Minimizing Test Time through Test Flow Optimization in 3D-SICs

by

Assmitra Dash

LIU-IDA/LITH-EX-A--13/066--SE

2013-11-26
Final Thesis

Minimizing Test Time through Test Flow Optimization in 3D-SICs

by

Assmitra Dash

LIU-IDA/LITH-EX-A--13/066--SE

2013-11-26

Supervisor: Breeta Sengupta
ESLAB, IDA, Linköpings universitet

Examiner: Zebo Peng
ESLAB, IDA, Linköpings universitet
Abstract

3D stacked ICs (3D-SICs) with multiple dies interconnected by through-silicon-vias (TSVs) are considered as a technology driver and proven to have overwhelming advantages over traditional ICs with a single die in a package in terms of performance, power consumption and silicon overhead. However, these “super chips” bring new challenges to the process of IC manufacturing; among which, testing 3D-SICs is the major and most complex issue to deal with. In traditional ICs, tests can usually be performed at two stages (test instances), namely: a wafer sort and a package test. Whereas for 3D-SICs, tests can be performed after each stacking event where a new die is stacked over a partial stack. This expands the set of available test instances. A combination of selected test instances where a test is performed (active test instance) is known as a test flow. Test time is a major contributor to the total test cost. Test time changes with the selected test flow. Therefore, choosing a cost effective test flow which will minimizes the test time is absolutely essential.

This thesis focuses on finding an optimal test flow which minimizes the test time for a given 3D-SIC. A mathematical model has been developed to evaluate the test time of any test flow. Then a heuristic has been proposed for finding a near optimal test flow which minimizes the test time. The performance of this approach in terms of computation time and efficiency has been compared against the minimum test time obtained by exhaustive search. The heuristic gives good results compared to exhaustive search with much lesser computation time.
Acknowledgements

My first gratitude goes to Dr. Erik Larsson for allowing me to start this master thesis work under his patient guidance.

I am grateful to my examiner Dr. Zebo Peng for generous support and guidance without whom I would not be able to finish with this thesis work.

I would like to thank my supervisor Breeta Sengupta for all of her help and support throughout my master thesis work (especially pulling me out of dead ends).

I also would like to thank all the people who have helped me with administration and arrangements involved in the thesis work, as well as my opponent Yi-Ching Chen, for taking her time to read, comment and discuss on my work.
# Contents

Abbreviations xi

## 1 Introduction 1
1.1 Motivation and Background ................................. 1
1.2 3D-SIC Production and Testing ............................. 4
1.3 Goal and Contribution ......................................... 7
1.4 Thesis Organisation ........................................... 7

## 2 Review of Previous Work 9

## 3 Problem Formulation and Test Time Calculation 11
3.1 Problem Description ........................................... 11
3.2 Cost and Test Time Calculation .............................. 13
3.3 Problem Definition ............................................ 20

## 4 Approach 21
4.1 Test Instance Matrix Model ................................. 21
4.2 Heuristic ..................................................... 23
4.2.1 Complexity Evaluation .................................... 24

## 5 Results and Analysis 27
5.1 Experimental Setup .......................................... 27
5.1.1 Exhaustive Search ......................................... 29
5.2 Test Flow Comparisons ....................................... 30
5.3 Execution Time ............................................... 34

## 6 Conclusion 37
6.1 Future Work .................................................. 38

## A Test Instance Matrix Model 41
List of Figures

1.1 Comparison of 2D IC with 3D-SIC ........................................ 2
1.2 Recent research trend on 3D-SIC related challenges ................. 3
1.3 (a) Test flow for 2D IC, (b) Test flow for 3D-SIC .................... 5

3.1 Attributes of overall low yield due to compound imperfect yield ... 12
3.2 Illustration of the optimal test flow after comparing test time of all valid test flows .................................................. 19
3.3 Illustration of the optimal test flow after comparing silicon investments of all valid test flows ........................................... 20

4.1 Test instance matrix (TI matrix) ........................................ 21
4.2 Transformation and mapping of a test flow into TI matrix (for n=3) . 22
4.3 Algorithm for Heuristic ................................................... 25
4.4 Flowchart for Heuristic .................................................... 26

5.1 Algorithm for Exhaustive search method ............................... 30
5.2 (a) Test flow from exhaustive and heuristic (b) Test flow from heuristic (in details) .................................................... 33
5.3 Relative comparison of average test time overhead (w.r.t exhaustive) 34
5.4 Execution time comparison (Exhaustive vs. Heuristic) ................. 36

A.1 Test instance matrix ....................................................... 41
A.2 Initial TI matrix (a) and Detailed TI matrix (b) ....................... 42
A.3 TI Matrix with n=2 ....................................................... 43
A.4 Task graph for test instances ........................................... 44
A.5 Component matrix ....................................................... 44
A.6 Test summary matrix ................................................... 45
A.7 Test forward matrix .................................................... 46
A.8 Testing matrix ......................................................... 46
A.9 Updated test summary matrix ......................................... 47
A.10 Output matrix ....................................................... 48
A.11 Quantity matrix .................................................... 49
A.12 Test time calculation .................................................. 51
List of Tables

3.1 Valid test flows ........................................... 14
3.2 Yield values for components and steps ....................... 15
3.3 Test times for components .................................. 16
3.4 Values for input parameters ................................ 18

5.1 Properties of dies in scenario 1 .............................. 28
5.2 Properties of dies in scenario 2 .............................. 28
5.3 Properties of dies in scenario 3 .............................. 29
5.4 Properties of dies in scenario 4 .............................. 29
5.5 Test time analysis table for scenario 1 ..................... 31
5.6 Test time analysis table for scenario 2 ..................... 31
5.7 Test time analysis table for scenario 3 ..................... 31
5.8 Test time analysis table for scenario 4 ..................... 32
5.9 Average execution time for varying stack size ............. 35
Abbreviations

\begin{itemize}
  \item \textit{B2B} Back to Back
  \item \textit{CMOS} Complementary Metal Oxide Semiconductor
  \item \textit{CUT} Circuit Under Test
  \item \textit{D2D} Die to Die
  \item \textit{D2W} Die to Wafer
  \item \textit{DfT} Design for Testability
  \item \textit{F2B} Face to Back
  \item \textit{F2F} Face to Face
  \item \textit{IC} Integrated Circuit
  \item \textit{KGD} Known Good Die
  \item \textit{KGP} Known Good Package
  \item \textit{KGS} Known Good Stack
  \item \textit{MCP} Multi Chip Package
  \item \textit{PCB} Printed Circuit Board
  \item \textit{PoP} Package on Package
  \item \textit{SIC} Stacked Integrated Circuit
  \item \textit{SiP} System in Package
  \item \textit{SoC} System on Chip
  \item \textit{TAT} Test Application Time
  \item \textit{TF} Test Flow
  \item \textit{TI} Test Instance
  \item \textit{TSV} Through Silicon Via
  \item \textit{ttu} Test Time Unit
  \item \textit{W2W} Wafer to Wafer
\end{itemize}
Chapter 1

Introduction

1.1 Motivation and Background

Moore’s Law [1] has acted as the main driving force for all major breakthroughs in circuit manufacturing industry since late 1960s. Integrated Circuit (IC) design has been influenced by the trend of increasing transistor counts in the circuit to make it more powerful and computational capable by operating at higher speed. As a result, the fabrication processes have been refined to highest possible levels in order to increase the transistor counts by reducing their sizes. But during the last decade the old law has started to fade away as the transistors size approaches to atomic level barring the growth of transistor density per square centimeter of silicon area. The ramifications of Moore’s law such as “More Moore” [2] has tried to supplement the continuation of the classic law by improving complementary metal oxide semiconductor (CMOS) performance with the help of materials for high speed channels, low access resistance and high dielectric constant gate stacks. But its ability for scaling has not improved much. So the researchers started to look at a different trend for designing future ICs rather than relying on scaling theory. This trend is commonly known as “More than Moore” [3], where non-digital components and functionalities (like clock, power, I/O etc) are to be migrated on the same package, hence increasing the component density per unit area. Irrespective of having limitation such as wiring crisis [4], it has gained popularity among research communities as well as industrial implementations.

Initially, the primary focus of this new trend was on exploiting the two dimensional space i.e. integrating multiple chips into one package or board. Multiple cores in a single chip (System-On-Chip: SoC), multiple dies in a single planar package (Multi-Chip Package: MCP) and multiple ICs on a Printed Circuit Board (PCB) are few examples of this evolution. Later, the third (vertical) dimension was also included to accommodate the increasing transistor density with reduced form factor of the packages. For example, System-in-Package (SiP), where multiple dies are stacked within a single package and interconnected by wire bonds to the substrate. Package-on-Package (PoP), in which multiple packaged chips are stacked is another example. More recently, the research
CHAPTER 1. INTRODUCTION

interests have been concentrated on the latest evolution of 3D stacked IC (3D-SIC), where “a single package containing a vertical stack of dies which are interconnected by means of Through-Silicon Vias (TSVs)” [5]. TSVs are conducting wires vertically etched through the silicon substrate interconnecting the dies of a 3D-SIC. The comparison between a 3D-SIC based on TSV interconnects and a traditional IC is illustrated in figure 1.1.

These TSV based 3D-SICs are termed as “super chips” as they hold numerous advantages over the traditional ICs [6], such as,

- Improved package volume density and compact form factor
- Reduced power consumption
- Reduced wire lengths. Hence reduction in interconnection delays
- High bandwidth communication as TSVs cross dies along the surface of the chip
- Heterogeneous integration facility
- Improved performance and reduced cost

![Figure 1.1: Comparison of 2D IC with 3D-SIC](image)

The manufacturing process of traditional ICs is carried out in two steps: fabrication of dies and assembly/packaging of dies into packages. This fixed manufacturing procedure facilitates to explore alternatives for more efficient design process and test architectures/techniques. As a result, we see a saturated and well established repository of solutions for traditional ICs. 3D-SICs differ from traditional ICs in the processes of design automation, manufacturing and testing. In 3D-SICs, stacks of dies are produced by stacking multiple dies on top of other dies or partial stacks. These stacking steps introduce alteration to 3D-SIC manufacturing process with inclusion of wafer thinning, back-side grinding, accurate alignment, TSV developments, etc. These alterations brings new types of faults which are yet to be addressed with appropriate work around. Most of these faults/defects stay undetected unless comprehensive tests are performed. So, 3D-SICs are complex to design and manufacture, but they are even
more complex and challenging to test [7]. Figure 1.2 acknowledges this highlighting 3D-SIC related test issues need more research attention.

![Figure 1.2: Recent research trend on 3D-SIC related challenges](image)

As untested products without sufficient quality assurance shall not be shipped out as they can hamper the customer satisfaction and industrial reputation. So, some tests (at least a final test on packages) are mandatory. 3D-SICs consist of many components (dies, TSVs) and are produced after multiple stacking steps. Any defect in either component or manufacturing step may damage the whole stack of dies. If these defects are detected at very late stage of production, then large amount of non-faulty components may waste due to a single defective component/step. This results in very low production yield. Due to low overall production yield, large amount of products need to be manufactured and tested for matching output demands. Which requires large test investments. So, conducting tests at very last stage are not advised as it requires larger test investments and results in higher amount of thrown away silicon. In order to minimize this thrown away silicon amount, frequent tests must be conducted to allow only non-faulty components to be forwarded in further manufacturing steps. To test all components in every available stage where a test can be performed (test instance), large amount of test investments are required. Overall production cost is a trade off between silicon (least/late test scenario) and test investments (frequent tests). Test investments are major contributors to overall production cost [8]. So, the research trends and industrial practices try to minimize the test investments for lowering overall production cost.

If we consider that adequate equipments and techniques are available for testing, then the test investment (test cost) is regarded as the amount of time spent during tests (test time). If a test is performed (by activating a test instance), it consumes specific amount of test time depending on the quantity of tested components and required test time for each component (which is being tested). So, different combinations of ac-
tive test instances (test flow: TF) differ in overall test time consumptions. In order to minimize overall test investments or test time consumptions, the optimal TF must be followed. The optimal TF is an efficient TF, which consumes the minimum test time by selectively activating appropriate test instances among all available alternatives.

The optimal TF can be obtained in two successive steps. In the first step, test time consumptions must be calculated for all available/possible TFs. In the second step, choose the TF which has lowest test time by comparing it with all participating TFs. This two step approach is followed and implemented in this thesis for obtaining the optimal TF (for any given 3D-SIC). For the first step, we develop a mathematical model for calculating test time consumptions of all listed TFs. For the second step, we propose a heuristic for finding a nearly optimal TF in an efficient and quick way. We evaluate the efficiency/optimality of our proposed heuristic by comparing the results with exhaustive search (which always finds the optimal TF). We find our heuristic is able to produce a nearly optimal TF which has small overhead compared to the exhaustive search, but it competes fairly (in finding a TF which minimizes the test time to a great extent) with the exhaustive search for any given 3D-SIC.

1.2 3D-SIC Production and Testing

In this section, we will introduce 3D-SIC briefly. Then we will discuss about categorization of 3D-SIC from manufacturing point of view. At last, we will address the complexity in scheduling appropriate tests and the lack of design for testability (DfT) solutions for 3D-SICs.

3D-SIC brings the era of “super chips”, where a package contains a stack of dies bonded by TSV interconnections and provides superior performance as well as cost efficiency when compared to traditional ICs. In a 3D-SIC, each die can be a memory element or a single/multi core processing unit. A die in 3D-SIC may dedicated to performing tasks related to one logic function or it may act as a shared resource (together with other dies) in order to contribute either in faster processing or in fault tolerance. A 3D-SIC that contains the same circuitry at every dies is called homogeneous integrated 3D-SIC. The multi-layered memory chips, which are already in production could be seen as an example [9]. But the semiconductor industry is headed towards further exploitation of the benefits provided by 3D-SICs with heterogeneous integration. In a 3D-SIC under heterogeneous integration, more than one type of circuit/die is stacked together with other types. The heterogeneous 3D-SIC gives the true advantage of integration of a complete SoC in a single package. This drives the research interest for more complex products in 3D-SIC, such as 3D Network-on-Chip (NOC) [10], 3D memory-on-processor [11] and 3D FPGA [12].

For the production of 3D-SICs, more than one die needs to be stacked together. The stacking process can be done in three ways: Wafer-to-Wafer (W2W), Die-to-Wafer (D2W) and Die-to-Die (D2D). In W2W stacking, complete wafers are stacked over one
another. In D2W and D2D stacking process, individual dies are placed over wafers and dies respectively. Each of the above mentioned process has its own benefits and drawbacks from the manufacturing point of view. W2W exhibits superior throughput and economic manufacturing cost but extremely poor on overall production yield. While D2D has improved production yield, it is extremely slow in throughput. This is because the rate of manufacturing is limited by the speed of the mechanical robot hand responsible for pick-and-placement of individual dies with desired alignment accuracy. The throughput and production yield of D2W comes in between of W2W and D2D. In another way of categorizing, the orientation of the stacked chips also divides the 3D-SIC into three categories: face-to-face (F2F), face-to-back (F2B) and back-to-back (B2B). The face of a die is the side of the transistors and metal interconnect layers and the back is the silicon substrate layer. In F2F face of one die will be bonded with face of the other die directly with interconnects. Similarly, in B2B back sides of both dies will be thinned and then bonded together. Both F2F and B2B lacks scalability and they are limited up to only two dies. But in F2B, the back side of top die will be thinned and bonded with the face of the bottom die. Hence F2B bonding is scalable to stack of more than two dies [8]. For the rest of this report we will consider F2B bonding only as it supports dies stack of any size.

Figure 1.3: (a) Test flow for 2D IC, (b) Test flow for 3D-SIC

As the industry is able to design and manufacture 3D-SICs, now testing issues
CHAPTER 1. INTRODUCTION

must be addressed with importance and precision. The 3D-SIC testing issues must
be treated from the early stage of manufacturing procedure. Because, a component
or circuit-under-test (CUT) can not be tested at any time stamp during the manufac-
turing process. A test can be conducted by activating a test instance only before or
after a fabrication event. So we can say, testing means following a TF where a TF is
a schedule of selective tests or combination of active test instances. A traditional
IC manufacturing process has two test instances: a wafer sort after wafer level fabri-
cation and a package test after the dies are packaged. This is illustrated in figure 1.3
(a), where white and grey boxes indicates the manufacturing events and test instances
respectively. So deriving TFs from two available test instances will result in four com-
bined test instances all total: no test, only wafer sort, only package test and a wafer sort followed
by a package test. As the number of TFs are fixed, it is easier to optimize and imple-
ment techniques that enhance production throughput as well as lower the production
cost. As 3D-SICs introduce significant alteration to the manufacturing process through
inclusion of stacking and bonding events, they have more test instances compared to
traditional ICs. Each stacking/bonding event introduces two additional test instances:
pre-bond test and post-bond test. The available components for a pre-bond test are the
dies (which are newly introduced to the stack). The post-bond test can be performed
on a partial/complete stack where individual/multiple/all dies and TSV layers will be
tested. The possible test instances are shown in figure 1.3(b). Tests need be performed
on partial stacks and untested dies to ensure that only known good dies (KGDs) and
known good stacks (KGSs) are forwarded to further manufacturing steps for achieving
higher production yield. So, for a 3D-SIC of size \( n \), there are \( n \) wafer sort instances, \( n-1 \)
intermediate test on partial stacks and one final package test, all total \( 2n \) test instances.

The test cost is the cost involved in testing an IC which is largely regarded as the
combination of test time consumed and amount of CUTs during tests or by following
a specific TF. TFs with least or most active test instances result in huge amount of sil-
icon (to be thrown away) or large test investments respectively. An efficient TF which
balances this tradeoff is absolutely essential. Finding an efficient and preferably the
optimal TF from \( 2^{2n} \) alternatives is not easy. We neither have established methods nor
enough investments for evaluating each TF to get the optimal one. From past research,
a few cost models have been proposed for evaluating a given TF. But, they produce
results in terms of deciding either following the given TF is cost effective or not. As
these solutions are based on selectively chosen manufacturing yield values from a lim-
ited range, the decision on following particular TF may change with respect to change
in yield values. Few cost models are able to calculate test cost efficiently, but only take
static/fixed TF as input and lacks scalability and adaption to change in situation/inputs.
Few proposals compare a handful of TFs and propose certain directives for reducing
test time and corresponding share of overall cost. But, those directives are intolerant to
variation in input parameters or considered situations. So, the current research searches
1.3 Goal and Contribution

The goal of this thesis is to find the optimal TF which minimizes the test time consumption for TSV based 3D-SICs. For this, we calculate the test time for any selected TF under a given 3D-SIC. Then we find out the optimal TF with minimum test time by comparing all considered TFs. For test time calculation, we use a Test Instance (TI) matrix model which is built on the base of quantity of tested CUTs and test time required for testing each CUT [13]. To obtain an optimized TF in a quick way, we propose a heuristic. The heuristic which uses the TI matrix model is able to find a near optimal TF which is fairly similar to the optimal TF with small test time overhead.

1.4 Thesis Organisation

The rest of the thesis is organized as follows. Chapter 2 has highlights of the prior work, where the test challenges for 3D-SIC and test optimizing methods were discussed. Chapter 3 introduces the problem in terms of necessity of finding an optimized TF with minimum test time investment. chapter 4 details the approach for calculating test time consumption of any given TF (TI matrix) and obtaining an optimal TF based on calculated test time (Heuristic approach). Chapter 5 discusses the experimental results highlighting the efficiency of the proposed heuristic method. Finally, Chapter 6 concludes this thesis.
Chapter 2

Review of Previous Work

In this section we will first discuss previous works addressing challenges for obtaining DfT solutions in 3D-SICs. Then we will discuss some of the established cost models followed by the ideas for minimizing test cost. Further, we will include the impact of test flows on overall test cost of 3D-SICs and state the necessity for searching an optimal test flow. Finally we will conclude with a comparison among approaches that help in solving the optimization problem.

In [7], Lee et al. discussed the test challenges for 3D-SICs throughout the manufacturing process till operating and testing stages. The authors described the challenges for 3D-SICs in terms of manufacturing, design automation and testing specific areas and highlighting the fact that in 3D-SIC manufacturing, the extra steps such as thinning, aligning and stacking will introduce new defect types. The unpredictable thermal behavior, lack of proven DfT solutions, test architectures, test scheduling schemes and limited physical access for wafer testing make 3D-SIC testing more complex which induced interest for further research. As a major portion of IC cost is test cost, economics of test must be considered at the time of 3D-SIC design. This encourages to focus only on test cost while building a cost model. Also the overall cost of 3D-SICs could be minimized to a greater extend by optimizing test cost (alone). In [8], the authors introduced modular testing which allows to decide where in the test flow a certain module is tested or/and retested. This enables the possibilities of including or excluding individual components and interconnects to be tested simultaneously or not.

Several cost models have been established and discussed in [13], [6], [14], [15] and [16]. In [13], Sengupta et. al. have proposed a cost model where test cost corresponds to the Test Application Time (TAT). The TAT is calculated as the sum of testing times consumed through a series of active test instances. These test instances could be wafer sorts, package tests and intermediate tests (on interconnects (TSVs) or on partial stacks). The authors also defined another cost parameter as ‘waste’ in their cost model. The waste could be seen as the portion of TAT which has been spent on testing faulty components. The optimality of test flows in terms of test economics is biased towards minimum amount in TAT as well as waste. For illustration, three different test
schemes (test flows) have been followed on 3D-SIC consisting of three heterogeneous chips. The total amount of TAT, waste and number of dies required to get one hundred good packages have been calculated under each test scheme. The authors found that the test scheme/flow which consists of wafer sorts on individual dies, interconnect test on top layer TSV only and a package test is most cost effective. Because this test flow had the lowest TAT and required the minimum number of dies (from each category) compared to the other two test schemes under the assumed values. Taouil et. al. have established a cost model in [6], where the authors divided the total chip cost into design, manufacturing, packaging, test and logistics costs and showed how the selection of test flows influences the variation in the overall cost. In [14] and [15], the same authors have extended their cost model from previous work and focused more on test cost for W2W and D2W stacking processes respectively. They have evaluated certain test flows and proved that pre-bond test (wafer sort) plays a great part in influencing the overall test cost and followed by interconnect test at post-bond testing. They have also stated the relevance to intermediate test for both top dies and top interconnect layers for low stacked die yield scenarios. Another cost model has been presented in [16] by Chakrabarty et. al. which the test cost as the sum of test preparation cost, test execution cost, silicon overhead cost and cost for test quality (fault coverage). Authors extended their framework of test cost calculation with exhaustive search and propose a heuristic for finding an optimal test flow which reduces test cost. The heuristic is based on the yield matrix partitioning method, where a large optimization problem is divided into smaller problems and the final solution is obtained by combining solutions of smaller problems.

In this thesis, we built a mathematical model for test time calculating by focusing on only the test time consumption. The test time calculation is influenced by the number of active test instances, test times of individual components, manufacturing yields of components or fabrication processes and the input quantity of CUTs. Instead of evaluating a few predefined test flows under a limited set of input values as in previous work, we enable evaluation of all possible test flows with a wide range of varying input parameters. For getting an optimal solution in short time, we propose a heuristic which solves a large optimization problem by splitting it into small problems and solving them iteratively. But unlike prior work, instead of partitioning the problem based on only yield values of dies, we have considered individual test instance as the problem divider. The details on our model for calculating the test time and the proposed heuristic are explained in the upcoming chapters.
Chapter 3

Problem Formulation and Test Time Calculation

This chapter formulates the problem of finding the optimal TF based on minimum test time consumption. First, we describe the problem in terms of unavailability of solutions and necessity of a robust and scalable approach to compensate the shortcomings. Then we will state our proposal with motivational example which addresses the mentioned problems. Finally, we define the problem with the help of our proposal.

3.1 Problem Description

The complexity in 3D-SIC testing grows exponentially due to additional stacking events in the manufacturing process which introduce more susceptible components that need to be tested. Each components and fabrication steps (stacking / packaging) have their own manufacturing yield. Hence, a 3D-SIC consisting of multiple components and produced after multiple manufacturing steps, suffers from low manufacturing yield in overall due to compound imperfect yields. Figure 3.1 explains the major factors for low (overall) production yield and how they limit the quantitative throughput of non-faulty 3D-SIC production. This small example illustrates that there are many possible defects which will lead to a defective final product even if there are only two dies in a 3D-SIC.

Depending on the amounts of dies to be stacked together, different TFs result in different test time consumptions. If tests are skipped or conducted at very later stage of production, then a very large amount of dies needs to be manufactured and tested due to combined low yield factors. If tests are performed at every possible step for maximizing overall production yield, then this precautionary testing for yield enhancement may result in unnecessary (and more than required) test time investment. Finding the optimal TF which balances these two extreme scenarios is the key to economic test and production investments and obtaining the optimal TF is still a persisting problem.
CHAPTER 3. PROBLEM FORMULATION AND TEST TIME CALCULATION

Figure 3.1: Attributes of overall low yield due to compound imperfect yield

which needs to be solved.

From the past research, few solutions establish the necessity of wafer sort in pre-bond testing. But this conclusion loses its potential in the situation where the wafer fabrication yield tends to be high. So, test time invested on testing these dies add unnecessary cost overhead. Few other solution propose the selection of components to be tested under intermediate test instance depends on stacking yield. If the stacking yield is low, then top dies followed by top interconnects should be tested. Otherwise for high stacking yield scenarios, only top interconnect test is sufficient and cost effective. But these solutions lack in defining the boundary between low to high yield range with exact precision. Similarly, most of the established solutions in 3D-SIC testing are based on optimizing only one input parameter (mostly a particular yield value) at a time and fails to maintain optimality with multiple varying parameters (compound yields). So, an effective approach is desired which is capable of solving optimization problems with high precision and multiple varying parameters. In our proposal, we will try to optimize an instance of global state including all input parameters instead of considering only one parameter at a time. Any change in parameter set will result in a new global state. Then a new optimal solution will be produced, which may differ from the previous one.
3.2 Cost and Test Time Calculation

In this thesis, we solve the test time minimization problem in two steps. First, we develop a mathematical model to calculate test time consumption of any given/possible TF. Then we propose a heuristic for obtaining a TF which helps to minimize the test time.

The mathematical model calculates the test time of a given TF over provided input parameters. The first input is the size of the 3D-SIC. The size is the number of dies present in 3D-SIC and it is denoted as \( n \). Then, one set of input is the individual test times, silicon investments and manufacturing yield values of the components in 3D-SIC. The components are dies, TSVs and packages. Another set of input parameters is yields of processing steps (fabricating events). These steps include intermediate stacking process and packaging event. It is assumed that the values for all input parameters are known or provided by the manufacturers.

Before discussing the test time calculation problem, we should first note that, for 3D-SIC, the selected test flow will not only influence the test cost, but also the production cost. For example, if we skip the wafer sort test of a die, the test cost can be reduced; however, defected dies will enter the 3D stacking process, and a KGD can be stacked onto such a defected die, leading to that the final package will be thrown away. This means that we waste some good silicon dies, which can have a very large impact on the production cost, since more dies will have to be manufactured in order to produce a given number of good packages at the end. Therefore, we need to consider both the test time and the number of dies required when computing the total cost of a given test flow.

The total cost of a test flow can be given in terms of a cost function which has two attributes: the cost related to test time consumption during testing and the amount of silicon invested on producing the final products. For example, let us assume that a single die requires 100 test time units (ttu) for testing and this amount of test time costs $5 for the die manufacturer. If the amount of silicon invested for manufacturing that die costs $10, then the total cost investment required for manufacturing that die and conducting a test on it can be deduced as $5+$10 = $15. Any variation in either of the cost attributes results a change in the total cost.

Let us consider a 3D-SIC with two dies (Die1 and Die2). Having size of two, this 3D-SIC will have four test instances in any of its TF. The test instances are: (1) wafer sort on Die1, (2) wafer sort on Die2, (3) intermediate stack test after Die2 is stacked over Die1 and (4) default package test on final packages. By combining these four test instances in different manner will result in sixteen \( 2^4 = 16 \) possible (unique) TFs. As untested products shall not shipped out, we assume that a comprehensive package test on final products are mandatory. With this, the available test instances will be three (1, 2 and 3) and total valid TFs will be eight \( 2^3 = 8 \). These eight valid TFs are listed in table 3.1. In table 3.1, the first column from the left lists the test instances. The second column holds a short description for the corresponding test instance from the
CHAPTER 3. PROBLEM FORMULATION AND TEST TIME CALCULATION

first column. From the third column to the end of the table, eight possible and valid TFs are presented (in column wise). The 1’s and 0’s indicate the active and inactive test instances respectively within a TF.

<table>
<thead>
<tr>
<th>Test instance</th>
<th>Type of test</th>
<th>TF0</th>
<th>TF1</th>
<th>TF2</th>
<th>TF3</th>
<th>TF4</th>
<th>TF5</th>
<th>TF6</th>
<th>TF7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wafer sort on Die1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Wafer sort on Die2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Intermediate test</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Package test</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.1: Valid test flows

We consider a TF is an unique combination of one or few selectively enabled test instances. Our mathematical model will treats each active test instance to be a separate entity and follows a modular approach to calculate test time consumption of the given TF. The steps in test time calculation approach are: (1) instance level input quantity calculation, (2) instance level test time calculation and (3) total test time calculation.

**Step 1: Instance level input quantity calculation**

Due to imperfect manufacturing yield, defects may be introduced to components during a manufacturing step. Each manufacturing step may be followed by a test instance for diagnosis of these newly introduced defects. If a test is performed in a test instance (active test instance) then, some of the CUTs (which are affected by the defects) will fail the test. So, the number of untested components is larger than the number of non-faulty components (outcome of the test as success/pass). As the test time is defined as the amount of time spent on testing components (irrespective of test outcome: pass or fail), the number of CUTs affects the test time directly. So, we need to determine the quantity of components to be tested (test inputs) at each activated test instance under a TF in order to generate the required number of CUTs as output.

The number of components to be tested in a test instance (say instance $i$) is calculated as:

$$Q(i) = \frac{Q_{out}(i)}{\prod Y_c}, \forall i \in \{1, 2n\}$$  \hspace{1cm} (3.1)

Where,

- $Q_{out}(i)$ is the desired output quantity;
- $Y_c$ is the individual manufacturing yield of untested components and steps; and
- $Q(i)$ is the required input quantity to be tested in the $i^{th}$ test instance in order to get $Q_{out}(i)$ amount of tested products after the test.

The input quantity, $Q(i)$, can be termed as the amount of components needs to be tested in an active test instance ($i$) for obtaining the desired output amount $Q_{out}(i)$ after
3.2. COST AND TEST TIME CALCULATION

If any component is tested for the first time in any active instances, then in order to get 100 units of non-faulty components with 50% manufacturing yield, we need to test $100/0.5 = 200$ untested units of that component.

For example, TF0 (table 3.1) which has only package test instance marked as active and no test has been performed till this instance. So in this scenario, untested packages will be tested against compound yield of all manufacturing steps at once. Therefore, $\prod Y_c = \text{Yields of (wafer sort on Die1} \times \text{wafer sort on Die2} \times \text{TSVs} \times \text{stacking step} \times \text{packaging event)}$.

<table>
<thead>
<tr>
<th>Components and steps (c)</th>
<th>Die1 (wafer sort)</th>
<th>Die2 (wafer sort)</th>
<th>TSVs</th>
<th>Stacking</th>
<th>Packaging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yield ($Y_c$)</td>
<td>0.9</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 3.2: Yield values for components and steps

For illustrating the usability of equation 3.1, we assume some input values (randomly chosen) which are listed in table 3.2. In this table, the first row lists the components and processing steps that are involved in the given 3D-SIC (with two dies) production. The second row holds the manufacturing yield values for the respective components and steps. Now consider our example with TF0, we have:

$\prod Y_c = 0.9 \times 0.8 \times 0.7 \times 0.6 \times 0.5 = 0.1512$

If we want 100 known good packages (KGP) then, (using equation 3.1) we need to test $100/0.1512 = 662$ untested packages during this test instance. In this case, the 662-100=562 packages which are failed during the test will be thrown away. As each package contains one die from both types (Die1 and Die2), a total of 562 Die1s and 562 Die2s are thrown away. It is important to note that some of these thrown-away dies are defected by themselves, while the others are originally good dies. Some of these good dies are stacked onto defected dies, and will be thrown away, therefore they are wasted.

To elaborate the illustration, let us consider another situation where we will take TF7 (table 3.1) as our given TF. Under TF7 (where all test instances are enabled), the instance level quantity calculations are:

In instance 1: $\prod Y_c = 0.9$ (yield of wafer sort on Die1)
In instance 2: $\prod Y_c = 0.8$ (yield of wafer sort on Die2)
In instance 3: $\prod Y_c = 0.7 \times 0.6 = 0.42$ (compound yield from TSVs and stacking step)
In instance 4: $\prod Y_c = 0.5$ (yield of packaging event)

In order to get 100 KGP under TF7, (using equation 3.1) we need to test:
In instance 4: $100/0.5 = 200$ untested packages
In instance 3: \( \frac{200}{0.42} = 477 \) untested partial stacks
In instance 2: \( \frac{477}{0.8} = 597 \) untested Die2
In instance 1: \( \frac{477}{0.9} = 530 \) untested Die1

With this TF, 100 units from both types of dies (Die1 and Die2) which are integrated in 100 KGP s out of 530 and 597 units respectively. So, 530-100 = 430 units from Die1 and 597-100 = 497 units from Die2 are thrown away for producing 100 good 3D-SICs. If we compare this TF (TF7) with TF0 where only the final package test is performed, less dies are thrown away.

**Step 2: Instance level test time calculation**

As the amount of components to be tested in any instance is already known from step 1, the test time can be calculated separately for each instance. So, the total test time required in an active test instance \( i \) is denoted as \( T(i) \) and evaluated as:

\[
T(i) \leq \left( Q(i) \cdot \sum_{c \in \text{tested components}} T_c \right), \quad \forall i \in \{1, 2n\} \quad (3.2)
\]

Where \( Q(i) \) is the amount of CUTs needed as input during test instance \( i \). \( T_c \) is the test time required for testing a component 'c' and 'c' can be any available dies or TSV layers or packages. As any component can get damaged arbitrarily during a stacking/packaging process, we test all the available components under an active test instance \( i \) even if a few of the components may be already tested in previous test instances.

The test time required to test a die under wafer sort (where a comprehensive test is performed) is usually different and more than testing a already tested die in an intermediate stack test where the intention of test is to diagnosis defects on KGDs that possibly introduced during the stacking/packaging process. As there is no precise and established relationship available for these two different types of test time consumptions for the same die, we consider the worst case scenario, where the dies are tested in similar manner (detail test) under an active test instance regardless if the test instance is a wafer sort or an intermediate stack test.

<table>
<thead>
<tr>
<th>Components (c)</th>
<th>Die1 (wafer sort)</th>
<th>Die2 (wafer sort)</th>
<th>TSVs</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test time (Tc) in ttu</td>
<td>100</td>
<td>200</td>
<td>10</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 3.3: Test times for components

Let us consider again the example we introduced at the beginning of this section. Table 3.3 lists some assumed test time values (randomly chosen) which we use in our example. The first and second rows of table 3.3 present the available components for testing and the test time required to test one unit of each individual component respec-
3.2. COST AND TEST TIME CALCULATION

tively.

TF0 has only one active test instance i.e. test instance 4 \( (i=4) \) : package test. By using equation 3.2, T(4) of TF0 will require \( 662 \times (100+200+10+25) = 221770 \) ttu in total to test 662 untested packages for getting 100 final KGP.

Now with the help of equation 3.2, the test time of the different instances during TF7 are:

In instance 1: T(1) will be \( 530 \times 100 \) ttu = \( 53000 \) ttu

In instance 2: T(2) will be \( 597 \times 200 \) ttu = \( 119400 \) ttu

In instance 3: T(3) will be \( 477 \times (100+200+10) \) ttu = \( 147870 \) ttu

In instance 4: T(4) will be \( 200 \times (100+200+10+25) \) ttu = \( 67000 \) ttu

In test instances 3 and 4 which represent intermediate stack test and final package test respectively, all available components are tested comprehensively even if Die1 and Die2 are already tested in previous test instances (instance1: wafer sort of Die1 and instance2: wafer sort of Die2). So, the test time consumed for testing one intermediate/partial stack during test instance 3 is the sum of individual test time consumptions for Die1, Die2 and TSV layer between them, which results in \( 100+200+10 = 310 \) ttu. Similarly during test instance 4, one package has test time consumption calculated as \( 100+200+10+25 = 335 \) ttu collectively.

We note these values of test time from each instance and proceed to the final step of test time calculation.

**Step 3: Total test time calculation**

The total test time is the sum of individual test time consumptions from all active test instances under the TF to generate 100 KGP. We denote the required total test time in a TF as \( T \) which can be evaluated as:

\[
T = \sum_{i = \text{active instances}} T(i) \quad , \forall i \in \{1, 2n\}
\]  

(3.3)

So, by using equation 3.3 and following our example, TF0 (having only test instance 4 as active) will have total test time as:

\[ T(4) = 221770 \] ttu for producing 100 KGP.

Whereas, TF7 will have total test time as:

\[ T(1)+T(2)+T(3)+T(4) = 53000+119400+147870+67000 = 387270 \] ttu for producing 100 KGP.

With the help of the discussed example we have demonstrated how the test time of a given TF is calculated using our mathematical model. By analysing the results, we can see that TF0 requires \( 221770 \) ttu for producing 100 KGP and throws away \( 562 \) ttu. 
dies from each type (Die1 and Die2) or $562 + 562 = 1124$ dies in total. On the other hand, TF7 consumes 387270 ttu for producing 100 KGPs and throws away 430 units from Die1 and 497 units from Die2 (total $430 + 497 = 927$ dies). By comparing TF0 and TF7, we notice: TF0 requires lesser test time (221770 ttu) than TF7 (387270 ttu), but TF0 throws away more silicon (total 1124 dies) compared to TF7 (total 927 dies). Therefore, TF0 is efficient in reducing the test time consumption whereas TF7 is better in lowering the thrown-away silicon amount. Here, we see that there is a tradeoff between the investments of test time and silicon while choosing a cost effective TF. It is up to the designer/manufacturer to decide whether the test time consumption or the silicon investment need to be optimized or how they should be considered together. In this thesis, we will focus mainly on the issue of test time minimization.

Now, we are able to calculate test time and amount of components (silicon) required by any given TF through our mathematical model. We can move to the next phase of our proposal on finding the optimal TF where the parameter for optimization is either test time or silicon investments. A simple way to achieve this objective is listing down all possible TFs for a 3D-SIC and calculating the required investments in terms of either test time or silicon for the given input values. We will then select the TF from this list which has the minimum value in the chosen optimization parameter (test time or silicon) as the optimal TF for given condition. We extend our example to illustrate the impact of input values (yields and individual test times) and optimization parameter (test time or silicon) on finding the optimal TF. We consider a 3D-SIC with two dies and evaluate all eight TFs from table 3.1 against different yield ranges in three different scenarios (presented by table 3.4). The first and second rows of table 3.4 present the list of components and steps that involved in 3D-SIC production. The second row presents some assumed test time consumptions for appropriate entries from the first row. The third, fourth and fifth rows hold some assumed manufacturing yield values for the listed components/steps (from the first row) and represent three different input sets for scenario 1, 2 and 3 respectively. Setting the desired output quantity to 100 KGPs, the test times and silicon requirements are calculated for the listed TFs under all specified scenarios and shown in figure 3.2 and figure 3.3 respectively.

<table>
<thead>
<tr>
<th>Components and steps (c)</th>
<th>Die1 (wafer sort)</th>
<th>Die2 (wafer sort)</th>
<th>TSVs</th>
<th>Stacking</th>
<th>Packaging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test time (Tc) in ttu</td>
<td>100</td>
<td>200</td>
<td>10</td>
<td>-</td>
<td>25</td>
</tr>
<tr>
<td>Yield (Y$_c$): Scenario1</td>
<td>0.9</td>
<td>0.9</td>
<td>0.9</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>Yield (Y$_c$): Scenario2</td>
<td>0.5</td>
<td>0.8</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>Yield (Y$_c$): Scenario3</td>
<td>0.7</td>
<td>0.5</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Table 3.4: Values for input parameters

Considering the test time consumptions, it is clearly seen (in figure 3.2) that different scenarios have different TFs as the optimal TF. In scenario 1, TF0 (only package test) is considered as the optimal TF with minimum test time. For scenario 2, TF1 (wafer sort on Die1 followed by package test) is the best TF for minimizing test time.
On the other hand, TF3 (wafer sorts on both the dies followed by package test) is the optimal TF for scenario 3. Similarly, figure 3.3 shows that different scenario has different TFs as the Optimal TFs when the amount of silicon investments are taken into account. Also, it is clear that under each scenario the optimal TF for test time and silicon investment are different from each other. Comparing the optimal TFs for corresponding scenarios in figure 3.2 and figure 3.3, it can be concluded that optimizing the test time consumptions does not optimize the silicon investments under a single TF. So, a tradeoff model must be developed, which is not in the scope of this thesis. Instead, we focus only on optimizing the test time consumptions from now on.

From the above example, it can be concluded that the optimality of TFs depends on input parameters and predefined TFs (from past research work) are unable to handle variations in input values. This motivates us to evaluate every possible TF under a set of (given) input parameters for getting the optimal TF with minimum test time. But, the number of possible TFs will grow exponentially with factor of $2^n$ as the 3D-SIC size ‘n’ increases. It will be practically impossible to evaluate every TF for larger ‘n’ values. So, we propose a heuristic to find a nearly optimal TF in a quicker manner for larger 3D-SIC sizes. The proposed heuristic divides the whole solution space into $2^n$ optimization steps. It solves each optimization problem iteratively and proceed to the
3.3 Problem Definition

We assume, given is a 3D-SIC of size \( n \), with test time \( (T_c) \) and manufacturing yield \( (Y_c) \) of each component \( (c) \). The problem is to find a test flow that minimizes the test time \( (T) \), by generating a test flow, which is described as a combination of active test instances and requires \( Q \) amount of CUTs for producing output quantity of \( Q_{out} \).
Chapter 4

Approach

In this chapter, we will discuss on the implementation details of our proposal. Our proposal has a mathematical model for test time calculation (implemented as TI matrix model at the first section in this chapter) and a heuristic for finding an optimal TF (described in further section).

4.1 Test Instance Matrix Model

We develop a mathematical model for representing a TF as a matrix in order to facilitate the calculation of the test time consumed by the TF using equations from section 3.2. We call this model as “Test Instance Matrix” (TI matrix) which can be represented in a single column matrix of size $2n \times 1$ for 3D-SIC with size '$n$'. TI matrix initiates/defines a combination of active test instances which is termed as the given TF. Briefly the model looks like as shown in figure 4.1.

![Figure 4.1: Test instance matrix (TI matrix)](image-url)
In TI matrix, each row represents one valid test instance. The rows are arranged from top to bottom according to the occurrence of test instances through the manufacturing process. All together there are $2n$ rows. The very first row indicates the wafer sort for first die and the last row describes the package test. Figure 4.2 illustrates the transformation and mapping process of a TF (with $n = 3$) into the TI matrix. Each element in the TI matrix lists the activeness of the test instances and use a binary value of '0' or '1' to represent it. A '0' represents a test instance (in corresponding row) is inactive, so no test will be conducted in that instance. A '1' represents that a test instance (in corresponding row) is active and all components available in that instance will be tested.

We consider a snapshot of whole matrix at once for deriving TFs from this matrix. Each of the instantaneous snapshots (combination of ones and zeros) of the TI matrix can be seen as one TF as it holds one unique combination of test instances. A custom TF can be generated by filling ones and zeros at desired rows. A filled matrix follows equations from section 3.2 to calculate total test time for the TF. The details of test time calculation can be found in appendix A.

Figure 4.2: Transformation and mapping of a test flow into TI matrix (for n=3)
4.2 Heuristic

The overview of heuristic is presented in algorithm 1 given in figure 4.3 and the flowchart given in figure 4.4.

The heuristic is used to find the optimal TF by dividing the optimization problem into smaller optimization problems and solving them iteratively. This approach can have maximum number of iterations the same as the total number of test instances \((2n)\). Each iteration can have steps equal to the number of inactive test instances in the corresponding iteration. In each step, a new TF is generated by activating one inactive test instance at a time. The newly generated TF is evaluated. The evaluation process consists of calculating the test time of the new TF (using TI matrix) and comparing the test time against a threshold value. The threshold value for any iteration is the test time of the default TF during that iteration. The default TF remains unchanged for all steps in one iteration. Each iteration has a different default TF based on results obtained from previous iteration. When all steps are followed under one iteration, the TF (among evaluated TFs during this iteration) with minimum test time will be awarded as the partially optimized TF. Every iteration results in one partially optimized TF. These partially optimized TFs become the default TF for successive iterations. The partially optimized TF obtained from the final iteration will be regarded as the optimal TF.

As a starting point, the default TF contains only package test (figure 4.3: line 1-3). By the end of every iteration, an inactive test instance (within the default TF) is activated and a new default TF is generated (figure 4.3: line 6-29). Within an iteration, every step enables one test instance (addition to default TF) resulting in an unique TF. This new TF is evaluated against a threshold where the threshold is the test time consumption of the default TF. If the TF does not cross the threshold, it is then regarded as a valid TF (figure 4.3: line 8-17). The purpose of defining the validity of a TF has two reasons. First, the exclusion of invalid TFs is beneficial for limiting the search space and avoiding unnecessary calculations (test time comparisons with all TFs). Second, as valid TFs have lesser test time consumptions than the default TF (of corresponding iteration), they impose strictness on the future iterations by lowering the threshold. This directs (positively) the approach of finding TF with minimum test time. Once all steps are completed resulting all possible unique TFs are evaluated, the valid TF with minimum test time is regarded as the partially optimized TF for current iteration and will become the default TF (threshold) for next iteration (figure 4.3: line 18-28). In this way, these partially optimized TFs build an optimal TF constructively on top of updated the default TFs. The process continues till the iteration count matches one less to the number of instances (as after assuming package test is default, we left with \(2n-1\) available test instance to exploit; (figure 4.3: line 4-6). If no partially optimized TF claims over the default TF in an iteration, the process terminates announcing the latest default TF as the best TF it can obtain for minimizing the test time (figure 4.3: line 18-21).

For a 3D-SIC with \(n\) chips, we will have \(2n\) test instances. As a starting point, we will consider the default TF to be only package test. The first iteration will have \(2n-1\) number of combinations (as package test instance is active and fixed among \(2n\) TFs).
CHAPTER 4. APPROACH

The partially optimized TF (if found, which has to be valid and possess lowest test time among all valid TFs) is treated as the default TF for next iteration. The second iteration has $2n-2$ remaining combinations after the default TF is updated and set for second iteration. In the same way the process continues until final iteration (iteration count: $2n-1$) or until unavailability of a partially optimized TF.

4.2.1 Complexity Evaluation

For complexity calculation, the total number of TFs to be evaluated under heuristic (in worst case scenario) is:

$$\sum_{n=1}^{2n-1} k \quad \text{or} \quad \frac{(2n - 1)2n}{2} \quad \text{or} \quad 2n^2 - n$$

The expression can be denoted as $O(n^2)$. So this time complexity form shows that the heuristic will have quadratic time complexity. In case an exhaustive search, the total number of available TFs is $2^{2n}$. Considering the mandatory final package test, $2^{2n-1}$ TFs will be evaluated by exhaustive search. So, in general the exhaustive search has exponential time complexity and can be represented as $O(2^n)$. So, the heuristics time complexity (quadratic: $O(n^2)$) is much lesser than exhaustive searches time complexity (exponential: $O(2^n)$).

We implement our TI matrix model and heuristic approach for experimental purpose and the next chapter details on these experiments and the analysis of results obtained from them. The heuristic is compared to other alternates (TFs) in terms of test time calculation and computation time taken for producing the desired results.
Algorithm 1: Heuristic approach for optimal TF

1. Set TF: Default = PackageTest
2. Call TI matrix
3. Record default test time
4. Set remaining_test_instances to 2n-1
5. Set enabled_test_instances to 1
6. While: remaining_test_instances ≠ 0 do
7. Set input matrix = TF: Default
8. For a = enabled_test_instances to 2n-1 do
9. Activate instance(a) on input matrix
10. Call TI matrix
11. Record test time
12. If test time > default test time then
13. Discard TF
14. Else
15. Mark the TF as valid
16. End if
17. End for
18. If number of valid TFs is zero then
19. Set TF: Optimal = TF: Default
20. Return TF: Optimal
21. Break
22. Else
23. Sort valid TFs in ascending test time order
24. Set TF: Default to be first TF in sorted TFs list
25. Set default test time to be test time of TF: Default
26. Decrement remaining_test_instances by 1
27. Increment enabled_test_instances by 1
28. End if
29. End while

Figure 4.3: Algorithm for Heuristic
Set default TF as (only) package test

Calculate test time for given TF and mark as $T_{\text{default}}$

Enable one inactive TI in default TF

Calculate test time for new TF and mark as $T_{\text{active}}$

If $T_{\text{active}} < T_{\text{default}}$:
- Add current TF to valid TF list
- All inactive TIs have been enabled

If $T_{\text{active}} < T_{\text{default}}$:
- Add current TF to valid TF list
- Set partially optimal TF = TF with lowest test time
- Set partially optimal TF = TF with lowest test time
- Set Optimal TF = Default TF

If $T_{\text{active}} < T_{\text{default}}$:
- Add current TF to valid TF list
- Set partially optimal TF = TF with lowest test time
- Set partially optimal TF = TF with lowest test time
- Set Optimal TF = Default TF

If $T_{\text{active}} < T_{\text{default}}$:
- Add current TF to valid TF list
- Set partially optimal TF = TF with lowest test time
- Set partially optimal TF = TF with lowest test time
- Set Optimal TF = Default TF

START

STOP

Figure 4.4: Flowchart for Heuristic
Chapter 5

Results and Analysis

In this chapter, we report the results by analyzing output data obtained from experiments which evaluate efficiency of the proposed heuristic for finding an optimal TF with short test time. First, we describe the setup for our experiments. Then, we discuss different approaches by comparing them in test time consumptions and execution time taken.

5.1 Experimental Setup

In order to evaluate the efficiency of our proposed heuristic algorithm, we compare it with the exhaustive search algorithm. The exhaustive search always finds the optimal TF by evaluating all possible TFs for a given 3D-SIC. The details of the exhaustive search implementation using TI matrix model will be discussed in sub-section 5.1.1. We evaluate both algorithms in terms of test time optimization by comparing the test time consumptions of best TFs obtained from respective algorithms. We introduce two more fixed TFs (active test instances are predetermined) for evaluating the efficiency of the heuristic against the extreme test scenarios (minimum and maximum testing). One of the fixed TF represents the final package test (minimum testing) which has only one (the last) test instance as active. We consider this TF as default TF for the rest of this chapter. The other fixed TF represents the maximum testing scenario by having all test instances as active. We name this TF as test-all TF. So, four different TFs will be compared and discussed in our experimental results.

For our experiment, we set the desired output quantity of the final product to 100 KGPs. So, each of the participating TF calculates the total amount of test time required to produce 100 KGPs. The input parameters such as properties of TSVs, packaging processes and stacking steps are set with constant values for our experiment and we choose these values randomly (based on suggested values from previous work and primary literature for this thesis). The test time and yield of each TSV layers are set to be 25 ttu and 0.9 respectively. Similarly the test time and manufacturing yield of
packaging step is considered to be 100 ttu and 0.9. We set the yield of each stacking event to 0.95. All of these values (input parameters) can be treated as variables and our proposed method can handle any input values. But for the sake of simplicity and considering the scope of this thesis (as to highlight and demonstrate the ability of obtaining optimal TF efficiently), we treat all input parameters to be set at the fixed values, as presented above. Only exceptions are stack size and manufacturing yield of individual dies corresponding to respective stack size.

Under an experimental scenario, we calculate test times on different stack sizes (varying from two to ten, \( n=2:10 \)). In total, we consider four different experimental scenarios. The 3D-SICs in different scenarios will differ from each other only in manufacturing yields of dies in the stacks. Table 5.1, 5.2, 5.3 and 5.4 shows the properties of dies which take part in building the 3D-SICs under the four different scenarios. The firstrows of the above mentioned tables have names of the dies and they are listed according to the order in which they enter into the 3D-SICs. For example, in a 3D-SIC with three dies, Die1 enters the stack at first place followed by Die2 (which enters only after Die1) and then Die3 introduces to the stack (where Die2 already stacked on top of Die1). The second rows indicate the test time of each die. The third rows list the manufacturing yields for the corresponding dies. In scenario1 (table 5.1), the yield range varies from .01 to .99 in ascending order and one yield value has been assigned to one die from die1 till die10. The yield range for scenario2 (table 5.2) covers the value from .99 to .01 in descending order. Scenario3 and 4 deal with mid-range yield values which varies from .4 to .6 (table 5.3) and .6 to .4 (table 5.4) respectively. The motivation for reversing yield values in alternating scenarios is for the purpose of observing the test time variation of stacks under test with wide coverage of possible yield range which is distributed among available dies. If we consider a 3D-SIC with only two dies, then scenario1 evaluates the 3D-SIC under low yield range (Die1:.01 and Die2:.9 ), scenario2 evaluates with high yield range (Die10:.99 and Die2:.9), whereas medium yield range is covered by scenario3 (Die1:.4 and Die2:.45) and scenario4 (Die1:.6 and Die2:.57). By combining all four scenarios, we can claim we have studied different types of yield ranges (low, medium and high) for a particular 3D-SIC size.

<table>
<thead>
<tr>
<th>Dies</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test time (in ttu)</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Yield ((Y_c))</td>
<td>0.01</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
<td>0.6</td>
<td>0.7</td>
<td>0.8</td>
<td>0.9</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Table 5.1: Properties of dies in scenario 1

<table>
<thead>
<tr>
<th>Dies</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test time (in ttu)</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Yield ((Y_c))</td>
<td>0.99</td>
<td>0.9</td>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.4</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 5.2: Properties of dies in scenario 2
5.1. EXPERIMENTAL SETUP

5.1.1 Exhaustive Search

The exhaustive search is one approach to find the optimal TF for a given 3D-SIC. In this method, the entire search space is covered by evaluating all possible TFs. The TF with the minimum test time consumption will then be the optimal TF. The implementation of exhaustive search is presented by algorithm 2, given in figure 5.1.

This method analyzes and compares the test time of all available TFs for a given 3D-SIC and then chooses the optimal one with minimum test time. The TI matrix model is used to calculate total test time for any given TF. As a starting point, the package test is selected as the default TF. A TI matrix instance with package test enabled (last row = '1') is considered as the input matrix. Then it enables the first row and combines all other rows (inactive test instances) by enabling one row in each iteration. At every iteration, a new TF (unique combination of active test instances) is generated and its test time is calculated. Once all TFs (resulted from combining first row with other available rows) are treated, then the input matrix is reset to default TF and starts combining the second row with other rows (which are available for traversing). All the newly generated TFs are treated in similar fashion as before. This procedure repeats until all rows are traversed without leaving any combination behind.

Once the test time of a TF is calculated, a validity evaluation will take place for considered/current TF by comparing it with the default TF in terms of test time consumption. If the current TF has higher test time consumption than the default one, then it is discarded in order to save memory space and computation time by avoiding further unwanted calculation. If the TF is evaluated as economic in test time saving, then it will be placed in a valid TF list. The valid TF list is a sorted list where TFs appear according to the ascending order of their test time consumptions. So, the very first TF in the sorted list has lowest test time and termed as the optimal TF. The optimal TF (list of active test instances) is returned to the user.

<table>
<thead>
<tr>
<th>Dies</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test time (in ttu)</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Yield ($Y_c$)</td>
<td>0.4</td>
<td>0.45</td>
<td>0.48</td>
<td>0.5</td>
<td>0.51</td>
<td>0.5</td>
<td>0.53</td>
<td>0.55</td>
<td>0.57</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 5.3: Properties of dies in scenario 3

<table>
<thead>
<tr>
<th>Dies</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test time (in ttu)</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Yield ($Y_c$)</td>
<td>0.6</td>
<td>0.57</td>
<td>0.55</td>
<td>0.53</td>
<td>0.5</td>
<td>0.51</td>
<td>0.5</td>
<td>0.48</td>
<td>0.45</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table 5.4: Properties of dies in scenario 4
CHAPTER 5. RESULTS AND ANALYSIS

Figure 5.1: Algorithm for Exhaustive search method for optimal TF

```
Algorithm 2: Exhaustive search method for optimal TF

1 Set TF-Default = PackageTest
2 Call TI matrix
3 Record default test time
4 For a = 1 to 2n-1 do
5   Set input matrix = TF-Default
6   For b = a to 2n-1 do
7     Activate instance (b) on input matrix
8     Call TI matrix
9     Record test time
10    If test time > default test time then
11       Discard TF
12       Else
13         Mark the TF as valid
14   End if
15 End for
16 Sort valid TFs in ascending test time order
17 Set TF-Optimal to be the first TF from sorted TF list
18 Return TF-Optimal
```

5.2 Test Flow Comparisons

The experimental results are presented in table 5.5, 5.6, 5.7 and 5.8 for scenario 1 (table 5.1), 2 (table 5.2), 3 (table 5.3) and 4 (table 5.4) respectively. The first columns of the above mentioned tables denote the size of the 3D-SIC. This size represents the number of dies (with inherited properties from table 5.1, 5.2, 5.3 and 5.4) present in the stack and denoted as ‘n’. If the size is two (n=2), then the first two dies (Die1 and Die2) will be considered. If the size is five (n=5) then first five dies (Die1 to Die5) is considered and so on. The first rows in the tables list the approaches used for test time calculation. As previously discussed, we have considered four TFs (Default, test-all, TF obtained from exhaustive search and heuristic) described in four columns in first rows. The exhaustive search is considered as the performance threshold or baseline in test time analysis for all scenarios. The columns represented by the exhaustive search list the test times for corresponding stack size (presented in first columns). The other three TFs have two columns each. The first columns hold the test time values received from experiments. The second columns show the percentage of test time overhead or additional test time invested under the listed TFs with respect to the exhaustive search.
### 5.2. Test Flow Comparisons

#### Table 5.5: Test time analysis table for scenario 1

<table>
<thead>
<tr>
<th>n</th>
<th>Exhaustive Test Time</th>
<th>Heuristic Test Time</th>
<th>Time overhead to Exhaustive</th>
<th>Test-All Test Time</th>
<th>Time overhead to Exhaustive</th>
<th>Default Test Time</th>
<th>Time overhead to Exhaustive</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>14576250</td>
<td>14576250</td>
<td>0.0%</td>
<td>14913275</td>
<td>2.3%</td>
<td>276154375</td>
<td>1794.6%</td>
</tr>
<tr>
<td>3</td>
<td>17958800</td>
<td>17958800</td>
<td>0.0%</td>
<td>18659200</td>
<td>3.9%</td>
<td>2393899200</td>
<td>13230.0%</td>
</tr>
<tr>
<td>4</td>
<td>21807150</td>
<td>21807150</td>
<td>0.0%</td>
<td>22954650</td>
<td>5.3%</td>
<td>1236984650</td>
<td>56623.8%</td>
</tr>
<tr>
<td>5</td>
<td>26215600</td>
<td>26215600</td>
<td>0.0%</td>
<td>27926400</td>
<td>6.5%</td>
<td>4504899800</td>
<td>171740.4%</td>
</tr>
<tr>
<td>6</td>
<td>31229900</td>
<td>31229900</td>
<td>0.0%</td>
<td>33719350</td>
<td>8.0%</td>
<td>2393899200</td>
<td>552319.2%</td>
</tr>
<tr>
<td>7</td>
<td>37011250</td>
<td>37011250</td>
<td>0.0%</td>
<td>40685325</td>
<td>11.3%</td>
<td>2393899200</td>
<td>777339.8%</td>
</tr>
<tr>
<td>8</td>
<td>43908175</td>
<td>43908175</td>
<td>0.0%</td>
<td>48885325</td>
<td>14.1%</td>
<td>2393899200</td>
<td>1066325.4%</td>
</tr>
<tr>
<td>9</td>
<td>51910000</td>
<td>51910000</td>
<td>0.0%</td>
<td>58534050</td>
<td>14.1%</td>
<td>2393899200</td>
<td>1066325.4%</td>
</tr>
<tr>
<td>10</td>
<td>61318825</td>
<td>61318825</td>
<td>0.0%</td>
<td>69942450</td>
<td>14.1%</td>
<td>2393899200</td>
<td>1066325.4%</td>
</tr>
</tbody>
</table>

#### Table 5.6: Test time analysis table for scenario 2

<table>
<thead>
<tr>
<th>n</th>
<th>Exhaustive Test Time</th>
<th>Heuristic Test Time</th>
<th>Time overhead to Exhaustive</th>
<th>Test-All Test Time</th>
<th>Time overhead to Exhaustive</th>
<th>Default Test Time</th>
<th>Time overhead to Exhaustive</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>310250</td>
<td>310250</td>
<td>0.0%</td>
<td>782275</td>
<td>152.1%</td>
<td>310250</td>
<td>0.0%</td>
</tr>
<tr>
<td>3</td>
<td>674100</td>
<td>674100</td>
<td>0.0%</td>
<td>2063650</td>
<td>36.4%</td>
<td>310250</td>
<td>0.0%</td>
</tr>
<tr>
<td>4</td>
<td>1371000</td>
<td>1371000</td>
<td>0.0%</td>
<td>3976400</td>
<td>84.7%</td>
<td>310250</td>
<td>0.0%</td>
</tr>
<tr>
<td>5</td>
<td>2153000</td>
<td>2153000</td>
<td>0.0%</td>
<td>4804400</td>
<td>89.9%</td>
<td>310250</td>
<td>0.0%</td>
</tr>
<tr>
<td>6</td>
<td>3148150</td>
<td>3424875</td>
<td>8.8%</td>
<td>51825589</td>
<td>413.7%</td>
<td>310250</td>
<td>0.0%</td>
</tr>
<tr>
<td>7</td>
<td>4587000</td>
<td>5191000</td>
<td>7.1%</td>
<td>4804400</td>
<td>44.6%</td>
<td>310250</td>
<td>0.0%</td>
</tr>
<tr>
<td>8</td>
<td>6815750</td>
<td>7192850</td>
<td>5.5%</td>
<td>58534050</td>
<td>51.3%</td>
<td>310250</td>
<td>0.0%</td>
</tr>
<tr>
<td>9</td>
<td>9929375</td>
<td>10396825</td>
<td>4.7%</td>
<td>6711350</td>
<td>35.9%</td>
<td>310250</td>
<td>0.0%</td>
</tr>
<tr>
<td>10</td>
<td>1197825</td>
<td>12196850</td>
<td>5.0%</td>
<td>137423100</td>
<td>935.9%</td>
<td>310250</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

#### Table 5.7: Test time analysis table for scenario 3

<table>
<thead>
<tr>
<th>n</th>
<th>Exhaustive Test Time</th>
<th>Heuristic Test Time</th>
<th>Time overhead to Exhaustive</th>
<th>Test-All Test Time</th>
<th>Time overhead to Exhaustive</th>
<th>Default Test Time</th>
<th>Time overhead to Exhaustive</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>890250</td>
<td>890250</td>
<td>0.0%</td>
<td>1123275</td>
<td>26.2%</td>
<td>1534250</td>
<td>72.3%</td>
</tr>
<tr>
<td>3</td>
<td>1513800</td>
<td>1513800</td>
<td>0.0%</td>
<td>2063200</td>
<td>36.4%</td>
<td>5544000</td>
<td>266.2%</td>
</tr>
<tr>
<td>4</td>
<td>2311150</td>
<td>2457050</td>
<td>6.3%</td>
<td>3276650</td>
<td>41.8%</td>
<td>17184300</td>
<td>643.5%</td>
</tr>
<tr>
<td>5</td>
<td>3326600</td>
<td>3413800</td>
<td>2.7%</td>
<td>4804400</td>
<td>44.6%</td>
<td>49076600</td>
<td>1377.1%</td>
</tr>
<tr>
<td>7</td>
<td>4576250</td>
<td>4615900</td>
<td>0.9%</td>
<td>6711350</td>
<td>46.7%</td>
<td>137423100</td>
<td>2903.0%</td>
</tr>
<tr>
<td>8</td>
<td>7607175</td>
<td>7896925</td>
<td>3.8%</td>
<td>11899325</td>
<td>56.4%</td>
<td>857240350</td>
<td>11168.8%</td>
</tr>
<tr>
<td>9</td>
<td>9607525</td>
<td>9830450</td>
<td>2.3%</td>
<td>13338050</td>
<td>59.6%</td>
<td>1976854500</td>
<td>20476.1%</td>
</tr>
<tr>
<td>10</td>
<td>1197825</td>
<td>12196850</td>
<td>1.8%</td>
<td>19470450</td>
<td>62.5%</td>
<td>4278236025</td>
<td>35613.0%</td>
</tr>
</tbody>
</table>
CHAPTER 5. RESULTS AND ANALYSIS

Exhaustive  Heuristic  Test-All  Default

<table>
<thead>
<tr>
<th>n</th>
<th>Test Time</th>
<th>Test Time</th>
<th>Time overhead to Exhaustive</th>
<th>Test Time</th>
<th>Time overhead to Exhaustive</th>
<th>Test Time</th>
<th>Time overhead to Exhaustive</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>722250</td>
<td>807500</td>
<td>11.8%</td>
<td>952275</td>
<td>31.8%</td>
<td>1534250</td>
<td>112.4%</td>
</tr>
<tr>
<td>3</td>
<td>1276800</td>
<td>1412140</td>
<td>10.6%</td>
<td>1831200</td>
<td>43.4%</td>
<td>3554000</td>
<td>334.2%</td>
</tr>
<tr>
<td>4</td>
<td>2013150</td>
<td>2251600</td>
<td>11.8%</td>
<td>2985650</td>
<td>48.3%</td>
<td>17184300</td>
<td>753.6%</td>
</tr>
<tr>
<td>5</td>
<td>2981600</td>
<td>3174825</td>
<td>6.5%</td>
<td>4468400</td>
<td>49.9%</td>
<td>49077600</td>
<td>1546.0%</td>
</tr>
<tr>
<td>6</td>
<td>4174250</td>
<td>4332600</td>
<td>3.8%</td>
<td>6314350</td>
<td>51.3%</td>
<td>137423100</td>
<td>3192.2%</td>
</tr>
<tr>
<td>7</td>
<td>5505250</td>
<td>6020075</td>
<td>9.4%</td>
<td>8597575</td>
<td>56.2%</td>
<td>353183750</td>
<td>6315.4%</td>
</tr>
<tr>
<td>8</td>
<td>7127175</td>
<td>7818875</td>
<td>9.7%</td>
<td>11403325</td>
<td>60.0%</td>
<td>857240350</td>
<td>11927.8%</td>
</tr>
<tr>
<td>9</td>
<td>9156525</td>
<td>10056775</td>
<td>9.8%</td>
<td>14820050</td>
<td>61.9%</td>
<td>1976854500</td>
<td>21489.6%</td>
</tr>
<tr>
<td>10</td>
<td>11631000</td>
<td>12569075</td>
<td>7.9%</td>
<td>18974430</td>
<td>62.9%</td>
<td>427826025</td>
<td>36619.9%</td>
</tr>
</tbody>
</table>

Table 5.8: Test time analysis table for scenario 4

The results for scenario 1 (table 5.5) shows that, both exhaustive search and heuristic gave exactly the same results with the lowest test time. The test-all method have test time little higher than the lowest value. But, the default TF is proven very costly by having highest test time which is significantly higher than other TFs. This can be explained as: a very large number of (untested) packages is needed for testing to get the desired output quantity of non-faulty products when the overall/compound manufacturing yield is low. Even if the default TF (only package test) spends the lowest amount of test time to test an individual CUT, the total test time consumption of this TF is very high due to the large number of CUTs needed to be tested in this scenario. The default TF is highly influenced by the amount of quantity to be tested. That is why, the default TF resulted in the highest test time overhead among all four TFs.

In scenario 2 (table 5.6), the heuristic has the same test time as the exhaustive search for high to medium yield range (n:2 to 5 with yield .99 to .7). But going further, the heuristic has 3.46% on average of the total test time overhead compared to the exhaustive search for higher stack sizes. This time overhead is due to additional active test instances under the heuristic compared to the exhaustive search. We illustrate it by describing a situation from scenario 2 with n=6 (marked by *) which is shown in figure 5.2. Figure 5.2 (a) gives an overview of how the TFs obtained from the heuristic and the exhaustive search look like and compares them in the number of active test instances. We can see that the exhaustive search has five active test instances (5, 6, 8, 10 and 12) whereas the heuristic has six active test instances (4, 6, 7, 8, 10 and 12). These additional and different test instances in the heuristic (compared to the exhaustive search) caused extra time overhead. Figure 5.2 (b) provide details of how the heuristic constructs its optimal TF iteratively by activating one additional test instance on top of default TF during each iteration. The newly activated test instances which contribute to optimizing test time are highlighted using grey boxes (iteration wise).

The default TF results in the lowest test time (the same for exhaustive/heuristic) for small stack size and very high yield range (n:2 to 3 with yield .99 and .9). For stack size 3 to 5 with medium to high yield values (0.6 to 0.8), the default TF shows
considerable growth in test time overhead which places it at the third place (but ahead of the test-all TF). Further onwards, as stack size increases and yield value drops below 0.5, the default TF proves to be most expensive among all. Again the low compound yield values can be blamed for this, as it affects the amount of CUTs to be tested (as explained in scenario 1).

Figure 5.2: (a) Test flow from exhaustive and heuristic (b) Test flow from heuristic (in details)

The test time analysis for scenario3 (table 5.7) shows that the heuristic follows the exhaustive search very closely with small test time overhead (2.3% on average). But the test-all and the default TFs are proven to be costly in all situations. Especially the default TF exhibits aggressive increment in test time overhead as the stack size increases, making this TF unsuitable to follow when the average manufacturing yield of dies are to be 50% or less. The sharp rise in input quantity for test due to lower compound yield (the same as scenario 1) can be held as accountable for this.

Scenario 4 (table 5.8) follows the same trend as scenario 3 for the test-all and the default TFs and making them unfavorable compared to the exhaustive search and the heuristic. The heuristic in this scenario shows test time overhead with the average of 9% against exhaustive search.
CHAPTER 5. RESULTS AND ANALYSIS

Figure 5.3 presents relative comparisons of the average test time overhead of each TF other than the exhaustive search. The exhaustive search is considered as the baseline (represented by x-axis baseline) for this comparison. The colored lines indicate the other TFs and present how much additional test time have been spent with respect to the test time of the exhaustive search. For each TF, we have taken the average values of the test time overhead from the four scenarios.

![Test Time comparison w.r.t Exhaustive method](image)

Figure 5.3: Relative comparison of average test time overhead (w.r.t exhaustive)

As a conclusion for this section, the heuristic approach proves itself to be significantly superior in test time minimization against the test-all TF and default TF. From the experimental results, the heuristic shows the test time overhead of 4.9% on average with respect to exhaustive search.

5.3 Execution Time

The execution time is referred as the amount of CPU time taken for finding an optimal TF under followed approach (exhaustive search or heuristic). We executed the exhaustive search and heuristic algorithms on a system which has 6 GigaByte of physical memory (RAM) and powered by an Intel ‘Xeon’ quad-core processor (model: W3550) running at 3 GHz as maximum clock frequency. We have recorded the start
and end time for both of the algorithms when executed separately on the same set of input data. The time duration obtained from differences of the two time stamps is considered as execution time (measured in seconds) of the followed algorithm. Table 5.9 presents the execution times obtained from experiments. In table 5.9, the first column (from left) indicates the size of the 3D-SICs (n=2:10). The second column presents the execution time taken by the exhaustive search. This column further divided down into five sub-columns. The first four sub-columns hold execution times of the exhaustive search under the four scenarios. The last/fifth sub-column presents the average execution time for the exhaustive search (from the four scenarios). The third column represents the execution times for the heuristic and these values can be read in similar manner as the exhaustive search (second column). The final column shows the (percentage) gain of the heuristic over the exhaustive search in average execution time.

Figure 5.4 presents the plot of average execution time comparison between the exhaustive search and the heuristic. The plot shows the execution time for exhaustive search increases exponentially compared to the heuristic as the 3D-SIC size increases. It could be explained as the number of TFs evaluated in exhaustive search is $2^{2n-1}$ whereas, the heuristic evaluates only $2n^2-n$ number of TFs.

<table>
<thead>
<tr>
<th>n</th>
<th>Execution time for Exhaustive search method (in seconds)</th>
<th>Execution time for Heuristic approach (in seconds)</th>
<th>% gain in Heuristic over Exhaustive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scenarios</td>
<td>Avg.</td>
<td>Scenarios</td>
</tr>
<tr>
<td>2</td>
<td>0.03 0.03 0.03 0.03</td>
<td>0.03</td>
<td>0.03 0.02 0.02 0.02</td>
</tr>
<tr>
<td>3</td>
<td>0.03 0.03 0.05 0.05</td>
<td>0.04</td>
<td>0.03 0.02 0.03 0.03</td>
</tr>
<tr>
<td>4</td>
<td>0.08 0.11 0.09 0.08</td>
<td>0.09</td>
<td>0.06 0.03 0.03 0.06</td>
</tr>
<tr>
<td>5</td>
<td>0.37 0.41 0.39 0.37</td>
<td>0.39</td>
<td>0.09 0.05 0.09 0.06</td>
</tr>
<tr>
<td>6</td>
<td>1.76 1.76 1.76 1.85</td>
<td>1.78</td>
<td>0.09 0.08 0.12 0.13</td>
</tr>
<tr>
<td>7</td>
<td>8.25 8.15 8.16 8.14</td>
<td>8.17</td>
<td>0.12 0.20 0.16 0.16</td>
</tr>
<tr>
<td>8</td>
<td>38.27 37.85 37.80 37.83</td>
<td>37.94</td>
<td>0.37 0.41 0.36 0.36</td>
</tr>
<tr>
<td>9</td>
<td>190.40 187.75 188.26 188.48</td>
<td>188.72</td>
<td>0.87 0.86 0.95 0.94</td>
</tr>
<tr>
<td>10</td>
<td>1500.92 1604.81 1550.26 1555.45</td>
<td>1552.86</td>
<td>2.85 3.03 2.87 2.90</td>
</tr>
</tbody>
</table>

Table 5.9: Average execution time for varying stack size
Figure 5.4: Execution time comparison (Exhaustive vs. Heuristic)
Chapter 6

Conclusion

This thesis project deals with the development of a technique to find the optimal test flow which minimizes the test time for TSV based 3D-SICs. The process of obtaining the optimal test flow is carried out in two phases. The first phase calculates the test time of any test flow. The second phase identifies the optimal test flow by comparing test times of the considered test flows.

The test time calculation uses the TI matrix model to calculate test time of any test flow. The TI matrix model is not only simple to use and implement but also capable of evaluating any test flow that is given with any input values. A motivational example (figure 3.2) highlights how variation in yield range affects the active test instances in the optimal test flow by evaluating three different scenario with different yield values. The optimal test flows obtained for the scenarios differ significantly from each other in combination of active test instances. This establishes the importance of adapting test flows to varying input parameters (especially to the manufacturing yield values of components and manufacturing steps, which might change over time in volume production) and evaluation of a large number of possible test flows while searching the optimal one.

A heuristic algorithm has been developed to determine a near optimal test flow with near minimum amount of test time spent on testing. The heuristic is compared with three alternatives for illustrating its efficiency in finding an optimal test flow with minimum test time. The three alternatives were: (a) an exhaustive search resulting in the actual optimal test flow, (b) a test flow with all test instances activated and (c) a default test flow with only the final package test. The experimental results show that the latter two test flows (b and c) were much behind in the competition. The exhaustive search is always able to produce the best results by suggesting the optimal test flow with minimum test time. But it takes significant amount of CPU time for larger stack size (hours when the stack size > 10). The heuristic, on the other hand, takes only a few seconds for small to medium stack size and a few minutes for larger stack size (stack size > 15) for finding a near optimal test flow. The test flow suggested by the heuristic may not be the optimal solution, But it competes very well with the test flow from exhaustive search with test time overhead of just 4.9% on average with four dif-
Therefore, if unlimited computation power is available, the exhaustive search method can be followed to get the best and most optimized test flow. But under real life circumstances, this situation cannot be achievable due to either bound design time (for meeting design deadlines) or nonexistence of such computational power. So, the heuristic approach can be followed for finding a near optimal test flow with small overhead in test time consumption.

6.1 Future Work

In this thesis, we have worked on a heuristic which resembles to a constructive greedy algorithm. It will be interesting to see the comparison of our proposed heuristic approach with other established heuristic approaches. So other heuristic approach such as genetic algorithm, simulated annealing, neighborhood search, etc. could be implemented and evaluated against our approach.

The TI matrix model is capable of handling calculations at individual component level. By exploiting this fact, one can consider intermediate test could be carried out separately on each individual component in the partial stack. A partial stack with two dies and one TSVs layer among them could be tested for (a) any one component (die1, die2 or TSVs), (b) any combination with two components (die1+die2, die1+TSVs or die2+TSVs), (c) all three components and (d) none. Scenario (c) and (d) has been implemented in this thesis. By adding (a) and (b) will expand the test flow space, but could accountable for producing more precise solutions.

The other directions to continue work on this thesis could be enhancing the cost model and inclusion of test architecture. In this thesis, we have calculated the test cost in terms of test time units. But more practical approach should be adapted by considering also the silicon cost and waste through the test flows, since the test flow has a very large impact on the number of KGDs that will be wasted during the manufacturing process.
Bibliography


Appendix A

Test Instance Matrix Model

Our objective is to allow the user to specify a test flow only in terms of selecting the test instances. The amounts of test time consumed in the specified test flow will then be generated by an algorithm. For this, we developed a mathematical model named as test instance matrix (TI matrix) for calculating the test time consumed by any given TF. The model is represented in a single column matrix of size $2n \times 1$ for a 3D-SIC with size $'n'$, where the user/designer needs to specify a TF by filling up the test instances. Briefly the model looks like as shown in figure A.1.

![Figure A.1: Test instance matrix](image)

In the TI matrix, each row represents one valid test instance. The rows are arranged from top to bottom according to the occurrence of the test instance through the manufacturing steps. All together there will be $2n$ rows. The very first row indicates the wafer sort for the first die and the last row describes the package test. Each element in the TI matrix lists the activeness of the test instances and use a binary value of '0' or '1' to represent it. A '0' represents a test instance (at the corresponding row) is disabled,
APPENDIX A. TEST INSTANCE MATRIX MODEL

so no test will be conducted in that instance. A ‘1’ represents that a test instance (at the corresponding row) is enabled and all the components available in that instance will be tested.

As different components in a 3D-SIC can have different manufacturing yields and affect the compound yield loss if they are listed for testing. So, in order to calculate the compound yield, we will decompose the initial TI matrix (figure A.1(a)) to a detailed TI matrix (figure A.1(b)).

![Initial TI matrix (a) and Detailed TI matrix (b)](image)

Figure A.2: Initial TI matrix (a) and Detailed TI matrix (b)

For a 'n' sized 3D-SIC, the corresponding detailed TI matrix will now have size of $2n \times 2n$. In this TI matrix, each column will be dedicated to one component in the 3D-SIC. The available components could be dies, interconnecting TSVs and final package. The columns will be arranged from the left to right as the inclusion order of components into the SIC during manufacturing process. As we know, there will be 'n' dies, 'n-1' layers of interconnects and one final package, hence all total $2n$ individual components represented by $2n$ columns. Each row of the detailed TI matrix represents one valid test instance (similar to one element of the initial TI matrix). The rows are arranged from top to bottom according to the available test instance through the manufacturing steps. In total there will be $2n$ rows with the very first row meant for wafer sort of the first die and the last row describes the package testing of completed but yet to test packages.

Each element in the TI matrix can have one of the three possible values from '0', '1' and 'NA'. A 'NA' value can be translated as the component (from respective column header) is not available in the SIC at the time of test instance (specified by the corresponding row number). A '1' represents that the test will be conducted on a component (specified by corresponding column header) in the test instance (specified by corresponding row number). Test exclusion can be indicated by '0'. So a row with all zeros and 'NA's describes that the test instance is disabled (similar to a '0' in initial TI matrix). Similarly, a column with all zeros and 'NA's says that the component, to whom the column belongs to has not been tested at all. A simplified version of TI with n=2 has been shown in figure A.3.
Each of the instantaneous snapshots of the TI matrix can be viewed as one test flow. Once the designer fills the TI matrix with 1s and 0s, the combination will be a given test flow. The output will show us the test time and efficiency for the given test flow. Also an alternate test flow will be suggested which will be the most effective one among all the test flows for provided properties (yield and individual test time consumption) of component set.

Now to calculate the total test time consumed by the given test flow, we will calculate some local parameters for each active test instance from the TI matrix. These local parameters are:

- $Q(i)$ is the quantity of components from each component category to be tested for instance $i$
- $T(i)$ is the test time for instance $i$

To calculate the test time in each instance, we can follow the task graph (figure A.4).

All the temporary matrices used by task graph will have dimension of $2 \times i$, where $i$ is the test instance. Each column of these matrices represents one component which is available in the stack at the time test instance I. The columns are arranged by the
order of entering components into the stack. The rows will represent other values and will be different for different states. Only the OUTPUT state has matrix of size $2n \times 3$. The details of these matrices will be described under each state of the task graph as follows:

**The GIVEN state:**

This state contains the parameters which should be given by the designer and manufacturer. The candidates in this state are:

1. **Component matrix** (figure A.5)

   This is a matrix with two rows. The first row holds the individual test time of the listed components. The second row indicates the manufacturing yield of the components.

   ![Component matrix](image)

   **Figure A.5: Component matrix**

2. Components to be tested: specifies which component or combination of components are need to be tested in this instance. So it is a copy of the $ith$ row from TI matrix till column length $i$. We will name this vector as $U$.

**The TEST PREPARATION state:**
This state consists of two matrices using which we will decide which components are prone to get damaged after a recent manufacturing step and are needed to be tested irrespective of the test flow. If those components are not listed among the testable components \((U)\) specified in the given test flow, then they need to be tested at any of the next available test instances.

1. **Test Summary matrix** (figure A.6)

The first row indicates that if a suspected component has been tested or not after the recent manufacturing step. We will consider only the newest components in the partial stack (top dies and top interconnects) are susceptible to get damaged during an intermediate stacking process. After wafer sort, the untested dies are the suspected components. Similarly, after the packaging step, the completed and packaged stack is the suspected component. The second row represents the number of times a component has been tested from the start till this instance. A component could be tested several times. We need to keep track of how many times each component has been tested until the current test instance.

![Figure A.6: Test summary matrix](image)

The \(f_{CX}\) can have value of ‘0’ or ‘1’. A ‘1’ indicates that the component has been tested after a recent manufacturing step during which the component is suspected for being damaged. A ‘0’ represents that the component has not been tested yet and it needs to be tested in a future instance.

The \(g_{CX}\) value incremented by one with each time the component \(cx\) has been tested and for an \(n\) sized test flow the value can be from 1 to \(n+1\).

2. **Test Forward matrix** (figure A.7)

The first row of this matrix represents whether a test is required for the component \((h_{CX} = 1)\) or not \((h_{CX} = 0)\) in the next test instance. The second row specifies at what yield \((y_{FWCX})\) the component will be tested if forwarded to the next test instance.

**The TESTING state:**

This state lists the components which will be under test in this test instance

*Testing matrix* (figure A.8)
APPENDIX A. TEST INSTANCE MATRIX MODEL

The first row indicates if a component will be tested or not. This value will be borrowed from the GIVEN states \( U \) vector. A '1' indicates confirmed test while a 0 indicates no test for the component. The second row specifies at which yield the component will be tested and represented by effective yield for that component \( y_{EFFC_X} \). If the component under test has skipped any previous test instance where it should have been tested then the forwarding yield value for that component needs to be taken into account while calculating the effective yield. Hence,

\[
y_{EFFC_X} = Y_{manufacturing \cdot YFWC_X} \tag{A.1}
\]

The **UPDATE state** :

The Test summary and the Test Forward matrices must be updated at each instances irrespective of a test is conducted or not. Because if no test has been performed in this instance then the untested and damage prone components must be mentioned and forwarded to the next test instance.

1. **Updated Test Summary matrix**

This is the same test summary matrix from TEST PREPARATION state with some of entries has been updated after TESTING state. If a test performed on components, then they were marked as tested in the first row and the number of times tested will be incremented by one.

\[
f'_{C_X} = 1 \quad \text{and} \quad g'_{C_X} = g_{C_X} + 1 , \quad \forall U_{C_X} = 1 \tag{A.2}
\]

2. **Updated Test Forward matrix**
This is also the same test forward matrix from TEST PREPARATION state with modified entries according to the test conducted at this instance. A component will be forwarded to be tested in the future instance if it has been marked as untested from last instance and remains untested during present test instance.

\[ h_{CX} = 1 \text{ if } U_{CX} = 0 \text{ and } f_{CX} = 0 \]  

Also if a component remains untested in present test instance, then its yield will decrease after another manufacturing event in future. So the effective yield of present instance should have forwarded to next so that the effective yield in next instance will be calculated from next manufacturing yield and present effective yield by using (equation A.1).

Each entry of the second row will be updated as follows:

\[ y_{FWCX} = y_{EFFCX} > \forall h_{CX} = 1 \text{ and } y_{FWCX} = 1 \forall h_{CX} = 1 \]  

**OUTPUT state**:

In the output state the test time will be calculated for the current instance. This state has one output matrix of size \(2n \times 2\). Each row will correspond to one iteration of the task graph. The \(i^{th}\) row stores all the local output values of instance \(i\). During each iteration of the task graph only one row \(i\) of the output matrix will be updated. The first and second columns represent the calculated values of test time \((T(i))\) and effective yield \((Y(i))\) respectively for current test instance.

Time spent on testing during this instance:

\[ T(i) = \sum t_{CX} : \forall U_{CX} = 1 \]  

The effective yield of the current test instance \(i\) will be \(Y(i)\)

\[ Y(i) = \prod Y_{EFFCX} : \forall U_{CX} = 1 \text{ and if } i = 1 \text{ or } i \text{ is even} \]  

If \(i\) is ’1’ means the test instance is a wafer sort of the first die and an even \(i\) indicates the wafer sort instance for subsequent dies or final package test. In all these cases the stacking yield will not play any role.

\[ Y(i) = \prod Y_{EFFCX} \cdot y_{stacking} : \forall U_{CX} = 1 \text{ and if } i = 1 \text{ or } i \text{ is odd} \]
The odd test instances from first intermediate test \((i=3)\) to final intermediate test \((2n-1)\), the partial stacks has underwent one stacking event each time. So in this situation the stacking yield will affect the effective yield of these particular test instances. We will iterate the task graph for each row of TI matrix. After traversing the TI matrix and calculating the local output parameters for each test instances, now we will calculate how much quantity of each component we will require for testing to get desired/specified quantity of final products. The desired quantity of final product could be specified by the designer.

**Quantity Calculation :**

We will use a quantity matrix with size same as output matrix to calculate the amount of components required for testing. Each row corresponds to one test instance. The first column holds the effective yield value of \(i^{th}\) iteration \((Y(i))\). The effective yield values could be copied directly from the OUTPUT matrix. The second and third column indicates the input quantity \((Q'(i))\) and output quantity \((Q(i))\) required for each instance.

While calculating the quantity, we will start from the last row. As the last row represents the package test, we will set the output quantity \((Q(2n))\) and determine how much input quantity \((Q'(2n):\) untested packages) is required during the package test when tested against the effective yield \((Y(2n))\) of this package test instance \((2n)\). Then we will move to previous instance \((2n-1)\) and calculate how many completed stacks (after final stacking event) are required \(Q'(2n-1)\) to be tested in order to produce \(Q'(2n)\) amount of tested stacks. Similarly we take one test instance prior to the current instance and calculate the amount of test input (number of components) required for each test instance till the first test instance.
So, \( Q(i) = Q'(i)/Y(i) \) \hspace{1cm} (A.8)

But at every stacking process, the input components will be partial stacks from previous stacking step and newly entered dies which will be at the top of the new stack. In this situation two of the previous instances output quantity must be same as required input quantity of current instance (\( Q(i-1) \) for newly entered dies and \( Q(i-2) \) for partial stacks from previous instance).

\[
Q(i-1) = \begin{cases} 
Q(i-2) = Q'(i), & \text{if } i > 1 \text{ and } i \text{ is even} \\
Q'(i) & \text{for all other instances}
\end{cases}
\]

\[TC = \sum Q'(i).T(i) \hspace{1cm} (A.9)\]

The following pseudo code and figure will summarize our methodology for calculating test time through the task graph,

**Part 1: Calculating instantaneous output parameters (by iterating the task graph)**

For all instances \( i \), where \( i = 1, 2, 2n \)

Given \( i \) state:

Take the components available to instance \( i \)
Take \( U \) vector, components to be tested specified by designer
\( U = i^{th} \) row of TI matrix

Test Preparation\( (i) \) state:
APPENDIX A. TEST INSTANCE MATRIX MODEL

Prepare test summary matrix
Add one column with zeros for each newly entered component to the stack

Prepare test forward matrix
Add one column for each newly entered component to the stack
The values will be NA and 1 for the first and second row respectively

Testing (i) state:

Prepare testing matrix
First row : copy the $U$ vector i.e. components to be tested (as asked by designer)
Second row : calculate effective yield for each component to be tested
for each $U_{CX} = 1$, effective yield, $y_{EFFCX} = $ manufacturing yield of component($y_{CX}$) .
forwarded yield of the component

Test Update (i) state:

Updating the test summary matrix
Mark all the components that been tested during this instance
First row : $f_{CX} = 1$ if $U_{CX} = 1$
Count how many times the component has been tested since beginning up to this instance (including this instance)
Second row : $g_{CX} = g_{CX} + 1$ if $U_{CX} = 1$
The updated test summary matrix will be the test summary matrix for next instance

Updating test forward matrix
If an untested component remains untested, then it needs to be tested in future
So, first row : $h_{CX} = 1$ iff $U_{CX} = 0$ and $f_{CX} = 0$
Second row : if $h_{CX} = 0$ then forwarding yield $y_{FWCX} = 1$
else forwarding yield $y_{FWCX} = y_{EFFCX}$
The updated test forward matrix will be the test forward matrix for next instance ($i+1$)

Output (i) state :

Calculate the test time of instance $i$, which will be the sum of test times of all the components tested during this instance
$T(i) = \sum t_{CX}$ , where all $U_{CX} = 1$
Calculate the effective yield of the instance $i$
For all instance after a stacking event (if $i > 1$ and $i$ is odd)
$Y(i) = \prod y_{EFFCX} \cdot y_{stacking}$, where all $U_{CX} = 1$
For all wafer sorts and package test ($i$ is even)
$Y(i) = \prod y_{EFFCX} \cdot$,where all $U_{CX}= 1$

Part 2 : Calculating the quantity of components required to test for getting desired output
Quantity \((Q)\)

Start with \(Q_{2n} = Q\)

For all instances \(i\), where \(i = 2n, 2n-1, 2, 1\)

Update \(i^{th}\) row of quantity matrix:

Column1: copy effective yield for this instance \(Y(i)\)

Column2: calculate required quantity to test \((Q(i)^{''})\) for getting \(Q(i)\) quantities as

Output of this instance: \(Q(i)^{''} = Q(i) \cdot Y(i)^{''}\)

Column3: update the required quantities of this instance as the output quantity for next instance

If empty then set

\(Q(i-1) = Q(i-2) = Q(i)^{''}\), for all instance after intermediate stacking \((i>1\ and\ i\ is\ odd)\)

\(Q(i) = Q(i-1)\), for all other instances

**Part 3 : Calculating the total test time for the given TF**

For all instances \(i\), where \(i = 1, 2, 2n\)

Calculate total instantaneous test time of \(i\)

Product of quantity required for testing and instantaneous test time

Total test time = sum of instantaneous test time

\(T = \sum Q(i) \cdot T(i)\) figure A.12

![Diagram](image)

Figure A.12: Test time calculation