PTR belongs to the PIR/PRR pair having the same values for HEAD\_NUM and SITE\_NUM.

**X\_COORD,Y\_COORD:** Have legal values in the range -32767 to 32767. A missing value is indicated by the value -32768.

X\_COORD,Y\_COORD, PART\_ID Are all optional, but you should provide either the PART\_ID or the X\_COORD and Y\_COORD in order to make the resultant data useful for analysis.

**PART\_FLG:** Contains the following fields:

bit 0:

0 = This is a new part. Its data device does not supersede that of any previous device.

1 = The PIR, PTR, MPR, FTR, and PRR records that make up the current sequence (identified as having the same HEAD\_NUM and SITE\_NUM) supersede any previous sequence of records with the same PART\_ID. (A repeated part sequence usually indicates a mistested part.)

bit 1:

0 = This is a new part. Its data device does not supersede that of any previous device.

1 = The PIR, PTR, MPR, FTR, and PRR records that make up the current sequence (identified as having the same HEAD\_NUM and SITE\_NUM) supersede any previous sequence of records with the same X\_COORD and Y\_COORD. (A repeated part sequence usually indicates a mistested part.) **Note:** Either Bit 0 or Bit 1 can be set, but **not both**. (It is also valid to have neither set.)

bit 2:

0 = Part testing completed normally

1 = Abnormal end of testing

bit 3:

0 = Part passed

1 = Part failed

bit 4: 0 = Pass/fail flag (bit 3) is valid

1 = Device completed testing with no pass/fail indication (i.e., bit 3 is invalid)

bits 5 - 7: Reserved for future use — must be 0

**HARD\_BIN:** Has legal values in the range 0 to 32767.

**SOFT\_BIN:** Has legal values in the range 0 to 32767. A missing value is indicated by the value 65535.

**PART\_FIX:** This is an application-specific field for storing device repair information. It may be used for bit-encoded, integer, floating point, or character information. Regardless of the information stored, the first byte must contain the number of bytes to follow. This field can be decoded only by an application-specific analysis program.

**PART\_TXT:** This field is used store any text description of the part. This field will be used also for storing the Electronic ID in this standard. Electronic ID will be stored in ASCII form at the end of the existing part description and will be separated by a colon.

**PART\_ID:** This field is used to store part identification in ASCII form. The same field will be used to store the Sr. No. It will be added at the end of existing PART\_ID string and will be separated by a colon.

## Test Synopsis Record (TSR)

This record contains a summary of the test results.

**Table 10**

<b>Test Synopsis Record (TSR)</b>			
<b>Function:</b> Contains the test execution and failure counts for one parametric or functional test in the test program. Also contains static information, such as test name. The TSR is related to the Functional Test Record (FTR), the Parametric Test Record (PTR), and the Multiple Parametric Test Record (MPR) by test number, head number, and site number.			
Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (10)	
REC_SUB	U*1	Record sub-type (30)	
HEAD_NUM	U*1	Test head number See note	
SITE_NUM	U*1	Test site number	
TEST_TYP	C*1	Test type space	
TEST_NUM	U*4	Test number	
EXEC_CNT	U*4	Number of test executions	4,294,967,295
FAIL_CNT	U*4	Number of test failures	4,294,967,295
ALRM_CNT	U*4	Number of alarmed tests	4,294,967,295
TEST_NAM	C*n	Test name	length byte = 0
SEQ_NAME	C*n	Sequencer (program segment/flow) name	length byte = 0
TEST_LBL	C*n	Test label or text	length byte = 0
OPT_FLAG	B*1	Optional data flag See note	
TEST_TIM	R*4	Average test execution time in seconds	OPT_FLAG bit 2= 1
TEST_MIN	R*4	Lowest test result value	OPT_FLAG bit 0= 1
TEST_MAX	R*4	Highest test result value	OPT_FLAG bit 1= 1
TST_SUMS	R*4	Sum of test result values	OPT_FLAG bit 4= 1
TST_SQRS	R*4	Sum of squares of test result values	OPT_FLAG bit 5= 1

**HEAD\_NUM:** If this TSR contains a summary of the test counts for all test sites, this field must be set to 255.

**TEST\_TYP:** Indicates what type of test this summary data is for. Valid values are:

- P = Parametric test
- F = Functional test
- M = Multiple-result parametric test
- S = Scan Test
- A = Memory Test

space = Unknown

**EXEC\_CNT,FAIL\_CNT,ALRM\_CNT:** Are optional, but are strongly recommended because they are needed to compute values for complete final summary sheets.

**OPT\_FLAG:** Contains the following fields:

bit 0 set = TEST\_MIN value is invalid

bit 1 set = TEST\_MAX value is invalid

bit 2 set = TEST\_TIM value is invalid

bit 3 is reserved for future use and must be 1

bit 4 set = TST\_SUMS value is invalid

bit 5 set = TST\_SQRS value is invalid

bits 6 - 7 are reserved for future use and must be 1

OPT\_FLAG is optional if it is the last field in the record.

**TST\_SUMS,TST\_SQRS:** Are useful in calculating the mean and standard deviation for a single lot or when combining test data from multiple STDF files

## Site Description Record (SDR)

This record is used to identify the equipment in the test cell where the part was tested. Following data model objects are stored in this record:

- Prober ID (HAND\_ID)
- Probe Card ID (CARD\_ID)
- Handler ID (HAND\_ID)
- Loadboard ID (In LOAD\_ID)

Table 11

Site Description Record (SDR)			
<b>Function:</b>		Contains the configuration information for one or more test sites, connected to one test head, that compose a site group.	
Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYP	U*1	Record type (1)	
REC_SUB	U*1	Record sub-type (80)	
HEAD_NUM	U*1	Test head number	
SITE_GRP	U*1	Site group number	
SITE_CNT	U*1	Number ( <i>k</i> ) of test sites in site group	
SITE_NUM	kxU*1	Array of test site numbers	
HAND_TYP	C*n	Handler or prober type	length byte = 0
HAND_ID	C*n	Handler or prober ID	length byte = 0
CARD_TYP	C*n	Probe card type	length byte = 0
CARD_ID	C*n	Probe card ID	length byte = 0
LOAD_TYP	C*n	Load board type	length byte = 0
LOAD_ID	C*n	Load board ID	length byte = 0
DIB_TYP	C*n	DIB board type	length byte = 0
DIB_ID	C*n	DIB board ID	length byte = 0
CABL_TYP	C*n	Interface cable type	length byte = 0
CABL_ID	C*n	Interface cable ID	length byte = 0
CONT_TYP	C*n	Handler contactor type	length byte = 0
CONT_ID	C*n	Handler contactor ID	length byte = 0
LASR_TYP	C*n	Laser type	length byte = 0
LASR_ID	C*n	Laser ID	length byte = 0
EXTR_TYP	C*n	Extra equipment type field	length byte = 0
EXTR_ID	C*n	Extra equipment ID	length byte = 0

**SITE\_GRP** Specifies a site group number (called a station number on some testers) for the group of sites whose configuration is defined by this record. Note that this is different from the station number specified in the MIR, which refers to a software station only. The value in this field must be unique within the STDF file.

**SITE\_CNT, SITE\_NUM, SITE\_CNT** tells how many sites are in the site group that the current SDR configuration applies to. **SITE\_NUM** is an array of those site numbers.

**\_TYP** fields These are the type or model number of the interface or peripheral equipment being used for testing:

**HAND\_TYP, CARD\_TYP, LOAD\_TYP, DIB\_TYP,  
CABL\_TYP, CONT\_TYP, LASR\_TYP, EXTR\_TYP**

**\_ID** fields These are the IDs or serial numbers of the interface or peripheral equipment being used for testing:

**HAND\_ID, CARD\_ID, LOAD\_ID, DIB\_ID,  
CABL\_ID, CONT\_ID, LASR\_ID, EXTR\_ID**

## Pattern Sequence Record (PSR)

PSR record contains the information on the pattern profile for specific executed scan test as part of the Test Identification information. In particular it implements the Test Pattern Map data object in the data model. It specifies how the patterns for that test were constructed. There will be a PSR record for each scan test (referenced by a test suite) in a test program. The PSR records are added under the major type 1 with a new sub-type 90.

**Table 12**

<b>Pattern Sequence Record (PSR)</b>			
<b>Function :</b> Contains information pertaining to the pattern profile of a specific executed scan test. It is referenced by the STR (Scan Test Record) through its PSR_IDX field. It Specifies the source pattern files created by ATPG tool that constitute a scan test.			
<b>Field Name</b>	<b>Data Type</b>	<b>Field Description</b>	<b>Missing/Invalid Data Flag</b>
REC_LEN	U*2	Bytes of data following header	
REC_TYPE	U*1	Record Type (1) Record sub-type (90)	
REC_SUB	U*1		
PSR_IDX	U*2	PSR Record Index (used by STR records)	
CONT_FLG	U*1	<b>Record Continuation Status plus FILE_ID &amp; SRC_ID present flags</b>	
TOTAL_CNT	U*2	<b>This field indicates the total no. of pattern information sets in this PSR record plus all following PSR continuation records</b>	
LOCAL_CNT	U*2	<b>No of Pattern Information sets in the current PSR (k)</b>	
PAT_BGN	kxU*8	<b>Cycle # pattern begins on (Cycle #1 is 1st cycle executed)</b>	<b>Note 1</b>
PAT_END	kxU*8	<b>Cycle # pattern stops at</b>	<b>Note 1</b>
PAT_FILE	kxC*n	<b>Pattern File Name</b>	<b>Note 1</b>
FILE_ID	kxC*n	<b>Unique file identifier key</b>	<b>CONT_FLG[2] =0</b>
SRC_ID	kxC*n	<b>PatternInSrcFileID</b>	<b>CONT_FLG[3] =0</b>

**PSR\_IDX:** This is a unique identifier for the set of PSRs that describe the patterns for a scan test.

**CONT\_FLAG:** This is flag used to handle the PSR records of size greater than the maximum record size in STDF V4 (64K). In these cases multiple PSR records are used to store the necessary information. Bit 0 and Bit 1 are used to indicate the state of PSR as follows

**Table 13**

<b>Bit</b>	<b>Name</b>	<b>Description</b>
0	CONT_FLG	0: this is a new PSR;

		1: It is a continuation PSR record
1	INCMP_FLG	0: The current PSR is complete and hence is the last one for a scan test; 1: PSR is not complete, more PSRs follow
2	FID_FLG	1: FILE_ID Present; all the pattern descriptions then contain this field
3	SID_FLG	1: SRC_ID present; All the pattern description then contain this field.

**TOTAL\_CNT:** This field indicates the total number of patterns that make up a scan test over all the PSRs. The description of all the patterns may not fit into a single PSR as mention above.

**LOCAL\_CNT:** This field indicate the total number of pattern that are described in the current PSR from a scan test.

**NOTE** The next set of fields is repeated for each pattern that is contained in a scan test. Each of this field is stored in its own array of size LOCAL\_CNT.

**PAT\_BGN:** The cycle count the specified ATPG pattern begins on. The 1st cycle executed is 1

**PAT\_END :** The cycle count the specified ATPG pattern ends on.

**PAT\_FILE :** The name of the ATPG file from which the current pattern was created

**FILE\_ID:** (Optional) - Unique character string that uniquely identifies the file. This field is provide as a means to additionally uniquely identify the source file. The exact mechanism to use this field is decided by the ATPG software, which will also provide this piece of information in the source files during the translation process.

**SRC\_ID:** (Optional) - The name of the [PatternExec](#) block in the source file. IN case there are multiple patterns being specified in the source file e.g. multiple PatternExec blocks in STIL.

## **Scan Test Record**

Scan Test Record (STR) is a new record that is added to the major record type 15 category (Data Collected Per Test Execution). This is the same category where functional and parametric fail record exist. Thus the scan test record becomes another test record type in this category. In this new record some fields have been brought over from the functional test record (The fields in black) and new fields (in Blue) have been added to handle the scan test data. Table shows the structure of the STR at a conceptual level.

**Table 14**

## Scan Test Record (STR)

**Function:** Contains all or some of the results of the single execution of a scan test in the test program. It is intended to contain all the individual pin/cycle failures that are detected in a single test execution. If there are more failures than can be contained in a single record then the INCOMP\_FLG it will be set in the CONT\_FLGS field and one or more additional STR records will be added containing remaining failure data

Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYPE	U*1	Record Type (15)	
REC_SUB	U*1	Record sub-type (30)	
CONT_FLG	U*1	Record Continuation Status	
TEST_NUM	U*4	Test Number	
HEAD_NUM	U*1	Test head number	
SITE_NUM	U*1	Test site number	
TEST_FLG	B*1	Test flags (fail, alarm, etc.)	
TEST_TXT	C*n	Descriptive text or label	length byte = 0
ALARM_ID	C*n	Name of alarm	length byte = 0
PROG_TXT	C*n	Additional Programmed information	length byte = 0
RSLT_TXT	C*n	Additional result information	length byte = 0
SPIN_MAP	D*n	Bit map of enabled comparators	length byte = 0
TOT_FAILS	U*4	Total failures (pin x cycle) detected in test execution	
LOG_FAILS	U*4	Total fails logged, including continuation records	
CYC_CNT	U*8	Total cycles executed in test	
MASK_MAP	D*n	Bit map of Globally Masked Pins	length byte = 0
FAL_MAP	D*n	Bit map of failures after buffer full	length byte = 0
LIM_CNT	U*2	Size of Array; 1 for global specification	
LIM_PMRS	kxU*2	Indexes of the pins; first location is always 0 and refers to all pins	
LIM_SPK	kxU*4	Fail Limits for the PMR_IDXs; 0th location for global/default value	
COND_CNT	U*2	No of Test Conditions and optional data specifications	
COND_NAM	kxC*n	Test Condition Description	COND_CNT=0
COND_VAL	kxC*n	Test Condition Value	COND_CNT=0
SSR_REF	U*2	SSR Index (Scan Structure Record)	
PSR_REF	U*2	PSR Index (Pattern Sequence Record)	
RCD_FAILS	U*4	Count ( <i>k</i> ) of fails logged in this record only	
CYC_OFST	U*8	Starting cycle no for the entries in the fail log	
DATA_SIZE	U*1	(1,2,4,8) bits per cycle; $m = k*(\text{no of Fields present})/\text{DATA\_SIZE}$	
Z_CHAR	C*1	Z Handling character	
OPT_FLG	U*1	Set of two bits specify the Optional fields present	
DATA_FLG	U*1	Individual bits specify what data arrays follow in this record.	
CYC_NUM	kxU*4	Array of cycle numbers (All cycle formats)	DATA_FLG bit 0 =0
PMR_IDX	kxU*2	Array of PMR Indexes (All Formats)	DATA_FLG bit 1 =0
CHAIN_NO	kxU*2	Array of Chain No for FF Name Mapping	DATA_FLG bit 2 =0
CAP_DATA	mxU*1	Captured Data (Cycle capture format)	DATA_FLG bit 3 =0
PAT_DATA	mxU*1	Pattern vector data contained in the test pattern	DATA_FLG bit 4 =0
NEW_DATA	mxU*1	New vector data (Vector change format)	DATA_FLG bit 5 =0
PAT_NUM	kxU*4	Pattern # (Pattern format)	DATA_FLG bit 6 =0
BIT_POS	kxU*4	Chain Bit Position (Pattern format)	DATA_FLG bit 7 =0

The fields in the STR records belong to various categories as shown in Table 14. The description of the fields is also classified according to those categories below:

### **Flags for Optional Fields**

**STR\_FLGS:** This is single byte file which stores various flags related to the STR record. The description of these flags is given below in Table:

**Table 16**

<b>Bit Position</b>	<b>Name</b>	<b>Description</b>
0	CONT_FLG	1: Indicates this is a continuation STR record to handle large STRs; 0: indicates its is the first record
1	INCOMP_FLG	0: Indicates that current STR is the last one in continued STR records;1: more records follow
2	FAL_MAP_FLG0	FAL_MAP flag bit 0 (Described later)
3	FAL_MAP_FLG1	FAL_MAP flag bit 1 (Described later)
4	MASK_MAP_FLG0	MASK_MAP flag bit 0 (Described later)
5	MASK_MAP_FLG1	MASK_MAP <del>flag</del> <u>flag</u> bit 1 (Described later)
6	<u>Unused</u>	
7	UserModified flag	1: indicates the patterns have been modified on the tester

**TEST\_NUM:** It is the identifier for the test for which data is collected. It should be populated with the Test Suite Number.

**HEAD\_NUM,SITE\_NUM:** If a test system does not support parallel testing, and does not have a standard way of identifying its single test site or head, these fields should be set to 1. When parallel testing, these fields are used to associate individual datalogged results with a PIR/PRR pair. An FTR belongs to the PIR/PRR pair having the same values for HEAD\_NUM and SITE\_NUM.

**TEST\_FLG** Contains the following fields:

- bit 0: 0 = No alarm  
1 = Alarm detected
- bit 1: Reserved for future
- bit 2: 0 = Test result is  
1 = Test result is
- bit 3: 0 = No timeout  
1 = Timeout occurred
- bit 4: 0 = Test was executed  
1 = Test not executed
- bit 5: 0 = No abort  
1 = Test aborted
- bit 6: 0 = Pass/fail flag  
1 = Test completed

bit 7: 0 = Test passed  
1 = Test failed

**TEST\_TXT:** This is a user given description name of the test

**ALARM\_ID** If the alarm flag (bit 0 of TEST\_FLG) is set, this field can optionally contain the name or ID of the alarm or alarms that were triggered. The names of these alarms are tester-dependent.

**PROG\_TXT:** This is also a user provided information. Any additional information regarding programming can be provided.

**RSLT\_TXT:** This is also a user provided information. Any additional information about the results can be provided

**NOTE: Shall we remove the above two fields as these could lead to ABUSE.**

### **Validation and Synchronization Fields**

**TOT\_FAILS:** The total number of failures that were detected in this test, logged or not. A failure is defined as a pin and cycle, such as in three pins failed in the same cycle, that would be counted as 3 failures. If 0 then it should be inferred that this value was not determinable

**LOG\_FAILS:** The total number of failures that were detected and logged in this test. A failure is defined as a pin and cycle, such as in three pins failed in the same cycle, that would be counted as 3 failures. This count includes the total failures from the test execution and includes any following continuation STR records.

**CYC\_CNT:** The total number of cycles that were executed in this test. If 0 then it should be inferred that this value was not determinable

**MASK\_MAP:** This optional field contains an array of bits corresponding to the PMR index numbers of the comparators that were disabled during the test. The 1st bit corresponds to PMR index 1, the second bit corresponds to PMR index 2 and so on. Each comparator that is disabled will have it's corresponding PMR index bit set to 1. Note that this field applies to the present record only plus any following continuation STR records). If this field is missing, the it is assumed that conventional pin enabling as defined in the source pattern source file specified in the referenced PSR records apply. The MASK\_MAP fields are controlled by the MASK\_MAP\_FLAG bits and the behavior is described in Table 17

**Table 17**

<b>MASK_MAP_FLG0</b>	<b>MASK_MAP_FLG1</b>	<b>Description</b>
0	0	Mask map not present; No pin is globally masked
0	1	Use the previous MASK_MAP definition (MASK_MAP in the current record absent)
1	0	Use the MASK_MAP in the current record and make it persistent
1	1	Unused

**FAL\_MAP** (Failures After Logging Map): : This optional field that contains an array of bits corresponding to the PMR index numbers of the pins that had any failure after the fail data logging was stopped during the test. The 1st bit corresponds to PMR index 1, the second bit corresponds to PMR index 2 and so on. Each pin that has at least one failure will have its corresponding PMR index bit set to 1. Note that this field applies to the present record only plus any following continuation STR records). If this field is missing, then it is assumed that conventional pin enabling as defined in the source pattern source file specified in the referenced PSR records apply. The FAL\_MAP field is controlled by the FAL\_MAP bits and the behavior is described in Table 18

**Table 18**

<b>FAL_MAP_FLG0</b>	<b>FAL_MAP_FLG1</b>	<b>Description</b>
0	0	No information of buffer status is provided
0	1	All fails were logged, FAL_MAP not present
1	0	FAL_MAP present
1	1	Unused

**LIM\_CNT:** This field specifies the number of pins for which the fail datalog limits specification is provided in the record.

**LIM\_PMRS:** This is an array which contains the PMR indexes of the pins for which the fail logging limits is provided. Please note that the first location in this array is location 0 and is reserved for indicating all pins or default pins. The number of entries in this array is specified by the LIM\_CNT described above.

**LIM\_SPK:** This array contains the fail log limits for the pins which are listed in the LIM\_PMRS array. The first location in this array is, location 0 and contains the value that is the default value that is applied to all the pins for which an explicit limit is not specified in this array. Thus in global case where a single limit applied to all the pins, this array contains only one element the element at location 0. The number of entries in this array is specified by the LIM\_CNT described above.

### **Test Condition Specification**

Test Conditions are specified using (Condition, Value) pairs. Each of them are represented by ASCII in two separate arrays using the following three fields.

**COND\_CNT:** This is the count of how many conditions are specified for the test.

**COND\_NAM:** This is array where the names of the conditions are stored for which the values are specified in the next field.

**COND\_VAL:** This is an array which contains the values for the conditions listed in COND\_NAM array. Please note that the values are stores in ASCII. Expressions are also allowed in this array, however if an expression is use for a value then it must start with a '=' sign and the expression is stored in ASCII as well. The processing of the expression is optional for the readers and it can be done by detecting the '=' symbol.

The standard provide following two reserved keywords for the frequency conditions with following meaning. If these keywords are used in the COND\_NAME array then the values must be for what the standard specifies.

- SHIFT\_FREQ: The shift frequency for the scan test.
- CAPTURE\_FREQ: The capture frequency for the scan test.

There is one exception to the test condition specification and that is Temperature specification. Since, the temperature is more of a global test condition and a field exist in MIR to represent that so the temperature need not be present in the STR. The standard however does not preclude one from putting another temperature parameter in the test conditions array mentioned above.

### **Datalog Format Specification**

Following fields in STR are provided for data format specification for the datalog.

**SSR\_REF:** This is the reference to the scan structure record which will be used for translating the chain no. and bit no. into flip/flop names.

**PSR\_REF:** This is the reference to the pattern sequence record that describes the pattern map for the current scan test.

**RCD\_FAILS:** This field is used to indicate the number of fails that are stored in the current STR record and is meant for preparing the STDF reader to RCS\_FAILS number of fails from the current record.

**CYC\_OFST:** This field is of type U\*8 and contains the offset for the cycle numbers in the fail log. This mechanism is allows only U\*4 values for the cycle numbers in the fail log (vs U\*8) and thus reduces the data volume. The actual cycle number for the fail log entries would be CYC\_OSFT+<fail Cycle No in the log>.

**DATA\_SIZE:** This field indicated the number of bit used to represent the captured data (CAP\_DATA), pattern/expected data (PAT\_DATA) and modified data (NEW\_DATA). The value of this field determines the size of the array required to store the respective data. The possible values for that this field can have is (1,2,4,8) with following convention.

- If 8 bits are used then the value in the byte represents the waveform character itself .
- For 1,2 and 4 bits cases, a default mapping tables provided to map the values in the array to corresponding waveform characters.
- For fail capture any of the possible values (1,2,4,8) can be used
- For pattern data only values possible are 4 or 8.

**Z-CHAR:** This is flag to indicate the how the Z (Dual side comparison) values were handled on the ATE. Table shows the modes that are supported. The first entry is for the ATEs which can't perform dual side comparison. While the rest of the entries are to be used by the ATEs that can and in that case it is meant to communicate how the Z-character was mapped in the datalog.

**Table 19**

Enumeration value	condition
0	Z mapped to L
1	Z mapped to H
2	Z mapped to Z
3	Z mapped to X
4	Not handled

**OPT\_FLGS:** This field is used to indicate the optional fields that are present in the datalog with each failure entry. In total a maximum of 4 optional fields are supported and the type of these fields is indicated by the two bits per field in the OPT\_FLGS field. The supported values are as follows:

**Table 20**

Bit Values	Field Type
01	U*1
10	U*2
11	U*4
00	End of the optional fields

If less than four optional fields are used then the two bits following the specification bits for the used optional fields must be 00.

**DATA\_FLGS:** This field is used as the mask for the datalog fields i.e. each bit in the DATA\_FLGS corresponds to a field in the datalog. If a bit in DATA\_FLGS is set to a 1

then the corresponding field is ~~not~~ present in the failure datalog that follows. Table ... Shows the mapping of DATA\_FLGS bits to the datalog fields:

**Table 21**

Bit position	Corresponding Field
0	CYC_NUM
1	PMR_IDX
2	CHAIN_NO
3	CAP_DATA
4	PAT_DATA
5	NEW_DATA
6	PAT_NO
7	BIT_POS

The fields in this table are described in the next section.

### **Datalog**

The fail datalog section of the record is organized as a super record which can support storing of each of the data fields described below. Each of these fields can however be removed by setting the appropriate bits in the DATA\_FLGS fields as mentioned above. In addition the fields are organized as one array for each field. i.e. the STR record contains and one array of the element for a particular field that is not masked in the order of the bit position in the DATA\_FLGS. All the array in an STR contain the same number of elements. The number of elements in stored in the RCD\_FAILS field in the data format specification section of STR.

**CYC\_NUM:** An array of 4 byte cycle numbers that correspond to the number of failures logged in this record when logging in the "Cycle" mode (vs. the "Pattern" mode). The first cycle executed is always "1".

**PMR\_IDX:** An array of PMR indexes that correspond either to the CYC\_NUM array or to the PAT\_NUM array

**CAP\_DATA:** An array of 1 byte characters that correspond to the CYC\_NUM ("Cycle" mode) or PAT\_NUM ("Pattern Mode") entries that indicate the "captured" data state. This function is generally the same as the MEAS\_DATA except is is intended to infer that this data does not represent a failure, but rather the data states "captured" by the tester, e.g. in a scan data dump application.

**PAT\_DATA:** An array of 1 byte characters that correspond to the CYC\_NUM ("Cycle" mode) or PAT\_NUM ("Pattern Mode") entries that indicate the data state that was specified in the source pattern file for this specific pin and Cycle or Pattern/Bit. This data has multiple applications:

1. Can be used when it is required to provide the expect data associated with each failure

2. Can be used to provide validation (when required) of the synchronization of cycle #'s to the source pattern file (as a note: Generally only a few initial representative cycles would be required for this function, and only once in an STDF file for each specific PSR utilization.)
3. Can also be used to represent the "Old" data states when used in conjunction with the following "NEW\_DATA array when transmitting an STR record for the purposes of defining test pattern modifications

**NEW\_DATA:** An array of 1 byte characters that correspond to the CYC\_NUM ("Cycle mode") or PAT\_NUM ("Pattern Mode") entries that indicate any test pattern data modifications made to the original data in the source patterns.

**PAT\_NUM:** When used in the "Pattern" format, an array of 4 byte integers specifying the pattern number. The 1st pattern executed is always "1"

**BIT\_POS:** When used in the "Pattern" format, an array of 4 byte integers specifying the bit position in the scan chain. The 1st bit position is always "1". This field can also be used in conjunction with the CHAIN\_NO field to support the applications where FF name needs to be derived using SSR. The CHAIN\_NO and BIT\_POS together will be used as an index in the SSR table for getting the FF name in that case.

## Signal Name Record (SNR)

This record is allow preservation of the ATPG names through the flow. STDF writes can use this information to provide the ATPG names in the log using following convention.

- If SNR record(s) is absent and then the PMR records should be used to communicate the ATPG signal names and logical name field in the PMR record should/would contain the ATPGsignal name
- If SNR(s) record is present then use then it should contain ATPG names and would replace the logical name from the PMR record.

**Table 22**

<b>Signal Name Record (SNR)</b>			
Function: Contains information on the mapping of PMR indexes to the ATPG Names It is used to store the mapping of ATE names to original ATPG names If this field is not present then the logic names of the PMR record should contain the ATPG names			
<b>Field Name</b>	<b>Data Type</b>	<b>Field Description</b>	<b>Missing/Invalid Data Flag</b>
REC_LEN	U*2	Bytes of data following header	
REC_TYPE	U*1	Record Type (1)	
REC_SUB	U*1	Record sub-type (91)	
NAME_CNT	U*2	No. of Entries	
PMR_IDX	U*2	PMR Index of the pin in the mapping	
ATPG_NAME	C*n	ATPG Signal Name for the pin	

Description of the Fields

**NAME\_CNT:** This is a count of the number of entries in the record and indicates the reader as to how many entries to read.

**PMR\_IDX:** It is the PMR indexes for which the ATPG names are provided

**ATPG\_NAME:** It is the ATPG names corresponding to the pin in PMR\_IDX array..

## Scan Cell Name Record

This record is used to store the mapping table from Chain and Bit position to the flip/flop names.

**Table 23**

<b>ScanCell Name Record (CNR)</b>			
Function:		Contains information on the mapping of Scan Chain, Bit position to the FF name It is used to support the applications where the log requires the FF names e.g. Logic Vision This is an optional record.	
Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYPE	U*1	Record Type (1)	
REC_SUB	U*1	Record sub-type (92)	
<b>CNR_IDX</b>	<b>U*2</b>	<b>CNR Index</b>	
<b>CNR_NAME</b>	<b>C*n</b>	<b>Name of the Scan Structure for reference</b>	
<b>CONT_FLGS</b>	<b>U*1</b>	<b>Flags for continuation record</b>	
<b>TOTAL_CNT</b>	<b>U*4</b>	<b>Total no of Table Entries</b>	
<b>LOCAL_CNT</b>	<b>U*2</b>	<b>No of Entries</b>	
<b>CHAIN_NO</b>	<b>kxU*2</b>	<b>Scan Chain No</b>	
<b>BIT_POS</b>	<b>kxU*2</b>	<b>Bit position</b>	
<b>FF_NAME</b>	<b>kxS*n</b>	<b>Name of the FF</b>	

### Field Description

**CNR\_IDX:** This is a unique number assigned to each SNR. It is used to reference a particular SNR while translating the chain no and bit position to an ATPG name.

**CNR\_NAME:** This is a ASCII unique name for the applicable scan structure record.

**CONT\_FLGS:** These are the flags used to indicate the continuation of a SSR record in case the size of scan structure exceeds the maximum size of a STDF record (64K). The fail

**Table 24**

Bit 0	Bit 1	Description
0	0	New and Complete
0	1	New and Not Continued (i.e. Not complete)
1	0	Continuation record and not complete
1	1	Continuation record and complete

**TOTAL\_CNT:** This field is used to store the total number of entries for FF names across all the SSRs

**LOCAL\_CNT:** This field is used to store the number of entries for FF names in the current SSR.

**Following three fields are stored in individual arrays. The number of elements in these arrays is determined by the LOCAL\_CNT field.**

**CHAIN\_NO:** This is the array for the chain identification for the target FF for which the name is provided in the table.

**BIT\_POS:** This is an array for the bit position within the chain identified by the CHAIN\_NO field for the target FF for which the name is provided in the table.

**FF\_NAME:** This is an array name of the of FF at the CHAIN\_NO and BIT\_POS. Please note the type of this field is S\*n where the length of the FF name is stored in the first two bytes and then the name follows.

## Scan Structure Record (SSR) and Scan Chain Records (SCR)

These records are used to transfer the information contained in a single STIL file ScanStructures block. They are optional records. The SSR record is a master record and contains an array of SCR records (by index)

**Table 23**

<b>ScanStructure Record (SSR)</b>			
Function:	This record is used to transfer the information contained in a STIL file ScanStructures block. It is an optional record and is not referenced by any other record.		
Field Name	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN	U*2	Bytes of data following header	
REC_TYPE	U*1	Record Type (1)	
REC_SUB	U*1	Record sub-type (93)	
<b>SSR_INDX</b>	<b>U*2</b>	<b>SSR Index</b>	
<b>SSR_NAME</b>	<b>C*n</b>	<b>Name of the Scan Structure for reference</b>	
<b>CONT_FLGS</b>	<b>U*1</b>	<b>Flags for continuation record</b>	
<b>CHAIN_CNT</b>	<b>U*2</b>	<b>Number of Chains (k)</b>	
<b>CHAIN_LIST</b>	<b>kxU*2</b>	<b>Array of SCR indexes</b>	

### Field Description

**SSR\_INDX:** This is a unique number assigned to each SSR. It is used to reference a particular SSR while translating the chain no and bit position to an ATPG name.

**SSR\_NAME:** This is a ASCII unique name for the scan structure record (normally provided by the STIL file).

## Scan Chain Record

Table 23

<b>ScanChain Record (SCR)</b>			
<p>Function: This record is used to transfer the information contained in a STIL file ScanStructures block for a single scan chain. It is an optional record. These records are referenced by a ScanStructure record (SSR)</p>			
<b>Field Name</b>	<b>Data Type</b>	<b>Field Description</b>	<b>Missing/Invalid Data Flag</b>
REC_LEN	U*2	Bytes of data following header	
REC_TYPE	U*1	Record Type (1)	
REC_SUB	U*1	Record sub-type (94)	
<b>SCR_INDX</b>	<b>U*2</b>	<b>SCR Index</b>	
<b>CHN_NAME</b>	<b>C*n</b>	<b>Chain Name</b>	
<b>SCAN_LEN</b>	<b>U*2</b>	<b>Chain Length (# of scan cells in chain)</b>	
<b>SCAN_CNT</b>	<b>U*2</b>	<b>k: # of scan cells listed in this record</b>	
<b>SIN_PIN</b>	<b>U*2</b>	<b>PSR/NMR index of the scan in signal</b>	
<b>SOUT_PIN</b>	<b>U*2</b>	<b>PSR/NMR index of the scan out signal</b>	
<b>MSTR_CNT</b>	<b>U*1</b>	<b>m: Number of master clock pins specified for this chain</b>	
<b>SLAV_CNT</b>	<b>U*1</b>	<b>n: Number of slave clock pins specified for this chain</b>	
<b>MSTR_CLKS</b>	<b>mxU*2</b>	<b>Array of PMR/NMR indexes of the master clocks for this chain</b>	
<b>SLAV_CLKS</b>	<b>nxU*2</b>	<b>Array of PMR/NMR indexes of the slave clocks for this chain</b>	
<b>INVRT_FLG</b>	<b>U*1</b>	<b>0: No chain Inversion; 1: Chain Inversion</b>	
<b>SCAN_CELLS</b>	<b>kxS*n</b>	<b>Array of Scan Cell Names</b>	

### Field Description

**SCR\_INDX:** This is a unique number assigned to each SCR. It is used by the SSR record to reference a particular SCR.

**CHN\_NAME:** This is a ASCII unique name for the scan chain (normally copied from the STIL file).

**SCAN\_LEN:** The number of scan cells contained within the scan chain

**SCAN\_CNT:** The number of scan cells listed in this record (the others are listed in continuation SCR records)

**SIN\_PIN:** The PMR/NMR record index for the chain's Scan In signal

<b>SOUT_PIN:</b>	The PMR/NMR record index for the chain's Scan Out signal
<b>MSTR_CNT:</b>	The # of master clock pins assigned to the scan chain
<b>SLAV_CNT:</b>	The # of slave clock pins assigned to the scan chain
<b>MSTR_CLKS:</b>	An array of PMR/NMR indexes for the chain's master clock pins The length of this array is specified in the MSTR_CNT field
<b>SLAV_CLKS:</b>	An array of PMR/NMR indexes for the chain's slave clock pins The length of this array is specified in the SLAV_CNT field
<b>INVRT_FLG:</b>	A boolean value indicating if the Scan_Out signal is inverted from the Scan_In signal
<b>SCAN_CELLS:</b>	The array of scan cell names

