Standard Test Data Format (STDF) V4 - 2007 Specification

Table of Contents

List of Figures

List of Tables

Preface

Advances in technology are making it imperative to collect and diagnose detailed structural IC fail data during manufacturing test to improve yield. However, current datalog formats (STIL and STDF V4) do not have efficient support for storing structural fail information. In particular STDF V4 which is widely used for functional and parametric datalogging does not have any intrinsic support for storing scan and memory fail requirements. Lack of any standard format to store these fails leads to many custom formats and corresponding translators which results in un-necessary investments in development and maintenance as well as inefficiencies in the diagnosis process for yield learning.

In order to address the above mentioned problems, an extension of STDF V4 called STDF-V4 2007 has been defined to provide formats for storing scan and memory fails as well as for supporting the volume diagnosis flows. This document contains the specifications of this standard.

It is our hope that this standard will be useful to all the companies involved in diagnosing the structural fail information for yield learning and would adopt this into their tools and environments.

Glossary of Terms

ATPG	Automatic Test Pattern Generator: A SW tool normally provided by EDA companies that is used to create the scan based test patterns eventually translated by and used on the ATE. Generally the ATPG tool will output this information as either a WGL, STIL file, or other proprietary language file.
STIL file	Standard Test Interface Language (IEEE1450): A file employing a standardized format for conveying test pattern, timing, and signal information (functional and/or structural)
WGL file	Waveform Generation Language: A file employing a regular format for conveying test pattern, timing, and signal information (functional and/or structural)
Chain	ScanChain: A sequence of linked internal scan cells that in the context of test patterns have a discrete scan input and a discrete scan output with some number of scancells between the two.
Pattern	Sometimes confusing term because it is sometimes used in multiple contexts. In the structural test context, a pattern is the scanning in of a data sequence on the scan-in pins that sets all of the scan cells within the IC with a set of stimulus data, the execution of a "capture" cycle which transfer the results of the scan pattern test into the scancells, then shifting out and testing the test results on the scan-out pin. Typically the stimulus data for the next scan pattern is shifted on the same cycles that the data from the present pattern is being shifted out. A **Pattern Block** will contain several hundreds or thousands scan patterns. While a single STIL / WGL file can support multiple pattern block definitions, generally ATPG tools only include a single pattern block in each file. And in turn, a single test on the ATE may apply several pattern blocks, built from multiple ATPG files.
Pattern/Chain/Bit	Term that specifies that the scan failure data is being logged in context of the failing scan pattern #, chain # (via PMR index), and scancell position within the scan chain. This format is an alternative to logging the cycle # and PMR index which leaves it to the downstream diagnostic tool to convert this to the corresponding pattern/chain/bit information.
PMR Index	An arbitrarily numeric index that is assigned to each pin/signal used by the ATE to provide/monitor test data. PMR indexes must be > 0. STDF file records failures based on their PMR index. The

	STDF **PMR** (**P**in **M**ap **R**ecord) record is used to provide textual information about a unique pin (one record per pin).
Signal	Synonymous with *Pin*. IC input/output pin name.
Scan Cell	Refers to the flipflop/latch that is used as the stimulus/observation node of a scan pattern.
Scan-In	The name of the signal(pin) on which scan data is applied to by the ATE
Scan-Out	The name of the signal(pin) on which scan data output is tested by the ATE
ScanStructure	A block of information that defines the structure of scan chains within the logic of an IC. This information will contain the list and attributes of each scan chain within the IC. Chain attributes generally include the ATPG signal names used for the Scan-In and Scan-Out functions, the # of scan cells within the chain, the clock(s) used to shift the data through the scan chain, and optionally the design instantiation name of each scan cell within the chain. When the latter is included, the size of a ScanStructure block can be significantly large.
Structural Test	A general term used for use of specialized on-chip test circuitry for testing the functionality of individual blocks of sub-circuits within an IC. Variations of structural test include Scan, Compressed Scan, BIST (logic Built-In-Test), MBIST (Memory Built-In-Test), and Boundary Scan.

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 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.

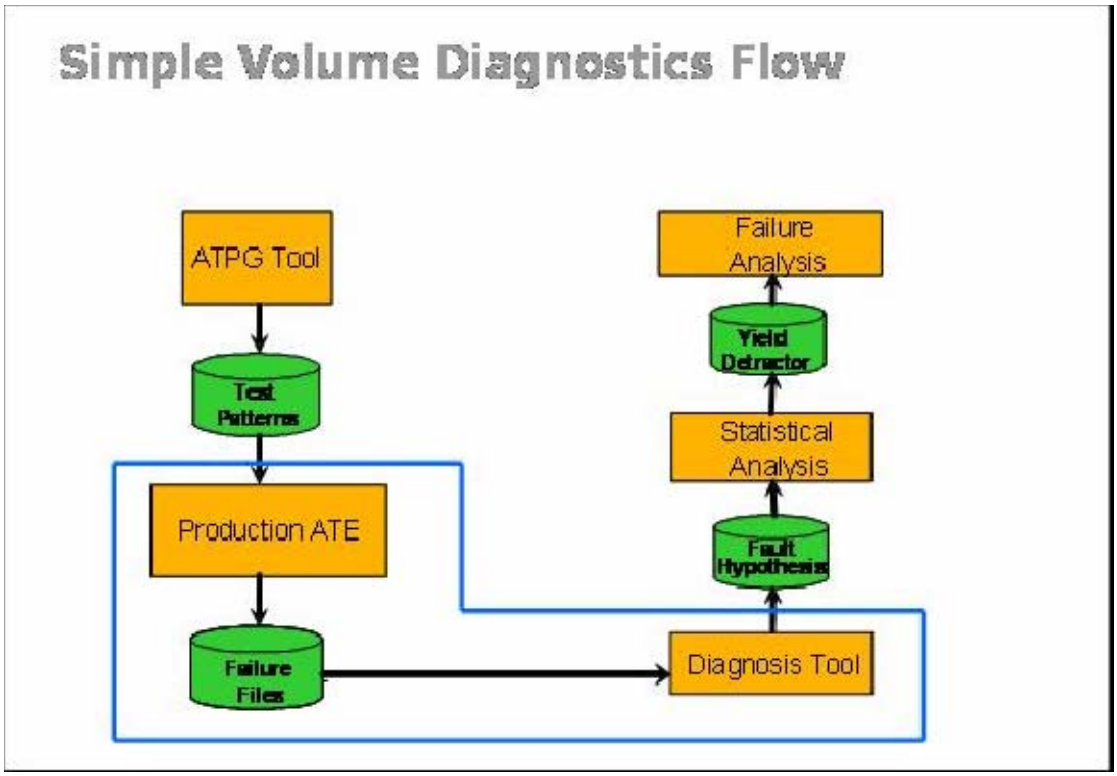

Figure 1: Volume Diagnostics Flow

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.

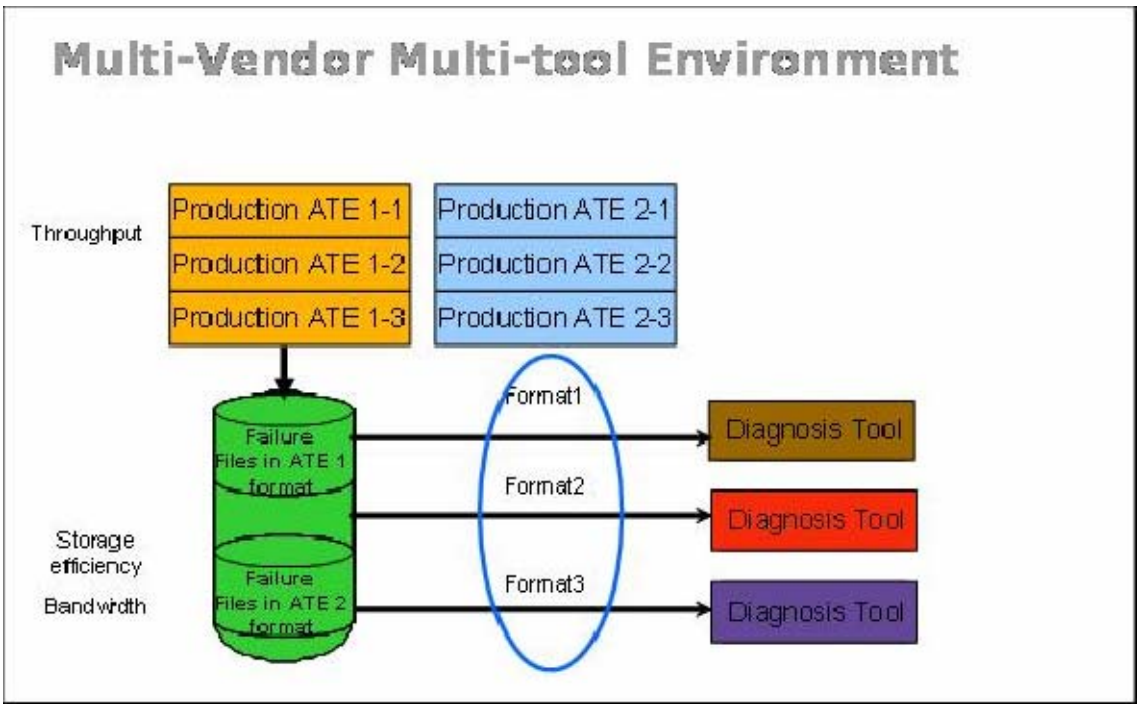

Figure 2: Diagnosis in Multi-Vendor Multi-Tool Environment

To complicate the matters further, each diagnosis tool also has its custom input format. 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 corresponding diagnosis tool. The responsibility for developing and supporting the translation tools can be with the EDA vendors, 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

co

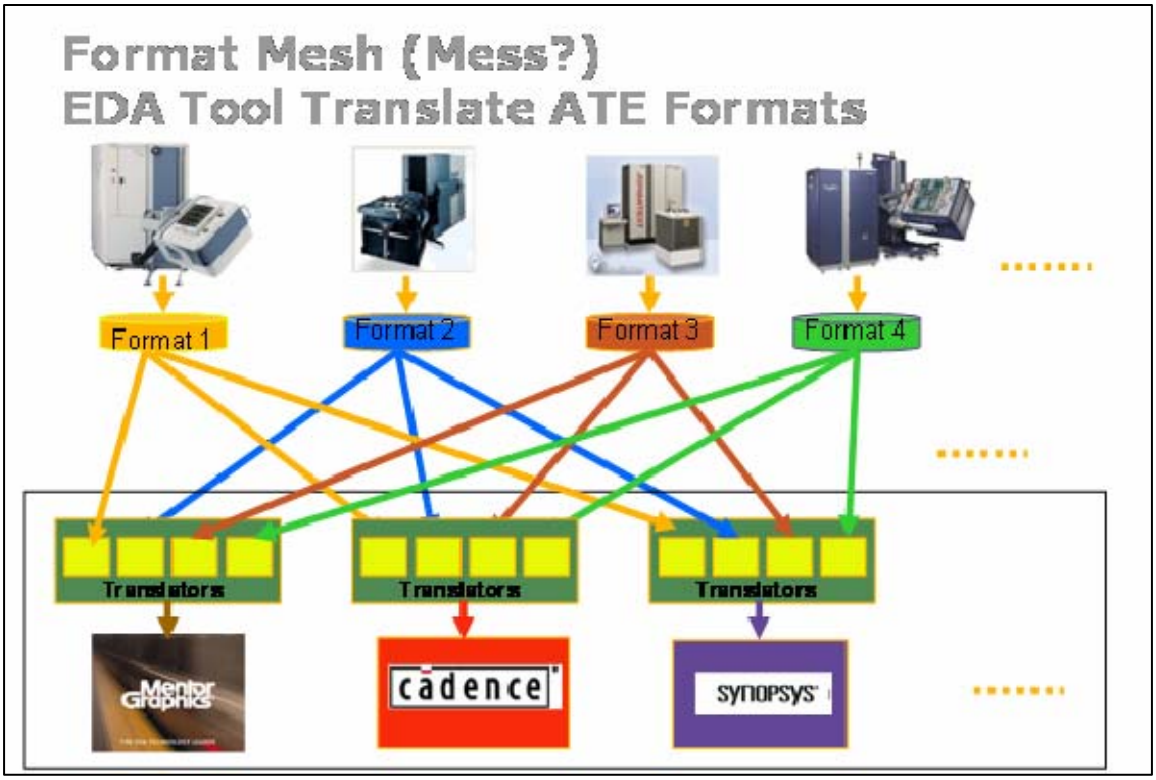

Figure 3: Format Mess, with ATE Custom Formats

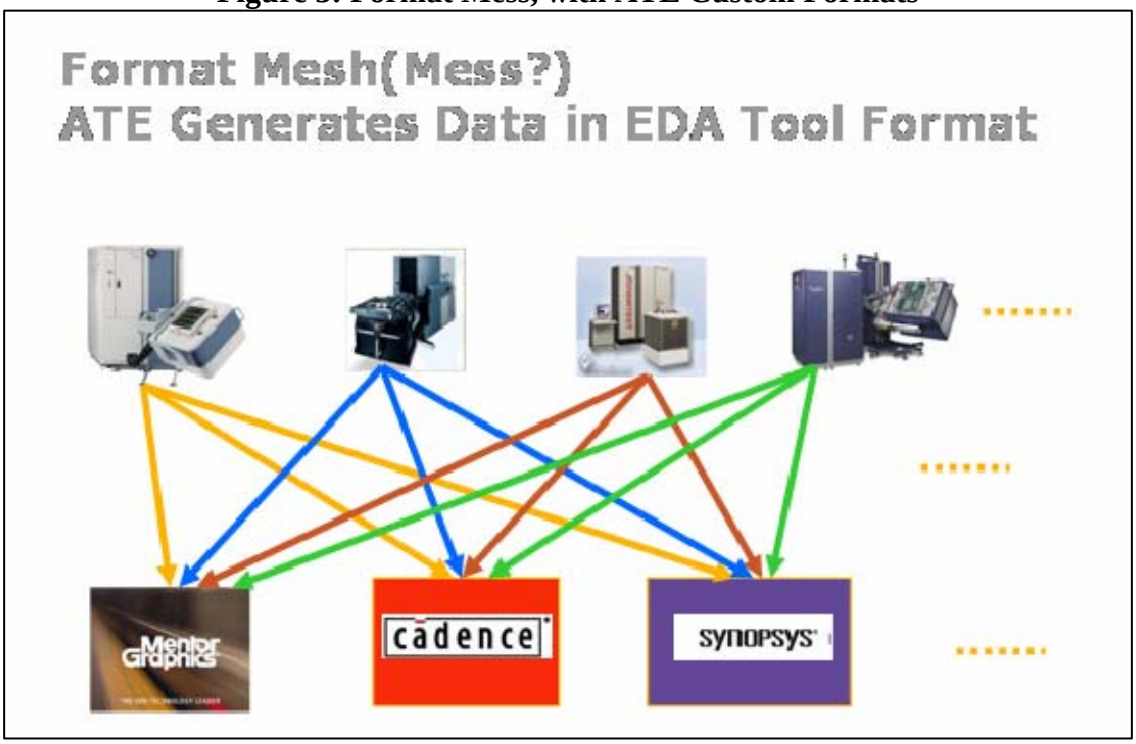

Figure 4: Format Mess with EDA Custom Formats

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.

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.

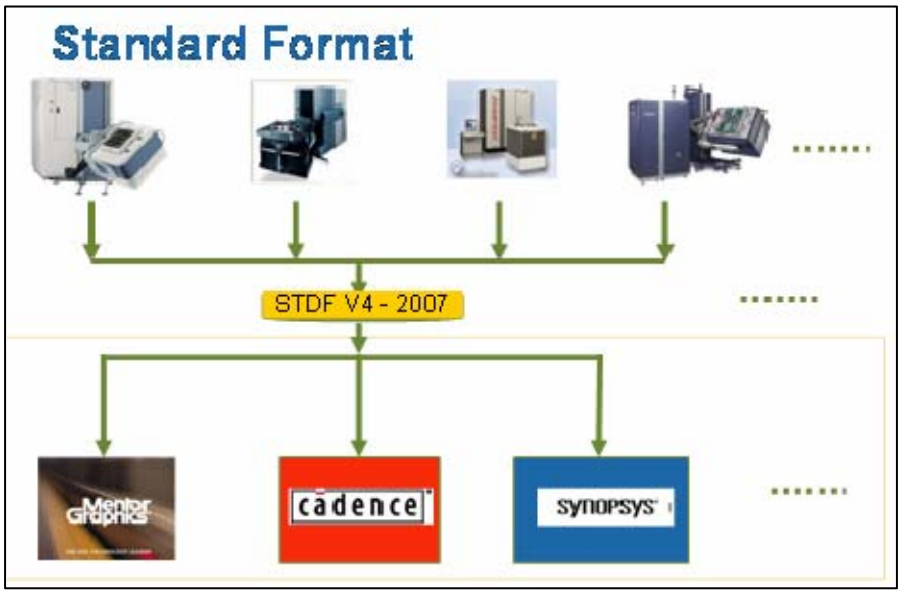

Figure 5: Standard Format Based Flow

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:

- Design Information
- 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 and expect data and how the data is handled on ATE (For example how Z state is handled)

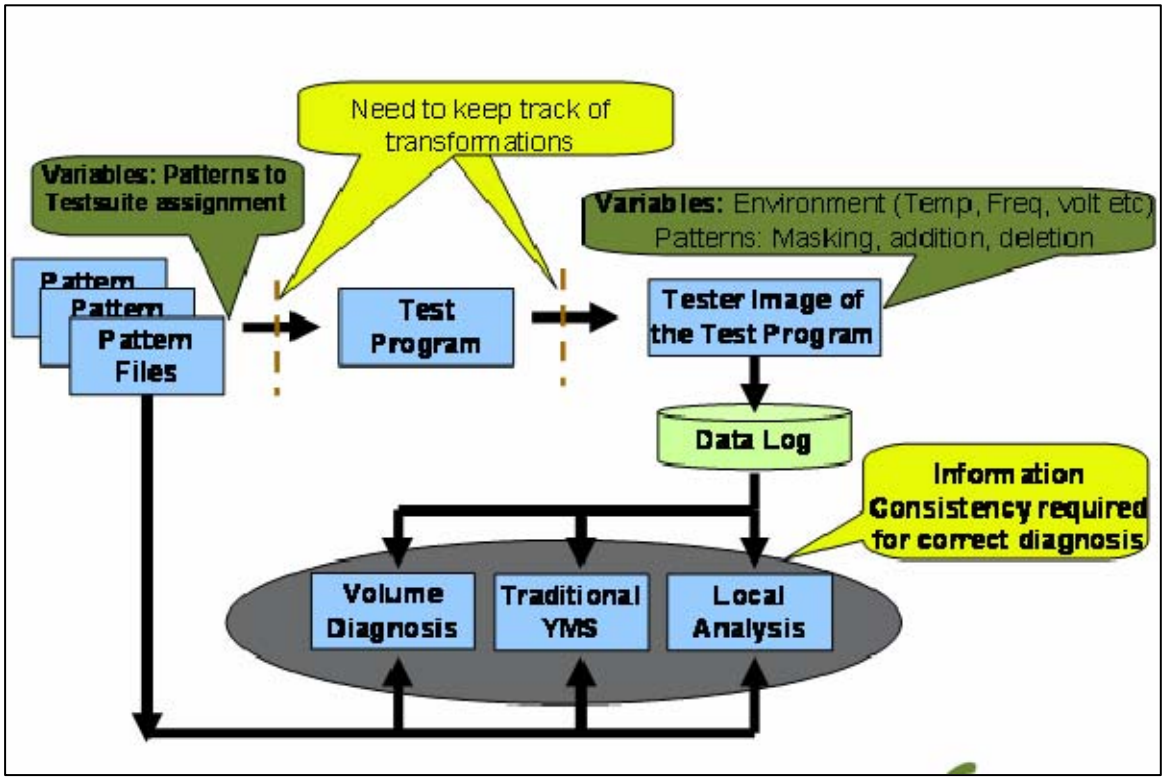

Figure 6: Conceptual Data Flow

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.

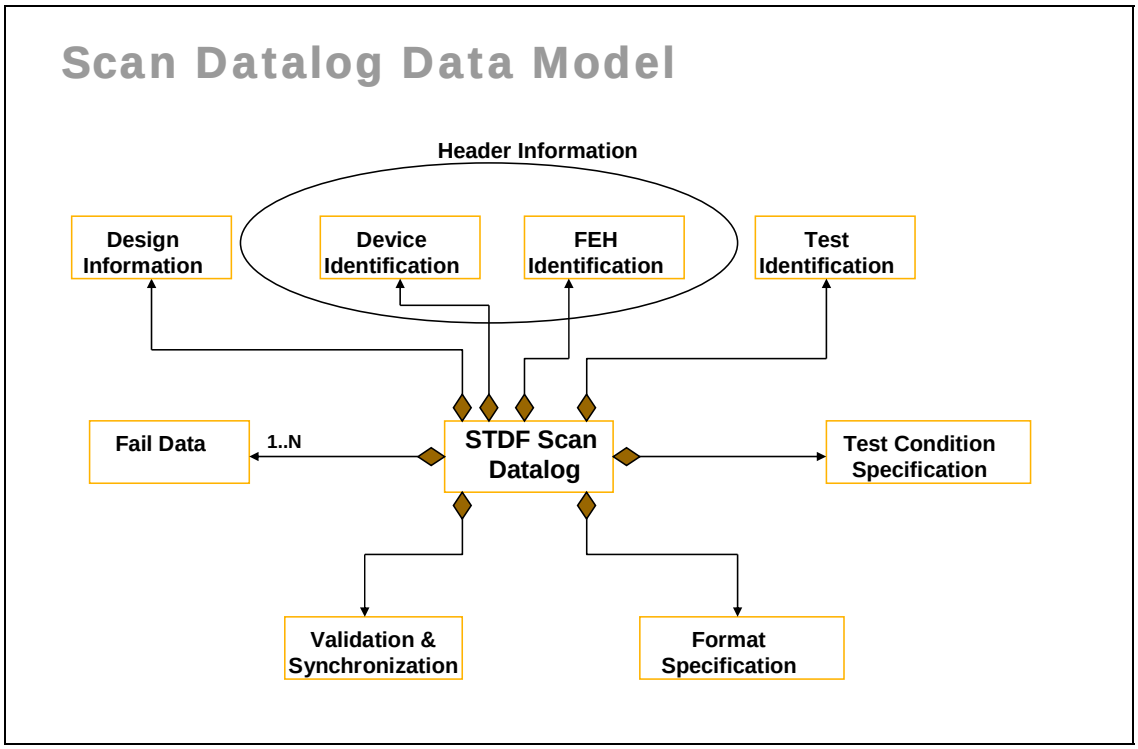

Figure 7: Data Model for Scan Fail Datalog

Design Informtion

The Design Information object allows the information related to design and DFT to be stored in the datalog. The design/DFT information that is relevant for volume diagnosis consists of:

- Design netlist identification
- Scan strcuture information.

Device Identification

The Device Identification Object 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.

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, Wafer Sort2, 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.

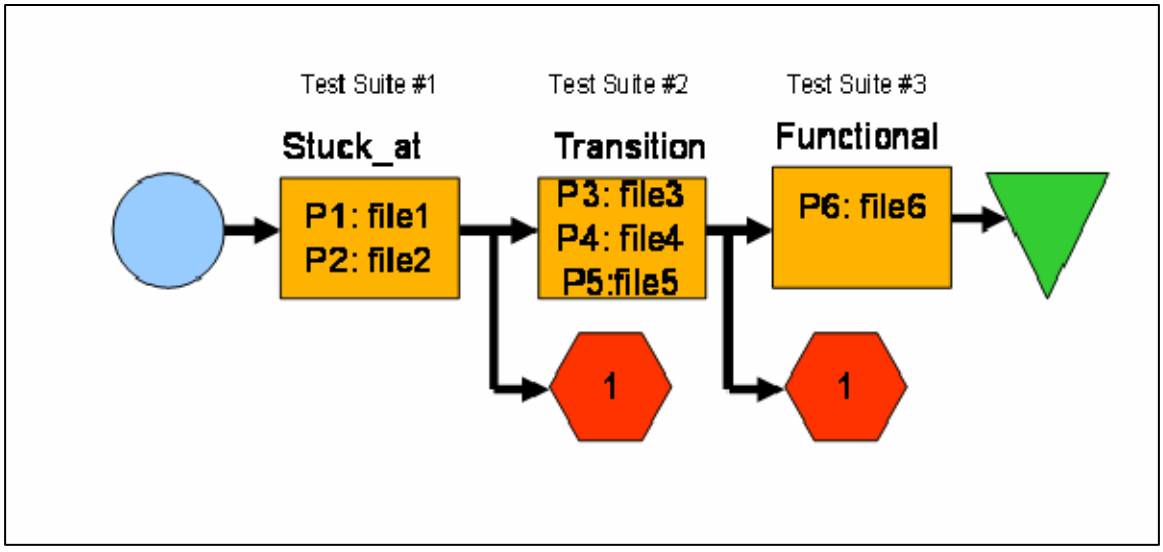

Figure 8: Simple Test Flow

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: It provides information on status of execution of a test e.g. whether all the information was captured or not w.r.t. a test.

Format Specification
Format specification data indicates the conventions used in the datalog for the fail data. In particular it contains following information:

- Information on the type of 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. A flag in the format specification indicates the log type that follows in the datalog.
- Information on how the Z state is handled in the datalog. The Z-handling information allows one to handle a tester's capabilities in terms of making a dual limit comparison.

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.

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 1 shows the existing major record types in STDF V4.

Table 1: STDF Major Record Types

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 1. Table 2 shows a structural overview of the standard

Table 2: Structural Overview of the New Standard

Purpose	Records
Version Identification	Version Update Record
Per Lot Information	Pattern Sequence Record Name Map Records Scan Structure Records
Device and Test Setup Information	Device Identification Records Test Identification Records FEH Identification Records
Per Test Execution Information	Scan Test Record ⋮ Scan Test Record

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 (existing usage)
- STDF V4 with existing plus the scan fail information
- STDF V4 with Scan fail information only

STDF Record Structure

This section describes the basic STDF record structure. It describes the following general topics, which are applicable to all the record types:

- STDF record header
- Record types and subtypes
- Data type codes and representation
- Optional fields and missing/invalid data

STDF Record Header

Each STDF record begins with a record header consisting of the following three fields:

Field	Field Description
REC_LEN	The number of bytes of data following the record header
REC_TYP	An integer identifying a group of related STDF record types
REC_SUB	An integer identifying a specific STDF record type within each REC_TYP group.

Record Types and Subtypes

The header of each STDF record contains a pair of fields called REC_TYP and REC_SUB. Each REC_TYP value identifies a group of related STDF record types. Each REC_SUB value identifies a single STDF record type within a REC_TYP group. The combination of REC_TYP and REC_SUB values uniquely identifies each record type. This design allows groups of related records to be easily identified by data analysis programs, while providing unique identification for each type of record in the file.

STDF Record Structure

The following table lists the meaning of the REC_TYP codes in STDF V4-2007.

REC_TYP	Code Meaning and STDF REC_SUB Codes
0	Information about the STDF file 10 File Attributes Record (FAR) 20 Audit Trail Record (ATR) 30 Version Update Record (VUR)
1	Data collected on a per lot basis 10 Master Information Record (MIR) 20 Master Results Record (MRR) 30 Part Count Record (PCR) 40 Hardware Bin Record (HBR) 50 Software Bin Record (SBR) 60 Pin Map Record (PMR) 62 Pin Group Record (PGR) 63 Pin List Record (PLR) 70 Retest Data Record (RDR) 80 Site Description Record (SDR) 90 Pattern Sequence Record (PSR) 91 Name Map Record (NMR) 92 Cell Name Record (CNR) 93 Scan Structure Record (SSR) 94 Scan Chain Record (SCR)
2	Data collected per wafer 10 Wafer Information Record (WIR) 20 Wafer Results Record (WRR) 30 Wafer Configuration Record (WCR)
5	Data collected on a per part basis 10 Part Information Record (PIR) 20 Part Results Record (PRR)
10	Data collected per test in the test program 30 Test Synopsis Record (TSR)
15	Data collected per test execution 10 Parametric Test Record (PTR) 15 Multiple-Result Parametric Record (MPR) 20 Functional Test Record (FTR) 30 Scan Test Record (STR)

20	Data collected per program segment 10 Begin Program Section Record (BPS) 20 End Program Section Record (EPS)
50	Generic Data 10 Generic Data Record (GDR) 30 Datalog Text Record (DTR)
180	Reserved for use by Image software
181	Reserved for use by IG900 software

Data Type Codes and Representation

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.

Code	Description	C Type Specifier
C*12	Fixed length character string: If a fixed length character string does not fill the entire field, it 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 bytes = unsigned count of bytes to follow (maximum of 65535 bytes)	char[]
C*f	Variable length character string: string length is stored in another field	char[]
U*f	Variable type U type fields where the type f is stored in another field can have value 1,2 or 4	
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	Unsigned long long
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
*k*x*TYPE*	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 ***k*xU*2**.	TYPE[]

Optional Fields and Missing/Invalid Data

Certain fields in STDF records are defined as optional. An optional field must be present in the record, but there are ways to indicate that its value is not meaningful, that is, that its data should be considered missing or invalid. STDF V4 – 2007 builds on mechanisms of STDF V4 for missing or invalid fields. It uses the same mechanism as the STDF V4 for indicating missing/invalid data as described below:

- Some optional fields have a predefined value that means that the data for the field is missing. For example, if the optional field is a variable-length character string, a length byte of 0 means that the data is missing. If the field is numeric, a value of -1 may be defined as meaning that the data is missing.
- For other optional fields, all possible stored values, including -1, are legal. In this case, the STDF specification for the record defines an Optional Data bit field. Each bit is used to designate whether an optional field in the record contains valid or invalid data. Usually, if the bit for an optional field is set, any data in the field is invalid and should be ignored.

Data Type	**Missing/Invalid Data Flag**
Variable-length string	Set the length byte to 0
Fixed-length character string	Fill the field with spaces.
Fixed-length binary string	Set a flag bit in an Optional Data byte.
Time and date fields	Use a binary 0.
Signed and unsigned integers and floating point values	Use the indicated reserved value or set a flag bit in an Optional Data byte.

However for the omission of the optional fields the new standard uses explicit control flags/bits to indicate the absence of optional fields and the first missing optional field does not mean that the rest of the optional fields in the records are missing as well.

Continuation of 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. 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. The continuation mechanism is implemented by adding two fields immediate after the Record Subtype fields. These fields are called REC_INDX and REC_TOT to implement an X of Y notation i.e. the REC_INDX field indicates the index of a record among the REC_TOT number of continuation records. Thus each record in a series of continuation records gets a unique REC_INDX between 1 and REC_TOT. In addition for each field in a record that span over multiple record, there is local and global count field associated with it (The exact names of these fields depend on the record) that indicate how many items of that field are present in the current record and how many items are there in total over all the continuation records. Also, the semantics for the readers in such cases is to concatenate all the items across the records to obtain the values for that field.

STDF Record Types

This section contains the definitions for the STDF record types. The following information is provided for each record type:

- a statement of function: how the record type is used in the STDF file.
- a table defining the data fields: first the standard STDF header, then the fields specific to this record type. The information includes the field name, the data type (see the previous section for the data type codes), a brief description of the field, and the flag to indicate missing or invalid data (see the previous section for a discussion of optional fields).
- any additional notes on specific fields.
- possible uses for this record type in data analysis reports. Note that this entry states only where the record type can be used. It is not a statement that the reports listed always use this record type, even if Teradyne has written those reports. For definitive information on how any data analysis software uses the STDF file, see the documentation for the data analysis software.
- frequency with which the record type appears in the STDF file: for example, once per lot, once per wafer, one per test, and so forth.
- the location of the record type in the STDF file. See the note on “initial sequence”

A note on Initial Record Sequences

For several record types, the "Location" says that the record must appear "after the initial sequence." The phrase "initial sequence" refers to the records that must appear at the beginning of the STDF file. The requirements for the initial sequence are as follows:

- Every file must contain one File Attributes Record (FAR), one VUR, one Master Information Record (MIR), one or more Part Count Records (PCR), and one Master Results Record (MRR). All other records are optional.
- The first record in the STDF file must be the File Attributes Record (FAR).
- If one or more Audit Trail Records (ATRs) are used, they must appear immediately after the FAR.
- Version Update Record (VUR) which is a required record for STDF V4 – 2007 must appear after the last ATR and before MIR
- The Master Information Record (MIR) must appear in every STDF file. Its location must be after the FAR , ATRs (if ATRs are used) and VUR.
- If the Retest Data Record (RDR) is used, it must appear immediately after the MIR.
- If one or more Site Description Records (SDRs) are used, they must appear immediately after the MIR and RDR (if the RDR is used).

Given these requirements, every STDF V4 2007 must contain one of these initial sequences:

FAR – VUR – MIR
FAR – ATRs – VUR – MIR
FAR – VUR – MIR – RDR
FAR – ATRs – VUR – MIR – RDR
FAR – VUR – MIR – SDRs
FAR – ATRs – VUR – MIR – SDRs
FAR – VUR – MIR – RDR – SDRs
FAR – ATRs – VUR – MIR – RDR – SDRs

All other STDF record types appear after the initial sequence.

Version Update Record (VUR)

Function: Version update Record is used to identify the updates over version V4. Presence of this record indicates that the file may contain records defined by the new standard. This record is added to the major type 0 in the STDF V4.

Data Fields:

Field Names	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN:	U*2	Bytes of data following header	
REC_TYP:	U*1	Record type (0)	
REC_SUB:	U*1	Record sub-type (30)	
UPD_NAM	C*n	Update Version Name	

Field Description:

UPD_NAM: This field will contain the version update name. For example the new standard name will be stored as "**V4–2007**" string in the UPD_NAM field.

A note on 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)
- Wafer Result Record (WRR)
- Part Information Record (PIR)
- 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)

Function: 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 VUR. This will allow any data reporting or analysis programs access to this information in the shortest possible amount of time.

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

Data Fields:

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 (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)	65535
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
SBLOT_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
TEST_COD	C*n	Test phase or step code	length byte = 0
TST_TEMP	C*n	Test temperature	length byte = 0
USER_TXT	C*n	Generic user text	length byte = 0

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

Field Description:

MODE_COD: Indicates the station mode under which the parts were tested. Currently defined values for the MODE_COD field are:

A = AEL(Automatic Edge Lock) mode
C = Checker mode
D = Development / Debug test mode
E = Engineering mode (same as Development mode)
M = Maintenance mode
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)

Function: 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.

Data Fields:

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

Field Description:

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.

Wafer Result Record (WRR)

Function: Contains the result information relating to each wafer tested by the job plan. The WRR and the Wafer Information Record (WIR) bracket all the stored information pertaining to one tested wafer. This record is used only when testing at wafer probe time. A WIR/WRR pair will have the same HEAD_NUM and SITE_GRP values.

Data Fields:

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
FINISH_T	U*4	Date and time last part tested	
PART_CNT	U*4	Number of parts Tested	
RTST_CNT	U*4	Number of parts Retested	4,294,967295
ABRT_CNT	U*4	Number of aborts during testing	4,294,967295
GOOD_CNT	U*4	Number of good (passed) parts tested	4,294,967295
FUNC_CNT	U*4	Number of functional parts tested	4,294,967295
WAFER_ID	C*n	Wafer ID	length byte = 0
FABWF_ID	C*n	Fab wafer ID	length byte = 0
FRAME_ID	C*n	Wafer Frame ID	length byte = 0
MASK_ID	C*n	Wafer mask ID	length byte = 0
USR_DESC	C*n	Wafer description supplied by user	length byte = 0
EXC_DESC	C*n	Wafer description supplied by exec	length byte = 0

Field Description:

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. A wafer ID in WRR supersedes Wafer ID found in WIR.

FABWF_ID: Is the ID of the wafer when it was in the fabrication process. This facilitates tracking of wafers and correlation of yield with fabrication variations.

FRAME_ID: Facilitates tracking of wafers once the wafer has been through the saw step and the wafer ID is no longer readable on the wafer itself. This is an important piece of information for implementing an inkless binning scheme.

Part Information Record (PIR)

Function: This record acts as a marker to indicate where testing for a particular part begins for each part tested by the test program. The PIR and Part Results Record (PRR) bracket all the stored information pertaining to one tested part.

Data Fields:

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 (10)	
HEAD_NUM	U*1	Test head number	
SITE_NUM	U*1	Test site number	

Field Description:

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.

Part Results Record (PRR)

Function: 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)

The PRR and the Part Information Record (PIR) bracket all the stored information pertaining to one tested part

Data Fields:

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

Field Description:

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. It is optional to store the Electronic ID, in that case the PART_TXT will be used in the same way as in STDF V4.

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. It is optional to add the part identification in that case, the PSRT_ID will be used as before in STDF V4.

Test Synopsis Record (TSR)

Function: Contains the test execution and failure counts for one parametric, functional, or scan 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), the Multiple Parametric Test Record (MPR), and the Scan Test Record (STR) by test number, head number, and site number.

Data Fields:

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 bit2= 1
TEST_MIN	R*4	Lowest test result value	OPT_FLAG bit0= 1
TEST_MAX	R*4	Highest test result value	OPT_FLAG bit1= 1
TST_SUMS	R*4	Sum of test result values	OPT_FLAG bit4= 1
TST_SQRS	R*4	Sum of squares of test result values	OPT_FLAG bit5= 1

Field Description:

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
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)

Function: This record is used to identify the equipment in the test cell where the part was tested and contains the configuration information for one or more test sites, connected to one testhead, that compose a site group. 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)

Data Fields:

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 (*k*) 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

Field Description:

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)

Function: PSR record contains the information on the pattern profile for a 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 in a test program. A PSR is referenced by the STR (Scan Test Record) using its PSR_INDX field

Data Fields:

Field Names	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 (90)	
REC_INDX	U*1	Record index # in context of total records used to contain a complete PSR data set	
REC_TOT	U*1	Record Total records used to contain a complete PSR data set	
PSR_INDX	U*2	PSR Record Index (used by STR records)	
PSR_NAM	C*n	Symbolic name of PSR record	length byte = 0
OPT_FLG	B*1	Contains PAT_LBL, FILE_UID, ATPG_DSC, and SRC_ID field missing flag bits and flag for start index for first cycle number.	
TOTP_CNT	U*2	Count of total pattern file information sets in the complete PSR data set	
LOCP_CNT	U*2	Count (k) of pattern file information sets in this record	
PAT_BGN	kxU*8	Array of Cycle #'s patterns begins on (Cycle #1 is 1st cycle executed)	
PAT_END	kxU*8	Array of Cycle #'s patterns stops at	
PAT_FILE	kxC*n	Array of Pattern File Names	
PAT_LBL	K*C*n	Optional pattern symbolic name	OPT_FLG bit 0 = 1
FILE_UID	kxC*n	Optional array of file identifier code	OPT_FLG bit 1 = 1
ATPG_DSC	kxC*n	Optional array of ATPG information	OPT_FLG bit 2 = 1
SRC_ID	kxC*n	Optional array of PatternInSrcFileID	OPT_FLG bit 3 = 1

Field Description:

REC_INDX: This is the index of the current PSR record as part of the current PSR set to complete the description of patterns for a test

REC_TOT: Total number of PSR records that make the PSR set to described all the patterns in a test.

PSR_INDX: This is a unique identifier for the set of PSRs that describe the patterns for a scan test.

PSR_NAM: It is a symbolic name of the test suite to which this PSR belongs. For example with reference to figure 8, it would be stuck-at for the test_suite #1.

OPT_FLG: This flag is used to indicate the presence of optional fields. The bit assignment for the optional fields in as shown in Table 3. If the bit is set to 1 the corresponding optional field is considered missing

Table 3: Optional Field Flags in PSR

Bit	Description
0	Symbolic pattern label missing
1	Unique File Identifier for the file is missing
2	Details of ATPG used to create the patterns
3	Identification of a pattern within a source file
4	The first cycle number is determined by the value of this bit.

TOTP_CNT: This field indicates the total number of pattern 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. For continuation records this should be the same count as for the first record (i.e. the final total)

LOCP_CNT: This field indicates the total number of patterns that are described in the current PSR from a scan test.

NOTE 1 The next set of fields is repeated for each pattern that is contained in a scan test. Each of **these** fields is stored in its own array of size LOCAL_CNT.

PAT_FILE : The name of the ATPG file from which the current pattern was created

PAT_BGN: The cycle count the specified ATPG pattern begins on. The 1st cycle number is determined by the OPT_FLG (bit 4).

PAT_END : The cycle count the specified ATPG pattern ends on.

PAT_LBL: (Optional) This is a symbolic name of the pattern within a test suite. For example, with reference to figure 8 it will be P1 for the pattern coming from file1.

FILE_UID: (Optional) - Unique character string that uniquely identifies the file. This field is provided 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 specific 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, this field specifies the one, which is the source of the pattern in this PSR

ATPG_DSC (Optional) – This field intended to be used to store any ASCII data that can identify the source tool, time of generation etc.

Scan Test Record (STR)

Function: 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 records exist. Thus the scan test record becomes another test record type in this category.

It 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 of 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 record may be followed by additional continuation STR records.

In this new record some fields have been brought over from the functional test record and some new fields have been added to handle the scan test data. Table 4 shows the structure of the STR at a conceptual level.

Table 4: STR Record Structure

Class of Information	Information Fields
Continuation Fields	(X of Y)
Flags for Optional Fields	FAL Flag, Mask Map Flag
Test Setup Information	Test No, Test Head Number, Test Site, Test Flags Etc (Brought over from FTR)
Validation and Synchronization Information	Buffer fail status, Global Mask Status, Fail limit Info,
Test Condition Specification with Optional Fields Specification	Scan_Freq, Capture Frequency, Voltage, Optional
Datalog Format Specification	Z-handling, Data Field Map
Datalog with Per fail optional Fields	Pin Info, Pattern/Cycle no/FF Name, Measured Data, Expected data, Fail data, optional data

The fields in the STR records correspond to various categories of table 4. Table below shows the details of an STR record. The description of the STR fields is classified according to those categories in Table 4.

Data Fields:

Field Names	Data Type	Field Description	Missing/Invalid Data Flag
REC_LEN:	U*2	Bytes of data following header	
REC_TYP:	U*1	Record type (15)	
REC_SUB:	U*1	Record sub-type (30)	
REC_INDX	U*1	Record index # in context of total records used to contain a complete STR data set	
REC_TOT	U*1	Record Total records used to contain a complete STR data set	
TEST_NUM	U*4	Test Number	
HEAD_NUM	U*1	Test head number	
SITE_NUM	U*1	Test site number	
PSR_REF	U*2	PSR Index (Pattern Sequence Record)	
TEST_FLG	B*1	Test flags (fail, alarm, etc.)	
LOG_TYP	C*n	User defined description of datalog	length byte = 0
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
Z_VAL	U*1	Z Handling Flag	
FMU_FLG	B*1	MASK_MAP & FAL_MAP field status & Pattern Changed flag	
MASK_MAP	D*n	Bit map of Globally Masked Pins	FMU_FLG bit 0 = 0 OR bit1 = 1
FAL_MAP	D*n	Bit map of failures after buffer full	FMU_FLG bit 2 = 0 OR bit3 = 1
CYC_CNT	U*8	Total cycles executed in test	
TOTF_CNT	U*4	Total failures (pin x cycle) detected in test execution	
TOTL_CNT	U*4	Total fails logged across the complete STR data set	
CYC_BASE	U*8	Cycle offset to apply for the values in the CYCL_NUM array	
BIT_BASE	U*2	BIT_POS offset during capture cycles in the pattern/chain/bit logging. (>> any valid BIT_POS and to be set by the environment)	

DATA_FLG	B*1	Specifies the presence of defined data arrays in this record	
COND_CNT	U*2	Count (g) of Test Conditions and optional data specifications in present record	
LOCL_CNT	U*4	Count (k) of fails logged in this record	
LIM_CNT	U*2	Count (j) of LIM Arrays in present record, 1 for global specification	
DATA_BIT	U*1	Number of (1,2,4,8) bits used per failure in the CAP_DATA, NEW_DATA, & EXP_DATA fields	
DATA_CHR	C*n	string containing 2, 4, 8, or 16 characters indexed by the bits specified in DATA_BIT	length byte = 0
DATA_CNT	U*2	Count (m) of number of data bytes recorded in present record. m = [(LOCL_CNT *DATA_BIT)/8)]	
USR1_LEN	U*1	Length (f) for the data type in USR1 field (f can only be 0,1, 2, or 4)	
USR2_LEN	U*1	Length (f) for the data type in USR2 field (f can only be 0,1, 2, or 4)	
USR3_LEN	U*1	Length (f) for the data type in USR3 field (f can only be 0,1, 2, or 4)	
TXT_LEN	U*1	Length (f) of each string entry in USER_TXT array	
LIM_INDX	jxU*2	Array of PMR indexes that require unique limit specifications	LIM_CNT=0
LIM_SPEC	jxU*4	Array of fail datalogging limits for the PMRs listed in LIM_INDX	LIM_CNT=0
COND_NAM	gxC*n	Array of test condition names	COND_CNT=0
COND_VAL	gxC*n	Array of test condition value	COND_CNT=0
CYCL_NUM	kxU*4	Array of cycle numbers relative to CYC_BASE	DATA_FLG bit0=1
PMR_INDX	kxU*2	Array of PMR Indexes (All Formats)	DATA_FLG bit1=1
CHN_NUM	kxU*2	Array of Chain No for FF Name Mapping	DATA_FLG bit2=1
CAP_DATA	mxU*1	Array of captured data	DATA_FLG bit3=1
EXP_DATA	mxU*1	Array of expected vector data	DATA_FLG bit4=1
NEW_DATA	mxU*1	Array of new vector data	DATA_FLG bit5=1
PAT_NUM	kxU*4	Array of pattern # (Ptn/Chn/Bit format)	DATA_FLG bit6=1
BIT_POS	kxU*4	Array of chain bit positions (Ptn/Chn/Bit format)	DATA_FLG bit7=1
USR1	kxU*f	Array of user defined data for each	USR1_LEN= 0;

		logged fail	
USR2	kxU*f	Array of user defined data for each logged fail	USR2_LEN= 0;
USR3	kxU*f	Array of user defined data for each logged fail	USR3_LEN= 0;
USER_TXT	kxC*f	Array of user defined fixed length strings for each logged fail	TXT_LEN = 0

Field Description:

TEST_NUM: It is the identifier for the test for which data is collected. It should be populated with the Test 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 reliable
1 = Test result is unreliable
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

LOG_TYP: This is a user defined description of the datalog type to indicate what type of information is stored in the current record.

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

PSR_REF: This is the reference to PSR(s) that describes the patterns for the test for which the data log is stored in the current STR. In case the patterns are described by multiple continuation PSRs then the reference would contain the index of the the first PSR.

Z_VAL: This is the flag to indicate the how the Z (Dual side comparison) values were handled on the ATE. Table 5 shows the modes that are supported. The first entry is for the ATEs which cann'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. Enumeration value '3' (Z mapped to Z) means that a compare to mid-level (between L and H state) is done for 'expected Z' in the patterns. This typically requires a dual-threshold comparator combined with an active load circuit on the ATE side, to pull the device output signals to mid-level when its output drivers are tristated.

Table 5: Z Handling Flags

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

FMU_FLG: This field describes the status of the fails after logging, Mask Map fields in STR as well as the flag to indicate pattern modified flag. Bits 0 and 1 indicate the status of the FAL_MAP and the Bit 2 and 3 described the status of Mask map. Table 6 shows the modes of the FAL_MAP and Table 7 shows the modes for Mask map

Table 6: FAL MAP Flag Description

FMU_FLG:bit 0	FMU_FLG:bit 1	Description
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

Table 7: Mask Map Flag Description

FMU_FLG:bit 2	FMU_FLG:bit 3	Description
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 is persistent
1	1	Unused

Bit 4 is used to indicate if the patterns on the ATE have been modified compared to the previous image. If bit 4 is set to 1 then indicates that patterns have been modified and a 0 means no change

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, then 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 Fail bits in FMU_FLG and the behavior is described in Table 7

FAL_MAP (Failures After Logging Map): : This is an 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, the 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 in FMU_FLG and the behavior is described in Table 6.

Validation and Synchronization Fields

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

TOTF_CNT: 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

TOTL_CNT: The total number of failures that were detected and loggeed 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_BASE: This field is of type U*8 and contains the offset for the cycle numbers in the fail log. This mechanism 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_BASE+ <fail Cycle No in the log>. The cycle numbers are the ATPG cycle numbers i.e. in case of multiple ATPG cycles being represented in a tester cycle the tester cycle numbers will have to be decoded into ATPG cycles.

BIT_BASE: This field is of type U*2 and contains the offset for the bit positions in the fail log for the capture cycles in the pattern/chain/bit format. Please note that in the pattern/chain/bit format the BIT_POS does not correspond to any scan chain position during the capture cycles. Therefore a separate number space is created for the BIT_POS values during the capture cycles to avoid confusion with the valid BIT_POS across all the scan chains.

DATA_FLG: This field is used as the mask for the datalog fields i.e. each bit in the DATA_FLG corresponds to a field in the datalog. If a bit in DATA_FLG is set to a 1 then the corresponding field is absent in the failure datalog that follows. Table 8 Shows the mapping of DATA_FLG bits to the datalog fields:

Table 8: Data Flag field Description

Bit position	Corresponding Field
0	CYC_NUM
1	PMR_INDX
2	CHN_NUM
3	CAP_DATA
4	EXP_DATA
5	NEW_DATA
6	PAT_NUM
7	BIT_POS

The fields in this table are described later in the datalog section.

LOCL_CNT: This is the count of the failures that are logged in the current STR record. Please note that the total number of failures for device may require multiple STR records to store them.

LIM_CNT: This field specifies the number of pins for which the fail datalog limits specification is provided in the record.

DATA_BIT: This field indicated the number of bit used to represent the captured data (CAP_DATA), pattern/expected data (EXP_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 mapping table (DATA_CHR) is provided in the STR record to map the values in the array to corresponding waveform characters.
- For 8 bit cases, the value is assumed to be the waveform character (ASCII)
- For CAP_DATA any of the possible values (1, 2, 4, or 8) can be used
- For EXP_DATA and NEW_DATA only values possible are 4 or 8.

DATA_CHR: This field contains the mapping table for converting bits in the log to the corresponding waveform character as mentioned above. It is organized as a string where byte of the string contains the waveform character corresponding the bits in the binary order.

DATA_CNT: This field contains the size of the array in bytes (U*1 used as a byte) to represent CAP_DATA, EXP_DATA and NEW_DATA as mentioned above. Its value is computed by the expression ceiling((LOCL_CNT*DATA_BIT)/8).

USR1_LEN, USR2_LEN,USR3_LEN: These fields define the data field type for the user define fields USR1, USR2 and USR3 respectively. They can have the values defined in the table below with their corresponding data size

Table 9: User Defined Optional Field Type Selection

USR<1,2,3>_LEN Value	Data Type
0	User defined field is not present
1	U*1
2	U*2
4	U*4

TXT_LEN: This field specifies the length of the optional string arrays in the datalog. If this is set to 0 then the optional field USER_TXT is considered missing in the record..

LIM_INDX: 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_SPEC: This array contains the fail log limits for the pins which are listed in the LIM_INDX array. The first location in this array is, location 0 and contain 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 using the COND_NAM and COND_VAL fields. The conditions are represented in ASCII in these arrays.

COND_CNT: This is the count of how many conditions are specified in the COND_NAM and COND_VAL fields 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 stored 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 reserves the following two COND_NAM keywords.

- 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, the temperature does not need to 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

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_FLG field as mentioned above. The fields are organized as one array for each field. i.e. the STR record contains one array of the element for a particular field that is not masked in the DATA_FLG. The number of elements in these arrays is specified by the LOCL_CNT and DATA_CNT fields.

CYC_NUM: An array of cycle numbers (U*4) 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 determined by the OPT_FLG (bit 3) in PSR record for the pattern. Please note that the CYC_NUM is relative to the CYC_BASE field described earlier and the actual cycle for the fail log entries would be CYC_BASE + <fail Cycle No in the log>. The size of this array is specified in the LOCL_CNT field.

PMR_INDX: An array of PMR indexes (U*2) that correspond either to the CYC_NUM array or to the PAT_NUM array. The size of this array is specified in the LOCL_CNT field.

CHN_NUM: An array of scan chain number at which the failure was observed. The size of this array is specified in the LOCL_CNT. The main purpose of this field for supporting the direct scan cell name based logging. This filed along with the BIT_POS should be used to find the CNR record (described later) that contains the scan cell name corresponding to this CHN_NUM and BIT_POS. The scan chain numbering is decided by the user environment. This field optionally can also be used with the PMT_INDX to store corresponding chain numbers in Cycle based on Pattern/Chain/Bit based fail logging **.**

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 EXP_DATA array except 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. The size of this array is specified in the DATA_CNT field.

EXP_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. The size of this array is specified in the DATA_CNT field. This data field has multiple applications:

1. Can be used when it is required to provide the expect data associated with each failure. This can be used to provide validation (when required) of the expected data for a cycle #' on the ATE to the expected data in 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.)
2. 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. The size of this array is specified in the DATA_CNT field.

PAT_NUM: When used in the "Pattern" format, an array of integers (U*4) specifying the pattern number. The size of this array is specified in the LOCL_CNT field.

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 CHN_NUM field to support the applications where Flip

Flop name needs to be derived using SSR. The CHN_NUM and BIT_POS together will be used as an index in the SSR table for getting the Flip Flop name in that case. The size of this array is specified in the LOCL_CNT field.

USR1, USR2, USR3: These are optional user defined fields. The presence and the type of these fields are defined by the corresponding USR1_LEN, USR2_LEN and USR3_LEN fields as mentioned earlier. The size of this array is specified in the LOCL_CNT field.

USER_TXT: This is an optional fixed length text field that can be added to the datalog with each fail. The length of this field is same for each fail in the datalog and is indicated by the TXT_LEN field in the STR record. The size of this array is specified in the LOCL_CNT field.

Name Map Record (NMR)

Function: This record contains a map of PMR indexes to ATPG signal names. This record is designed to allow preservation of ATPG signal names used in the ATPG files through the datalog output. This record is only required when the standard PMR records do not contain the ATPG signal name

Data Fields:

Field Names	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 (91)	
REC_INDX	U*1	Record index # in context of total records used to contain a complete NMR map set	
REC_TOT	U*1	Record Total records used to contain a complete NMR map set	
TOTM_CNT	U*2	Count of PMR indexes and ATPG_NAM entries	
LOCM_CNT	U*2	Count of (k) PMR indexes and ATPG_NAM entries in this record	
PMR_INDX	kxU*2	Array of PMR indexes	
ATPG_NAM	kxC*n	Array of ATPG signal names	

REC_INDX: Index within all the REC_TOT number of NMR records that store the mapping table, starting at 1.

REC_TOT: Total number of NMR records that are required to store the mapping table.

TOTM_CNT: This the count of total number of entries in the mapping table across all the NMR records

LOCM_CNT: The count of number of entries in the current NMR record.

PMR_INDX: It is the array of PMR indexes for which the ATPG names are provided

ATPG_NAM: It is the array ATPG signal names corresponding to the pin in PMR_INDX array.

Scan Cell Name Record (CNR)

Function: This record is used to store the mapping from Chain and Bit position to the Cell/FlipFlop names. A CNR record should be created for each Cell for which a name mapping is required. Typical usage would be to create a record for each failing cell/FlipFlop. A CNR with new mapping for a chain and bit position would override the previous mapping.

Data Fields:

Field Names	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 (92)	
CHN_NUM	U*2	Chain number. Referenced by the CHN_NO array in an STR record	
BIT_POS	U*2	Bit position in the chain	
CELL_NAM	S*n	Scan Cell Name	

Field Description:

CHN_NUM: This is the array for the chain identification for the target Flip Flop 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 Flip Flop for which the name is provided in the table.

CELL_NAM: The is an array name of the of Flip Flop at the CHN_NUM and BIT_POS. Please note the type of this field is S*n where the length of the Flip Flop name is stored in the first two bytes and then the name follows.

Scan Structure Record (SSR)

Function: This record contains the Scan Structure information normally found in a STIL file. The SSR is a top level Scan Structure record that contains an array of indexes to SCR records which contain the chain descriptions.

Data Fields:

Field Names	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 (93)	
SSR_NAM	C*n	Name of the STIL Scan Structure for reference	Length byte = 0
CHN_CNT	U*2	Count (k) of number of Chains listed in CHN_LIST	
CHN_LIST	kxU*2	Array of SCR Indexes	

Field Descriptions:

SSR_NAM: This is a ASCII unique name for the scan structure record that is normally provided by the STIL **(IEEE 1450)** file**.**

CHN_CNT: It is the count of number of scan chains in a scan structure**.**

CHN_LIST: It is an array of index of each chain that is part of this scan structure. No particular order of the scan chain indexes is specified by the standard.

Scan Chain Description Record (SCR)

Function: This record contains the description of a scan chain in terms of its input, output, number of cell and clocks. Each SCR record contains description of exactly one scan chain. Each SCR is uniquely identified by an index.

WARNING: THESE RECORDS COULD RESULT IN A LOT OF DATA VOLUME. PLEASE USE CAUTION.

Data Fields:

Field Names	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 (94)	
REC_INDX	U*1	Record index # in context of total records used to contain a complete chain data set	
REC_TOT	U*1	Record Total records used to contain a complete chain data set	
SCR_INDX	U*2	SCR Index	
CHN_NAM	C*n	Chain Name	Length byte = 0
TOTS_CNT	U*2	Chain Length (# of scan cells in chain)	
LOCS_CNT	U*2	Count (k) of scan cells listed in this record	
SIN_PIN	U*2	PMR index of the chain's Scan In Signal	0
SOUT_PIN	U*2	PMR index of the chain's Scan Out Signal	0
MSTR_CNT	U*1	Count (m) of master clock pins specified for this scan chain	
SLAV_CNT	U*1	Count (n) of slave clock pins specified for this scan chain	
M_CLKS	mxU*2	Array of PMR indexes for the master clocks assigned to this chain	MSTR_CNT=0
S_CLKS	nxU*2	Array of PMR indexes for the slave clocks assigned to this chain	SLAV_CNT=0
INV_VAL	U*1	0: No Inversion, 1: Inversion	255
CELL_LST	kxS*n	Array of Scan Cell Names	LOCS_CNT=0

Field Descriptions:

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

CHN_NAM: This is an optional ASCII unique name for the scan chain (user defined or derived/copied from ATPG files)**.**

TOTS_CNT: The number of scan cells contained within the scan chain

LOCS_CNT: The number of scan cells listed in this record (the others are listed in continuation SCR records)

SIN_PIN: The PMR record index for the chain's Scan In signal

SOUT_PIN: The PMR record index for the chain's Scan Out signal

MSTR_CNT: The # of master clock pins assigned to the scan chain

SLAV_CNT: The # of slave clock pins assigned to the scan chain

M_CLKS: An optional array of PMR indexes for the chain's master clock pins The length of this array is specified in the MSTR_CNT field

S_CLKS: An optional array of PMR indexes for the chain's slave clock pins The length of this array is specified in the SLAV_CNT field.

INV_VAL: A Boolean value to indicate if the Scan_Out signal is inverted from the Scan_In signal. A 0 value indicated no inversion. A value of 255 indicates unknown status.

CELL_LST: The array of scan cell names.

Appendix

A. ATDF Representation

ATDF is the ASCII representation of the binary STDF. ATDF representation will have the same format as the V4 specification i.e.

Field name: <Value>
….
…..
…..
Field name: <Value>

The field names are the names that are mentioned in the tables for each field. The value will be the value of each field.

B. Data Model to STDF Record Mapping Table

The following table shows the mapping of data model objects onto STDF record. Please note that the information in the data model object in some cases is spread over multiple STDF records

Data Object	Data Field	Mapped Record	Mapped Field	Note
Design Information				
	Design Netlist Name	MIR	DSGN_REV	
	Scan Structure Description	***SSR***		
	Scan Structure Name	SSR	SSR_NAM	
	Num of Scan Chains	SSR	CHN_CNT	
	Chain Descriptions	SSR	CHN_LIST	List of chain description records
	Scan Chain Description	***SCR***		
	Chain ID	SCR	SCR_INDX	
	Chain Name	SCR	CHN_NAM	
	Scan Len	SCR	TOTS_CNT	
	Scan In	SCR	SIN_PIN	
	Scan Out	SCR	SOUT_PIN	
	Master Clock(s)	SCR	MSTR_CNT, M_CLKS	
	Slave Clock(s)	SCR	SLAV_CNT, S_CLKS	
	Inversion	SCR	INV_VAL	
Device Identification				
	Lot ID	MIR	LOT ID	
	Wafer ID	WIR/WRR	WAFER_ID	
	X-Coord	PRR	X_COORD	
	Y-Coord	PRR	Y_COORD	
	Part Num	***PRR***		
	Electronic ID	PRR	PART_TXT	Added as string at the end of existing information
	Serial Number	PRR	PART_ID	Added as string at the end of existing information
Equipment				

Identification				
	Site ID	MIR	FACIL_ID	Same as V4
	TestCell ID	MIR	FLOOR_ID	Same as V4
	Tester ID	MIR	SERL_NUM	Same as V4
	TestHead ID	WIR	HEAD_NUM, SITE_GRP	Same as V4
	Probe Card ID	SDR	CARD_TYP, CARD_ID	Same as V4
	Prober ID	SDR	CONT_TYP, CONT_ID	Same as V4
	Loadboard ID	SDR	LOAD_TYP, LOAD_ID	Same as V4
	Handler ID	SDR	CONT_TYP, CONT_ID	Same as V4
Test Identification				
	Test Program ID			
	Test Stage ID			
	Test Suite ID	STR	TEST_NUM	
	Test PatternMap	***PSR***		
	Number of Patterns	PSR	TOTP_CNT	
	Source File ID	PSR	PAT_FILE	
	Pattern In File	PSR	SRC_ID	
	Unique File Qualifier	PSR	FILE_UID	
	Start Cycle	PSR	PAT_BGN	
	End Cycle	PSR	PAT_END	
	ATPG_REV	PSR	ATPG_DSC	
Test Condition				
	Temp	MIR	TST_TMP	
	Shift Freq	STR	COND_CNT COND_NAM COND_VAL	
	Capture Freq	STR		
	User Defined	STR		
Validation and Synchronization				
	Total Fails	STR	TOTF_CNT	
	Total Logged Fails	STR	TOTL_CNT	
	Total Cycles	STR	CYC_CNT	
	Buffer Depth	STR	LIM_CNT, LIM_PRMS, LIM_SPK	
	Mask Map	STR	FMU_FLG,MASK_MAP	
	Fails After Buffer Full	STR	FMU_FLG, FAL_MAP	
	PatternsModified	STR	FMU_FLG (bit 4)	

	ATPG Signal name map	***NMR***		
	ATPG Signal names	NMR	ATPG_NAM	
Format Specification				
	Z-Handling Flag	STR	Z_VAL	
	Fail Datalog Format	STR	DATA_FLG, USR_FLG	
Datalog				
	Cycle Num	STR	CYC_BASE, CYCL_OFST	
	Pin	STR	PMR_INDX	
	Captured Data	STR	CAP_DATA	
	Expected Data	STR	EXP_DATA	
	New Data	STR	NEW_DATA	
	Pattern Num	STR	PAT_NUM	
	Chain Num	STR	CHN_NO	
	Bit Position	STR	BIT_POS	
	Optional Data 1 (Type - U*1)	STR	USER1_U1	Only one of these types can be used at a time i.e. these are mutually exclusive
	Optional Data 1 (Type – U*2)		USER1_U2	
	Optional Data 1 (Type U*4)		USER1_U4	
	Optional Data 2 (Type - U*1)	STR	USER2_U1	Only one of these types can be used at a time i.e. these are mutually exclusive
	Optional Data 2 (Type – U*2)		USER2_U2	
	Optional Data 2 (Type U*4)		USER2_U4	
	Optional Data 3 (Type - U*1)	STR	USER3_U1	Only one of these types can be used at a time i.e. these are mutually exclusive
	Optional Data 3 (Type – U*2)		USER3_U2	
	Optional Data 3 (Type U*4)		USER3_U4	

C Application Primer and Examples

This section provides the reader with some usage examples of datalogging scan test failures using the new record structures available in STDF V4-2007.

The following depicts an example STDF record sequence for 2 devices where two scan tests are applied to each device. This record exemplifies a typical minimum configuration and does not deploy all of the new STDF records added. Note that in this example multiple records are indicated in some cases. This is because some record data sets will exceed the maximum STDF record size of 65,536 bytes. To accommodate this, the NMR, STR, PSR, and SCR records feature the ability to concatenate multiple records together in order to contain all of the required information of a single data set.

Record Type	Comments
FAR	File Attributes Record (Required)
VUR	Version Update Record (Required for V4-2007)
MIR	Master Information File (Required)
PMR(s)	Signal Pin Records (note 1)
NMR	List of Scan out Pins using ATPG signal names (note 1)
PSR	Applies to test #1
PSR(s)	Applies to test #2 (2 records required)
PIR	Start of Device #1
STR	from scan test #1
STR(s)	From scan test #2 (2 records required)
PRR	End of Device #1
PIR	Start of Device #2
STR	from scan test #1
STR	from scan test #2
PRR	End of Device #2

(Note 1) Each PMR record defines a single signal's attributes and subsequent records will refer to these records by their 'PMR Index". Downstream diagnostic tools will often need to reference the signal name used in the pattern source files (e.g. WGL or STIL). The NMR record is required in those cases where, for whatever reason, the PMR is not able to accommodate the ATPG signal name. The NMR will list those pins that may be referenced by the STR records (generally the "Scan out" pins) and provide the required ATPG signal name.

The Version Update Record (VUR)

VUR record is enables downstream tools to quickly identify that this SDTF file potentially contains Scan Test Failure information. It is a required record for the V4-2007 specification.

VUR

UPD_NAM = "V4-2007"

Name Map Record (NMR)

If we assume that the PMR records (Pin Map Record) were unable to use the signals names specified in the ATPG pattern file (e.g. WGL or STIL) in their LOG_NAM fields, then a NMR record needs to be created. In this example the NMR record needs as a minimum to list all of the scan-out pins that will be subsequently referenced in the STR (Scan Test records), e.g. if we have 32 scan out pins then create the 1st record. Since only one NMR is required in this scenario to list all of the signal indexes and name, REC_TOT is set to 1. REC_INDX specifies which NMR record in the sequence of NMR records the present record resides in. (e.g. "Record m of n")

NMR

REC_INDX = 1
REC_TOT = 1
TOTM_CNT = 32
LOCM_CNT = 32
PMR_INDX = { 34, 17; 83; 22; . . . 99} // 32 Scan out PMR indexes
ATPG_NAM = {"SO1", "SO2", . . . "SO32" } // ATPG signal names

Note: If the corresponding PMR record for any of these pins listed in the NMR record used the ATPG signal in it LOG_NAM field then listing it here again is optional.

Pattern Sequence Record (PSR)

One or more PSR records are required to convey the information to downstream tools about the files created by ATPG tools that were used to construct any specific scan test. In this example we have defined that test #1 and test #2 used a different set of or sequence of patterns file, thus a unique PSR record must be created for each test. In this example PSR #1 will be assumed to be small enough to fit in a single record, but we will spread out PSR#2 across two records for the sake of understanding the use of the REC_INDX and REC_TOT fields. PSR record #1 will be created with the minimum amount of information required, and PSR record #2 will be filled out with all of the optional fields specified.

PSR #1

REC_INDX = 1
REC_TOT = 1
PSR_INDX = 1
OPT_FLAG = 7 // The 3 optional arrays are not included
TOTP_CNT = 2 // Test #1 is comprised from 2 ATPG files
LOCP_CNT = 2 // Both ATPG files are defined in this record
PAT_BGN = {10, 4011 } // *See note*
PAT_END = {4010, 7010} // *See note*
PAT_FILE = {"File1.std", "File2.std" }

File	Starting Cycle	Ending Cycle
Initial Cycles	1	9
File1	10	4,010
File2	4,011	7,010

The 2nd PSR record (for test # 2) is depicted below in two concatenated records. Note that the optional field arrays FILE_UID, ATPG_DSC, and SRC_ID were added.

PSR #2A

```
REC_INDX   = 1          // Record 1 of 2
REC_TOT    = 2
PSR_INDX   = 2
OPT_FLAG   = 0          // All 3 optional arrays are included
TOTP_CNT   = 5          // Test #2 is comprised from 5 ATPG files
LOCP_CNT   = 3          //First 3 ATPG files are defined in this record
PAT_BGN    = {5, 2006, 6006}
PAT_END    = {2005, 6005, 8505 }        // Cycle # pattern file ends on
PAT_FILE   = {"File3.std", "File4.std" , "File5.std" }
FILE_UID   = { "15467289", "54223491", "89923414" }
ATPG_DSC   = { "TetraMax V4.0", "TetraMax V4.0", "TetraM…" }
SRC_ID     = { "PatternExec01", "PatternExec01", "PatternExec01" }
```

PSR #2B

```
REC_INDX   = 2          // Record 2 of 2
REC_TOT    = 2
PSR_INDX   = 2
OPT_FLAG   = 0          // All 3 optional arrays are included
TOTP_CNT   = 5          // Test #3 is comprised from 5 ATPG files
LOCP_CNT   = 2          // Last 2 ATPG files are defined in this record
PAT_BGN    = {8505, 12525}
PAT_END    = {12505, 13405}
PAT_FILE   = {"File6.std", "File7.std" }
FILE_UID   = { "96634527", "94336752" }
ATPG_DSC   = { "TetraMax V4.0", "TetraMax V4.0" }
SRC_ID     = { "PatternExec01", "PatternExec01" }
```

The following table depicts the resultant tester cycle range that each pattern file resides in based on the information supplies in the PSR #2A & 2B.

File	Starting Cycle	Ending Cycle
Initial Cycles	1	4
File3	5	2,005
File4	2,006	6,005
File5	6,006	8,505
File6	8,506	12,505
File7	12,526	13,405

Scan Test Record (STR)

Each failing test execution can create a STR record to contain some or all of the failing cycles detected in the test. In additional any desired prevalent test conditions associated with this test can be added. The amount of information logged in an STR record can be at a minimal a set of two data arrays (failing cycle # and failing pin #), or be expanded to include such additional data as captured data state, expected data state, as well as user defined data. The example below depicts a minimum data set configuration while expanded data sets will be described later in this document.

STR (#1)

```
REC_INDX   = 1
REC_TOT    = 1
TEST_NUM   = 1
HEAD_NUM   = 1
SITE_NUM   = 1
PSR_REF    = 1        // Uses the pattern files defined in PSR #1
TEST_FLG   = “”
TEST_TXT   = “Scan Test #1”
PROG_TXT   = “”
RSLT_TXT   = “Test Failed”
Z_VAL      =4         // All pattern “Z” states are mapped to “X” states
FM_FLG     =  2       // All fails logged, no pins masked (no FAL_MAP or MASK_MAP)
CYC_CNT    =7010      //Total # of cycles in test
TOTF_CNT   = 55       // 55 failures detected in this test
TOTL_CNT   = 55       // All 55 failures are being logged
CYC_BASE   = 0        // Add 0 to all values in the CYCL_OFST array
DATA_FLG   = 252      // Use the CYCL_OFST  & PMR_INDX arrays
COND_CNT   = 2        // Two test conditions are listed  in COND_NAM/COND_VAL
LOCL_CNT   = 55       // All 55 failures are contained in this record
LIM_CNT    = 0        // No fail limit specifications apply
DATA_BIT   = 0        //  n.a. (no EXP_DATA,  CAP_DATA, & NEW_DATA arrays)
DATA_CHR   = 0        // n.a. (no EXP_DATA,  CAP_DATA, & NEW_DATA arrays)
DATA_CNT   = 0        // n.a. (no EXP_DATA,  CAP_DATA, & NEW_DATA arrays)
USR1_LEN   = 0        // USR1 field is not present
USR2_LEN   = 0        // USR2 field is not present
USR3_LEN   = 0        // USR3 field is not present
TXT_LEN    = 0        // Not logging USER_TXT array
COND_NAM   = { “VCC1”, “VCC2” }
COND_VAL   = { “1.3V”, “3.2V” }
CYCL_OFST  = { 233, 456, . . . }          //   array of 55 failing Cycle #’s
PMR_INDX   = { 22, 83, 22, … }            //   array of 55 failing pin #’s
```

Below is the STR record created for the 2nd test. For sake of explanation the record is split into two records (STR 2A & STR 2B) . Note the treatment of certain fields in the 2nd STR record that are "redundant" with those in the 1st record. The general rule is that for fields that don't support multiple values for a single test execution, only the values in the 1st record are applicable. Test #2 also adds the EXP_DATA ("expected data") array to the fail log information.

STR (#2A)

```
REC_INDX    = 1        // Record 1 of 2
REC_TOT     = 2        // Two record constitutes the STR data set
TEST_NUM    = 2
HEAD_NUM    = 1
SITE_NUM    = 1
PSR_REF     =2         // Uses the pattern files defined in PSR #2
TEST_FLG    = ""
TEST_TXT    = "Scan Test #2"
PROG_TXT    = ""
RSLT_TXT    = "Test Failed"
Z_VAL       = "L"
FM_FLG      =  5       // Both MASK_Map and FAL_MAP fields are present
MASK_MAP    = {70, 0, 0 , 32, 0, 0, 0, 0, 0, 0, 0}(1)  // 70 pins, PMR # 13 is masked
FAL_MAP     = {70, 0, 4 , 32, 0, 0, 0, 0, 0, 0, 0}(1)  // 70 pins, PMR # 3 has unlogged fails
CYC_CNT     =13405   // Total cycles in the test
TOTF_CNT    = 325    // 325 failures detected in this test
TOTL_CNT    = 325    // All 325 failures are being logged
CYCLE_BASE  = 0
DATA_FLG    =236     // Use CYCL_OFST, PMR_INDX, and EXP_DATA  arrays
COND_CNT    = 2      // Two test conditions are listed  in COND_NAM/COND_VAL
LOCL_CNT    = 200    // 1st 200 failures are contained in this record
LIM_CNT     = 0      // No fail limit specifications apply
DATA_BIT    =1       //   (1 bit per fail/8 fails per byte) contained in EXP_DATA array
DATA_CHR    = "LH"   //  EXP_DATA Bit(n) = 0 : "L" failure,   = 1: "H" failure
DATA_CNT    = 25     //  25 bytes (200 fails) contained in EXP_DATA array
USR1_LEN    = 0      // USR1 field is not present
USR2_LEN    = 0      // USR2 field is not present
USR3_LEN    = 0      // USR3 field is not present
TXT_LEN     = 0      // Not logging USER_TXT array
COND_NAM    = { "VCC1", "VCC2" }
COND_VAL    = { "1.1V", "2.8V" }
CYCL_OFST   = {1321, . . .,  8013 }          //   200 failing Cycle #'s
PMR_INDX    = { 99, 99, 17, … }              //   200 failing pin #'s
EXP_DATA    = { 103, 87, 101, … }(1)         //   200 failing data states (in 25 bytes)
```

STR (#2B)

```
REC_INDX    = 2        // Record 2 of 2
REC_TOT     = 2        // Two record constitutes the STR data set
TEST_NUM    = 0        // Ignored, inherited from STR 2A
HEAD_NUM    = 0        // Ignored, inherited from STR 2A
SITE_NUM    = 0        // Ignored, inherited from STR 2A
```

```
PSR_REF      = 0        // Ignored, inherited from STR 2A
TEST_FLG     = 0        // Ignored, inherited from STR 2A
TEST_TXT     = 0        // Ignored, inherited from STR 2A
PROG_TXT     = 0        // Ignored, inherited from STR 2A
RSLT_TXT     = 0        // Ignored, inherited from STR 2A
Z_VAL        = 0        // Ignored, inherited from STR 2A
FM_FLG       =  0       // Ignored, use MASK_MAP & FAL_MAP data from STR #2A
CYC_CNT      =0         // Ignored, inherited from STR 2A
TOTF_CNT     = 0        // Ignored, inherited from STR 2A
TOTL_CNT     = 0        // Ignored, inherited from STR 2A
CYCLE_BASE   = 0        // Ignored, inherited from STR 2A
DATA_FLG     = 236      // Use CYCL_OFST, PMR_INDX, and EXP_DATA  arrays
COND_CNT     = 1        // One test condition are listed  in COND_NAM/COND_VAL
LOCL_CNT     = 200      // 1st 200 failures are contained in this record
LIM_CNT      = 0        // Ignored, inherited from STR 2A
DATA_BIT     =1         //   (1 bit per fail/8 fails per byte) contained in EXP_DATA array
DATA_CHR     = "LH"     //  EXP_DATA Bit(n) = 0 : "L" failure,  = 1: "H" failure
DATA_CNT     = 16       //  16 bytes (125 fails) contained in EXP_DATA array
USR1_LEN     = 0        // USR1 field is not present
USR2_LEN     = 0        // USR2 field is not present
USR3_LEN     = 0        // USR3 field is not present
TXT_LEN      = 0        // Not logging USER_TXT array
COND_NAM     = { "CAP_FREQ" }
COND_VAL     = { "765MHz" }
CYCL_OFST    = { 8025, 754, . . . }              //   125 failing Cycle #'s
PMR_INDX     = { 17, 23, 17 …                    //   125 failing pin #'s
EXP_DATA     = { 103, 87, 101, … }(1)            //   125 failing data states (in 16 bytes)
```

Note (1): a list of bytes

Additional Data Arrays Available in the STR Record

In addition to the arrays used in the previous example, there are other optional arrays that can be for logging either additional failure or other test information

As shown in the example for Test #2, the optional **EXP_DATA** array was added to log the expected data state for each failure. In this example only "L" or 'H" data state failures were possible, so one bit per failure was defined for this array **(DATA_BIT** field) which compressed the data into 8 failiyre per byte. The **CAP_DATA** array is intended for logging "captured" data its use is identical to that defined for the **EXP_DATA**. Generally the usage of **CAP_DATA** array is in a test specifically used for the capture of actual devices data independent of expect data, e.g. a "Scan Dump"

The **NEW_DATA** array is provided for the purpose of logging changes made to a pattern within the ATE test program that results in a difference from the original ATPG pattern. Its use is identical to that defined for the **EXP_DATA and CAP_DATA**. It may or not be used in conjunction with the **EXP_DATA** array which would contain the original pattern data. Generally the usage of this field is in a test specifically used for the one time recording of the pattern changes made to a specific test.

Finally, the **USER_TXT** array provides the facility to log a unique fixed string for each logged failure (the string length is set in the **TEXT_LEN** field).

In summary, the typical array combinations in an STR record are:

1) CYCL_OFST + PMR_INDX
2) CYCL_OFST + PMR_INDX + EXP_DATA
3) CYCL_OFST + PMR_INDX + CAP_DATA
4) CYCL_OFST + PMR_INDX + NEW_DATA
5) CYCL_OFST + PMR_INDX + EXP_DATA +NEW_DATA

Logging Changes Made to a Pattern

One of the capabilities added is the ability to log any modifications made to a test pattern that deviate from the pattern data that was in the original ATPG files. First a STR record is created just for logging the changes as per the following example: (LH1X0HHLHH1X0 is being changed to XL0L1XLXXX0L1 on the specified cycles/pins)

STR (Pattern Change)

```
REC_INDX    = 1
REC_TOT     = 1
- - -
PSR_REF     = 1         // Uses the pattern files defined in PSR #1
- - - - -
LOG_TYP     = "Pattern_Change"  // Indicates record purpose to downstream tool
- - - - -
TOTL_CNT    =13         // 13 pattern changes are being logged
CYC_BASE    = 0         // Add 0 to all values in the CYCL_OFST array
DATA_FLG    = 204       // Use CYCL_OFST , PMR_INDX, EXP_DATA, & NEW_DATA
- - - - -
LOCL_CNT    = 13        // All 5 pattern changes are contained in this record
- - - - -
DATA_BIT    = 4                             // 4 bits per cycle used
DATA_CHR    = "LHXZ01xxxxxxxxxx"            //  the data to pattern map for 4 bits
DATA_CNT    = 7                             // 7 bytes are required for 13 pattern changes
- - -
CYCL_OFST   = { 233, 456, . . . }           //  Array of the 13 cycles  being changed
PMR_INDX    = { 22, 83, 22, … }             //  Array of the 13  pins being changed
EXP_DATA    = { 16, 37, 20, 1, 17, 37, 4 }  // Array of original pattern data
NEW_DATA    = { 33, 66, 81, 0, 33, 66, 80 }  // Array of new pattern data
```

Subsequently whan an STR is being logged that uses the changes described above, the FMU_FLG bit 4 should be set to a 1 to indicate the test is being performed with modifications made to the original pattern.

Logging Pattern/Chain/Bit Data in an STR Record

The previous conventions were based on the tester capturing the cycle and pin # of a failure and logging the data in that format. The downstream tool would then, based on information provided in the PSR records, extract the Pattern/Chain/Bit information on which the diagnosis is performed. The facility for the logging of Pattern/Chain/Bit data directly in the STR record is provided by the **PAT_NUM** and **BIT_POS** arrays. The **PAT_NUM** array would contain a list of pattern numbers (starting at 0 or 1) and the BIT_POS array would contain the bit position with the chain (Bit position "1" is the 1st location read out of a scan chain).

When the test contains multiple pattern files, the proposed convention is to start the pattern numbering over at "1" on each pattern. To indicate when a new file starts the proposed convention is to indicate that by inserting a "file change marker". This could be done by inserting an entry in the **PAT_NUM** array of 65535 and a corresponding file number in a new **BIT_POS** entry.

This format can be used with the **PMR_INDX** array to indicate what pin # the failure was observed on. Instead of or in addition to the **PMR_INDX** array, the **CHN_NUM** (chain number) can be used as a direct reference to the chain number.

In addition, the **USER_TEXT** field could potentially be used to log the failing flipflop names. However, a more effective and compact alternative would probably be to use the **CNR** records for this purpose.

Using the User Defined Data Arrays in an STR Record

The user may add up to three optional generic data arrays to an STR file logging user specific data. Each of the three arrays can be specified to be a 1, 2, or 4 byte data type. If present, the array sizes will be specified by the **LOCL_CNT** field (this is the same field that specifies the **CYCL_OFST, PMR_INDX**, and **CHN_NUM** array sizes). Like other data arrays, these arrays may be concatenated across contiguous STR records. The presence/size of these arrays is specified by the **USR1_LEN**, **USR2_LEN**, and **USR3_LEN** fields respectively.

Logging Failures by Scan Cell Name in an STR Record

There are two provisions provided for logging explicit scan cell failures by name. The simplest (but most memory intensive) is to insert a string entry into the **USER_TXT** array for each failure. Each string entry must be a fixed length string which is specified in the **TXT_LEN** field. Note that the usage of the **USER_TXT** field is not exclusively for scan cells name and can be used for other user specific applications.

A more efficient method of logging explicit scan cells failures is to create a **ScanCell Name Record** (**CNR**) for each scan cell failure observed (the same **CNR** record can be referenced each time it fails) The structure of a **CNR** record is:

CNR

CHN_NUM	= Chain number
BIT_POS	= Scan cell position in chain
CELL_NAM	= Scan cell Name

Notes:

CHN_NUM would be referenced by the CHN_NO array in an STR record
BIT_POS would be referenced by the BIT_POS array in an STR record

The FMU_FLG, MASK_MAP, and FAL_MAP Fields

The **STR** record provides three fields (two of which are optional) that provide additional information to downstream diagnostic tools about the failure attributes of specific signal pins.

The **MASK_MAP** field will specify which signals have been masked off in this specific test

The **FAL_MAP** field will specify which signals have additional failures that were detected beyond what was logged in the data arrays.

These presences of the **MASK_MAP** and **FAL_MAP** arrays are optional and their attributes are declared in the **FMU_FLG** field. (Note: that bit 4 of the **FMU_FLG** field is used to indicate that patterns have been modified)

The set of available field attributes for the **MASK_MA**P field are:

- No pins are masked and the **MASK_MAP** field is not present in this record
- Some pins are masked and the **MASK_MAP** field is present in this record
- Some pins are masked and the masked pins were specified in the **MASK_MAP** field contained in a previous STR record used for this same test suite. The field is not replicated in this STR record again (the information is "inherited" form the last STR applicable to this test)

The set of available field attributes for the **FAL_MA**P field are:

- No information on any potential additional failures after the last logged failure is available and the **FAL_MAP** field is not present in this record
- All failures were logged thus the **FAL_MAP** field is not present in this record
- They were some pins that have failures beyond what was logged and those pins are specified in the **FAL_MAP** field included in this record.

The LIM_CNT, LIM_INDX, and LIM_SPEC Fields

The **STR** record provides three fields (two of which are optional) that provide additional information to downstream diagnostic tools about the failure logging characters of specific signal pins.

The **LIM_INDX** array contains a list of pins (via their PMR indexes) that have unique maximum failure limits that were imposed by the tester for this specific test.

The **LIM_SPEC** array contains a list of the maximum failure limits imposed by the tester for this test. Each entry is the maximum failure limit imposed on the corresponding pin in the **LIM_INDX** array. If the If the 1st entry in **LIM_INDX** is= 0 (an otherwise invalid PMR index) then this indicates that the first specification listed in the **LIM_SPEC** array is the "default limit" and will apply to all pins not otherwise explicitly listed in the **PMR_INDX** array.

The **LIM_CNT** field specifies the number of entries in the **LIM_INDX** and **LIM_SPEC** arrays.

Example: Assume that pin #17 has a max failure limit of 1000, pin #99 has a maximum failure limit of 1500, and all other pins have a maximum failure limit of 3000, the field values would then be:

LIM_CNT = 3

LIM_INDX = { 0, 17, 99 }

LIM_SPEC = { 3000, 1000, 1500 }

Logging Scan Structure Information

The **SSR** and **SCR** records provide the ability to include scan chain structure information in an STDF file. Normally this information would be extracted from the ScanStructures block of a STIL file. These records are optionally and are referenced or required by any other record type in STDF file. These records will tend to be very large and should be included only if required by a downstream diagnosis tool.

The **SSR** (Scan Structure Record) contains the top level information of a scan structure and includes the following fields:

SSR_NAM = the name of the ScanStructure block

CHN_CNT = the number of chains in the block

CHN_LIST = the list of SCR (Scan Chain record) record indexes

A **SCR** (Scan Chain record) must be created for each chain in the structure. Generally, multiple SCR records will be required to list all of the scan cell names that are present in one chain. One option to the user is to exclude the scan cell name list (by setting **FLOCS_CNT** = 0) to achieve smaller files but still provide the other chain information .

SCR_INDX = Scan chain index (as referenced by the CHN_LIST array)
CHN_NAM = Chain name
FTOTS_CNT = Chain length (# of scan cells in chain)
FLOCS_CNT = # of scan cells in chain listed in this record
SIN_PIN = Scan-in pin (PMR index)
SOUT_PIN = Scan-out pin (PMR index)
MSTR_CNT = # of master clocks used in this scan chain
FSLAV_CNT = # of slave clocks used in this scan chain
M_CLKS = List of master Clocks (PMR Indexes)
S_CLKS = List of slave clocks (PMR Indexes)
INVRT_FlG = Chain Inversion state (0 or)
CELL_LST = List of scan cell names

Note: The CELL_LST list is a "kxS*n" data type, which is a new data type added in STDF 4-2007. The "S*n" type provides allows strings longer then the 255 character limit imposed by the "C*n" data type.

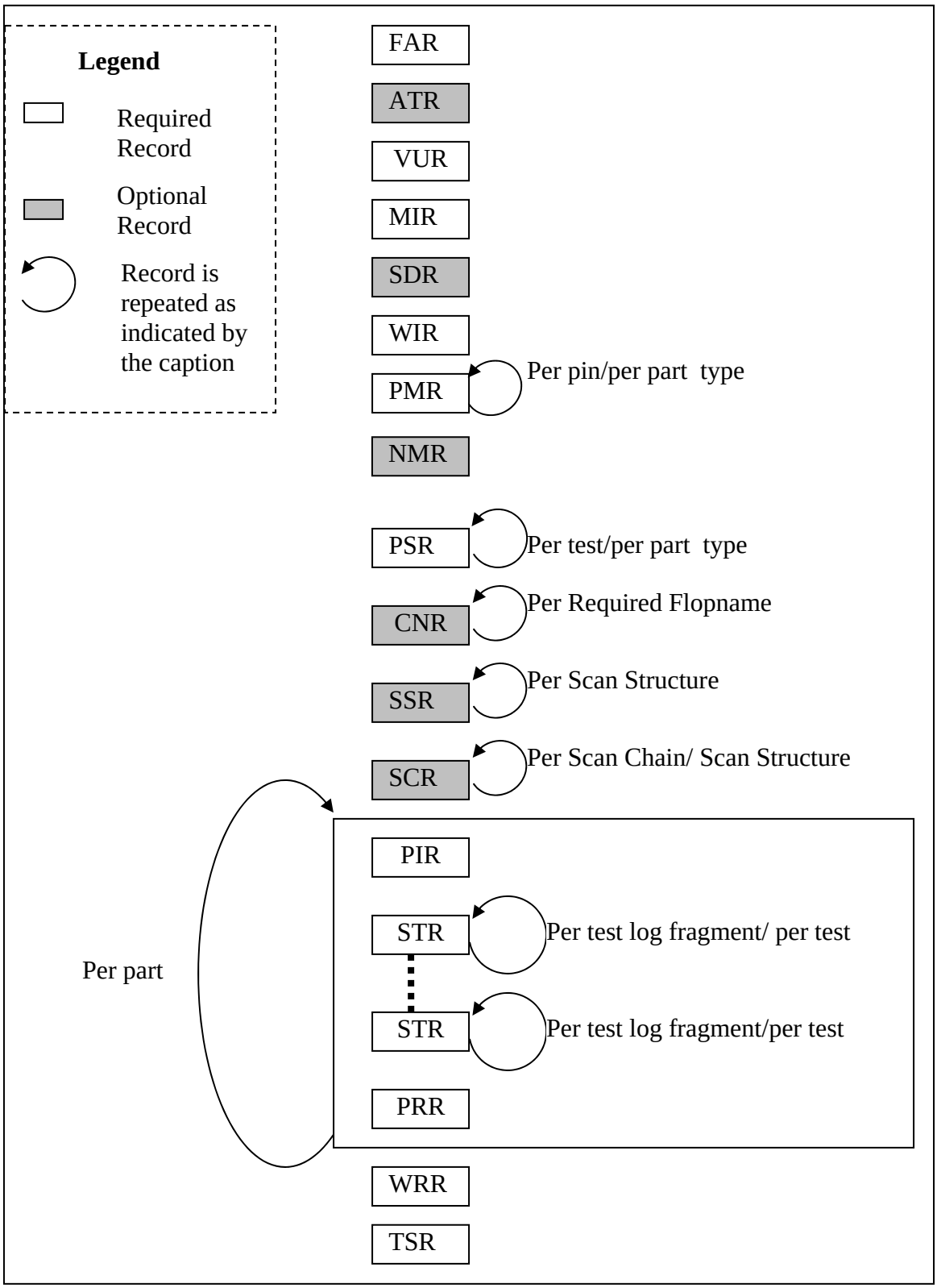

Figure 9: Sample STDF V4-2007 File Structure