3D-COSTAR: User Guide

Mottaqiallah Taouil[1] Said Hamdioui[1]
[1]Delft University of Technology
Faculty of EE, Mathematics and CS
Mekelweg 4, 2628 CD Delft, The Netherlands
{m.taouil, s.hamdioui}@tudelft.nl

Erik Jan Marinissen[2]
[2]IMEC vzw
3D Integration Program
Kapeldreef 75, 3001 Leuven, Belgium
erik.jan.marinissen@imec.be

Sudipta Bhawmik[3]
[3]Qualcomm
5000 Somerset Corp. Blvd.
Bridgewater, NJ, USA
sbhawmik@qualcomm.com

I. Introduction

One of the major challenges that has to be addressed in order to make 2.5D technology commercially successful is overall cost optimization. A 2.5D-SIC consists of two or more active dies stacked on a passive silicon interposer that forms the interconnection between the active dies and to the external world. As is the case for any IC, TSV-based 2.5D-SICs must be tested in order to guarantee the outgoing product quality and reliability. Hence, test cost is indispensable. Inherent to their manufacturing process, 2.5D-SICs provide several test moments such as before stacking, during manufacturing of partial stacked IC, after the complete manufactured stack, etc. This results into a large space of test flows; each with its own cost. Determining the optimal and most efficient test flow requires the analysis of all test flows, as different design and/or manufacturing parameters may impact the cost differently. Therefore, an appropriate cost model is required. The cost model should be able to evaluate the cost of each test flow, while considering all relevant incurred costs in the production chain of the 2.5-SIC. In this document, we present 3D-COSTARthat is able to perform this. In order to use this tool, whether for reports, scientific publications or other documents, the following two publications must be referenced:

- M. Taouil, S. Hamdioui, E.J. Marinissen, "Quality versus cost analysis for 3D Stacked ICs," IEEE 32nd VLSI Test Symposium, pp.1-6, 13-17 April 2014.
- M. Taouil, S. Hamdioui, E.J. Marinissen, S. Bhawmik, "Using 3D-COSTAR for 2.5D test cost optimization," IEEE International 3D Systems Integration Conference, pp.1-8, 2-4 Oct. 2013

II. Interface

Currently, 3D-COSTAR is only accessible through our web-server available at http://www.ce.ewi.tudelft.nl/3dcostar. A single input file, compliant to 3D-COSTAR's format, must be uploaded through the web-interface as depicted in Figure 1. The executable is automatically activated once the input file is uploaded through the web-server. After 3D-COSTAR finishes the evaluation and execution of the uploaded input file, a zipped file will be made available for download containing all the produced output results. The download link will only be shortly available.

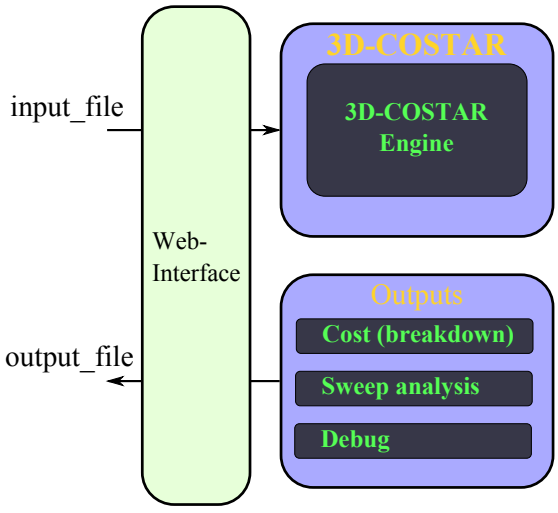

Fig. 1. 3D-COSTAR Interface

The zipped output file contains several files/folders:

- A log file *log_file.txt* showing providing feedback to the user. If the provided input file contains errors they can be found in this file.
- A file containing the cost and cost breakdown analysis for the main experiment. This default output file is described in Section V.

- A folder called *sweep* is created that contains the remaining information such as cost breakdown, escapes etc. per experiment. The sweep input and outputs are described in Section V. An additional file *sweep_summary.txt* is generated that contains a summary output of the sensitivity experiments.
- A debug file called *dump_file.txt*. This file is optionally generated and contains debug information and detailed outputs related to the cost classes.

III. Input File

3D-COSTAR runs experiments using a single ASCII input file. This file contains inputs either directly related to cost classes or to sweep analysis. Figure 2 shows the general architecture of classes the input files of 3D-COSTAR contain, which can evaluate 2.5D-/3D- and 5.5D-SICs. The tool has five input classes which symbolize the costs involved in the whole SIC production flow; these include emphdesign, *manufacturing*, *packaging*, *test*, and *logistics*. Input parameters of these classes can be recognized by their prefix. For example, parameters from the manufacturing class start with the prefix *M*. The cost classes are described first and a general overview with some of the inputs are shown in Figure III-G. Thereafter, a subsection is dedicated to the inputs that control the sweep analysis in Section III-G.

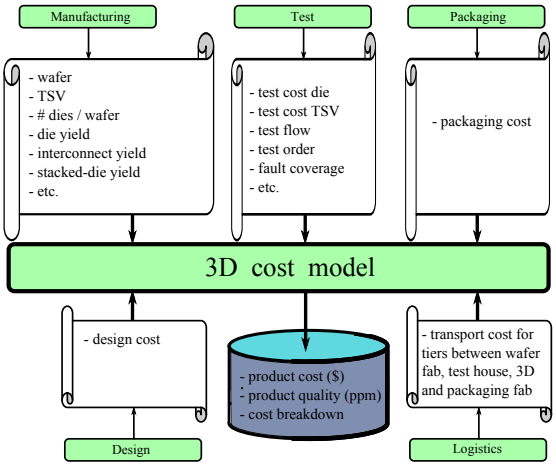

Fig. 2. 3D-COSTAR: Cost Classes Organization.

A. Background and Terminology

Figure 3(a) shows the conventional 2D test flow for planar wafers [1]; it consists of two test moments: a wafer test prior to packaging and a final test after packaging. The wafer test can be cost-effective when the yield is low as it prevents unnecessary assembly and packaging costs, while the final test is used to guarantee the final quality of the packaged chips. 2.5D/3D-SICs, however, provide additional test moments; e.g., additional test moments can be defined for each partial stack. Moreover, at each moment a distinction can be made between different tests such as die tests and interconnect tests. In general, four test moments can be distinguished for a 2.5D-SICs consisting of n dies and $n-1$ stacking operations as depicted in Figure 3(b): (1) n *pre-bond* wafer tests, (2) n-2 *mid-bond* tests, (3) one *post-bond* test prior packaging and (4) one *final* test. Note that the mid-bond, post-bond and final tests could contain several sub-tests for the different dies and interconnects.

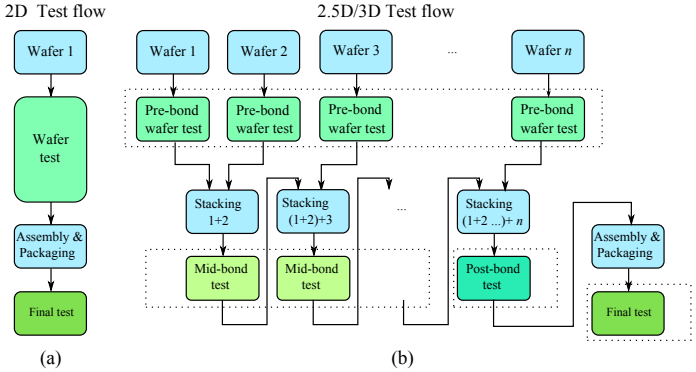

Fig. 3. 2D versus 2.5D/3D D2W test flows.

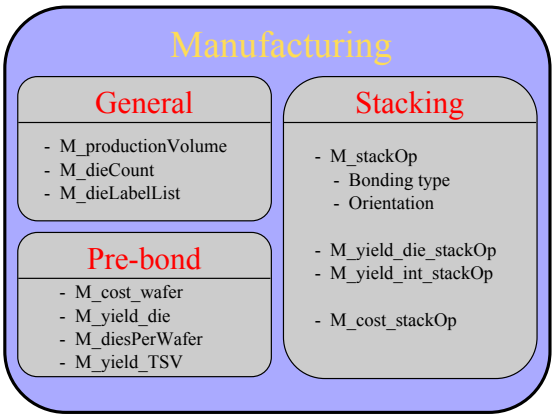

Fig. 4. Manufacturing.

Most parameters of the input file are either of type integer, double, an array of integers or an array of doubles. In order to distinguish between them we use the following notations *i_value*, *d_value*, *ai_value[x]* and *ad_value[x]*, respectively. For the arrays, the term *x* denotes the array length. In case arrays are initialized with all the same value, the user has the option to initialize them by using the letter *a* standing for array (i.e., a *i_value* is equivalent to *ai_value[x]*). In this case, it is sufficient now to specify only item as all the entries are equal. Examples of this will be given later. Note that all parameters and inputs are case-sensitive. Next, the inputs of all the five classes are described.

B. Design

The tool considers only a single direct parameter for design, referred to as non-recurring engineering costs. This parameter is specified as follows:

```
> D_nreCost <d_value>              > D_nreCost 0.0
```

These design costs are shared among the production volume.

C. Manufacturing

The manufacturing parameters have the prefix M (Manufacturing) and consist of several subgroups as depicted in Figure 4. In general, there are three global input parameters:

```
> M_productionVolume <i_value>     > M_productionVolume 10
> M_dieCount <i_value>             > M_dieCount 4
> M_dieLabelList <ai_value[n]>     > M_dieLabelList 1 2 3 4
```

The keyword *M_productionVolume* specifies the number of final shipped ICs (after testing), i.e., the considered good 3D-SICs shipped to the customers. In the example, this values is set to 10. In general, the number must be positive. The keyword *M_dieCount* specifies the number of dies that the stack contains, which must be positive. For the remainder of this document, we assume the stack size to be 4. In addition, we use the symbol n to denote this stack size. To identify the position of the dies in the stack a label list of dies should be provided. The keyword *M_dieLabelList* takes as argument a list of integers equal to the stack size, where each position identifies the label of the die in the stack. This identification is for important for the stacking order (2.5D-SIC, 3D-SIC or 5.5D-SIC) and the yields that belong to the dies etc. In the given example, the dies are simply labeled from 1 to 4.

The next group of parameters are related to the manufacturing of dies prior to be bonded. Each wafer comes with a certain cost. This cost can be specified in the keyword *M_cost_wafer*. For all the dies in the stack, a value must be specified. Each index in the array corresponds to the die specified on the position in *M_dieLabelList*. This holds in general for all arrays related to dies. In the example below, die with label 1 has a wafer cost of 500, while the other dies (i.e., those with label 2, 3 and 4) have a cost of 5000. Cost units can be specified in any quantity (e.g., $, €, etc.).

```
> M_cost_wafer <ad_value>          > M_cost_wafer 500.0 5000.0 5000.0 5000.0
```

Similarly, the number of dies per wafer and the die yield are specified in the keywords *M_yield_die* and *M_diesPerWafer*.

```
> M_yield_die    <ad_value[n]>     > M_yield_die 0.8165 0.7670 0.4006 0.8006
> M_diesPerWafer <ai_value[n]>     > M_diesPerWafer 622 1278 1605 1605
```

There exists an option to estimate the die yields and the number of dies per wafer by providing the die area. This option is controlled through the *M_enableArea* keyword. For each zero entry in this array, the yields and GDWs provided in *M_yield_die* and *M_diesPerWafer* are taken, otherwise (i.e., a non-zero value) it ignores the corresponding entries in *M_yield_die* and *M_diesPerWafer* and instead uses the binomial yield formula [2] and GDW counting algorithm in [3] to compute the yield and number of gross dies per wafer respectively. To calculte the GDW using the algorithm in [3] requires two arguments: (a) die length, (b) die width and (c) effective wafer radius.

```
> M_enableArea <ai_value[n]>        > M_enableArea 1 0 1 0
> M_dieSize_length <ad_value[n]>    > M_dieSize_length 10.0 0.0 5.0 0.0
> M_dieSize_width <ad_value[n]>     > M_dieSize_width  10.0 0.0 5.0 0.0
> M_radiusEff <ad_value[n]>         > M_radiusEff a 147.0
```

In the particular example above, dies with label 2 and 4 keep the GDW values specified by *M_diesPerWafer* (as their corresponding value in *M_enableArea* is zero), while the GDW for die 1 and 3 will be ignored in that array. The die length and width are specified by the keywords *M_dieSize_width* and *M_dieSize_width* respectively. In this example, the die areas for the dies with label 1 and 3 are equal to 100 mm^2 and 25 mm^2 respectively. The effective wafer radius is specified by the keyword *M_radiusEff*. For simplicity, we use the letter *a* here, to denote that all entries have a wafer radius of 147 mm (a 300 mm wafer with 3 mm edge clearance). Again, the values for the dies with label 2 and 4 will be ignored here.

Similarly, dies 2 and 4 keep the die yields specified in M_yield_die, while the yields for die 1 and 3 will be ignored in that array. To determine the yield using the equation in [2] three parameters are needed: (a) the die area, (b) the defect density and (c) the defect clustering parameter. The die area is obtained from its length and width as explained above. The defect density is specified by the keyword *M_defectDensity* and the clustering parameter by *M_clustering*. The units of the keywords *M_dieSize_length*, *M_dieSize_width*, *M_radiusEff* and *M_defectDensity* must match. For example, these terms could be specified in mm, mm, mm and mm^{-2} respectively. Example values for the defect density and clustering parameters are provided below.

```
> M_defectDensity <ad_value[n]>     > M_defectDensity 0.005 0.00 0.01 0.00
> M_clustering <ad_value[n]>        > M_clustering a 0.5
```

3D-COSTAR requires that all arrays must be fully specified, even if some elements within the array are not used. 3D-COSTAR ignores the actual values of these unused items, but needs them for correct indexing.

3D-COSTAR provides two functions that related the die yield and test escape equations and is used as follows.

```
> M_testEscapeEquation <0 or 1>     > M_testEscapeEquation 1
```

When *M_testEscapeEquation* is set to 0, the yield equation in [2] is used to calculate the test escapes in the pre-bond phase as expressed in Equation 1. It defines the relation between test escapes TE, actual or ingoing yield Y_{in} and outgoing yield Y_{out}; TE, the test escape rate, describes the ratio of faulty dies that pass the test. The ingoing yield is the actual yield (which is an estimated number), the outgoing yield is the fraction of dies that is considered to be good after testing (includes both good dies and test escapes). Equation 2 [2] shows an example of the relation between test escapes, ingoing yield and the fault coverage (FC). By combining equations 1 and 2, the outgoing yield as a function of the ingoing yield and fault coverage is obtained in Equation 3.

$$TE = \frac{Y_{out} - Y_{in}}{Y_{out}} \quad (1)$$

$$TE(FC) = 1 - Y_{in}^{1-FC} \quad (2)$$

$$Y_{out}(FC) = \frac{Y_{in}}{1 - TE} = \frac{Y_{in}}{Y_{in}^{1-FC}} = Y_{in}^{FC} \quad (3)$$

However, Equation 2 assumes an infinite defect clustering parameter which has the tendency to report too pessimistic number of test escapes [4]. Therefore, when *M_testEscapeEquation* is set to 1 the clustering parameter for more accurate and realistic estimation can be used. In that case, the outgoing yield changes into [4]:

$$Y_{out}(FC) = \left(1 + \frac{FC \cdot A \cdot d_0}{\alpha}\right)^{-\alpha} \quad (4)$$

This parameter can only be used when the *M_enableArea* is set to 1.

The next parameters are related to the stacking operation and stacking order. In order to represent the stack, a number of stacking operations must be provided. The keyword that is used is *M_stackingOp*. The number of stacking operations depends on the stack size and equals n-1. For our example, three stacking operations must be specified (as n=4). To differentiate between the stacking operations a stacking id must be directly concatenated to this keyword. After that, the user must provide the two die labels of the dies that are going to be stacked together separated from each other by a hyphen symbol (-). The die

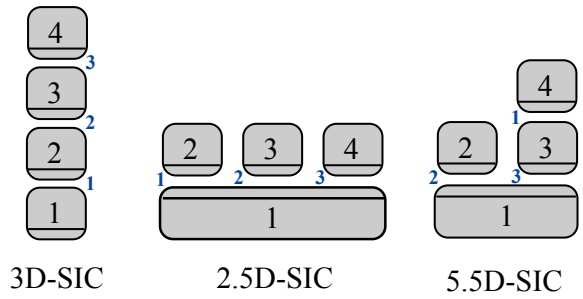

Fig. 5. Different stacking orders and configurations.

label on the left side presents the lower die in the stacking operation and the die on the right side of the hyphen the top die. Subsequently, a vertical line — is used to specify thereafter the bonding type and stacking operation. The currently supported bonding types are either be D2D (for die-to-die stacking) or W2D (for die to wafer stacking). The stacking orientations can be either Face-to-Face (F2F), Face-to-Back (B2B), Face-to-Back (F2B, bottom die face, top die back), and Back-to-Back (B2F, bottom die back, top die face). Examples are given next.

```
> M_stackOp<id> <die_label>-<die_label> | <bonding_type>, <orientation_type>

3D-SIC:                          2.5D-SIC:                          5.5D-SIC:
> M_stackOp1 1-2 | W2D, B2F      > M_stackOp1 1-2 | W2D, F2F      > M_stackOp1 3-4 | D2D, F2F
> M_stackOp2 2-3 | W2D, B2F      > M_stackOp2 1-3 | W2D, F2F      > M_stackOp2 1-2 | W2D, B2F
> M_stackOp3 3-4 | W2D, B2F      > M_stackOp3 1-4 | W2D, F2F      > M_stackOp3 1-3 | W2D, F2F
```

We provide three stacking examples, for a normal 3D-SIC, for a 2.5D-SIC and for a 5.5D-SIC as shown in Figure 5. The numbers in the dies denote the die label and the ones between two dies (in blue) present the stacking order.

The 3D-SIC contains four stacked dies stacked in a linear fashion from the bottom to the top. In the figure, the numbers between two dies (in blue) present the stacking order. In the box above, the equivalent input representation are provided. First die label 2 is stacked on die label 1, using die to wafer stacking with a B2F stacking orientation. Subsequently, die label 3 is stacked on die label 2 in the same manner. The tool internally understands that die 2 is actually a partial stack that contains both the dies with label 1 and 2. Similarly, die label 4 is stacked on die label 3 (i.e, partial stack with the dies with label 1, 2, and 3).

The stacking order for the 2.5D-SIC is different as compared to the 3D-SIC. In this case, dies with labels 2, 3 and 4 are stacked in sequence directly on bottom die 1 (passive silicon interposer) using Face-to-Face stacking.

In the last case, we consider a 5.5D-SIC. Here, in the first stacking operations die with label 4 is stacked on 2. Subsequently, a new partial stack is created with the dies with labels 3 and 4 using D2D stack with a B2F orientation. Finally, in the third stacking operations both partial stacks are stacked on each other (denoted by the stacking operation of die 3 stacked on die 1). The tool internally figures out which dies are valid for which partial stacks.

Each stacking operation involves two kinds of yields, i.e., yields that are related to the dies (stacked-die yields) and yields that are related to interconnects. We will first describe the stacked die yields. The number of keywords for stacked die yields should be equal in size as the number of stacking operations, i.e., 3 in our case. Similarly as in *M_stackOp<id>* an id is used to differentiate between these $n-1$ stacking operation. The stacked die yields should be specified with the keyword *M_yield_die_stackOp<id>*. For each existing die in the stack a valid yield must be specified. Nevertheless, the input array should always be of length n even if some dies are not part of the stack. We will use in the example below a yield value of 1.00 to identify the ignored entries.

```
> M_yield_die_stackOp<id> <ad_double[n]>

3D-SIC:                                    2.5D-SIC:                                    5.5D-SIC:
> M_yield_die_stackOp1 0.98 0.98 1.00 1.00  > M_yield_die_stackOp1 0.98 0.98 1.00 1.00  > M_yield_die_stackOp1 1.00 1.00 0.98 0.98
> M_yield_die_stackOp2 0.98 0.98 0.98 1.00  > M_yield_die_stackOp2 0.98 0.98 0.98 1.00  > M_yield_die_stackOp2 0.98 0.98 1.00 1.00
> M_yield_die_stackOp3 0.98 0.98 0.98 0.98  > M_yield_die_stackOp3 0.98 0.98 0.98 0.98  > M_yield_die_stackOp3 0.98 0.98 0.98 0.98
```

For the 3D-SIC, in the first stacking operations only die with labels 1 and 2 are being stacked. Therefore, the tool will ignore any value specified on index 3 and 4 of *MC_stacked_die_yield1*. As the 3D-SIC and 2.5D-SIC have the same dies involved in each stack, their stacked die yield pattern is the similar. However, for the 5.5D-SIC, in the first stacking operation dies with label 3 and 4 are stacked and only the values belonging to these dies are considered. The user is free to specify any yield

values in the range (0,1].

The interconnects, consist similarly as the dies out a pre-bond component and a mid-bond/post-bond/final component. In the pre-bond phase, we refer to the interconnections as TSVs (they are not entirely formed yet) and in the remaining phases, we consider them parts of interconnect. Note that the interconnects could be TSVs + micro-bumps or only micro-bums (in case of F2F stacking). For the pre-bond phase, the keyword *M_yield_TSV* is used to denote the yield in each die. 3D-COSTAR reads and uses all specified values. In case a die does not have TSVs, a yield of 1.0 must be specified.

```
> M_yield_TSV <ad_value[n]>
> M_yield_int_stackOp<id> <ad_double[n-1]>

3D-SIC:                                  2.5D-SIC:                                  5.5D-SIC:
> M_yield_TSV 0.99 0.99 0.99 1.00        > M_yield_TSV 1.00 1.00 1.00 1.00        > M_yield_TSV 1.00 1.00 0.99 1.00
> M_yield_int_stackOp1 0.98 1.00 1.00    > M_yield_int_stackOp1 0.98 1.00 1.00    > M_yield_int_stackOp1 0.98 1.00 1.00
> M_yield_int_stackOp2 0.98 0.98 1.00    > M_yield_int_stackOp2 0.98 0.98 1.00    > M_yield_int_stackOp2 1.00 0.98 1.00
> M_yield_int_stackOp3 0.98 0.98 0.98    > M_yield_int_stackOp3 0.98 0.98 0.98    > M_yield_int_stackOp3 0.98 0.98 0.98
```

The interconnect yields are specified in a similar manner as the stacked die yields. However, there are some slight differences. First, interconnects are labeled automatically internally in 3D-COSTAR according to the stacking order. For example, for the 5.5D-SIC, the first index in the interconnect belongs to the interconnection between dies 3 and 4 as these two dies are stacked first. A second difference is the array length; the length of *M_yield_int_stackOp<id>* is 1 less as the number of interconnects is $n-1$.

There is a cost involved in each 3D stacking operation. This cost can be specified by the keyword *M_cost_stackOp*. This cost should include all costs related to this step, such as aligning, heating and cohesive material cost. Each entry represents the cost related to that stacking operation. As an example, we ascribe a cost of 0.05 for the first stacking operation, 0.15 for the second one, and 0.12 for the final stacking operation. Again the cost can be specified in any unit as desired as long as they are expressed in the same unit as the other cost values.

```
> M_cost_stackOp <ad_double[n-1]>          > M_cost_stackOp 0.05 0.10 0.12
```

D. Packaging

The packaging parameter considers two sets of inputs; its cost and yield. The cost for packaging is denoted by the keyword *P_cost*. A single value of type double either zero or positive is expected here. For example, we can attribute a cost of 1.00 ($ or €) to it.

```
> P_cost <double>                          > P_cost 1.00
```

For each die and interconnect in the stack, a packaging yield can be attributed. The array lengths equal to the number of dies and number of interconnects respectively. For the dies, the values of each index belong to their die label. While for the interconnects this is dependent on the stacking order. The first index, belongs to the interconnect between the two dies specified in *MC_stacking1*, the ith element to the interconnection between the two dies specified at *MC_stacking*$< i >$.

```
> P_yield_die <ad_double[n]>               > P_yield_die 1 0.9 1 0.67
> P_yield_int <ad_double[n-1]>   > P_yield_int 0.89 0.9 0.87
```

E. Test

The inputs from the test class have the prefix T and are grouped into pre-bond, mid-/post-bond, final, test sequence, and probing as depicted in Figure 6. Each of these groups is described next.

Both dies and TSVs may be tested in a pre-bond test prior to stacking. 3D-COSTAR supports a fault coverage related to wafer manufacturing defects (*M_yield_die*) and TSV defects *M_yield_TSV* using the keywords *T_fc_die_prebond* and *T_fc_TSV_prebond* respectively. If a die does not contain TSVs/micro-bumps a fault coverage of zero must be specified. In addition, for each test a cost can be attributed to it using *TC_prebond_die_cost* and *TC_prebond_TSV_cost* for the dies and TSVs respectively.

```
> T_fc_die_prebond <ad_value[n]>      > T_fc_die_prebond 0.9 0.7 0.988 0.80
> T_cost_die_prebond <ad_value[n]>    > T_cost_die_prebond 1.00 0.50 1.40 0.00
> T_fc_TSV_prebond <ad_value[n]>      > T_fc_TSV_prebond 0.9 0.9 0.99 0.00
> T_cost_TSV_prebond <ad_value[n]>    > T_cost_TSV_prebond a 0.99
```

After the pre-bond test, the next opportunity for dies and interconnects to be tested is the mid-/post-bond phase. The defects that can be tested for are either related to stacking defects (*M_yield_die*) or previous defects from the wafer manufacturing that escaped during pre-bond testing (*M_yield_die_stackOp*). In case dies have not been tested in the pre-bond phase, they may be tested now with a higher test quality. Similarly as for *M_yield_die_stackOp*, some dies might not exist for partial stacks. In the examples below the fault coverages for such cases have been set to 0.

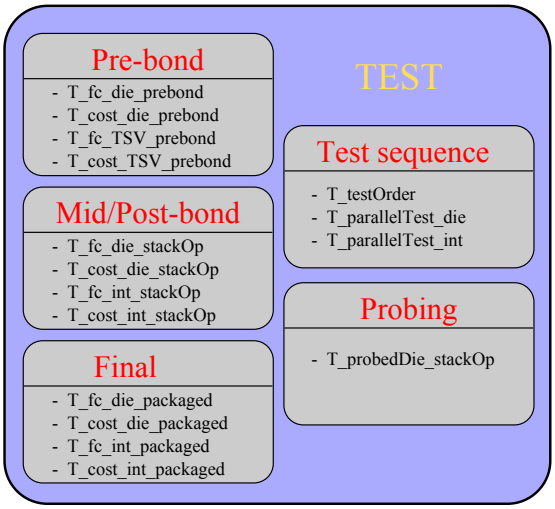

Fig. 6. Test.

```
> T_fc_die_stackOp<id> <ad_double[n]>
> T_cost_die_stackOp<id> <ad_double[n]>

3D-SIC:                                2.5D-SIC:                                5.5D-SIC:
> T_fc_die_stackOp1 0.99 0.99 0.00 0.00 > T_fc_die_stackOp1 0.99 0.99 0.00 0.00  > T_fc_die_stackOp1 0.00 0.00 0.99 0.99
> T_fc_die_stackOp2 0.99 0.99 0.99 0.00 > T_fc_die_stackOp2 0.99 0.99 0.99 0.00  > T_fc_die_stackOp2 0.99 0.99 0.00 0.00
> T_fc_die_stackOp3 0.99 0.99 0.99 0.99 > T_fc_die_stackOp3 0.99 0.99 0.99 0.99  > T_fc_die_stackOp3 0.99 0.99 0.99 0.99

> T_cost_die_stackOp1 1.00 1.00 0.00 0.00
> T_cost_die_stackOp2 1.00 1.00 1.00 0.00
> T_cost_die_stackOp3 1.00 1.00 1.00 1.00
```

In the example above, the test cost has been only provided for the 3D-SIC due to space shortage. Non-existing fault coverages must be set to zero. Note the mid-bond phase is represented by id¡$n-1$ and the post-bond phase id=$n-1$ (see also Figure 3).

Similarly, the keyword *T_fc_int_stackOp<id>* is used to denote the fault coverage of interconnect testing for mid-bond and post-bond tests.

```
> T_fc_int_stackOp<id> <ad_double[n-1]>
> T_cost_int_stackOp<id> <ad_double[n-1]>

3D-SIC:                          2.5D-SIC:                          5.5D-SIC:
> T_fc_int_stackOp1 0.99 0.00 0.00  > T_fc_int_stackOp1 0.99 0.00 0.00  > T_fc_int_stackOp1 0.99 0.00 0.00
> T_fc_int_stackOp2 0.99 0.99 0.00  > T_fc_int_stackOp2 0.99 0.99 0.00  > T_fc_int_stackOp2 0.99 0.99 0.00
> T_fc_int_stackOp3 0.99 0.99 0.99  > T_fc_int_stackOp3 0.99 0.99 0.99  > T_fc_int_stackOp3 0.99 0.99 0.99

> T_cost_int_stackOp1 0.05 0.00 0.00
> T_cost_int_stackOp2 0.05 0.05 0.00
> T_cost_int_stackOp3 0.05 0.05 0.05
```

Again, special note must be taking to correct indexing. The values of *T_fc_int_stackOp1* belong to the testing of interconnects after the first stacking operation (mid-bond tests). The first value after the keyword presents the interconnect between the first two stacked dies etc. For the 3D-SIC, the first number belongs to the interconnects between die 1 and 2, while for the 5.5D-SIC this number the interconnect between die 3 and 4.

Finally, after assembly and packaging a screen test can be applied (final test). Again, fault coverages can be specified for each die and interconnect in the stack. Note that these tests can detect faulty dies/interconnects during all phases, i.e., pre-bond (pre-bond die and TSV yield), mid/post-bond (stacked-die and interconnect yield) and due to packaging. In the example below, all the n dies are tested with a fault coverage of 99.5% and the interconnects with 100% fault coverage at a cost of 1.0$ and 0.1$ respectively. Also in these arrays the die indexes correspond to their label order, and the interconnects based on their stacking position.

```
> T_fc_die_packaged <ad_value[n]>      > T_fc_die_packaged a 0.995
> T_cost_die_packaged <ad_value[n]>    > T_cost_die_packaged a 0.995
> T_int_die_packaged <ad_value[n-1]>   > T_int_die_packaged a 0.1
> T_cost_die_packaged <ad_value[n-1]>  > T_cost_die_packaged a 1.0
```

The next group of parameters are related to the test sequence, i.e., the order in which dies are tested and whether some of

the dies tested in parallel. The keyword *T_testOrder* is used to specify the order of die testing. The keyword takes as argument all existing interconnects and dies of the final stack. Interconnects are prefixed with *i* and dies width *d*.

```
> T_testOrder <die and int id's>      > T_testOrder i1 d4 i2 i3 d1 d2 d3
```

For the given example above and the 5.5D-SIC of Figure 5, first interconnect one is tested (i.e., the interconnnect between dies 3 and 4), subsequently die 4 followed by interconnect 2 (i.e., between die 1 and 2) followed by interconnect 3 (i.e., between die 1 and 3) followed by the dies 1, 2 and 3. This order is used in both the mid/post-bond and final test phase. In case a partial stack does not contain certain dies or interconnects, they will be ignored by 3D-COSTAR. For example, in the first stacking operation of the 5.5D-SIC only dies 3 and 4 and interconnect 1 are part of the stack. In this case, the test order will be interconnect 1, die 4 followed by die 3.

3D-COSTAR also supports testing of ICs simultaneously. This does not refer to scanning multiple dies using huge concatenated chains. Rather, it means that individual dies could be tested simultaneously, such as memories. Imagine for example that dies 3 and 4 of the 5.5D-SIC are memory dies each with their own BIST engine. They can be tested in parallel. Dies and interconnects that are tested in parallel have to be provided in sets separated by comma's.

```
> T_testOrder <die and int id's>        > T_testOrder i1 d4 i2 i3 d1 d2 d3
> T_parallelTest_die <sets of dies>     > T_parallelTest_die {3,4}
> T_parallelTest_int <sets of int>      > T_parallelTest_int {1,3}{2}
```

3D-COSTAR will automatically search the die with the largest test cost and only attribute that cost. For example, if the test cost of die 4 is larger than die 3 in the example above in keyword *T_parallelTest_die*, it will be equivalent as if die 3 is tested with zero cost. Multiple sets can be provided by having The argument of the keywords *T_parallelTest_die* and *T_parallelTest_int* can be left empty in case each die or interconnect test is performed sequentially.

The last set of test parameters are related to probing. The tool automatically keeps track of the number of times a die is probed during testing. In the pre-bond phase, the tool automatically tracks if dies are tested on not based on the fault coverages for dies and interconnects. In the mid/post-bond phase, for each partial stack the user has to specify a single die which is probed as given by the following example.

```
> T_probedDie_stackOp<id> <i_value>
                                        > T_probedDie_stackOp1 3
                                        > T_probedDie_stackOp2 1
                                        > T_probedDie_stackOp3 1
```

For example, for the 5.5D-SIC of Figure 5 the values in the example denote that die 3 is probed during the test after the first stacking operation; similarly, during the tests after the second and third stacking operation die 1 is probed. After packaging, ICs are typically tested through the package interface. Probing does not apply in this case. There is no support for probe yield in the current version.

F. Logistics

The parameters in the logistics class start with the prefix L. In order to understand this class we first explain Figure 7. There are 11 arrows that represent movement of tiers between five possible companies. The companies are the following: in Design company, Wafer fab, Test house, Stacking fab, and Packaging. In the worst case, logistic costs can to be attributed the worst case scenario in which each activity (manufacturing, test, 3D stacking and packaging) is separated from another. Each letter next to the arrow depicts the cost to move from one company to another one. It covers the following cost movements:

- L_{wd}: Cost between the *Design company* and *Wafer fab*
- L_{wt}: Cost of moving tiers from *Wafer fab* to *Test house* (e.g., needed for pre-bond test)
- $L_{ts,d}$: Cost of moving tiers from *Test house* to *Stacking fab* (e.g., to transport pre-bond wafers or dies)
- $L_{ts,s}$: Cost of moving tiers from *Test house* to *Stacking fab* (e.g., to transport partial stacks)
- L_{st}: Cost of moving tiers from *Stacking fab* to *Test house* (e.g., for mid-bond or post-bond tests)
- L_{tp}: Cost of moving tiers from *Test house* to *Packaging fab* (e.g., after post-bond test)
- L_{pt}: Cost of moving tiers from *Packaging fab* to *Test house* (e.g., for final/packaged test)
- L_{td}: Cost of moving tiers from *Test house* to the *Design company* (e.g., after final test)
- L_{ws}: Cost of moving tiers from *Wafer fab* to *Stacking fab* (e.g., to perform stacking in case no pre-bond test is used)
- L_{sp}: Cost of moving tiers from *Stacking fab* to *Packaging fab* (e.g., no post-bond test)
- L_{pd}: Cost of moving tiers from *Packaging fab* to *Design* (e.g., no final test)
- L_{ss}: Cost of moving tiers between 2 different *Stacking fabs* or possibly within the same 3D fab. (e.g. no mid-bond test)

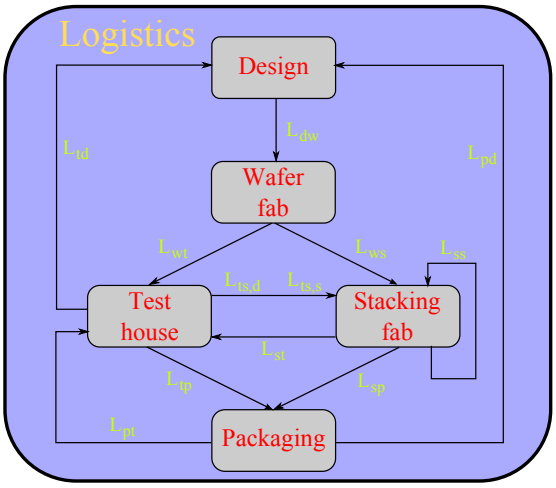

Fig. 7. Logistics Input.

Depending on the test flow, some of the costs are not applicable. For example, in case a pre-bond test is skipped (arrow between wafer fab and stacking fab), the cost associated with the arrow wafer fab to test house for that particular die is ignored. Furthermore, for some companies some of these values are zero. For example, if a company performs both the testing and stacking in-house, costs associated with arrows $L_{ts,s}$ and L_{st} could be set to zero. In the same order as the list presented above, the following keywords represent these cost in 3D-COSTAR.

```
> L_cost_company2waferfab        <d_value>
> L_cost_waferfab2tester       <ad_value[n]>
> L_cost_tester2fab3D_wafer    <ad_value[n]>
> L_cost_tester2fab3D_stack  <ad_value[n-2]>
> L_cost_fab3D2tester        <ad_value[n-1]>
> L_cost_tester2packaging        <d_value>
> L_cost_packaging2tester        <d_value>
> L_cost_tester2company          <d_value>
> L_cost_waferfab2fab3D        <ad_value[n]>
> L_cost_fab3D2packaging         <d_value>
> L_cost_packaging2company       <d_value>
> L_cost_fab3D2fab3D         <ad_value[n-2]>
```

All the cost in this class have to be specified per die, even if wafers are transported. In case (partial) stacks are transported, the logistics cost will depend on the number of transports dies/wafers from by the bottom layer of the partial stack and the particular cost values in *L_cost_**.

Note that the arrays have different lengths based on whether dies or stacks are transported. For example, for the keyword *L_cost_company2waferfab* we assume a single double value as an input. As there are n wafers in the pre-bond phase, each element can have a different cost in *L_cost_waferfab2tester*, *L_cost_waferfab2fab3D* and *L_cost_tester2fab3D_wafer*. The tool requires full arrays of length n to be specified and filters the unused values, based on whether a die/wafer is tested or not, out. It achieves this by looking at the fault coverage of the dies and TSVs.

The length of the array *L_cost_tester2fab3D_stack* equals $n-2$, and considers the movement of partial stacks. The length of this array does not equal $n-1$, as the last stack has to be moved to the packaging house (*L_cost_tester2packaging*). This also applies to *L_cost_fab3D2fab3D*.

The last argument in this class is *L_dieBased*. This arguments specifies whether tiers are transported per die (denoted by *y*) or wafer (*n*) and only is valid for the cases where applicable. For example, a D2D stack can only be transported by dies as the wafers are already sliced for the stacking operation. However, the movements from the test house to stacking fab for pre-bond wafer is affected by this parameter in case the dies enter the stack as individual dies.

```
> L_dieBased  <y/n>        > L_dieBased y
```

There is currently no support for logistic yields.

G. Sweep Analysis

This section describes how to perform sweep analyses. The global keyword here that controls the number of variables*Sweep*. This variable takes two arguments. The argument -t specifies the total number of sweep variables. The second argument, -dim,

specifies the number of dimensions this sweeps have. An example will make the difference between the more clear. Lets consider for now three variables using two dimensions.

```
> MSens -t <i_value> -dim <i_value>          > MSens -t 5 -d 2
```

Some variables are correlated with each other (e.g., test cost and test coverage) and can be combined in a single dimension. The example below gives more details. The tool expects for each variable the dimension (-dim) it belongs too, which keyword is being looped (keyword), the begin (-b) and end (-e) value of this variable and its step size (-s).

```
MSens<i_value> -dim <i_value>  keyword -b <d_value>   -e <d_value>    -s <d_value>

MSens1 -dim 1 T_fc_die_prebond[1]            -b 0   -e 1    -s 0.1
MSens2 -dim 1 T_cost_die_prebond[1]          -b 0   -e 5    -s 0.50
MSens3 -dim 2 P_cost                         -b 1.5 -e 2.25 -s 0.125
MSens4 -dim 2 M_yield_int_stackOp[1][1]      -b 0.6 -e 0.9  -s 0.05
MSens5 -dim 2 M_yield_die_stackOp[1][3]      -b 0.6 -e 0.9  -s 0.05
```

In the example above, the keywords *T_fc_die_prebond[1]* and *T_cost_die_prebond[1]* are sweeped along the same dimension, i.e., their values are applied simultaneously during experiments.

The total number of sweeps per dimension equals $\lfloor \frac{e+s-b}{s} \rfloor$. In this example, a total of 11 (for the first dim) * 7 (for the second dim) = 77 are performed. There are basically three types of keywords. The first group consist of single entries, such as *P_cost*. The second group consists of one-dimensional arrays, such as *T_cost_die_prebond*. In case the array is related to a die, a valid die label between [ ] brackets must be placed, i.e., it must exist in *M_dieLabelList*. In case the array is related to interconnects, a valid interconnect id must be placed (i.e., a value between 1 and (*M_dieCount*-1). The last group consists of two-dimensional variables such as *M_yield_die_stackOp* and *M_yield_int_stackOp*. In these types of variables, two index values must be placed. The first index identifies the stacking operation (a number between 1 and (*M_dieCount*-1), while the second index identifies the interconnect and/or die label. The tool validates all entries, but does currently not check if the same variable is entered multiple times. In addition, sensitivity analysis will only be printed when one or two dimensions are used.

IV. Input Validation

A. Direct specified inputs

Table I shows the boundary values of the input variables. For each keyword (shown in the first column), the min and max values are provided (second and third column). Additional comments that are of value are provided in the last column.

B. Limitations and Considerations

There are some limitations and considerations in using the inputs:

- The minimum and maximum values where ∞ is involved in Table I are actually limited by the data size.
- Some keywords require arrays as arguments. 3D-COSTAR does not check the length of these arrays. In case too few elements are specified unexpected behavior can occur, such as a program crash.
- 3D-COSTAR does not check the inputs in the order as specified in the order in the table. The keywords may be placed in any order. In addition, keywords that are unnecessary are not checked.
- Insufficient error printing in case die labels are wrong. Only the keyword will be printed, but not the index location.
- All arrays must be fully specified, even if they are not used. This is necessary for correct indexing. For example, inputs costs related to die and wafer transport must always be fully specified, even if some of the arrow in Figure 7 are inapplicable.
- The values that are specified in sweep variables are not checked against the min and max values of Table I. However, if the sweep variable is part of a single or two dimensional array the indexes it has are checked.
- Lines that contain random data are ignored and not considered.
- If a keyword is specified multiple time, only the first encounter will be considered. No error will be printed.

V. Outputs

3D-COSTAR generates in general three output files:

1) Main output file, this file contains for example:
   - the cost per shipped IC.
   - the test quality per shipped IC in defects per million.
   - the cost break down.
2) Debug file
3) Output files related to sensitivity if applicable

TABLE I
RESTRICTIONS ON INPUTS AND TYPE CHECKING

Keyword	min	max	Additional Restrictions/Comments
D_nreCost	>0.0	∞	—
M_productionVolume	1	∞	—
M_dieCount	1	∞	—
M_dieLabelList	-∞	∞	Each entry must have a unique value
M_cost_wafer	0.0	∞	—
M_yield_die	>0.0	1	Applies only for entries where M_enableArea is 0
M_diesPerWafer	1	∞	Applies only for entries where M_enableArea is 0
M_enableArea	0	1	—
M_radiusEff	>0.0	∞	Applies only for entries where M_enableArea is 1
M_dieSize_width	>0.0	∞	Applies only for entries where M_enableArea is 1
M_dieSize_length	>0.0	∞	Applies only for entries where M_enableArea is 1
M_defectDensity	0.0	∞	Applies only for entries where M_enableArea is 1
M_clustering	>0.0	∞	Applies only for entries where M_enableArea is 1
M_yield_TSV	>0.0	1	—
M_testEquation	0.0	1	The argument must be either 0 or 1
M_stackOp<i> j-k | ...			
i	1	M_dieCount -1	
j,k	-∞	∞	Both specified dies j and k must exist in M_dieLabeList and j != k
M_yield_die_stackOp<i>	> 0.0	∞	$1 \leq i \leq$ M_dieCount -1
M_yield_int_stackOp<i>	> 0.0	∞	$1 \leq i \leq$ M_dieCount -1
M_cost_stackOp	>0.0	∞	—
P_cost	>0.0	∞	—
P_yield_die	>0.0	1	—
P_yield_int	>0.0	1	—
T_fc_die_prebond	0.0	1	—
T_cost_die_prebond	0.0	∞	Costs are ignored when fault coverage is zero
T_fc_TSV_prebond	0.0	1	—
T_cost_TSV_prebond	0.0	∞	Costs are ignored when fault coverage is zero
T_fc_die_stackOp<i>	0.0	1	—
T_cost_die_stackOp<i>	0.0	∞	Costs are ignored when fault coverage is zero
T_fc_int_stackOp<i>	0.0	1	—
T_cost_int_stackOp<i>	0.0	∞	Costs are ignored when fault coverage is zero
T_fc_die_packaged	0.0	1	—
T_cost_die_packaged	0.0	∞	Costs are ignored when fault coverage is zero
T_fc_int_packaged	0.0	1	—
T_cost_int_packaged	0.0	∞	Costs are ignored when fault coverage is zero
T_test_order	n.a.	n.a.	Each die or interconnect must have a valid id and all dies and interconnects must be specified Each interconnect index should satisfy $1 \leq i \leq$ M_dieCount -1 The die entries should be in M_dieLabelList
T_parallelTest_die	n.a.	n.a.	Dies must be defined in M_dieLabelList and can exist in 1 set only
T_parallelTest_int	n.a.	n.a.	Interconnects must be valid and can exist in 1 set only
T_probeDie_stackOp<i>	n.a.	n.a.	The probed die must be defined in M_dieLabelList, this keyword applies only for mid-bond and post-bond phases.
L_dieBased	n.a.	n.a.	Must have as argument either y or n
*L_**	0.0	∞	All the other keywords from the logistics class have the same min and max boundary

A. Main Output File

The output file consists of several parts. It first reports the accounting of the number of probes. Here, the tool generates a small report that shows which ICs were probed and how frequently they were probed. Thereafter, the output report shows the overall design cost and the design cost per shipped IC. After that, manufacturing information is showed both for die and interconnects. Similarly, costs are shown for test, packaging and logistics cost. Finally, overall cost per shipped IC, test escapes and cost break down are provided. The main output file is generated in all case (even if sensitivity analysis are performed). An example of such a file is provided on the web-page.

B. Debug File

The debug file that is generated along with the default output file is called dump_stack.txt and contains detailed information. Debug file are only generated if requested by the user. The keyword *verbose* takes as argument y (yes) or n (no) and based on the value debug files are generated or not.

```
> verbose <y/n>                    > verbose y
```

The content of a debug file can be divided in two parts:

- The first part is related to the inputs. All inputs are printed in a different from to correct and test if 3D-COSTAR understands the inputs correctly. The parameters that are related to this part are depicted in Figure 8.
- The second part is related to the outputs and gives an in-depth analysis of the cost breakdown. It shows for all phases precisely how many dies are required for the particular stack, how often dies are tested in each phase, how many dies are transported etc.

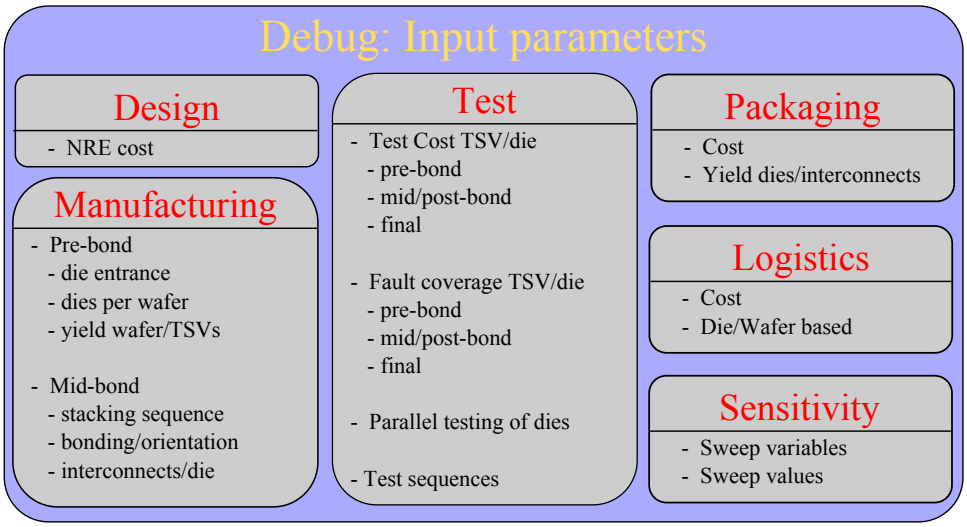

Fig. 8. Debug of Inputs.

C. Sensitivity Output File

Finally, the last outputs are related to sensitivity analysis. 3D-COSTARwill generate for each input combination of the sweep variables an output file where its name is a string concatenation of the the sweep variables and their values. The tool creates all sweep experiments (main files and optionally debug files) in a sub-folder *sweep* for each input combination. In addition, if the number of dimensions is one or two, a comma-separated values (CSV) file (sweep_summary.txt) will be created that contains a one-dimensional or two-dimensional array respectively with the results of the sensitivity analysis. In case the dimension equals 2, the file will contain an array in which the row elements belong to the variables of the first dimensions, while the columns to the second dimension. The sweep_summary.txt summarizes all total cost outputs, test escape rates, and cost breakdown.

REFERENCES

[1] E. J. Marinissen and Y. Zorian, "Testing 3D Chips Containing Through-Silicon Vias," in *Test Conference, 2009. ITC 2009. International*, nov. 2009, pp. 1 –11.

[2] V. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*, ser. Frontiers in Electronic Testing. Springer, 2000. [Online]. Available: http://books.google.nl/books?id=CgnLg99GumMC

[3] D. de Vries, "Investigation of Gross Die per Wafer Formulas," *Semiconductor Manufacturing, IEEE Transactions on*, vol. 18, no. 1, pp. 136 – 139, feb. 2005.

[4] J. De Sousa and V. Agrawal, "Reducing the complexity of defect level modeling using the clustering effect," in *DATE*, 2000, pp. 640–644.