

# The Next Step in Volume Scan Diagnosis: Standard Fail Data Format

Andreas Leininger

Ajay Khoche<sup>1</sup>, Martin Fischer<sup>2</sup>

Nagesh Tamarapalli, Wu-Tung Cheng, Randy Klingenberg, Wu Yang

Infineon Technologies AG  
Am Campeon 1-12  
Munich

Agilent Technologies  
<sup>1</sup>Santa Clara – CA, USA  
<sup>2</sup>Boeblingen, Germany

Mentor Graphics Corporation  
8005 S.W. Boeckman Road  
Wilsonville, OR 97070, US

## Abstract

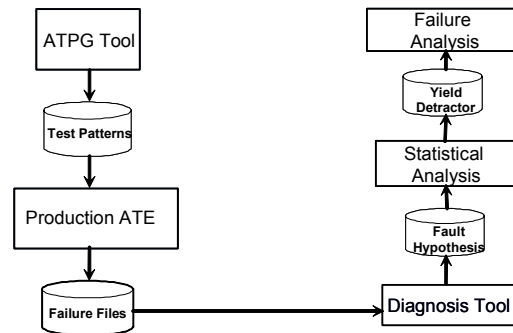
*The need for faster and more reliable yield ramp-up when introducing new CMOS technologies is driving the effort to acquire and analyze valuable information from production test, for the process of identification of yield detractors. This paper addresses a key step in the phase of “industrialization” of these processes: standardization. The objective of standardization is to enable a seamless flow for production integrated scan diagnosis in a multi-tool and a multi-vendor environment. To make analysis of chips failing the production test more efficient, a process flow and a file format to store the chips failing response is proposed. This will enable a smooth exchange of production test data from ATE to diagnosis, failure analysis, design and process.*

## 1. Introduction

Semiconductor industry is realizing the growing importance of data acquisition capabilities of the Automatic Test Equipment (ATE) [1][2]. SEMI Standards Roadmap for Probe & Test [3] identified Result Data Collection as the key topic for 2005. Two main trends can be held responsible for this. The first trend is to analyze the production test data off-line by statistical methods e.g. to identify reliability risks and to reduce or replace burn-in. The second trend is to generate data for yield learning purposes.

One evolving yield learning method to identify design-process interaction, and systematic yield issues is the production integrated scan diagnosis [4][5]. Figure 1 illustrates a high-level view of a production integrated diagnosis flow. Instead of performing just go/no-go production test, the idea behind the flow is to log a reasonable number of failures from the production test for all or a large number of failing dies. These failures are analyzed by a diagnosis tool, which comes up with fault hypotheses that best explain the observed failures. Statistical analysis of the diagnosis results for the large number of failing dies is then performed to identify systematic yield detractors. Finally, few dies may be selected for failure analysis and identification of the defect mechanisms.

Yield learning based on production integrated volume scan diagnosis requires a more complex data flow than the traditional yield learning approach based on memory fail bitmaps. Unlike the bitmaps, the result in scan diagnosis, which is typically the suspect net(s) causing a chip to fail, cannot be calculated by the ATE itself. Hence, a scan diagnosis tool has to be integrated in the flow.

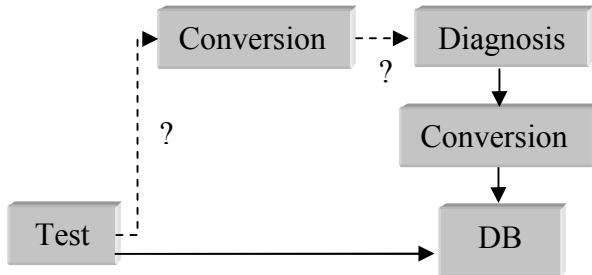


**Figure 1 Flow for yield learning through production integrated volume diagnosis**

As shown in Figure 1, communication between the ATE and the scan diagnosis tool occurs through failure files. Though flow shown in Figure 1 and the type of analysis and functionality described for e.g. in [6][7] can be implemented today, the motivation for a failure file standard is driven by the need to optimize the diagnosis flow. The different tasks in this flow need to be understood and assigned clearly. This will allow to efficiently employ resources and to improve the necessary competencies.

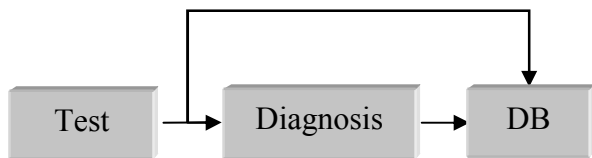
Today, diagnosis flow is quite hampered every time a new test platform is introduced or a new subcontractor is engaged. The data collection routines need to be adapted on every platform to a variety of IT infrastructures at the subcontractor or manufacturing sites. This adds unnecessary complexity to the test program as the capture routines in such a scenario cannot be maintained by the ATE vendor. In addition, conversion of the failure files to a format acceptable to diagnosis tool is a quite tricky task

especially when complex test patterns such as sequential patterns with variable number of capture cycles are involved. Thus, at the moment the owner of such a diagnosis flow has to build up competence on ATE data collection and data formatting instead of concentrating on the key task of chip yield and process control.



**Figure 2** Current Test Data Flow

The current test data flow is illustrated in Figure 2, where a conversion, performed either offline or directly by the ATE equipment, is needed. The multi-vendor and multi-tool scenario generates a big variety of implementation solutions with the need to have a large number of converters in place. In addition, another converter is typically needed to process the diagnosis results and prepare the data base loading, i.e. to generate a table which links the diagnosed net information to the tested chip, identified by lot number, wafer number, X and Y position. Test program and diagnosis flow are unnecessarily complex and experts are needed to guarantee 24 hours up time. This reduces the benefit of knowing how to use the diagnosis data for the improvement of production processes.



**Figure 3** Desired Test Data Flow

The traditional approach to yield analysis has been to store and analyze the test results in a production data base. Now, a third party diagnosis tool is added, which ideally should read the same input format and write to the same data base as illustrated in Figure 3.

ATE systems now are employed for diagnosis data acquisition. This potentially has an impact on test time, thus cost of test [8]. In order to reduce test time and excessive data storage needs, external interfacing to the ATE for adaptive data collection is needed. Actually, the integration of a control mechanism into the test program is

the only possible solution. This makes the test program again more complex, presumably data acquisition will be product specific and without a specific knowledge of the test program the acquisition of the necessary data is at a risk. Therefore the ATE vendor has to improve their already existing “datalog” functionality for test time and data collection control in order to support standard test methods for later diagnosis. If every company has its own specifications of the interface to the diagnosis tool this optimization of the ATE functionality will hardly be started.

In short, standardization of how to store scan fail data will offer the advantage to optimize the production integrated volume diagnosis flow. Our hope is this paper will raise the awareness that a standard will be a benefit for all users. To help the standardization process a proposal of the format is presented in this paper. We expect this format to evolve with input from multiple vendors and user community.

The rest of the paper is organized as follows. Section 2 gives an outlook of how a high volume diagnosis flow can be implemented. The requirements for a scan data collection standard are described in Section 3. Section 4 presents a brief introduction to the standard tester datalog format STDF. The proposed format for storing scan data in STDF is presented in Section 5. Finally, Section 6 presents the conclusions.

## 2. Production Integrated Diagnosis Flow

As illustrated in Figure 1, diagnosis of production scan test failures plays a central role in the evolving yield learning methodology. This section discusses the production integrated diagnosis flow in more detail.

Before scan diagnosis can be run for a die it is necessary that the die fails the production scan test. Thus in the Automatic Test Pattern Generation (ATPG) step, shown in Figure 1, it is paramount to create a very high quality production scan test such that very few, if not zero, defective die escape the test. For the current nanometer designs with newer types of defects, in order to achieve the quality levels, it is necessary to augment the standard stuck-at test patterns with patterns such as at-speed transition / path delay patterns and multiple-detect / targeted bridging patterns. Given the complexity of the present designs, achieving the quality levels with such an augmented test set while maintaining the test cost is possible only if dramatic test compression is employed. In fact, in the past few years test compression with embedded logic has become more prevalent [10]-[13]. While compression facilitates high test quality, it impacts diagnosis since the data that is applied to IC and the response that is captured undergo a transformation. In

order to assist diagnosis it is not acceptable to switch to a mode bypassing the compression logic as this would cause enormous test time penalty and reduce the throughput of the test floor. Furthermore, it would necessitate ATE upgrades because of the vast amount of test data and fail response data that needs to be transferred from the test floor into the diagnosis database. For a truly production integrated diagnosis flow, targeted at analyzing all production fails, the diagnosis tool must be able to work with compressed patterns i.e. take the fail log corresponding to the compacted response stream and determine the suspects inside the logic cones.

The high quality test patterns generated by the ATPG tool along with the netlist are usually archived for later use by the diagnosis tool. The test patterns, after proper validation, are ported to ATE, which applies the test stimuli and compares the device responses with the expected responses. Typically during the production test, ATE is set up to stop-on-first-fail. However, for diagnosis purposes it is necessary for the ATE to log some number of failures instead of stopping on the first fail. Failure logging for one test set such as stuck-at is well understood, however, given the variety of test sets present these days, logging is becoming a challenge. In order to minimize the test time increase incurred due to failure logging, a trade-off must be drawn between the number of failures and the resulting diagnosis accuracy and resolution. In general, logic failures for a die can be diagnosed with good accuracy and resolution with fewer failing cycles, whereas scan chain failures require many more cycles to be logged. Multi-site testing, which is

is usually the case except in extremely low yield scenario).

The objective of the production integrated volume diagnosis is to analyze most, if not all, failing die to quickly identify the yield detractors. Figure 2, illustrates an example, abstracted flow for the high volume diagnosis. A production test floor at any point of time may be processing parts corresponding to multiple designs. As failures from the tested parts are logged and stored in a database, the diagnosis tool should analyze them and store the suspect reports in results database. Hence there is an ongoing diagnosis server process that continually monitors the user specified locations for the appearance of new failure files and dispatches the analysis of these failure files to appropriate number of diagnosis tools. Note that once the diagnosis sever is set up, very minimal manual intervention is expected as is usually the case on production floor. Thus for the flow to be successful it must incorporate extreme level of automation, ease of use, and fault tolerance. In addition, the diagnosis tool must have high throughput to facilitate analysis of a large number of failing die with reasonable number of tool licenses. Furthermore, the diagnosis tool should be able to distinguish between the library cell internal and interconnect defects and utilize physical information to provide results with good accuracy and resolution. Due to the processing of large number of failing die and statistical nature of the analysis, certain degradation of the accuracy and resolution can be tolerated.

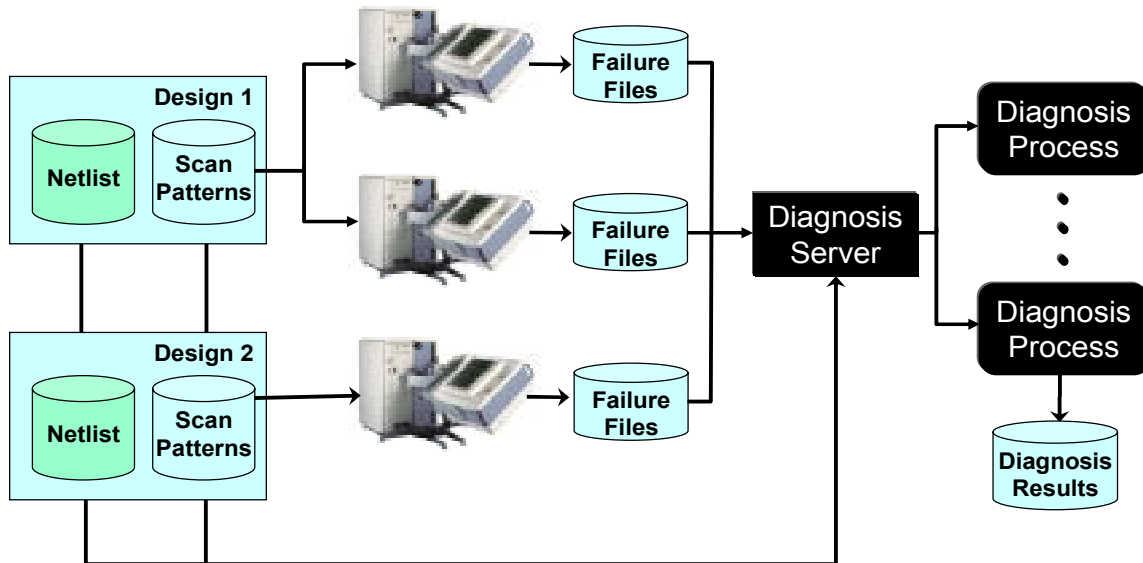


Figure 2 Production integrated high-volume diagnosis flow

more prevalent these days, makes the test time penalty due to failure logging more acceptable (since all tests would have to be applied if at least one site passes, which

As mentioned earlier, failures logged by ATE are typically stored in STDF and are in time-based or more commonly in cycle-based format. That is each mismatch

contains the cycle number and name of the pin where the mismatch has occurred. In addition, expected / measured value information may also be available. Diagnosis tools on the other hand typically require failure files to be in tool specific ASCII format. This means that the user is burdened with the task of converting the failure files in ATE format to tool specific format. This conversion is quite tricky, especially when complex patterns such as sequential patterns with variable number of capture cycles are involved and it is further compounded by the fact that there is a time lag between diagnosis and ATPG activities which are typically performed different teams. Such a flow based on custom scripts is untenable when the objective is to perform highly automated volume diagnosis as outlined in this section. Thus it is necessary to have a standard format for the failure files such that the failure files produced by any ATE can be directly consumed by the diagnosis tools. The next section, Section 3 outlines the requirements for the standard.

### 3. Requirements

When proposing a new standard the level of acceptance depends on the benefit for the adaptors. The idea and the requirement is to create a win-win situation for the different parties involved and thus to have a high probability of acceptance.

In order to allow fast adoption of the proposal, the diagnosis data should be stored in a file format which is well accepted in semiconductor industry and requires the smallest change in the IT landscape. It would be best to maintain compatibility with existing data processing software. Fast introduction of the new format is necessary to avoid a competitive situation with multiple formats in place and not to benefit few players. Especially, as the diagnosis data is written by the ATE, a requirement is to allow the different test system architectures to take advantage of their individual strengths.

As the objective is to store the data for a large number of devices, a compact format is needed, such that the requirements for data storage and transmission bandwidth are reduced. On the other hand the format should offer certain flexibility for future enhancements and contain all necessary information for the best diagnosis resolution. As data collection is the competence of the ATE and the analysis is the competence of the diagnosis tool, the format should allow storing “raw data”, hereby reducing the need of data conversion and preprocessing on the ATE.

To summarize, following is the list of requirements:

- Data sufficiency: Provide ability to store “raw” data as well as all the necessary data for the downstream diagnosis tool to perform its function.

- Data Integrity: Provide means to check for data validity, to perform data synchronization and to uniquely identify a data set.
- Compact representation: To reduce the size of data necessary to convey the requisite information.
- Flexibility: So that multi-tool, multi-vendor environment can be supported
- Ease-of-adoption: To enable fast adoption

Section 4 below discusses STDF, the commonly used language for test results representation. Section 5 presents a proposal for the failure file standard in STDF.

## 4. STDF for Volume Diagnosis

### 4.1 Motivation

As described in the previous sections, volume diagnostics is fast becoming an integral part of the overall yield learning solution along with other conventional yield learning tools. All these tools use the data collected by the ATE from the product test. Thus the ATE data is meant to be used by multiple tools, which may have different internal data formats. In addition, in some environments individual tools may come from different vendors. While it is certainly possible to develop ATE software such that a separate solution exists for each possible tool’s and vendor’s format, it is definitely not practical from the point of view of development effort, maintenance and inter-tool data consistency. The multi-tool and multi-vendor requirements are best served by having a vendor and tool neutral standard. Standard Test Data Format (STDF), developed by Teradyne and available for use by anybody in the industry, provides such a vendor and tool neutral platform. STDF is also portable across various compute platforms. It is widely used in the industry for storing the test result data and is well supported by all the major ATE vendors. A broad range of standard and in-house tools also exist today to process STDF test results for statistical analysis, yield management and yield learning. Therefore it makes a good candidate for representing scan fail information for volume diagnostics. However, the current STDF standard does not have any predefined way of describing volume diagnostics information in terms of scan fail log or memory fail map. This section first provides an overview of STDF and then evaluates the alternatives for storing scan fail log information in STDF.

### 4.2 Overview of STDF

STDF provides a common platform to allow tester, database management systems and data analysis software to store and communicate test data in a general and flexible format. STDF data is organized in the form of a

partially ordered set of records. Each record consists of a header section and information section. The header section contains the size of the record, the record type (which indicates the general category of the information) and a record sub type (which indicates the specific information in that category). Table 1 shows the categories of data and the record types that are supported in STDF V4.

**Table 1 STDF Record Types and Subtypes**

Rec. Type	Rec. Sub type	Pneumonic	Description
0			<b>Information about the STDF File</b>
	10	FAR	File Attribute record
	20	ATR	Audit trail record
1			<b>Data Collected on a per lot basis</b>
	10	MIR	Master Information Record
	20	MRR	Master Results Record
	30	PCR	Part Count record
	40	HBR	Hardware BIN record
	50	SBR	Software Bin record
	60	PMR	Pin Map Record
	62	PGR	Pin Group Record
	63	PLR	Pin List Record
	70	RDR	ReTest Data Record
	80	SDR	Site Description Record
2			<b>Data collected Per wafer</b>
	10	WIR	Wafer Information Record
	20	WRR	Wafer Result Record
	30	WCR	Wafer Configuration Record
5			<b>Data Collected on per part basis</b>
	10	PIR	Part Information Record
	20	PRR	Part Result Record
10			<b>Data Collected per test in the test program</b>
	30	TSR	Test Synopsys Record
15			<b>Data Collected Per Test Execution</b>
	10	PTR	Parametric Test Record
	15	MPR	Multiple-Result Parametric Record
	20	FTR	Functional Test Record
20			<b>Data Collected Per program segment</b>
	10	BPS	Begin Program Section Record
	20	EPS	End Program Section Record
50			<b>Generic Data</b>

	10	GDR	Generic Data Record
	30	DTR	Datalog Text Record

STDF also supports various data types for describing data in each record. Table 2 shows the data types supported in STDF.

**Table 2 Data Type Codes in STDF**

Code	Description
C*n	Character string of length n
U*1,U*2,U*4	One, two and four bytes unsigned integer
I*1,I*2,I*4	One, two and four bytes signed integer
R*4,R*8	Four, Eight byte floating point number
V*n	Variable data type field, data type specified in the first byte. (max 255)
B*n	Variable length Bit encoded field; first byte denotes the count; first data bit in LSB of second byte
D*n	Variable length bit-encoded field; First two bytes specify the count
N*1	Unsigned integer stored in a nibble

### 4.3 STDF Record Alternatives

It is evident from Table 1 that there is no explicit record to store the scan fail log information. While a single failure e.g. first fail in functional test, can be captured in a FTR record, there is no single record that can capture multiple scan failures. The reason for not having any record appears to be that before volume diagnosis, there was no need to store this information in the production environment. Therefore in order to store the information necessary to enable volume diagnosis in the production flow, some of the existing records will have to be used in their non-intended purpose. Specifically the possible alternatives are

- FTR (Functional Test Record)
- FTR (Functional Test Record) + DTR (Datalog Text Record)
- DTR (Datalog Text Record)
- GDR (Generic Data Record)

One point to note here is that these alternatives are for representing the information necessary for diagnosis. The test result (Pass/fail) can still be stored in a FTR.

The FTR record is meant for storing the information from the functional test. This record has fields to describe information on test site, test number, vector number, failing pin and expected and received values. However, it can only support logging of one fail in one vector/cycle in a single record. This would mean that a separate FTR would be required for each failing vector/cycle. Given the

large size of the FTR record (>50bytes)[9], this would add a lot of overhead for storing the scan fail log information (A comparison of the data sizes for all the alternatives is shown at the end of this section). Moreover the record does not have predefined fields to provide information for data validity/synchronization, and format specification information. This will have to be described in its “additional result information” field in the text form, which could affect the existing STDF reader. Hence storing information in FTR is not a viable option.

The other alternative is to store the test setup, data validity/synchronization and format specification information in the FTR (with the use of “additional result information field” for missing pieces) and then use DTR which allows a single text field (up to 255 characters) for specifying any user information, for logging the failure information on each fail cycle. Moreover this information can be added to the same FTR that is used for storing test result. While this option is better than the first one, it has the problem that it creates an indirect link between DTR and FTR and the sequence of FTR->DTR is not well known to the standard STDF readers today and hence such a dependency may break them. In addition, the DTR record will add four bytes of header overhead for each scan fail cycle data and depending on the size of fail information this could be substantial. DTR record also forces the fail information to be represented in text form. This will result in a sub-optimal representation since ASCII is not the best representation for numbers. So overall the combination of FTR and DTR is not optimal either due to restriction of only one data field per record in the DTR and its associated header overhead and the restriction of ASCII representation.

The third option could be to store everything in DTR in text form. This eliminates the indirect link problem mentioned above, but still has the overhead and ASCII representation problem. In addition, as mentioned before, it also has the limitation of 255 maximum characters per record, which restricts how much information can be stored in a single DTR record.

The last option is GDR. It is meant to be used for storing any generic data. Table 3 shows the structure of a GDR record, where the data is stored in a list of GEN\_DATA fields. Each GEN\_DATA field has its own data type. The data type is specified in first byte of the GEN\_DATA field. The list of supported data types is shown in Table 4.

**Table 3 GDR Structure**

Field Name	Data Type	Description
REC_LEN	U*2	Bytes of data following the header
REC_TYPE	U*1	50: for Generic
REC_SUB	U*1	10:for GDR

FLD_CNT	U*2	Count of data fields in the record.
GEN_DATA	V*n	Data type code and the data for one field. GEN_DATA repeated for each data field

**Table 4 Data Type Codes for GEN\_DATA in GDR**

Data Type code	Data Symbol	Description
0	B*0	Special pad filed used for alignment
1	U*1	One byte unsigned integer
2	U*2	two bytes unsigned integer
3	U*4	four bytes unsigned integer
4	I*1	One byte unsigned integer
5	I*2	two bytes unsigned integer
6	I*4	four bytes unsigned integer
7	R*4	Four byte floating number
8	R*8	Eight byte floating number
10	C*n	ASCII char string of length n. first byte after code contains n
11	B*n	Binary data string. first byte after code contains n. Data is LSBs
12	D*n	Bit Encoded data. First two bytes contain n.
13	N*1	Unsigned nibble

The property of GDR to allow multiple data items in a single record allows both the fail and non-fail (setup, format, data validity/synchronization) information to be stored in the same record. In addition, the ability to use appropriate data type for each information field enables an optimal representation and reduces the overall data size.

Table 5 shows a comparison of the alternatives in terms of data size for storing only the fail information i.e. it does not include the storage required for other information (e.g. test result, setup, data validity/handling). An additional DTR+ column is added to table. The difference between DTR and DTR+ is that, in DTR, only one fail cycle record is stored in a single DTR record, whereas in DTR+ multiple fail cycle records are packed into one single record up to a maximum of 255 allowed characters. The “Ratio” row shows the data size relative to GDR.

Following assumptions were used in the calculation of data size numbers:

- For each fail, only the cycle number and the pin number are stored. (No Expected data)
- Cycle numbers are four byte numbers and pin numbers are two byte numbers.

- Hex representation is used to store a number in ASCII
- FTR record size for storing each fail information is 50 bytes.

**Table 5 Datalog Size Comparison for Alternatives**

Property	FTR	FTR+DTR	DTR	DTR+	GDR
<b>Size for 500 fails (bytes)</b>	25000	8000	8000	5997	3000
<b>Size for 10K Fails (bytes)</b>	500K	160K	160K	123.7	60K
<b>Ratio</b>	8.33x	2.66x	2.66x	1.99x	1x

Please note that:

- In the case of FTR the 50 byte assumption for storing fail information is conservative and in reality it would be higher than that and thus would make the numbers in the FTR column even higher.
- The numbers in the FTR+DTR and the DTR columns are identical. This is due to the fact that in the case of FTR+DTR, the FTR is used to store the non-fail information only. Therefore its size does not account towards the fail only data size shown in the table.

Therefore use of GDR for storing scan data log information for volume diagnostics is the most optimal in the current STDF version and hence is used in the proposal described in this paper.

## 5. Proposed format to store Scan Fail Data in STDF file format

This section describes the details of GDR that are proposed for use in storing scan data log information for the purpose of volume diagnosis.

### 5.1 Diagnosis Information Representation

This section presents the information that can be described in the proposal to meet the requirements mentioned above. The information is classified in six categories described below. Each of these categories is described in a table form where the “Type” column indicates the data type for the GEN\_DATA field. The “Req” column indicates whether that particular field is mandatory. The “Comment” column adds some field specific information.

### 5.1.1 Die/Core Identification

This set of information identifies the die/core for which the data is collected. The information includes following elements:

**Table 6 Die/Core Information**

Information	Type	Req	Comment
Wafer Lot Id	C*n	Y	n is the length of the ID string
Wafer Id	U*2	Y	Assumed to be a number
X-Coord	U*2	Y	
Y-Coord	U*2	Y	
Core Id	U*2	N	Assumed to be a number

### 5.1.2 Test Identification

This set of information identifies the test for which the data collection is done.

**Table 7 Test Identification Information**

Information	Type	Req	Comment
Test Program ID	C*n	N	A text field for the test program version identification. A 0 in the n field indicates missing/unspecified field.
Test Mode	C*n	N	A text field is for test insertion identification. A 0 in the n field indicates missing/unspecified field
Test Num	U*2	Y	Assumed to be a number
Pattern Id	C*n	N	Any ATPG given name; A 0 in the n field indicates a missing field
Test Suite Id	U*2	Y	Assumed to be a number indicating a subset of the complete ATPG test set.

### 5.1.3 Environment Specification

This information set contains the data about the environment in which the test specified in the Test Identification section was run. Each test could be run under different environment conditions at different point in the test flow.

**Table 8 Environment Specification Information**

Information	Type	Req	Comment
Scan Freq	U*4	Y	Shift Frequency
Capture Freq	U*4	N	
Test Nominal Voltage	U*2	Y	
Test Stress Voltage	U*2	N	To be used in the case of low voltage scan or bumped voltage scan
Test Temp	I*2	N	To be used if either self-contained GDR is needed or if there is a temperature that is different than the ambient that needs to be captured
User Env Spec	C*n	N	To specify any custom specification (string type)

### 5.1.4 Datalog Format Specification

This information allows the specification of the format in which the fail data is provided.

**Table 9 Datalog Format Specification Information**

Information	Type	Req	Comment
Fail Data Format	U*1	Y	0: ATPG pattern based, 1: cycle based
Expected Data Spec	U*1	Y	A single field is used to indicate the presence/absence of expected data in the fail records
Z-Handling flag	U*1	Y	Used to indicate Mapping to Z –states 0:L, 1:H, 2:Z, 3:X, 4-not handled

### 5.1.5 Data Capture Information

This set of data is provided to perform data validation and synchronization by the diagnosis tools to ensure that there is no discrepancy in the data used by the tester, the ATPG and the diagnosis tool, which may result in inaccurate result from the diagnosis tool.

**Table 10 Test Execution Summary Information**

Information	Type	Req	Comment
First Pattern Executed	U*4	Y/N	Mandatory if format is ATPG and the first pattern executed is not the first pattern of the in the pattern memory. Not present if format is cycle based
Last Pattern Executed	U*4	Y/N	Mandatory if format is ATPG and the last pattern executed is not the last pattern in the pattern memory. Not present if format is cycle based
First Cycle executed	U*4	Y/N	Mandatory if format is cycle based and the first cycle is not the first cycle of the first pattern. Not present if format is ATPG
Last Cycle executed	U*4	Y/N	Mandatory if format is cycle based and the last cycle executed is not the last cycle with entire pattern run. Not present if format is cycle based
Total Fails	U*4	Y	
Total Cycles	U*4	Y/N	Required if cycle based format is used and there are multiple test suites
Total Patterns	U*4	N	Used only if ATPG format is used
Pattern 0 fails	U*1	Y	A global field for all the scan out pins
Buffer Full	C*n	Y	One flag per scan out pin to indicate buffer overrun

### 5.1.6 Fail data

This is the actual fail data during the test run and contains the pattern/cycle information, the failing pin information and the expected data information. The fail log consists of a list of fail records, one for each observed failure. The fail log is represented by a single GEN\_DATA field of type D\*n and the fail records themselves don't have their own GEN\_DATA fields. The fail record fields are represented as shown in the table below

**Table 11 Fail Data Information**

Information	Type	Req	Comment
Pattern Num	U*4	Y/N	Not required if cycle based format is used
Offset	U*4	Y/N	Not required if cycle based format is used
Pin Num	U*2	Y	Assumes that pin name to pin number mapping is done is some other record if not it can be added to this record itself.
Measured and Expected data	U*1	Y/N	Measured data in upper(left) nibble and expected in lower(right) nibble. Expected data not required if format specifies not expected data
Cycle Num	U*4	Y/N	Not required if ATPG format is used

Please note that depending on the format settings in the previous section not all fields are present in a particular fail record and all the fail records in a log have the same format. The possible combinations for a fail record format are

- Pattern\_num, offset, pin\_num, measured data, expected data - **11 bytes/fail**
- Pattern\_num, offset, pin\_num, measured\_data - **11 bytes/fail**
- Cycle\_num, pin\_num, measured\_data, expected data - **7 bytes/fail**
- Cycle\_num, pin\_num, measured\_data - **7 bytes/fail**

Among these formats, in the cases where no expected data is written in the log, the assumption is that the only expected data from a given setup will be written out by the tester is a single special run. A diagnosis tool can then use this data for all the failures and eliminate the need for specifying it with each fail.

### 5.2 Example

Figure 5 shows an example circuit with its associated test and environment parameters that need to be communicated to the diagnosis tool. In this example the design contains three scan chains of identical length of 100. A total of 10 (0:9) scan patterns are applied which

take 1000(0:999) cycles. Two failures were detected in this test run. Table 10 shows the corresponding GDR description.

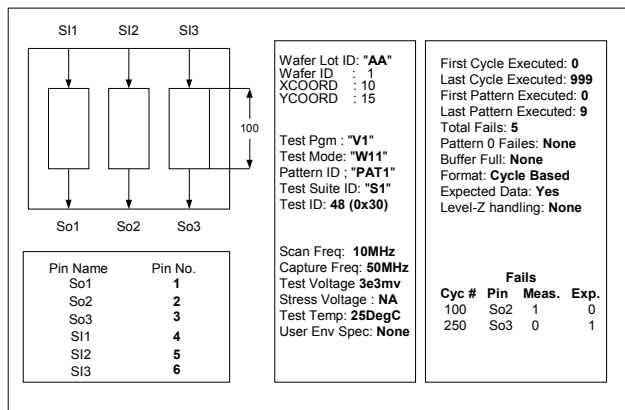


Figure 5 Example

Table 12

Even Byte	Odd Byte	Comment
00	8a	138 bytes after header
32	0a	Type 50 subtype 10
00	2a	42 GEN_DATA fields in the record
0a	0e	C*14
44	49	"DIAGNOSIS_DATA" string
41	48	
54	55	
59	49	
59	5f	
44	41	
54	41	
0a	02	C*2
41	41	Wafer lot ID "AA"
00*	02	Pad byte (0) and U*2
00	01	Wafer ID 01
00*	02	Pad byte (0) and U*2
00	0a	XCOORD = 10
00*	02	Pad byte (0) and U*2
00	0f	YCOORD = 15
01	00	U*1; Core 0
0a	02	C*2
56	31	Test Program Version ID "V1"
0a	03	C*3
57	31	Test Mode ID "W11"
31	02	U*2 in the second byte
00	30	Test ID = 48 (0x30)
0a	04	C*4
56	41	Pattern ID = "PAT1" in the four bytes
60	31	
0a	02	C*2
53	31	Test Suite ID = "S1"
00*	03	Pad (0) U*4
00	00	Scan Freq in four bytes in KHz (10000 KHz - 00:00:27:10)
27	10	
00*	03	Pad (0) U*4

00	00	Capture Frequency in KHz
c3	50	50000KHz - (00:00:c3:50)
00*	02	Pad (0) U*2
0B	B8	Nominal voltage (3000 mV)
00*	02	Pad (0) U*2
FF	FF	Stress Voltage
00*	02	
00	19	Test Temperature +25C (00:19)
0a	00	No User Env Spec
01	01	U*1; cycle based format (01)
01	01	U*1; Expected data in the log
01	04	Z not handled
00*	03	Pad (0) U*4
00	00	First Cycle 0 (00:00:00:00)
00	00	
00*	03	Pad (0) U*4
00	00	Last Cycle 999 (00:00:03:E7)
03	E7	
00*	03	Pad(0); U*4
00	00	
00	05	5 Total fails
00*	03	Pad(0) U*4
00	00	
03	E8	Total cycles 1000(00:00:03:E8)
01	00	U*1;No Pattern 0 fail
0a	03	C*3
54	54	Buffer full "NNN" one for each scan out
54	0c	Expected data for fails in the output; D*n
00	23	Overusing the 0c format total 35 bytes for 5 failure
00	00	First Fail Info
00	64	Cycle no 100 (00:00:00:64)
00	02	Pin no 2(00:02)
10	00	Measured value 1 and Expect value 0 in first fail 1; 2nd Fail Info Starting on Odd byte
00	00	
FA	00	Cycle number 250 (00:00:00:FA);
03	01	Pin 3 (00:03); measured value 0 and expected value 1 in 4 <sup>th</sup> fail 0

## 6. Conclusion

The proposal for storing scan fail data in STDF is motivated by the broad use of STDF Version 4 throughout the semiconductor industry for storing and processing test result data. Although several alternative data format standards have been proposed in the meanwhile, replacing an existing standard – such as STDF – will take some time and effort. However, without a doubt – and as shown in this paper – the STDF format specification will need to be continuously enhanced to keep track with the newest developments in semiconductor testing. Since the definition of the format, there are new trends coming up, like massive parallel test, adaptive testing, part average testing and production integrated diagnosis, which are not fully reflected in the latest published version (V4) of the standard.

The fact that more and more subcontracting for manufacturing and testing is used, increases the need to have a standard format for storing, exchanging and processing test result data. Thus, future test data formats as well need to provide a definition on how to efficiently store scan fail data.

The proposal builds on the flexibility designed into the current STDF format description to define a new and efficient data structure for scan fail data. Of course, enhancing the standard itself by adding standard data records would help to broaden the usage of STDF for diagnosis in volume manufacturing. In that case, the GDR definition described in the paper can serve as a starting point for enhancing STDF itself. Scan test is the current state of the art to test the digital logic of a semiconductor device. As discussed in the introduction, a standard to store the test result of such an important test method is missing. A standard would offer an important optimization potential for the semiconductor industry to improve the interface between design, test and production. Hopefully, this proposal will lead to a fast consolidation of such a standard, not only taking into account functional needs but also processing times, storage and efficiency and "cost of change".

There is also ongoing work on having a similar synergy potential for representing the memory bitmap information. This is especially necessary, when a MBIST approach is in place and again some, for sure less complex, calculation is necessary to extract the diagnosis information out of in most cases a serial bit stream.

Design for test and manufacturing is well accepted. Test for design and manufacturing is evolving as a method to target upcoming challenges with the shrinking silicon feature sizes. This method needs to be supported by the ATE by providing a well defined interface to EDA world and vice versa.

## 8. References

- [1] Rehnani et al. "ATE Data Collection – A comprehensive requirements proposal to maximize ROI of test", ITC 2004"
- [2] R. Arnold, A. Leininger "Evaluating ATE-Equipment for Volume Diagnosis", ITC 2005, paper 41.1, Austin/Tx
- [3] R. Nageswaran, G. Crispieri, B. O'Grady, C Richardson, "Automation Guidelines and Standards for Probe and Test", SEMI 2002.
- [4] C. Hora, R. Segers, S. Eichenberger, M. Lousberg, "On a statistical fault diagnosis approach enabling fast yield ramp-up", *Proc. of Euro. Test Workshop*, pp. 193-198, 2002
- [5] A. Leininger, P. Muhmenthaler, W.-T. Cheng, N. Tamarapalli, W. Yang, H. Tsai "Compression Mode Diagnosis enables High Volume Monitoring Diagnosis Flow", ITC 2005, paper 7.3, Austin/Tx
- [6] H. Erb, C. Burmer, A. Leininger "Yield enhancement through fast statistical scan test analysis for digital logic", ASMC 2005, Munich
- [7] D. Appello, A. Fudoli, K. Giarda, V. Tancorre, E. Gizdarski, and B. Matthew, "Understanding yield losses in logic circuits", *IEEE Design & Test of Computers*, pp. 208-215, Vol, 21, Issue 3, May-June 2004
- [8] "International Technology Roadmap for Semiconductors, Test and Test Equipment", 2003 Edition
- [9] Standard Test Data Format Specification, Version 4.0, Teradyne Inc
- [10] J. Rajski, M. Kassab, N. Mukerjee, N. Tamarapalli, J. Tyszer, and J. Qian, "Embedded deterministic test for low-cost manufacturing", *IEEE Design & Test of Computers*, Vol. 20, Sept.-Oct. 2003, pp 58-66
- [11] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, A. Ferko, B. Keller, D. Scott, B. Koenemann, T. Onodera, "Extending OPMISR beyond 10x scan test efficiency", *IEEE Design & Test of Computers*, Vol 19, Sept.-Oct 2002, pp 65-73
- [12] S. Mitra and K.S. Kim, "XPAND: An efficient test stimulus compression technique", *IEEE Trans. On CAD*, Vol. 55, Feb 2006, pp 163-173
- [13] P. Wohl, J. Waicukauski, S. Patel, F. Da Silva, T.W. Williams, R. Kapur, "Efficient compression of deterministic patterns into multiple prpg seeds", *Proc. Of ITC*, Nov. 2005, pp 916-925