A Method for Storing Semiconductor Test Data to Simplify Data Analysis

Jeremy W. Webb
J. Webb Consulting
Woodland, California 95695
jeremy.webb@jwebb-consulting.com

***Abstract*—The automated testing of semiconductor wafers, integrated circuits (IC), and surface mount devices generates a large volume of data. These devices are tested in stages and their test data is typically collected in log files, spreadsheets, and comma separated value files that are stored at both the foundry and on-site on file shares. Building a cohesive picture of the quality and performance of these devices can be difficult since the data is scattered throughout many files. In addition, tracking any deviations in performance from wafer to wafer by analyzing historical process control monitoring (PCM) test data is a manual, laborious task. Collecting the test data from multiple sources (e.g., foundries and contract manufacturers) can be cumbersome when manual intervention is required. Automating the transfer of test data from third party servers to on-site file shares is crucial to providing a consistent method of accessing the data for analysis. This paper provides a method of both implementing a database for storing test data collected at the various stages of IC manufacturing, as well as automating the retrieval and import of test data into the database from all relevant sources.**

I. Introduction

A large volume of data is generated during semiconductor fabrication, wafer testing, and package testing. This data can be stored in the following file types:

- GDSII files
- Foundry process control monitor data
- Test executive software output files
- Touchstone files (e.g., S-Parameters)

The GDSII files are mainly used to extract the die names and X/Y coordinates within a reticle for conversion into wafer level X/Y coordinates. In the case of foundry PCM data, a single file may be generated, which could be in the form of a spreadsheet or a physical piece of paper. The test executive output files may either be separated into individual files per die or all lumped into a single spreadsheet. When using individual files per die there can be thousands upon thousands of individual files containing both parametric and swept data, such as touchstone S-Parameter files.

Extracting any useful information from this collection of files can be challenging. Often custom scripts, Excel spreadsheets, R scripts, and GNU Plot scripts are employed to try and make sense of the data to determine yield for individual wafers or wafer lots, wafer-to-wafer performance, or even to generate a pick list of good parts to be used during the packaged IC production process.

This paper describes the steps necessary to develop a parametric database for storing semiconductor test data. These steps include identifying producers and consumers of data, understanding all file formats used for storing the data, locating the directories where the data is stored, developing the database structure, and finally automating report generation and providing a method of accessing the data for analysis.

II. Parametric Database

A. Data Producers and Consumers

Before a database can be implemented, both the producers and consumers of data must be identified and interviewed to understand the type of data being manipulated. In this case, the main goal of the parametric database development was to identify all sources of data from the start of the IC design to production of the wafers and surface mount devices.

An interesting occurrence during the development of the parametric database was that producers and consumers sometimes gave limited answers to initial interview questions and it became necessary that more focused questions be asked for further clarification. This additional probing made it possible to learn more about the data and processes, and slowly the full picture emerged regarding how the data was generated or used and how it influenced the next stage of the design or manufacturing process.

1) Producers:

Common producers of data include IC design engineers, wafer foundries, and test executive software. The data generated by producers will lay the foundation for the tables in the database. Data producers have a deep understanding of the data they generate, but they are sometimes unable to visualize the bigger picture of how the data is connected.

a) Interview Questions:

- What type of data do you generate?
- Where do you store your data files?
- What are the stages of the production process?
- How many process stages exist for each type of device (e.g., wafers, ICs, multi-chip modules)?
- Do you typically re-test a device during production? If so, how do you tag the data to distinguish between previous tests?
- What type of specifications exist for devices?
- How do you keep track of specification revisions?

978-1-5090-0790-5/16/$31.00 ©2016 IEEE

- Do you ever manually record data on paper or in a spreadsheet?

2) Consumers:
Consumers of data include test engineers, quality engineers, managers, production planners, and IC design engineers. Consumers generally have knowledge about how to analyze the data and an understanding of how all of the data is connected.

a) Interview Questions:

- What type of data do you access to do your job?
- Where do you access data (e.g., Foundry file transfer protocol (FTP) server, networked file share)?
- If you connect to an FTP site, what are the credentials used to gain access? Is the password changed at a regular interval?
- Do you have to reformat data prior to analysis?
- What programs do you use to analyze data?
- Do you generate reports? If so, how are they formatted, what is the purpose, who is the intended audience?

B. Data Sources

The next phase of parametric database development is often referred to as the extract, transform, and load (ETL) phase. This section addresses the extract and transform steps for the development of parsing scripts, and the following section will discuss the load steps.

Understanding the file formats used to store data is key to determining how many parsers need to be developed. When developing the parsing scripts, it is recommended to choose an output format that is common to all parsers. For example, the extensible markup language (XML) file format.

1) Design Data:
As either multi-chip or production wafers are completed and the reticle map documentation layer has been generated, then the GDSII file can be processed to extract a list of devices under test (DUT) with the corresponding chip name, width and area, and the origin positions (row/column) on the reticle for each DUT with its corresponding x and y position. The DUT chip name describes what type of devices are present on the reticle. The reticle is tiled to create a wafer mask, which in turn is used by the foundry to fabricate individual wafers. Due to the complexity of parsing the GDSII files, this task is often an iterative process and close collaboration is required between the test team and the IC design team to ensure that the proper information is present in the GDSII documentation layer.

The overall wafer X/Y coordinates are calculated along with the die serial labels or chip identifiers (IDs) (e.g., A5500) using the wafer and reticle dimensions, DUT X/Y coordinates, and other miscellaneous information. The chip IDs will be used to associate data in the parametric database to a specific wafer mask, wafer lot, and wafer number. The chip IDs located on the edge of the wafer should not be tested, and can either be tagged as being on the wafer edge at the time of GDSII data extraction or after the GDSII data has been loaded into the parametric database. Figure 1 shows the processing steps for extracting, transforming, and loading the GDSII file data into the parametric database. Listing 1 shows an example of an XML file containing information extracted from a GDSII file for a wafer mask.

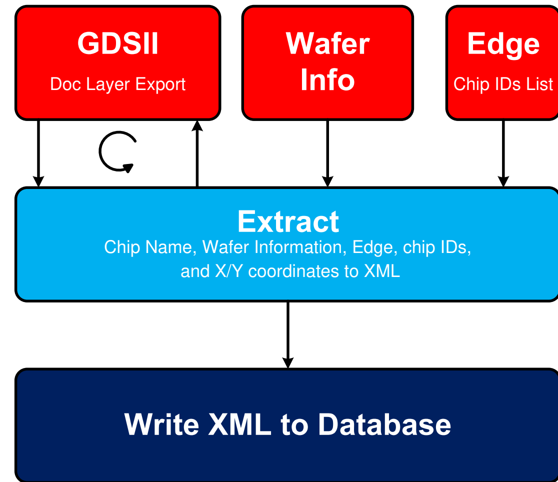

Fig. 1. Wafer mask GDSII file processing steps.

2) Foundry Data:
Semiconductor foundries generate a variety of data files during the fabrication and testing process, including open purchase orders, work in progress (WIP) status, PCM data, and wafer test data. This data is typically stored in either comma separated value (CSV) files or Excel spreadsheets. Figure 2 shows the processing steps for extracting, transforming, and loading the foundry open purchase order and WIP data into the parametric database.

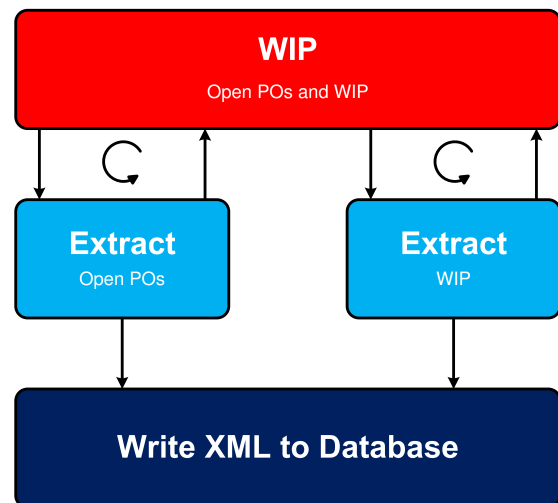

Fig. 2. Wafer fabrication work in progress and open purchase order data processing steps.

When starting a new relationship with a foundry, the formatting of the files can vary, but after two or three runs through

Listing 1. Example Wafer Mask XML File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- CSS Include File for display in web browser -->
<?xml-stylesheet type="text/css" href="../css/mask.css"?>
<mask>
    <info>
        <mask_name>aa001</mask_name>
        <timestamp>1404616217</timestamp>
        <material>gaas</material>
        <device_type>wafer</device_type>
        <makeup>whole</makeup>
        <foundry>foundry1</foundry>
    </info>
    <mask_params>
        <x_width>150000.0</x_width>
        <y_height>150000.0</y_height>
        <center_column>4.0</center_column>
        <center_row>4.0</center_row>
        <x_offset>0.0</x_offset>
        <y_offset>7.0</y_offset>
        <rim_scale>1.2</rim_scale>
        <max_reticle_cols>9.0</max_reticle_cols>
        <max_reticle_rows>8.0</max_reticle_rows>
        <wafer_x_offset>-1000.0</wafer_x_offset>
        <wafer_y_offset>0.0</wafer_y_offset>
        <inner_ring>0.96</inner_ring>
        <middle_ring>0.98</middle_ring>
        <outer_ring>1.0</outer_ring>
        <pcm_xoffset>0.0</pcm_xoffset>
    </mask_params>
    <gdsii>
        <filename>mychip.gds</filename>
        <filesize>51200</filesize>
        <filepath>/prod/wafer/mychip/</filepath>
        <fileblob><snip></fileblob>
        <filedump><snip></filedump>
        <timestamp>1404616217</timestamp>
        <quadrant>Q</quadrant>
    </gdsii>
    <reticle>
        <width>18400</width>
        <height>21320</height>
        <scale>0.00000100</scale>
        <units>meters</units>
        <makeup>whole</makeup>
    </reticle>
    <dut id="DEV-01">
        <dut_key>DEV-01</dut_key>
        <dut_family>DEV</dut_family>
        <dut_type>01</dut_type>
        <dut_x>920</dut_x>
        <dut_y>1640</dut_y>
    </dut>
    <dut id="PCM-01">
        <dut_key>PCM-01</dut_key>
        <dut_family>PCM</dut_family>
        <dut_type>01</dut_type>
        <dut_x>920</dut_x>
        <dut_y>1640</dut_y>
    </dut>
    <die id="A5500">
        <die_key>DEV-01</die_key>
        <die_family>DEV</die_family>
        <die_type>01</die_type>
        <die_quadrant>A</die_quadrant>
        <die_x>73600</die_x>
        <die_y>63960</die_y>
        <die_edge>0</die_edge>
    </die>
    <die id="snip">
    </die>
    <die id="D5501">
        <die_key>DEV-01</die_key>
        <die_family>DEV</die_family>
        <die_type>01</die_type>
        <die_quadrant>D</die_quadrant>
        <die_x>74520</die_x>
        <die_y>63960</die_y>
        <die_edge>0</die_edge>
    </die>
</mask>
```

the fabrication process the files should converge to a consistent format. This is especially true when foundries are performing production level wafer testing since there tends to be more variability in the data being collected. Subsequent parsing of the foundry files becomes straightforward and predictable, which is key for automating the data extraction.

a) Open Purchase Orders: Open purchase order information can be used to trigger creation of new wafer mask definitions in the parametric database. As new wafer masks are detected, an automated message can be sent to the test team to inform them that GDSII files are available for upload to the parametric database.

b) Work in Progress: Work in progress data contains information relating to the current stage of wafer fabrication. This type of data can be very useful for manufacturing operations and planning departments when scheduling activities.

Some of the process steps reported in the WIP data can take longer than a day, therefore the data is typically compared to the WIP table within the parametric test data to see if the wafer mask, lot, and number has advanced to a new process step. If it has, then a new entry is made. If not, then no entry is made in the parametric database. This information can be used to determine the amount of time a wafer will spend in any given process step and the average time it takes to fabricate a new wafer.

An interesting observation during the development of the work in progress parsing scripts was that the steps performed during wafer fabrication were not always the same from wafer to wafer. Initially, a request was made for a list of all process steps for storage in the parametric database. However, after storing these "known" process steps the parsing script began detecting unknown process steps. Important insight can be gained by observing the steps performed during wafer fabrication. For example, if the process steps used for two wafers fabricated from the same mask are different, this information may help in identifying the root cause of extreme wafer performance deviations.

c) Wafer Process Control Monitoring Data: Wafer PCM data contains X/Y coordinates of the PCM test structures on the wafer and their associated measurement data. This data can be used to indicate the performance of devices distributed across the wafer. Where PCM parameters align with those parameters collected for the DUT (e.g., the drain current of a field-effect transistor (I_{dss}) or the maximum frequency of operation) the X/Y coordinates can be used to map the measurements to the location on a wafer and assigned a color based on a range of values similar to a 2-D contour plot. Figure 3 shows the processing steps for extracting, transforming, and loading the foundry data into the parametric database.

d) Wafer Test: When foundries perform wafer level testing they typically generate either a CSV or Excel spreadsheet file containing

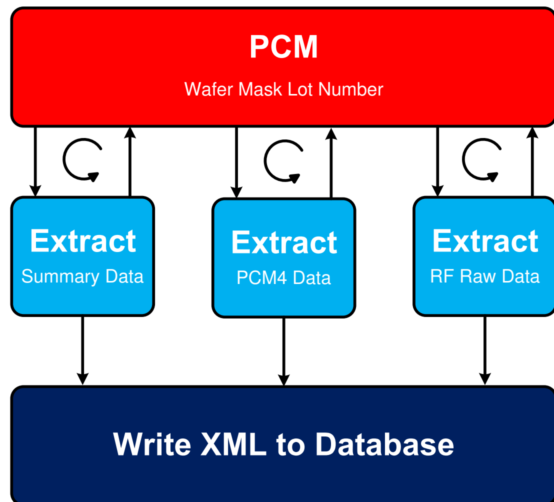

Fig. 3. Wafer fabrication process control monitoring data processing steps.

test data for all die on the wafer. It is recommended to have the foundry include the specifications for each parameter measured in the data file, as well as the conditions for the measurement. For example, how a specific DUT was biased for the measurement. Tracking revisions made to specifications can be useful when the yield of a wafer changes or a new trend occurs regarding a specific parameter or set of parameters. Generally, the more information a foundry can include in the header of the test data file the better because it reduces the number of descriptors needed by a script when importing into the parametric database. The following is a sample list of information that should be included in the header of the data file:

- Wafer mask name
- Wafer lot number or identifier
- Wafer number
- Test station identifier
- Test date and time
- Measurement conditions

If the measurement data for each DUT is provided its own file, then the DUT or chip name should be included in the header as well.

3) Production Data:
New product introduction (NPI) and production tests generate a significant number of data files, including parametric test data files and swept data measurement files. For wafer testing, the test executive software is usually controlling a wafer probe station using the DUT X/Y coordinates to identify the probing location on the wafer. Depending on the makeup of the wafer (e.g., multi-chip wafer (MCW) or production wafer), the test executive software will either test the entire wafer in one pass or perform multiple passes per unique DUT type. In the case of an MCW, multiple test and measurement equipment setups may be required in addition to an adjustment of the die probe configuration, which adds more variance to the type of data being collected. The test executive software will either generate a combined set of data in a single file or individual data files for each unique DUT. It can be convenient to use a combined data file for production wafers containing only a single type of DUT. However, most wafer tests do not complete successfully in one pass, and using a single data file can be difficult to keep up to date after the overall wafer tests have been completed. Generating a separate data file per device may end up being more efficient for post-processing tasks. Additionally, when processing a batch of data files using the parsing script, separate data files can allow for quicker recovery when restarting the parsing script after an error occurs during the parsing (e.g., server power loss or disk failure). Figure 4 shows the processing steps for extracting, transforming, and loading the parametric test data files and swept data measurement files into the parametric database.

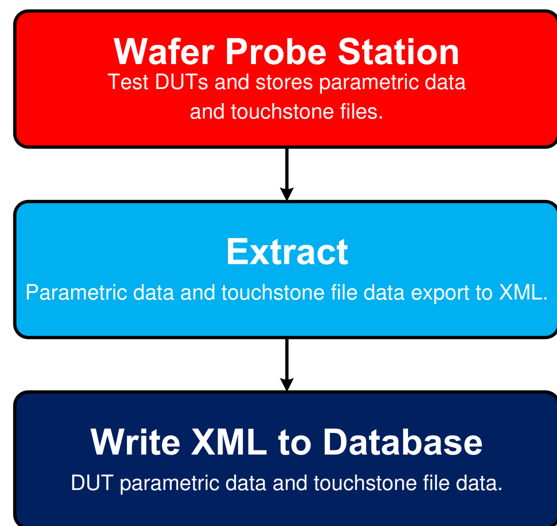

Fig. 4. Wafer parametric test data and swept data measurement file processing steps.

The format of the data files generated by the test executive software tends to change over time due to changes in the test method, parameters measured and stored, updates to the file header, or even to the personnel managing the test executive software. The parsing scripts must be written in such a way that they can detect the deviations in the data file formatting. The type of deviations that can occur range from differences in header formatting to adding entire sections for documenting which test equipment was used to perform the measurements. It is recommended that a file version be included in all data files that indicates the version of the data file template. Using a file version creates a straightforward process for detecting the different data files in a parsing script. When uploading a large set of historical data files it can be difficult to ascertain how many data file variants exist in the set of files. The simplest method is to execute a parsing script on a large set of files and add provisioning in the parsing script to flag an error upon detecting deviations. Depending on the severity of the

deviation, only a minor change may be required to allow the parsing script to process the data file successfully. In the case of major deviations, new parsing sub-routines may need to be written as well as a sub-routine to determine which parsing sub-routine should be executed.

a) Parametric Test Data:
Listing 2 shows an example of an XML file containing information extracted from a parametric test data file for a unique device.

Listing 2. Example Parametric Test Data XML File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- CSS Include File for display in web browser -->
<?xml-stylesheet type="text/css" href="../css/data.css"?>
<die_data>
    <info>
        <file_version>1.1</file_version>
        <username>jwwebb</username>
        <timestamp>1404616217</timestamp>
        <mask_name>aa001</mask_name>
        <lot_number>1</lot_number>
        <wafer_id>1</wafer_id>
        <quadrant>q</quadrant>
        <device_type>wafer</device_type>
        <material>gaas</material>
        <dut_family>DEV</dut_family>
        <dut_type>01</dut_type>
        <dut_chipid>A5500</dut_chipid>
        <dut_status>PASSED</dut_status>
        <test_sequence>first-pass</test_sequence>
        <test_system>TS10</test_system>
        <test_type>final_test</test_type>
    </info>
    <specifications>
        <spec_class id="DEV-01">
            <specification id="Ambient">
                <spec_low_value>15</spec_low_value>
                <spec_high_value>35</spec_high_value>
                <spec_units>degC</spec_units>
                <spec_desc>Ambient Temperature</spec_desc>
                <spec_type>parameter</spec_type>
            </specification>
            <specification id="Idss">
                <spec_low_value>0.22</spec_low_value>
                <spec_high_value>0.33</spec_high_value>
                <spec_units>Amps</spec_units>
                <spec_desc>Drain Current</spec_desc>
                <spec_type>parameter</spec_type>
            </specification>
        </spec_class>
    </specifications>
    <measure>
        <measure_class id="DEV-01">
            <parameter id="Ambient Temperature">
                <param_value>21.264</param_value>
                <param_units>degC</param_units>
                <param_cond>Vg=0.0, Vd=8</param_cond>
                <param_status>passed</param_status>
            </parameter>
            <parameter id="Idss">
                <param_value>0.2869048</param_value>
                <param_units>Amps</param_units>
                <param_cond>Vg=0.0, Vd=8</param_cond>
                <param_status>passed</param_status>
            </parameter>
        </measure_class>
    </measure>
</die_data>
```

b) Swept Measurement Data:
In the case of swept data measurements, using a separate file makes more sense because the measured data is typically contained in a proprietary format. A common format for swept network measurements containing S-Parameters is a touchstone file. These files tend to be made up of columns containing frequency and the individual S-Parameter measurements. Storing the touchstone option string (e.g., # HZ S RI R 50) will allow future extraction of the S-Parameters from the parametric database into a touchstone file for further analysis using tools such as Keysight Technologies' Advanced Design System software. Listing 3 shows an example of an XML file containing information extracted from a touchstone data file for a unique device.

Listing 3. Example Touchstone Data XML File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- CSS Include File for display in web browser -->
<?xml-stylesheet type="text/css" href="../css/data.css"?>
<touchstone>
    <info>
        <mask_name>aa001</mask_name>
        <username>jwwebb</username>
        <timestamp>1404616217</timestamp>
        <material>gaas</material>
        <quadrant>q</quadrant>
        <lot_number>1</lot_number>
        <wafer_id>1</wafer_id>
        <device_type>wafer</device_type>
        <dut_family>DEV</dut_family>
        <dut_type>01</dut_type>
        <dut_status>FAILED</dut_status>
        <dut_chipid>A5500</dut_chipid>
        <test_sequence>single-pass</test_sequence>
        <test_system>TS10</test_system>
        <test_type>final_test</test_type>
        <option_line># HZ S RI R 50</option_line>
    </info>
    <sparameters>
        <data_row id="100000000">
            <freq>100000000</freq>
            <s11_im>0.02047161</s11_im>
            <s11_re>-0.04349013</s11_re>
            <s12_im>-0.0001343436</s12_im>
            <s12_re>1.809457e-05</s12_re>
            <s21_im>1.051201</s21_im>
            <s21_re>-3.672653</s21_re>
            <s22_im>0.07070011</s22_im>
            <s22_re>-0.1499106</s22_re>
        </data_row>
        <data_row id="snip">
        </data_row>
        <data_row id="65000000000">
            <freq>65000000000</freq>
            <s11_im>0.04317693</s11_im>
            <s11_re>0.1639288</s11_re>
            <s12_im>0.02116442</s12_im>
            <s12_re>-0.07889209</s12_re>
            <s21_im>1.004514</s21_im>
            <s21_re>2.728517</s21_re>
            <s22_im>-0.222041</s22_im>
            <s22_re>0.1913727</s22_re>
        </data_row>
    </sparameters>
</touchstone>
```

C. Data Import

Once all sources of data have been identified and parsing scripts have been developed to extract data to an XML file, then the database import scripts can be developed. As mentioned in Section II-B, choosing a common output format such as XML will greatly simplify the database import operations because the data will be uniform and can be processed the same way every time.

1) XML Data Import:
The database import script will use the XML root element to detect the type of data contained within the XML file. Common XML root elements employed in the parametric database are:

- **mask** - wafer mask information XML file
- **foundry1_data** - foundry #1 test XML data
- **die_data** - production DUT XML data
- **touchstone** - production DUT touchstone XML data

Listing 4 shows an example of how to extract XML file data, in this case a parametric test data XML file, into an associative array using Perl. In this example, an XML file with a die_data root element is shown, but the steps will be similar for other XML root elements with only slight variations in the child element processing. The extraction process consists of creating a new XML::LibXML object, invoking the *parse_file* method, verifying the XML root element exists, and then proceeding to loop through the XML child elements while inserting the data into an associative array. The extraction process will be similar in other languages, such as Python or C/C++.

a) XML Child Elements:

The **info** XML child element contains parameters that describe the data held within the XML file. For example, the **info** XML child element within the parametric test data XML file (shown in Listing 2) contains parameters that describe the file version, test technician, test timestamp, wafer mask name, wafer lot, wafer number, wafer quadrant, device type, material, DUT family/type, DUT chip ID, DUT status, test sequence, test system, and test type. These parameters help map the data into the parametric database. SQL queries are performed on these parameters to verify existence in the parametric database and determine the primary IDs from the map tables.

The **specifications** XML child element contains the specification classes and parameters, which include the low and high values, the parameter's units, a description, and the specification type. The parametric test data files can contain any number of specification classes. For example, the single-pass test sequence may contain the following specification classes:

- **session** - measurement limits and analysis boundaries.
- **base** - default specifications.
- **dut** - end product specifications.

All specification classes contain the *parameter* specification type, which indicates a parameter/value assignment. The session specification class contains three additional specification types: boundary, limit, and bias. The boundary specification type can be used to describe the boundaries or ranges of S-Parameter data. The limit specification type can be used to describe a voltage or current limit. The bias specification type can be used to specify the bias conditions for a DUT in certain modes of operation.

As the specifications are imported into the parametric database, the specification classes are compared to existing classes stored within the wafer_die_specs table. If an identical match occurs, then the specification class revision ID is retrieved from the wafer_die_specs table. Otherwise, if any specification parameter of a specification class is different than the specifications stored in the wafer_die_specs table, the specification class revision ID is incremented. Keeping track of specification class revisions can help with identifying the cause of product performance issues.

All specification classes, with the exception of the session specification class, contain pass and fail status for each parameter measured by the test executive software. Therefore, the **measure** XML child element will contain the same number of measurement classes as there are specification classes with the exception of the session specification class.

2) Batch Data Import:

Storing both an individual parametric test data file and a swept data measurement file for each die will result in thousands of files being generated for a single wafer. Using an import script to load the *N* data files into the parametric database would require *N* operations. Automation scripts can be developed to operate on batches of files. It is suggested that a script be developed to import all files related to a specific wafer test lot directory path. The process may require more than one script be developed (e.g., one script to convert all data files into XML files and another script to load the data contained within the XML files into the parametric database).

Further automation can be achieved by developing scripts that can operate on lists of wafer test lot directory paths. Parsing the wafer test lot directory paths can be challenging due to the diverse naming conventions used. A sample list of directory paths is shown below:

- /wafer/test/aa001_001-1q/DEV/DEV01/
- /wafer/test/aa001_001-1q/DEVSampleTest/DEV01/
- /wafer/test/aa001_001-1q/DEVSampleTestData/DEV01/
- /wafer/test/aa001_001-1q/DEV_1stPass/DEV01/
- /wafer/test/aa001_001-1q/DEVSampletest1/DEV01/

The script will need to extract the following information from the wafer test lot directory paths:

- Wafer mask name
- Wafer lot number
- Wafer number
- Wafer quadrant
- Test type

When operating on a large volume of historical test data, the load time can be quite long (i.e., days or weeks) depending on the computing resources available. Listing 5 shows an example how to automate the XML file creation for unique parametric test data files. Similar scripts can be written for swept data measurement XML file creation, as well as for XML file uploading to the parametric database.

Depending on how the parametric database is integrated with the test executive software, a method to stage data for import can be useful. Due to the iterative nature of wafer testing, it is difficult to automate the staging of data for import. An HTML form was created to allow test technicians to stage data for import into the parametric database after a wafer test lot had completed the testing process.

3) Foundry Data Import:

The data generated by foundries is typically accessed via a secure website that allows individual data files to be downloaded. Sometimes foundries will provide an FTP server that can be used to download files to a local directory. However, there can be discrepancies between which files are accessible

Listing 4. Example XML File Importer

```
#--------------------------------------------------------------
# 1. Create XML::LibXML Object:
#--------------------------------------------------------------
my ($pXML) = XML::LibXML->new();

#--------------------------------------------------------------
# 2. Get XML File Parse Object:
#--------------------------------------------------------------
my ($dXML) = = $pXML->parse_file($self->{x}->{ifile});

#--------------------------------------------------------------
# 3. Check Root Key for existence:
#--------------------------------------------------------------
$self->{isXMLDieData} = $dXML->exists('/die_data');

#--------------------------------------------------------------
# 4. Extract Die Data from XML File Parse Object:
#--------------------------------------------------------------
my (%ddata); # Parametric Die Data Hash.
if ($self->{isXMLDieData} == 1) {
  for my $tn ( $dXML->findnodes('/die_data/*') ) {
    my ($tnn) = $tn->nodeName();
    if ($tnn eq "info") {
      #------------------------------------------------------
      # 4a. Get second level child element(s): info
      #------------------------------------------------------
      for my $isn ( $tn->findnodes('./*') ) {
        my ($isnn) = $isn->nodeName();
        $ddata{$tnn}->{$isnn} = $isn->textContent();
      }
    } elsif ($tnn eq "specifications") {
      #------------------------------------------------------
      # 4b. Get third level child element(s):
      #      specifications/spec_class
      #------------------------------------------------------
      for my $sdn ( $tn->findnodes('./spec_class') ) {
        my ($sdnn) = $sdn->nodeName();
        my ($sdnga) = $sdn->getAttribute("id");
        #----------------------------------------------------
        # 4c. Get fourth level child element(s):
        #      specification
        #----------------------------------------------------
        for my $pn ( $sdn->findnodes('./specification') ) {
          my ($pnga) = $pn->getAttribute("id");
          for my $psn ( $pn->findnodes('./*') ) {
            my ($psnn) = $psn->nodeName();
            $ddata{$tnn}->{$sdnga}->{$pnga}->{$psnn} =
                $psn->textContent();
          }
        }
      }
    } elsif ($tnn eq "measure") {
      #------------------------------------------------------
      # 4d. Get third child element(s):
      #      measure/measure_class
      #------------------------------------------------------
      for my $sdn ( $tnode->findnodes('./measure_class') ) {
        my ($sdnn) = $sdn->nodeName();
        my ($sdnga) = $sdn->getAttribute("id");
        #----------------------------------------------------
        # 4e. Get fourth level child element: parameter
        #----------------------------------------------------
        for my $pn ( $sdn->findnodes('./parameter') ) {
          my ($pnga) = $pn->getAttribute("id");
          for my $psn ( $pn->findnodes('./*') ) {
            my ($psnn) = $psn->nodeName();
            $ddata{$tnn}->{$sdnga}->{$pnga}->{$psnn} =
                $psn->textContent();
          }
        }
      }
    } else {
      my ($emsg) = sprintf("Unknown Node (%s)!\n", $tnn);
      warn($emsg);
    }
  }
}

#--------------------------------------------------------------
# 5. Assign Die Data Hash to Self Object:
#--------------------------------------------------------------
$self->{x}->{fdata}->{die_data} = \%ddata;
```

Listing 5. Example Bash File Batch Importer

```
#!/bin/sh

#--------------------------------------------------------------
#
# gen_pdata_xml.sh module
#
#--------------------------------------------------------------

#--------------------------------------------------------------
# Input Arguments:
#--------------------------------------------------------------
MASK=$1
LOT=$2
WAFER=$3
QUAD=$4
TTYPE=$5
FPATH=$6
WTLOTID=$7
#--------------------------------------------------------------
# Create XML Filename Prefix:
#--------------------------------------------------------------
QUADLC=`lc ${QUAD}`
PDATAPRE=`printf "pdata_%s_%03d-%d%s_%d" ${MASK} ${LOT} \
          ${WAFER} ${QUADLC} ${WTLOTID}`
echo ${PDATAPRE}
#--------------------------------------------------------------
# Create XML Storage Directory:
#--------------------------------------------------------------
XMLDIR="/tmp/${PDATAPRE}"
mkdir ${XMLDIR}
#--------------------------------------------------------------
# Store Parametric Data File Names into FILES variable:
#--------------------------------------------------------------
FILES=`ls ${FPATH}/*.dat`
#--------------------------------------------------------------
# Set Persistent Variables:
#--------------------------------------------------------------
./perl/pdata2xml.pl -test ${TTYPE}
./perl/pdata2xml.pl -quad ${QUAD}

for f in $FILES
do
    echo "Processing $f"
    if [[ -z $WTLOTID ]]
    then
        #------------------------------------------------------
        # If WTLOTID is empty, then don't pass it
        # to pdata2xml.pl.
        #------------------------------------------------------
        ./perl/pdata2xml.pl -mask ${MASK} -lot ${LOT} \
            -wafer ${WAFER} -file $f -gx
    else
        #------------------------------------------------------
        # If WTLOTID is not empty, then pass it to
        # pdata2xml.pl.
        #------------------------------------------------------
        ./perl/pdata2xml.pl -mask ${MASK} -lot ${LOT} \
            -wafer ${WAFER} -file $f -wtlid ${WTLOTID} -gx
    fi
done
```

on the foundry website versus the FTP server. During the development of the parametric database, one foundry provided access to PCM and WIP data via the website, but only the PCM data was available on the FTP server. A request was made to make the WIP data accessible via the FTP server, and it became possible to automate the daily extraction of open purchase order and work in progress data. A cron job was created to check the FTP server for new files on a nightly schedule, and import new data into the parametric database as it became available.

D. Database Structure

The parametric database will contain many tables with unique relationships that describe the makeup of wafers and devices. After interviews have been completed, initial tables can be created based on each user's input. The data files referenced by the users will also play a crucial role in shaping the table definitions. At this stage, table creation does not have to happen within an actual database. Alternatively, the table definitions can be outlined on paper or in text files using SQL CREATE TABLE statements. Once the initial table definitions have been created, they are reviewed and common columns can be identified for possible use as either foreign keys or to be extracted into a new mapping table to reduce the duplication of data within the parametric database. As an example, consider parametric test data files that contain chip IDs, wafer mask, wafer lot, and wafer number information. This data should not be duplicated within the parametric measurement data table because it is already present in the wafer mask, wafer lot, and wafer chip ID with X/Y coordinate tables. The best solution to this problem is to reference the primary keys that describe the chip IDs, wafer mask, wafer lot, and wafer number information.

1) Table Creation:

The process of creating the database structure is iterative. As relationships between tables are better understood, new tables are created to minimize duplication of data within the parametric database. The act of creating new tables and redefining tables can be laborious. However, simple scripts can be written to support removal and creation of tables as well as uploading sample data to use with SQL queries when testing the data integrity of the parametric database.

Once the database structure has been proven and the data integrity has been verified, the database can be built using a database integrated development environment (IDE) tool. Some database companies will include a default database IDE (e.g., MySQL Workbench), but third party database IDE tools are also available (e.g., Navicat®). A database IDE tool will perform checks on data operations, such as checking that no foreign key dependencies exist prior to deleting data. The creation of custom views is more reliable when using a database IDE due to its ability to access table and column names directly from the database. Without a database IDE tool, view creation can be quite error prone.

To visualize the table relationships, a database IDE tool can be used to create a model of the database by reverse engineering the relationships. Figure 5 shows the model of the parametric database created by the Navicat® database IDE. Foreign keys are indicated by the lines connecting the tables and the arrows indicate where the foreign keys originate.

2) Table Structure:

The table structure of the parametric database can be viewed as three main sections:

- System Tables
- Map Tables
- Wafer Tables

Figure 5 shows a visual representation or model of the final implementation of the parametric database.

a) System Tables:

System tables are tables that do not contain any information that describes a wafer or device. Instead, these tables are used to store information about the parametric database users (e.g., Active Directory Roles and Usernames), a log of which users have viewed or generated reports, a log of events that have occurred, a list of the table and view names in the parametric database, and a history of the parametric database version as well as what changes were made in each version.

Below is a list of the system tables shown in Figure 5:

- **ad_login_failures**
- **ad_roles**
- **ad_users**
- **cgi_view_log**
- **event_log**
- **schema_list**
- **sessions**
- **version**

b) Map Tables:

Map tables are tables that map an integer identifier to a name and description. These tables are used to store descriptive information that gets shared across many tables or rows within a table. For example, the pass or fail status of a DUT would use an ID indicating either a pass or fail, which has the benefit of taking up less space within the database and allows the *pass* or *fail* terms to be changed to *passed* or *failed* without affecting the contents of the status tables.

Below is a list of the map tables shown in Figure 5:

- **device_pfui_map**
- **device_status_map**
- **device_type_map**
- **process_flow**
- **test_provider_map**
- **test_sequence_map**
- **test_system_map**
- **wafer_foundry_bin_status_map**
- **wafer_foundry_map**
- **wafer_foundry_wip_stage_map**
- **wafer_makeup_map**
- **wafer_material_map**
- **wafer_quad_map**

c) Wafer Tables:

The wafer tables contain information that describe either the wafer mask or the devices on the wafer. The main purpose of these tables is to provide a method of describing and referencing the parametric test data, S-Parameter touchstone data, and the DUT status.

Below is a list of the wafer tables shown in Figure 5:

- **ic_die_map**
- **stage_wafer_test_data_upload**
- **wafer_chipid_xy_map**
- **wafer_die_specs**
- **wafer_die_status**

- **wafer_mask_params**
- **wafer_gds_duts**
- **wafer_gds_files**
- **wafer_gds_origins**
- **wafer_lots**
- **wafer_mask_map**
- **wafer_param_meas_data**
- **wafer_param_spec_status**
- **wafer_pcm_files**
- **wafer_pcm_raw_data**
- **wafer_pcm_summary_data**
- **wafer_test_lots**
- **wafer_to_die_map**
- **wafer_touchstone_data**
- **wafer_foundry_wip_purchase_orders**
- **wafer_foundry_wip_status**

E. Data Analysis

Once the data has been imported into the parametric database, analysis can be performed by querying the tables and views. Basic analysis can be performed from the database console, but typically this type of analysis is restricted to database maintenance. Detailed data analysis is usually performed using third party tools, such as R, JMP® from SAS, Matlab® from Mathworks, Microsoft Excel®, or even scripting languages such as Perl and Python. These tools access the parametric database via an open database connection (ODBC), which is made available by the hosting server. Typical ODBC connection parameters for a MySQL or MariaDB database are shown below:

- **Platform**: MySQL
- **Database Name**: pdb
- **Host Name**: myserver.acme.com
- **Port Number**: 3306
- **User Name**: myusername
- **Password**: mypassword

Listing 6 shows an example of how to query a list of wafer mask information from the parametric database using Perl.

III. Conclusion

Building a cohesive picture of the quality and performance of semiconductor devices can be greatly simplified when employing a parametric database to store semiconductor test data. Without the use of a parametric database, the tracking of any deviations in performance from wafer to wafer can be a manual, laborious task. A parametric database can be deployed either along side existing test executive software programs or fully integrated within the testing flow. This flexibility allows data to be organized in a database for easy extraction during analysis, thus providing great benefit to its users.

Listing 6. Example Perl Query of Wafer Mask Information

```
#!/usr/bin/env perl

use strict;

#-------------------------------------------------------------
# Database Modules
#-------------------------------------------------------------
use DBI;
use DBD::mysql;
use Data::Dumper;

#-------------------------------------------------------------
# Get Database Configuration Variables:
#-------------------------------------------------------------
my (%cmdH);
$cmdH{mysql}->{platform} = sprintf("mysql");
$cmdH{mysql}->{database} = sprintf("pdb");
$cmdH{mysql}->{host}     = sprintf("myserver.acme.com");
$cmdH{mysql}->{port}     = sprintf("3306");
$cmdH{mysql}->{user}     = sprintf("myusername");
$cmdH{mysql}->{pw}       = sprintf("mypassword");

#-------------------------------------------------------------
# MySQL Data Source Name
#-------------------------------------------------------------
$cmdH{dsn} = sprintf("dbi:%s:%s:%s:%s",
                     $cmdH{mysql}->{platform},
                     $cmdH{mysql}->{database},
                     $cmdH{mysql}->{host},
                     $cmdH{mysql}->{port}
                    );

#-------------------------------------------------------------
# Perl DBI Connect
#-------------------------------------------------------------
$cmdH{mysql}->{attrs} = {
                          AutoCommit => 0,
                          RaiseError => 1,
                        };
$cmdH{dbh} = DBI->connect(
                          $cmdH{dsn},
                          $cmdH{mysql}->{user},
                          $cmdH{mysql}->{pw},
                          $cmdH{mysql}->{attrs}
                         );

#-------------------------------------------------------------
# Query Database:
#-------------------------------------------------------------
# Create Select Statement:
#-------------------------------------------------------------
$cmdH{select} = sprintf("SELECT * FROM wafer_mask_map;");

#-------------------------------------------------------------
# Prepare SELECT statement:
#-------------------------------------------------------------
$cmdH{sth} = $cmdH{dbh}->prepare( $cmdH{select} );

#-------------------------------------------------------------
# Execute SELECT statement
#-------------------------------------------------------------
$cmdH{sth}->execute();

#-------------------------------------------------------------
# Fetch the data
#-------------------------------------------------------------
my ($hash_ref);
while ( $hash_ref = $cmdH{sth}->fetchrow_hashref ) {
    # Push Hash Rows onto an Array of Hashes stack:
    push( @{ $cmdH{maskAoH} }, $hash_ref );
}

#-------------------------------------------------------------
# Close Query Statement Handle:
#-------------------------------------------------------------
$cmdH{sth}->finish();
undef $cmdH{sth};

#-------------------------------------------------------------
# Print Query Results:
#-------------------------------------------------------------
print Dumper($cmdH{maskAoH});

#-------------------------------------------------------------
# Disconnect from Database:
#-------------------------------------------------------------
$cmdH{dbh}->disconnect();

exit;
```

Fig. 5. Parametric Database Model