Utah State University

DigitalCommons@USU

All Graduate Theses and Dissertations

Graduate Studies

5-2010

Micronetworking: Reliable Communication on 3D Integrated Circuits

Andres A. Contreras
Utah State University

Follow this and additional works at: <https://digitalcommons.usu.edu/etd>

Part of the [Electrical and Electronics Commons](#)

Recommended Citation

Contreras, Andres A., "Micronetworking: Reliable Communication on 3D Integrated Circuits" (2010). *All Graduate Theses and Dissertations*. 728.
<https://digitalcommons.usu.edu/etd/728>

This Thesis is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.

MICRONETWORKING: RELIABLE COMMUNICATION
ON 3D INTEGRATED CIRCUITS

by

Andres A. Contreras

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Dr. Todd K. Moon
Major Professor

Dr. Jacob H. Gunther
Committee Member

Dr. Reyhan Baktur
Committee Member

Dr. Byron R. Burnham
Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2010

Copyright © Andres A. Contreras 2010

All Rights Reserved

Abstract

Micronetworking: Reliable Communication
on 3D Integrated Circuits

by

Andres A. Contreras, Master of Science
Utah State University, 2010

Major Professor: Dr. Todd K. Moon
Department: Electrical and Computer Engineering

The potential failure in through-silicon vias (TSVs) still poses a challenge in trying to extend the useful life of a 3D integrated circuit (IC). A model is proposed to mitigate the communication problem in 3D integrated circuits caused by the breaks at the TSVs. We provide the details of a low-complexity network that takes advantages of redundant TSVs to make it possible to re-route around breaks and maintain effective communication between layers. Different configurations for the micronetwork are analyzed and discussed. We also present an evaluation of the micronetwork's performance, which turns out to be quite promising, based on several Monte Carlo simulations. Finally, we provide some directions for future research on the subject.

(97 pages)

Acknowledgments

The completion of this thesis has been possible thanks to the help and contribution of many. I would like to acknowledge their extraordinary support in making this research a reality. First, I would like to thank the government of the Dominican Republic and the Ministry of Higher Education for giving me the opportunity to come to study at Utah State University.

I would like to thank my major advisor, Dr. Moon, for giving me the opportunity to work under his supervision and guidance. I am also thankful to Dr. Gunther, who was a co-advisor during the first part of this research work. In addition, I am grateful to Dr. Baktur for being part of my committee. I am truly grateful for all their teachings inside and outside of the classroom. I would like to thank the many other professors who contributed to my education at Utah State University, in particular Dr. James Powell and Dr. Donald Cripps. I would also like to acknowledge the help and support from Kathy Bayn, who was more than an advisor to me.

I would like to thank my friends and family at Utah State University who gave the necessary support to make this thesis a reality. Among them I have to mention my beloved girlfriend, Lori Diaz, and my friends Marcos Chalas, Ricardo Estevez, Alberto Garcia, Vaibhav Ghadiok, Eduardo Monzon, Augusto Rodriguez, Rohan Sreeram, Erick Tejada, Abiezer Tejada, Roger West, and Elias Valdez. I have to give special thanks to Franyell Silfa, who helped me with the development of the code and the process of learning Java, and to Manuel Diaz, who helped me clarify my ideas and edited my writing. I also would like to thank my friend Omar Rodriguez for taking the time to help me with the errands of the thesis correction process.

Finally, I would like to thank my family for always standing by my decisions and having faith in me. Especially, my mother, Elizabeth Reyes, and my father, Rafael Contreras.

Andres A. Contreras

Contents

	Page
Abstract	iii
Acknowledgments	iv
List of Figures	vii
1 Introduction	1
1.1 Background Work	4
1.1.1 Alternative	4
1.1.2 Related Work	4
1.2 Thesis Overview	6
1.3 Main Features	6
2 Micronetwork Description	8
2.1 Nodes Description	10
2.2 Dealing with Breaks	10
2.3 Node Logic	13
2.3.1 Non-redundant Sender	15
2.3.2 Redundant Sender	16
2.3.3 Non-redundant Receiver	17
2.3.4 Redundant Receiver	19
2.4 Node Internal Structure	19
3 Performance Evaluation	26
3.1 Relation Between Via Failure and Layer Failure	26
3.2 Performance Under Multiple Breaks	31
3.3 Delay Analysis	34
4 On-layer Interconnection	44
5 Conclusions	53
5.1 Summary	53
5.2 Future Work	54
References	58
Appendix	61
Code for the Nodes Processing Logic	62

List of Figures

Figure	Page
1.1 A vision of future 3D hyper-integration of infotech, nanotech, and biotech systems - a new paradigm for future technologies.	2
1.2 IBM cools 3D chips with H ₂ O.	4
2.1 Schematic representation of a micronetwork.	8
2.2 3D representation of a micronetwork.	9
2.3 Node representation with $m = 4$	11
2.4 Dealing with breaks example. Layers A and B are both on the XY-plane and the TSVs are in the Z-direction.	12
2.5 Internal architecture of a node.	21
2.6 Sender internal structure with switches separated into groups	23
3.1 Simple abstraction of a multilayer chip.	26
3.2 Layer failure probability vs. single via failure probability, $\frac{R}{N}$ constant.	29
3.3 Layer failure probability vs. single via failure probability, $\frac{R}{N}$ constant, logarithmic scale.	29
3.4 Layer failure probability vs. single via failure probability, N constant.	30
3.5 Layer failure probability vs. single via failure probability, N constant, logarithmic scale.	31
3.6 Layer failure probability vs. percentage of redundancies.	32
3.7 Layer failure probability vs. percentage of redundancies, Y-log scale.	32
3.8 Layer failure probability vs. percentage of redundancies, fixed α	33
3.9 Performance curve.	34

3.10	Screenshot of second simulator menu.	35
3.11	Simulator screenshot of an even distribution, group size = 1.	36
3.12	Simulator screenshot of an even distribution, group size = 2.	36
3.13	Delay analysis, L=10, B=20, R=20, I=10, default.	39
3.14	Delay analysis, L=10, B=20, R=20, I=10, random 1.	40
3.15	Delay analysis, L=10, B=20, R=20, I=10, random 2.	40
3.16	Delay analysis, L=10, B=20, R=20, I=10, random.	41
3.17	Delay analysis, L=10, B=20, R=10, I=10, average delay.	41
3.18	Delay analysis, L=10, B=20, R=10, I=10, max delay.	42
3.19	Delay analysis, L=10, B=20, R=10, I=10, average max delay.	42
3.20	Delay analysis, L=10, B=20, R=20, I=10, even 1.	43
3.21	Delay analysis, L=10, B=20, R=20, I=10, even 2.	43
4.1	On layer distribution, case 1.	46
4.2	Modified nodes interconnection.	46
4.3	Modified interconnection, example of successful routing.	48
4.4	Modified interconnection, example of a routing failure.	48
4.5	Delay analysis, two layers, single iteration.	50
4.6	Delay analysis, two layers, single iteration, bursty.	50
4.7	Modified nodes interconnection.	51
5.1	Example of bidirectional routing.	57

Chapter 1

Introduction

Integrated circuits (ICs) are the building blocks of many of the technological advances seen nowadays. Moore's law drives improvement, but physical limits are being reached [1]. A major determining factor in the performance of an IC is wiring, which is closely related to communication time and to physical proximity [2]. In order to continue increasing packaging density, integrated circuit design is moving to the third dimension. Great time and effort is now being devoted to find a way to stack two or more layers of a semiconductor device to create a 3D integrated circuit. This next step in the evolution of the architecture of integrated circuits will allow expedited information processing and reduced power consumption.

Another major advantage of 3D integrated circuits is the possibility of heterogeneous integration. They give the flexibility to build different components individually, perhaps under incompatible processes, and later integrate them into a single chip [3,4]. Figure 1.1 illustrates this idea [4]. By replacing chip-to-chip interconnections by intra-chip interconnections, performance, power, reliability, and portability are improved [5]. The high interconnection of the layers creates new possibilities for large caches and massively parallel designs.

A proposed technique for communication between layers is through silicon vias (TSVs), in which fine copper, or tungsten, interconnecting wires are produced between the silicon layers, making connection with the metal layers on each silicon slice [6,7]. TSVs could be formed either before or after staking the different layers [8]. TSV technology results in significantly shorter wires than conventional planar

Fig. 1.1: A vision of future 3D hyper-integration of infotech, nanotech, and biotech systems - a new paradigm for future technologies.

interconnections, resulting in lower power requirements and simpler wire buffers.

Nevertheless, in order to take advantage of the full benefits of 3D integrated circuits, there are several problems that must be addressed [9]. For instance, the potential failure in through-silicon vias still presents a challenge in trying to extend the useful life of a 3D integrated circuit [10, 11]. The failures may be the result of thermal stresses acting on the silicon [12], or as a result of a manufacturing defect. 3D assemblies are difficult to cool adequately and thermally induced mechanical stresses or breakdown due to age can break TSV connections with metal layers on the silicon slices. Furthermore, the TSV manufacturing process is not yet perfected, resulting in low chip yield.

Even a single break in a via compromises the functionality of the 3D integrated circuit, because the circuit is no longer able to accurately transmit information. It is also possible that different portions of a TSV are functional across different layers, resulting in a spatially intermittent connection. Since the open connections may be thermally induced, there may be temporal intermittence as well. The failure of one or a few TSV interconnections still leaves potentially thousands of effective connections

which could still be operable.

Another issue is that 3D packaging makes test difficult. Test techniques and design for testability solutions for 3D ICs have remained largely unexplored [13]. There must be sufficient confidence in the design and manufacturing that an extremely complicated chip can be used, even when much of it is not available to test. The interlayer communications system needs to be sufficiently robust that without direct test access it can be relied upon with high confidence.

What is needed to address these problems is a mechanism to efficiently re-route data through TSVs which are still operating, ensuring reliable overall throughput. This may be achieved by adding additional TSVs and equipping each silicon layer with circuitry, interfacing between the metal layer and the TSVs on the layer, to route the data among the TSVs. We propose the introduction of routing nodes and redundant TSVs to create a micronetwork in which information is re-routed around broken TSVs, and then re-routed back to the initial via position [14].

One of the approaches to deal with the problem of high temperature in 3D ICs is the use of thermal TSVs. By lowering the thermal resistance between different device layers [15], it is possible to create a thermal path from the hot areas to the heat sinks [16]. Even though thermal vias are beneficial, they take up valuable routing space, which can be up to 10-20% of total chip area [16]. As a result, algorithms are needed to minimize their usage [17].

Another approach is the use of microchannel cooling as presented by Sekar (2008) [18]. They discuss the possibility of using cooling fluid that could flow through tubes on the back side of the 3D stack or using fluidic channels on the substrate. IBM also presents a similar solution. They implement a cooling layer that is placed between other active layers [19]. IBM uses water as the cooling fluid. Figure 1.2 shows the concept.

Fig. 1.2: IBM cools 3D chips with H₂O.

1.1 Background Work

1.1.1 Alternative

Instead of routing the information around broken TSVs, a conventional binary error correction code in an erasure setting could be implemented. This has the benefits of a wide range of choices and years of supporting research. Moreover, it will avoid the re-routing delays due to the increment in wire length introduced by the micronetwork. However, this approach will require decoding in every layer. The decoding will require information from all the TSVs, which is a disadvantage when compared to the localized logic of the micronetwork. In addition, the decoding could take several clock cycles. The decoding overhead could be significantly greater than the routing latency.

1.1.2 Related Work

3D integrated circuits are a relatively new concept and there are still many aspects of it that have not been fully explored. For instance, the problem of the breaks at the through-silicon vias is mostly approached by looking at how to avoid

the breaks, but little work exists about how to keep the system functioning when breaks occur. The 3D crossbar proposed by Nomura (2006) [2] was early work found to be somewhat related. Despite the fact that this work is not presented as a way to deal with the breaks, it could be adapted so that the switches can handle the breaks at the TSVs. A routing performed by the 3D crossbar could potentially achieve an optimal routing path, where optimality could be measured as a function of the delay or the number of bits that are perturbed by the rerouting. Moreover, the crossbar could find the optimal path not only between adjacent layers, but also between any two layers in the 3D integrated circuit. Nevertheless, implementing this crossbar requires a great amount of resources when compared to the micronetwork approach. We believe that the extra complexity is not justified by the potential optimality. This could change as technology and needs evolves, but it might not be in the near future.

More recently, we became aware of the work presented by Kang (2009) [20]. The concept presented in that paper is similar to what is being presented in this thesis. Another similar work is presented by Loi (2008) [21]. However, the node architecture and the routing protocol that we are proposing are different. For instance, we do not require a centralized controller; this reduces the system complexity considerably. Another difference is that in Kang’s approach, a single node is directly connected to multiple vias, whereas in our approach we connect each node to a single via and interconnect the nodes so that they can reach multiple vias indirectly through other nodes. A major difference from Loi’s approach is that they do not account for multiple failures in the same group, which means that the routing is limited to a more specific failure pattern. Moreover, their recovery scheme can only deal with failures discovered during the testing stage.

1.2 Thesis Overview

This thesis presents a model to deal with the breaks at the TSV in 3D integrated circuits. The thesis has been divided into five chapters. In this chapter, Chapter 1, we present the motivation for this work and examine the related work. Chapter 2 presents a detailed description of the micronetwork idea, the routing logic for the nodes, and some aspects about the internal structure of the nodes. Chapter 3 presents the results concerning the evaluation of the performance of the micronetwork. Performance was evaluated by looking at the number of breaks the micronetwork can handle effectively and what is the delay introduced by the micronetwork. Chapter 4 discusses the feasibility of an alternative on-layer interconnection of the nodes. Finally, Chapter 5 presents the conclusion and some ideas for future work.

1.3 Main Features

The overall goal of the research is to design a micronetwork with a low complexity routing protocol, which will serve as an error correction circuitry for 3D integrated circuits. Below is the list of the main features that the micronetwork possesses.

- The nodes circuitry has small real estate on the silicon. This is very important since neither power nor functionality requirements allow for complicated routing circuitry on each layer.
- The latency introduced by the routing should be effectively minimized by properly placing the redundant TSVs on the layer. As a result, the interchip communication is only minimally impacted.
- The nodes are able to reconfigure as the connections change. This is an important feature because connections may change due to thermal stresses, making necessary for reconfiguration to be on the fly, and not just during chip startup.

- Routing decisions are mainly local and the nodes only require information that is available on the same layer that they are found. This avoids the need of interlayer traffic for the purpose of carrying routing information.

Finally, it is important to mention that the scope of this work is somewhat abstract. The performance is evaluated with the use of simulations, which are not at the transistor level. This means that no particular fabrication technology has been considered. This thesis focuses on the behavioral description of the micronetwork.

Chapter 2

Micronetwork Description

The micronetwork in each layer consists of two sets of nodes: receivers and senders. The receivers are those nodes within a particular layer that will receive information from the senders of an adjacent layer. Receivers and senders can each be classified as redundant or non-redundant. Non-redundant nodes are those located at non-redundant vias. Redundant nodes are used to transmit information if the TSV to which a non-redundant node is attached suffers a break. Every node in the 3D integrated circuit thus belongs to one of four categories: redundant receiver (RR), redundant sender (RS), non-redundant receiver (NR), and non-redundant sender (NS). See figs. 2.1 and 2.2.

Fig. 2.1: Schematic representation of a micronetwork.

Fig. 2.2: 3D representation of a micronetwork.

Figure 2.2 gives a better idea of the concept described in fig. 2.1. Nevertheless, they are both describing the same system. For this particular case we are representing the micronetwork in a 3D integrated circuit which has three layers, four non-redundant through-silicon vias, and four redundant through-silicon vias (only two are shown). Communication is assumed to flow from the bottom layer, which we call layer A, to the middle layer, referred to as layer B, and from the middle layer to the top layer, called layer C. Notice that this labels are not explicitly shown in the figure. This is why layer A only has nodes of the type sender and layer C only has nodes of the type receiver, whereas layer B has both senders and receivers. In the figure the blue nodes are non-redundant, the yellow nodes represent redundant nodes, the gray planes represent the layer area (it is not shown on layer C to allow a better appreciation of the details), and the cylinders are the through-silicon vias. The different colors in the through-silicon vias are only due to aesthetic reasons.

2.1 Nodes Description

Nodes can be modeled as simple input/output devices with one main input/output pair to send bits of information from one layer to the next. In addition, a node has m other input/output pairs that can be used for intra-layer communication, which is essential to deal with breaks. As a result, there are a total of $n = m + 1$ input/output pairs. Figure 2.3 is a schematic representation of a node for the case $m = 4$. The figure shows a sender and a receiver. Notice that they look pretty similar. In fact, the receiver is a mirror image of the sender.

2.2 Dealing with Breaks

The proposed fault-tolerant algorithm will now be illustrated with an example in which six bits of information are transmitted between two layers, denoted A and B. Figure 2.4 is a pictorial representation of this example. For this discussion, unidirectional communication from layer A to layer B will be assumed. Moreover, the communication system will have two extra through-silicon vias in order to tolerate two breaks. This means that there will be a total of eight through-silicon vias between layers A and B. Layer A has a total of eight nodes, six of which are NS and the remaining two are RS. Likewise, layer B has eight nodes, but these are receivers (NR and RR) as opposed to senders. Senders in layer A are assumed to be numbered from left to right in the following manner: NS1, NS2, $\dots$, NS6, RS1, and RS2. Thus, RS1 is to the right of NS6. Receivers in layer B are numbered in a similar fashion.

Suppose now that two breaks occur in the through-silicon vias, specifically in those connecting NS2 and NS5 to NR2 and NR5, respectively. In order to transmit the six bits from layer A to layer B, each bit is placed at the main input of a non-redundant sender and is normally expected to exit through the main output of its corresponding receiver. The second and fifth bits, however, will not be directly transmitted to their

Fig. 2.3: Node representation with $m = 4$.

Fig. 2.4: Dealing with breaks example. Layers A and B are both on the XY-plane and the TSVs are in the Z-direction.

corresponding non-redundant receiver because of the break.

In order to successfully transmit the information across the layers, some of the non-redundant nodes must redirect their bits through the redundant nodes. First note that NS1 remains unaffected, while NS2 will pass its bit of information (b2) to NS3 through the intra-layer channel, and NS3 will send this bit (b2) to NR3 through its main output. Moreover, the bit that NS3 receives through its main input (b3) is passed along to NS4, again through the intra-layer. NS4 will behave similar to NS3; it sends the bit b3 through its main output and the bit b4 to NS5 through the intra-layer channel. NS5, which has a break in its main output, will pass along to NS6 two bits, b4 and b5. So, NS6 will receive a total of three bits, two bits through the intra-layer channels and one at its main input. This node will transmit the fourth bit (b4) through its main output and pass the others two along to RS1. Finally, RS1 will output the fifth bit through its main output and the sixth bit to RS2, which transmits this last bit to RR2.

The receivers in layer B will operate in a similar manner to reverse the action of the senders in layer A in order to align the bits with their original output positions. This idea can be extended to any number of layers with any number of bits. More-

over, any node could be set, in real time, to behave as non-redundant or redundant. The only requirement is that the number of breaks does not exceed the number of redundancies or intra-layer channels.

It is important to note that intra-layer vias are assumed to remain intact throughout the operational life of the integrated circuit. This is due to the extremely small probability of these communication links breaking when functioning under the conditions specified by the manufacturer of the circuit.

2.3 Node Logic

In order to describe the logic of the different nodes, the input conditions will be represented as an ordered n -tuple in the following way:

$$\mathbf{I} = (a_1, a_2, \dots, a_n),$$

where a_1 represents the current state of the principal input to the node and the others a_i , $2 \leq i \leq n$ represent the current state of the intra-layer inputs. The values assumed by these entries are either 1 or 0, where a value of 1 is used to represent that information is being sent through that particular input and a 0 is used to represent a high impedance state, (i.e., not information is being sent). It is important to recognize that these entries do not represent the value of the actual bits being transmitted but the condition of the transmission, namely if it is active or not. Moreover, a_2 corresponds to the first intra-layer input, a_3 to the second input, and a_i to the $i - 1$ input.

From the way in which communication between nodes was described in the previous section, it is easily seen that there is an order associated with the a_i representing the state of the intra-layer inputs. Particularly, if j bits are being received by a node from an adjacent one, then the first j intra-layer inputs, represented by

$a_2, a_3, \dots, a_{j+1}$, will be in use, and the terms listed will all assume a value of 1 in $\mathbf{I}$.

It is also convenient to use an ordered n -tuple $\mathbf{O}$ to represent the output conditions of a node. Thus,

$$\mathbf{O} = (b_1, b_2, \dots, b_n),$$

where b_1 represents the current state of the principal output of the node and the other b_i represent the current state of the intra-layer outputs. As before, the values assumed by these entries are either 1 or 0. It should also be noted that $\mathbf{O}$, like $\mathbf{I}$, also has an order associated with its entries. Moreover, $\mathbf{O}$ is determined by $\mathbf{I}$ and they both depend on the type of node represented, as will be illustrated next.

For the node to be able to properly interconnect its inputs with its outputs, it must identify whether or not there is information at any of its intra-layer inputs. This will be done by setting the unused intra-layer channels to a high-impedance state. When a node needs to send information through any of its intra-layer outputs, it will remove the high-impedance state from those outputs. This change in the state of the intra-layer channels will be sensed by the right-adjacent node in the case it is a sender (left-adjacent in the case it is a receiver). Once a node senses a change in any of its inputs, it will know that it is necessary to change its internal configuration in order to properly interconnect its inputs with its outputs. It has been assumed that the nodes obtain this information either from an external signal (coming from an adjacent node) or by the use of an embedded sensing mechanism. Moreover, it is necessary to sense breaks at both sides of the TSVs. The shared detection of the states of the TSVs contributes to overcome the need of interlayer communication. The state of the TSV will be determined with the help of a sensing mechanism. The characteristic of the sensing mechanism is something that needs to be researched, and it is not addressed in this thesis.

2.3.1 Non-redundant Sender

Since the main objective of a NS is to transmit information to another layer, it is expected that these nodes will be actively receiving information at their main input. As a result, a_1 , which represents the status of the principal input for this node, will be equal to 1. The status of the other inputs, namely the intra-layer inputs, will be determined by the state of the intra-layer outputs of the left-adjacent node and the following condition: if $a_i = 1$ for $i > 2$, it is necessary that $a_{i-1} = 1$.

Once the status of the inputs is fixed for a particular NS, the status of its outputs can be easily determined. The behavior of a non-redundant sender will be greatly affected by the status of its principal output, namely if it has a break in its through-silicon via or not. The former case, in which there is no break, will be considered first.

If the main output's via is intact, the NS is sending bits of information over to the next layer. As a result, the output n -tuple will be as follows:

$$\mathbf{O} = (a_k, a_1, \dots, a_{k-1}, 0, \dots, 0),$$

where a_k is the rightmost element of $\mathbf{I}$ with a value of 1 and $a_i = 1$ for $1 \leq i \leq k \leq n$. This assumes that some of the intra-layer inputs, specifically $n - k$, are not being utilized. In the event that all of the intra-layer inputs are utilized, $\mathbf{O}$ will have the following form:

$$\mathbf{O} = (a_n, a_1, \dots, a_{n-1}).$$

Let us now consider the case where the main output has a break in its through-silicon via. The output n -tuple will look as follows:

$$\mathbf{O} = (0, a_1, a_2, \dots, a_k, 0, \dots, 0).$$

As before, this assumes that $n - k$ intra-layer inputs are not being utilized. It is also important to notice that all the intra-layer inputs could not be in use because when the main output is broken the output vector can handle at most $n - 1$ non-zero elements. Thus, it will be only possible to handle the main input and $n - 2$ of the intra-layer inputs, and the output vector would have the following form:

$$\mathbf{O} = (0, a_1, \dots, a_{n-1}).$$

2.3.2 Redundant Sender

These nodes behave in a very similar manner to the non-redundant senders. The only difference is that they do not receive any information at their main input. As a result, a_1 will always be equal to zero and the input n -tuple is:

$$\mathbf{I} = (0, a_2, a_3, \dots, a_n),$$

where, as before, a_i could be either zero or one.

Now, the form of the output vector will be considered. The first case to be analyzed is when the main output's via is intact. For this case the output vector looks as follows:

$$\mathbf{O} = (a_k, a_2, \dots, a_{k-1}, 0, \dots, 0),$$

if $n - k$ intra-layer inputs are not being utilized, or

$$\mathbf{O} = (a_n, a_2, \dots, a_{n-1}, 0),$$

if all the intra-layer inputs are being utilized.

The remaining case is when the main output has a break in its through-silicon via, resulting in the following form of the output vector:

$$\mathbf{O} = (0, a_2, \dots, a_{k-1}, \dots, a_k, 0, , 0),$$

if $n - k$ intra-layer inputs are not being utilized, or

$$\mathbf{O} = (0, a_2, \dots, a_{n-1}, a_n),$$

if all the intra-layer inputs are being utilized.

Both non-redundant and redundant senders are responsible for assuring that information sent from a layer successfully arrives to the adjacent layer. However, in the case of a break, some of the bits of information will not arrive to the same location in the adjacent layer that they would have if there were no breaks at all. Therefore, it is necessary to have a mechanism that restores the arriving bits to the position they would have originally arrived at in the event that there were no breaks. This restoring process is done by the receivers at each layer. The functions of the receivers will be explained by dividing these into non-redundant and redundant receivers, as it was done for the senders.

2.3.3 Non-redundant Receiver

The non-redundant receiver is very similar to the non-redundant sender. However, since the main objective of a NR is to receive information from another layer, it is expected that these nodes will be actively sending information through their main output. As a result, b_1 , which represents the status of the principal output for this node, will be equal to 1. The status of the other outputs, namely the intra-layer

outputs, will be determined by the state of the intra-layer inputs of the right-adjacent node. Nevertheless, if $b_i = 1$ for $i > 1$, it is necessary that $b_{i-1} = 1$.

Once the status of the inputs is fixed for a particular NR, the status of its outputs can be easily determined. The behavior of a non-redundant sender will be greatly affected by the status of its principal input, namely if it has a break in its through-silicon via or not. The first case, in which there is no break, will be considered first.

If the main input's via is intact, the NR is receiving bits of information from the previous layer. As a result, the output n -tuple will be as follows:

$$\mathbf{O} = (a_k, a_1, a_2, a_3, \dots, a_{k-1}, 0, \dots, 0),$$

where a_k is the rightmost element of $\mathbf{I}$ with a value of 1. This assumes that some of the intra-layer inputs, specifically $n - k$, are not being utilized. In the event that all of the intra-layer inputs are utilized, $\mathbf{O}$ will have the following form:

$$\mathbf{O} = (a_n, a_1, a_2, \dots, a_{n-1}).$$

If none of the intra-layer inputs are being utilized the output vector will look as follows:

$$\mathbf{O} = (a_1, 0, 0, \dots, 0),$$

where the value of each a_i is equal to 1 for all the previous cases.

In the event that the main input's through-silicon via is broken, the output vector will have the following form:

$$\mathbf{O} = (a_k, a_2, a_3, \dots, a_{k-1}, 0, \dots, 0), \quad 1 < k \leq n.$$

Note that if the main input's through-silicon via is broken at least one of the intra-layer inputs has to be utilized.

2.3.4 Redundant Receiver

These nodes behave in a very similar manner to the non-redundant receivers. The only difference is that they do not send any information through their main output. As before, their output vector can be analyzed by looking at whether or not their main input's through-silicon via is broken.

First, the case in which the through-silicon via is not broken will be described. For this case the output vector will look as follows:

$$\mathbf{O} = (0, a_1, a_2, \dots, a_{k-1}, a_k, 0, \dots, 0).$$

For the case where the through-silicon via is broken the vector will be:

$$\mathbf{O} = (0, a_2, a_3, \dots, a_{k-1}, a_k, 0, \dots, 0).$$

2.4 Node Internal Structure

After exploring the internal logic of the nodes it is possible to see that the key concept consists in properly connecting the nodes' inputs to the outputs. There are, possibly, several ways in which we can create the internal structure for a node to accomplish this task. For instance, implementing the nodes as $n \times n$ cross-bars is one possibility. This will, in fact, allow connecting the inputs to the outputs in any possible configuration. However, cross-bars are very costly to implement in terms of the number of switching elements they require, which is $\mathcal{O}(n^2)$. Moreover, a full cross-bar implementation would not fully exploit the structure of the nodes routing protocol.

If we analyze the fact that for the intra-layer inputs we have the property that $a_i = 1$ only if $a_{i-1} = 1$ and, similarly, for the intra-layer outputs $b_i = 1$ only if $b_{i-1} = 1$, it is possible to identify that only a certain number of connections between the inputs and the outputs are allowed. As a result, it is not necessary to have a full cross-bar for the nodes to effectively reroute. Figure 2.5 shows a simple circuit that can be used to achieve the functionality of a node at a much lower cost. The number of switching elements required for this configuration is $\mathcal{O}(n)$, specifically $4n + 2$. This is considerably better than the full cross-bar in terms of complexity. The figure shows both a sender and a receiver. Note that, as we mentioned before, they are mirror images of each other. This configuration makes it possible for a given node to behave as either non-redundant or redundant. Nevertheless, an external signal will be required to select the type of behavior.

The decision to close and open the switches are based on three parameters: the status of the through-silicon via (i.e., whether or not it is broken), the type of node (redundant or non-redundant), and the information available at the inputs. The status of the TSV could be obtained by measuring its impedance, since the impedance is expected to change when the TSV breaks. Now, we consider three possible ways to detect if information is available at the inputs. The first one is by using 3-state switches at the outputs and implementing a way to sense the high impedance state at the inputs. For example, suppose that the outputs of Node A are connected to the inputs of Node B. The sensing mechanism in Node B will detect the status of the outputs of Node A.

The second approach is to double the number of intra-layer outputs and pair them in groups of two, so that one wire carries the information while the other wire signals that information is being carried. The third approach is similar to the second approach, but instead of doubling the number of intra-layer outputs only k

Fig. 2.5: Internal architecture of a node.

additional intra-layer outputs are introduced. The value of k is chosen such that $2^k \geq m$, where m is the same as defined in the nodes description section. So, instead of signaling which specific outputs are being used we only indicate how many outputs are being utilized. This is possible, again, because of the order associated with the rerouting process. This approach represents a substantial save in terms of the number of outputs. Nevertheless, it will require additional circuitry at the inputs of the adjacent node to interpret the received signal.

In order to control a given switch, only the status of some of the inputs, at most three, are required to be known. For instance, to control the switch at the output b_3 it is only necessary to know the status of the inputs a_2 , a_3 , and a_4 , even in the case when there are more than four inputs. In general, to control the switch at the output b_i we need to know the status of inputs a_{i-1} , a_i , and a_{i+1} . This has a great impact in reducing the complexity of the controller.

The control logic for the switches in a sender node will be described by separating the switches into four groups. As shown in fig. 2.6, the groups are denoted as X, Y, Z , and W . Switches in group X are used when the node is configured as non-redundant

and switches in group Y are used for the redundant configuration. Switches in groups Z and W are used in both configurations.

Let S_i be the signal that indicates that the i^{th} intra-layer input needs to be active, (i.e., information will be received at this input). Recall that this signal will come from the adjacent node or it will be produced by a sensing mechanism at the inputs of the node. Let R be the signal that indicates that the node should be configured as redundant. Finally, V is the signal that indicates a failure in the TSV to which the node is connected. With this notation we can now describe the control logic for the switches in a sender node. The letters x , y , z , and w are the signals that control the switches.

Switches in group X are controlled by two signals, R and S_i . These switches will be active only if the node is configured as non-redundant, (i.e., R is low), and there is information at the intra-layer input to which the switch is connected. Below is the Boolean expression for the switches' control signal:

$$X_i : x_i = \bar{R} \cdot S_i.$$

Similarly, switches in group Y are controlled by R and S_i . However, contrary to those in group X , R needs to be high in order for them to be active. The boolean expression is:

$$Y_i : y_i = R \cdot S_i.$$

Switches in group Z are used to create a path from the inputs to the main output. There is a total of $m + 2$ switches in this group. The control logic is the same for all the switches in the group except for the one connected to the main input and the one connected to the main output. The logic for the first one is the following:

Fig. 2.6: Sender internal structure with switches separated into groups

$$Z_1 : z_1 = \bar{R}.$$

This switch will be closed whenever the node is configured as non-redundant and open otherwise. The only function of this switch is to disconnect the node from the main input. The switch is not truly necessary since there should be no information at the main input of the node when it is configured as redundant, but it convenient to have it available.

The Z_{m+2} switch will always be closed except when there is a break in the TSV to which the node's output is connected. The signal to control it is the following:

$$Z_{m+2} : z_{m+2} = \bar{V}.$$

The control signal for the i^{th} switch in group Z is the following:

$$Z_i : z_i = \bar{R} \cdot \bar{V} \cdot \bar{S}_{i-1} + R \cdot \bar{V} \cdot \bar{S}_i, \quad 1 < i \leq m + 1.$$

First note that this switch is open whenever the TSV is open, (i.e., V is high). Also, the behavior is different depending on whether the node is configured as redundant or non-redundant. When the node is configured as non-redundant the switch will be closed if there is no information at the $(i - 1)^{th}$ intra-layer input. When the node is configured as redundant the state of the switch is determined by the status of the i^{th} input.

Following a similar reasoning we can write the following expressions for the control logic of the switches in group W :

$$W_1 \quad : \quad w_1 = \bar{R} \cdot V + \bar{R} \cdot S_1 + R \cdot V \cdot S_1 + R \cdot S_2,$$

$$W_i \quad : \quad w_1 = \bar{R} \cdot V \cdot S_{i-1} + \bar{R} \cdot S_i + R \cdot V \cdot S_i + R \cdot S_{i+1},$$

$$W_m \quad : \quad w_1 = \bar{R} \cdot V \cdot S_{m-1} + \bar{R} \cdot S_m + R \cdot V \cdot S_m.$$

Chapter 3

Performance Evaluation

Now that we have described the concept behind the micronetwork, it is necessary to evaluate its performance. The two key factors that we investigated are the ability of the micronetwork to handle multiple breaks on multilayer systems and the delays introduced in the re-routing process. In addition, we analyzed the relationship between the probability of failure of a single via and the probability of failure of a layer in the 3D integrated circuit.

3.1 Relation Between Via Failure and Layer Failure

In order to find the relationship between the probability of layer failure and the probability of via failure a simple model was created. The model was created using fig. 3.1 as an abstract representation of the multilayer chip.

In this specific figure, there are five bits that need to find a path to go from one

Fig. 3.1: Simple abstraction of a multilayer chip.

layer to the next. Therefore, it is required to have at least five TSVs. However, if the number of TSVs is the same as the number of bits to be transmitted, a single via failure will make impossible the transmission of all the bits resulting in the entire failure of the chip. As a result, it is necessary that the number of TSVs exceeds the number of bits to be transmitted. The extra TSVs are denoted by R , the total number of bits to be transmitted is denoted as B , and the total number of TSVs as N , where $N = B + R$.

Note that a failure in the communication of any two layers could result in the failure of the entire chip. Thus, finding the probability of failure between layers will, in some sense, be similar to finding the probability of error of the entire chip. To do so, it is assumed here that all the vias will fail independently with the same probability α . The probability of failure between layers is the probability that the number of vias that fail is greater than number of redundant vias,

$$p_l = \sum_{i=R+1}^N \binom{N}{i} \alpha^i (1 - \alpha)^{N-i},$$

where p_l is the probability of failure between layers; N , as mentioned before, is the total number of TSVs and R is the number of redundancies. This probability essentially follows a binomial distribution. Finally, the failure probability for the chip will be given by: $p_c = 1 - (1 - p_l)^{l-1}$, where l is the number of layers.

To better visualize how the probability of failure between layers relates to the number of redundancies and to the probability of failure of a single via (α), some plots are presented. The plots show the relation between the layer failure probability and the probability of failure of a single via. For all the curves the ratio R to N was held constant at 20% while the values of N and B were changed.

Figures 3.2 and 3.3 show how the layer failure probability will increase as the

single via failure probability increases. It can also be seen that the relation not only depends on the percentage of redundancies, but also depends on the total number of TSVs. Figure 3.3, which is the same as fig. 3.2 but in a logarithmic scale, gives a better idea on how the number of TSVs affects the layer failure probability. The plot shows how increasing the number of TSVs will reduce the layer failure probability for small values of α for the same ratio of R to N .

Figures 3.4 and 3.5 also show the relation between the layer failure probability and the probability of failure of a single via. However, now N is kept constant while B is allowed to take different values. This allows visualizing what happens for different values of $\frac{R}{N}$.

Finally, we analyzed the relation between the layer failure probability and the percentage of redundancies. These results are shown in figs. 3.6 and 3.7. These figures have a fixed value $B = 100$ and $B = 500$, and there are four curves for different values of α . As expected, increasing the number of redundant vias decreases the layer failure probability. This information may help to properly select the amount of redundancy that will be introduced into the system. Moreover, this shows that it is possible to achieve a low probability of layer failure despite a relatively high probability of via failure. For instance, the upper part of fig. 3.7 shows that for $\alpha = 0.02$ it is possible to achieve a layer failure probability of 10^{-10} with less than an 18% redundancy. Since the redundancy requirement depends on α , it is desired to minimize the probability of via failure as much as possible because this minimizes the number of redundant TSVs that are required.

The curves in fig. 3.7 show that the amount of redundancy required to maintain a given probability of layer failure does not depend only on the probability of via failure, but also on the number of non-redundant TSVs. This relationship is better portrayed in fig. 3.8. Note that as the number of non-redundant TSVs increases

Fig. 3.2: Layer failure probability vs. single via failure probability, $\frac{R}{N}$ constant.

Fig. 3.3: Layer failure probability vs. single via failure probability, $\frac{R}{N}$ constant, logarithmic scale.

Fig. 3.4: Layer failure probability vs. single via failure probability, N constant.

the required redundancy to maintain a given probability of layer failure decreases. In particular, when $B = 100$ it is necessary to have a redundancy of 17% in order to maintain the probability of layer failure below 10^{-10} . On the other hand, when $B = 500$ the redundancy requirement is of only 8%. For this particular example, increasing the number of non-redundant TSVs by a factor of five requires that the number of redundancies be increased by only a factor of three. This means that the redundancy requirement does not grow linearly with the number of non-redundant TSVs.

It is important to keep in mind that this model assumes that the TSV failures are independent and that α does not depend on B . These assumptions may not be true in reality. Nevertheless, it is expected that the TSV requirements will still follow a similar trend.

Fig. 3.5: Layer failure probability vs. single via failure probability, N constant, logarithmic scale.

3.2 Performance Under Multiple Breaks

The ability of the proposed micronetwork to handle multiple breaks on multilayer systems was simulated using a Monte Carlo simulation. The simulator accepts as primary inputs the number of layers (L), the number of non-redundant TSVs (B), the number of redundant TSVs (R), and the number of iterations (I). The code used to emulate the nodes' logic is shown in the Appendix.

In order to determine how many breaks are required for the circuit to no longer be able to accurately transmit information, we created models with $L=10$, $B=100$, $I = 500$, and $R=35, 40, 45, 50, 100$. We started by randomly creating $(R+1)$ breaks, performing 500 instances of the simulation, and recording the number of failures. The number of breaks was increased by one until we reached $R * (L - 1) + 1$ breaks, which is the minimum number of breaks that guarantees that the circuit will fail. Figure 3.9 shows the results which are quite satisfactory. The x -axis in the top plot is the

Fig. 3.6: Layer failure probability vs. percentage of redundancies.

Fig. 3.7: Layer failure probability vs. percentage of redundancies, Y-log scale.

Fig. 3.8: Layer failure probability vs. percentage of redundancies, fixed α .

actual number of breaks that the system tolerated and the x -axis in the bottom plot is the number of breaks as a percentage of the total number of TSVs in our model of the 3D IC. It is possible to see that with 100 redundant TSVs we can tolerate, for all the 500 iterations, around 38% of all the possible breaks. There are 1800 TSVs, which is the same as the total number of possible breaks.

For these simulations, the breaks are assumed to be at random locations, but in reality they may be more likely to occur in certain regions and in some specific pattern. If this is the case, the performance could decrease. Nevertheless, some of the redundant TSVs that are between layers with low probability of failure could be removed and placed between those layers with higher probability of failure. This way we can keep the performance constant, or perhaps improve it, without increasing the total number of redundant TSVs.

The problem of quantifying the probability of failure for the 3D IC is analogous to solving an occupancy problem. It can be thought of as having $M = (L - 1)$ cells,

Fig. 3.9: Performance curve.

where L is the number of layers, and T balls are thrown at random to the cells. Then, it is necessary to find the probability that at least one cell contains at least K balls. The range of T is $R+1 = K \leq T \leq M \times (R+B)$, where R is the number of redundant TSVs between any two layers and B is the number of non-redundant TSVs between layers. A possible approach to solve this problem can be found in the work presented by Williamson (2009) [22].

3.3 Delay Analysis

In addition to the parameters mentioned previously, it is also possible to select how the redundant nodes are distributed in the layer. Figure 3.10 shows a screenshot of the simulator menu. As we can see, there are three options for the type of distribution.

The first option, the default option, places all the redundant nodes together as

Number Of Layers:

Number Of Bits:

Number Of Redundants:

Number Of Iterations:

Delta break:

☐ Default
☐ Evenly Distributed
☐ Random

Fig. 3.10: Screenshot of second simulator menu.

a single group next to the group of non-redundant nodes. This type of distribution is the one shown in figs. 2.1 and 2.2.

The second option, the evenly distributed option, distributes the redundant nodes as evenly as possible, (i.e., alternating between small groups of redundant nodes and small groups of non-redundant nodes). The size of the redundant group can be specified by entering a value in the box adjacent to the evenly distributed label. Figures 3.11 and 3.12 present this type of arrangement. These figures are snapshots of the actual simulator. The black nodes are redundant and the blue nodes are non-redundant. For this type of configuration it is necessary that the intra-layer outputs of the last node are connected to the intra-layer inputs of the first node, this is not explicitly shown in the figures.

The third option, the random option, places the redundant nodes at random. This configuration allows us to better understand the impact that redundancy placement has on the magnitude of the delay introduced to the system.

Fig. 3.11: Simulator screenshot of an even distribution, group size = 1.

Fig. 3.12: Simulator screenshot of an even distribution, group size = 2.

The way the delays are analyzed is by keeping track of how many nodes a particular bit goes through when moving from the first layer to the top layer. The delay is described as delay in the node plus delay in the vias and intra-layer connections. The delay introduced inside the nodes will only have an impact when a break occurs, since a node may need several clock cycles to reconfigure correctly. The delay with the greatest impact comes from the time required to cover the distance from one node to another node. By assuming that the TSVs are equidistant, it is possible to obtain the change in the interconnect length introduced in the rerouting process. We chose to do it this way because the specific separation between the TSVs is not known since it will depend on various different factors: the number of TSVs, the technology used to fabricate the 3D IC, and the specific application [10]. Nevertheless, the count of the number of nodes could be easily converted to actual units of time by multiplying by the specific per node delay value. Thus, counting the number of nodes will provide essentially the same information as measuring actual delays; the only difference will be a constant factor.

The purpose of the delay analysis is to give a sense of the amount of delay that the micronetwork could introduce and to understand how the redundancy placement affects delays. For the first set of simulation we used the following parameters: $L = 10$, $B = 20$, $R = 20$, and $I = 10$. Figures 3.13, 3.14, 3.15, 3.20, and 3.21 show five curves, which correspond to the following: first non-redundant via, last non-redundant via, average delay, average maximum delay, and maximum delay. First [last] non-redundant via shows the amount of delay that will be encountered as information travels from the first layer to the last layer through the first [last] via. In fig. 2.4, the first non-redundant via would be the one connecting nodes NS1 and NR1, and the last non-redundant via would be the one connecting NS6 and NR6.

The “average delay” is calculated by taking the average of all the TSVs delays

for each iteration, and then the average over all the iterations is taken. The “average maximum delay” is calculated by taking the maximum delay from all TSV for each iteration, and averaging these values over all the iterations. The “maximum delay” is taken to be the greater delay at any particular TSV for any of the iterations; it is the maximum delay over all the iterations. The y-axis shows the number of nodes a particular bit of information has to go through. Note that a TSV has two nodes between layers, which means that a 3D IC with 10 layers will have 18 nodes per TSV. So, for the cases shown below, in the absence of breaks, each bit of information has to go through 18 nodes when traveling from the first layer to the last layer.

Figure 3.13 shows the results for the default distribution. This configuration will, on average, introduce a great amount of delay into the system. For example, at 90 breaks, which is half of what the system can tolerate, we see that on average every bit goes through 80 nodes. This is 4.5 times greater than the number of nodes that it goes through when there are no breaks in the system. This is because when a break occurs, all the bits to the right of the break are shifted. This is why the first non-redundant via has a very small delay, while the last non-redundant via has an extremely large delay.

Since the default configuration is not very practical, we decided to try other types of configurations. To do this we selected the random distribution option. Figures 3.14 and 3.15 show the results for two cases of the random distribution. We can see that in both cases there is a huge improvement over the default case. The increment in the number of nodes that every bit goes through, at 90 breaks, is only double than when there are no breaks. Moreover, these figures suggested that the average delay was about the same in both cases, so we decided to run a few more simulations with a random distribution and compared the results. These results are shown in fig. 3.16. It is possible to see that there is not a lot of variation in the average delay for

Fig. 3.13: Delay analysis, $L=10$, $B=20$, $R=20$, $I=10$, default.

the different cases. This could allow us to estimate the expected delay of the system for any random configuration. To support this observation another set of simulations was executed. For these simulations the number of redundancies was decreased to 10 in order to speed up the process of collecting the data. Decreasing the number of TSVs produces a higher standard deviation. Nevertheless, it is still possible to appreciate that variation in the average delay for the different cases is not excessive. For a total of 100 simulations the standard deviation and the mean were calculated for the average delay (fig. 3.17), the maximum delay (fig. 3.18), and the average maximum delay (fig. 3.19).

Finally, we evaluated the performance of the micronetwork under the evenly distributed configuration. Figures 3.20 and 3.21 show the results for a group size of one and a group size of two, respectively. At 90 breaks the group of size one shows an average number of nodes travelled which is 1.5 times higher than the average number of nodes travelled when there are no breaks, and this value is around 1.6 for the group of size two. As the group size increases the delay increases. It can be seen that the even distribution, with group of size one, introduces the smallest amount of delay.

Fig. 3.14: Delay analysis, $L=10$, $B=20$, $R=20$, $I=10$, random 1.

Fig. 3.15: Delay analysis, $L=10$, $B=20$, $R=20$, $I=10$, random 2.

Fig. 3.16: Delay analysis, L=10, B=20, R=20, I=10, random.

Fig. 3.17: Delay analysis, L=10, B=20, R=10, I=10, average delay.

Fig. 3.18: Delay analysis, $L=10$, $B=20$, $R=10$, $I=10$, max delay.

Fig. 3.19: Delay analysis, $L=10$, $B=20$, $R=10$, $I=10$, average max delay.

Fig. 3.20: Delay analysis, $L=10$, $B=20$, $R=20$, $I=10$, even 1.

Fig. 3.21: Delay analysis, $L=10$, $B=20$, $R=20$, $I=10$, even 2.

Chapter 4

On-layer Interconnection

So far we have presented the nodes in the micronetwork as connected one next to another in a linear (1D) distribution. Moreover, in the previous chapter it was possible to see how the distribution of the redundant nodes plays a role in the delay introduced by the rerouting process. This suggests that it may be beneficial to modify the intra-layer connections so that it is possible to deal with a broken TSV by rerouting the information through any TSV in the layer and not necessarily to an adjacent TSV. Thus a more efficient way to handle bursty breaks could be accomplished by creating a better 2D distribution of the TSVs in a given layer.

The goal is to find a way to interconnect the nodes so that it is possible to minimize the number of non-redundant nodes that are affected when a TSV breaks. The idea was to connect one node to several nodes and not only to the adjacent nodes, while maintaining the same level of complexity. When a group of j adjacent non-redundant nodes suffer breaks at their TSVs, information will have to be rerouted through at least j nodes and the number of nodes that are affected will depend on how the redundant nodes are distributed. This idea will be clarified with the use of an example.

Figure 4.1 shows 16 sender nodes on a layer, four of which are redundant. The redundant nodes are evenly distributed throughout the circuit. In Chapter 3 this was shown to be the best distribution. Now, suppose that nodes NS1 and NS2 suffer a break at their TSV, forcing them to send the information they are receiving at their main input to the adjacent nodes. The information that was at the main input of

NS1 will have to travel all the way to NS3 before it can reach the next layer and the information at the main input of NS2 will have to travel to node RS1. Moreover, nodes NS3, NS4, NS5, and NS6 will be affected as well. This “domino” effect is an issue that needs to be addressed. The first attempt to alleviate this problem was to modify the way the nodes are interconnected.

The objective was to give the nodes the possibility of re-routing in more than one direction. The initial approach was to find a way to send all the m intra-layer outputs of a given nodes to i other nodes. This creates a better set of options for the re-routing process. However, a greater number of intra-layer connections will be required, specifically $m \times i$, which is not desired. Moreover, the node will have to make a decision regarding to which node it should send the information. As a result, the complexity of the nodes routing logic will increase. Another disadvantage is that the receiver at the next layer will need to know what path was chosen by the sender, which represent a further increment in the complexity. Due to the high complexity this approach is not a very viable option.

The second approach also aimed to have a single node connected to multiple nodes. However, instead of connecting each of the intra-layer outputs to i different nodes, each intra-layer output is connected to a single, different, node. See fig. 4.2. Note that the number of intra-layer connections will be equal to m , which is the same as when all the intra-layer outputs connect to a single node. So, there exists the possibility of maintaining the same routing logic. This solves the issues of the higher complexity introduced by the need of choosing a path and could, with the proper interconnections, reduce the number of non-redundant nodes that are affected in the re-routing process.

It is necessary to address the issue of how the receivers will deal with the information in order to route it to the right position. Since symmetry is the key piece

Fig. 4.1: On layer distribution, case 1.

Fig. 4.2: Modified nodes interconnection.

behind the micronetwork's low complexity, it was assumed that maintaining the symmetry was the way to solve the problem. So, if the k intra-layer output of node NSx is connected to the k intra-layer input of node NSy , the assumption was that connecting the k intra-layer output of node NRy to the k intra-layer input of node NRx was the right approach. NRx represents an arbitrary non-redundant receiver and NSx represents an arbitrary non-redundant sender. The same logic applies for the connections involving redundant nodes. This configuration is able to handle effectively some break patterns, as is shown in fig. 4.3. Nevertheless, this will not work in general. Figure 4.4 shows an example of a re-routing failure. Moreover, note that fig. 4.3 shows a worse performance than the one from the standard interconnection. Other interconnections were tested and similar results were obtained.

It may be possible to find a way to interconnect the nodes that will work for all the cases, perhaps with a small modification of the logic. Unfortunately, we were unable to find such an interconnection. Moreover, we believe that achieving this type of interconnectivity will require nodes which have a considerably more complex logic. Moreover, it may require that information for rerouting decisions is exchanged between adjacent layers. Inter-layer communication is not desirable and it could offset any benefits obtained from the modified intra-layer interconnection.

The potential inconveniences that could be introduced by a change in the interconnections suggest that it might be necessary to find another way to deal with the problem of increased delays in the presence of bursty breaks. In order to do so, we will first try to better understand the effect of bursty errors in the amount of delay introduced by the micronetwork. The analysis presented is for a simulation with 100 non-redundant TSVs, 25 redundant TSVs, and two layers. Figure 4.5 shows the delay analysis when the TSV fails independently and the redundant nodes are evenly distributed, for a single iteration. Several simulations of a single iteration were per-

Fig. 4.3: Modified interconnection, example of successful routing.

Fig. 4.4: Modified interconnection, example of a routing failure.

formed, with the results following a similar trend each time. It can be seen that on average the amount of delay introduced is quite low. However, the maximum delay measured shows that there is at least one via that encounters a substantial delay. But this is an issue that would not be solved even with a different arrangement of the interconnections. Any particular TSV could face a delay as big as the number of breaks, even in the presence of an optimal interconnection. This is due to the routing logic and the random nature of the breaks.

Figure 4.6 shows the results for the same simulation parameter that is shown in fig. 4.5, but now considering the possibility that bursty breaks can occur. For this example, every time a TSV breaks, there is a 90% chance that the either of the two adjacent TSVs will break. We see that the delays are higher than in the previous case. This is what we were expecting from the discussion that led to the desire of finding an alternative way of interconnecting the nodes. However, we propose that instead of looking for an alternative way to interconnect the nodes, we remain with the same interconnection, and to avoid interconnecting nodes that are expected to fail together.

Once there is information about which TSVs are more likely to fail, it is possible to connect the nodes at those TSVs to nodes at TSVs that are less likely to fail and with independent probability of failure. Moreover, the distance between the nodes needs to be considered as well. This way the wiring length could be minimized. Hence, the expected delay is minimized. We illustrate this idea with the use of an example.

Suppose that in fig. 4.7 the TSVs associated to nodes NS1, NS2, NS7, and NS8 have a higher probability of failure and that their probability of failure is not independent. The scenario described could be typical in a 3D IC, since the center part of the layer tends to be at a higher temperature. The failure pattern that was described in fig. 4.1, failure of the TSVs at NS1 and NS2, is likely to emerge in

Fig. 4.5: Delay analysis, two layers, single iteration.

Fig. 4.6: Delay analysis, two layers, single iteration, bursty.

Fig. 4.7: Modified nodes interconnection.

the described scenario. Nevertheless, the way the nodes are now interconnected will prevent the undesired domino effect that was present in the configuration shown in fig. 4.1. Nodes associated to TSVs that are more likely to fail are directly connected to non-redundant nodes, which reduces the interaction between non-redundant nodes.

The previous example shows that with the appropriate information concerning failure patterns it is possible to minimize the expected delay. However, this only improves expected values while the worst case scenario remains the same. The worst delay is experienced whenever TSVs at interconnected nodes fail together and this depends on the number breaks we intent to tolerate. The maximum delay could be as big as the number of breaks we want to tolerate, because that is the maximum number of interconnected TSVs that could break together. Together does not necessarily mean at the same time, but that at some point in time they all have failed. Nevertheless, by interconnecting nodes that are associated to TSVs with independent probability

of failure we reduce the chances of getting anywhere close to the maximum delay. So, we do not change the maximum delay, but we decrease the probability of ever experiencing it.

Chapter 5

Conclusions

In this chapter a summary of the results and some ideas for future work are discussed.

5.1 Summary

A model to mitigate the communication problem in 3D integrated circuits caused by the breaks at the through-silicon vias (TSVs) was developed in this thesis. Using a low complexity network, the introduction of redundant TSVs makes it possible to re-route around breaks to maintain effective communication between layers. This was developed while maintaining a local on-layer communication, a small area, a low complexity circuitry, and local routing.

Chapter 2 describes the principles of operation of the micronetwork. In addition, a detailed description of the routing logic is presented. An idea for the internal structure of the nodes is described. This shows that the cost of creating the nodes circuitry is relatively low, due to their low complexity.

In Chapter 3, an analysis of the micronetwork performance is presented. The relationship between the probability of failure of a single via and the probability of failure of a single layer was studied. This helps assessing the number of redundancies that are required for the system to operate reliably. It was shown that with the introduction of a relative small amount of redundancy it is possible to improve the reliability of the system considerably. In most cases the robustness improved by several orders of magnitudes with a redundancy of less than 15%.

In addition, the performance of the micronetwork under multiple breaks was assessed, and promising results were obtained. The chapter also covers an analysis of the delay introduced in the rerouting process. It was observed that delays are closely tied to the distribution of the nodes in the layer.

Chapter 4 further expands on the delay analysis. The idea is to modify the intra-layer connections in order to reduce the delay introduced in the re-routing process. Two approaches were discussed. The first, connecting all the intra-layer outputs to different nodes, has the potential of reducing the delay to the very minimum, but it would be extremely complex to implement. The other, connecting each intra-layer output to a single, different, node is more appealing since it has a lower complexity than the first approach. Unfortunately, the level of complexity is still too high when compared to the interconnection described in Chapter 2. We conclude that the best way to deal with the delay is by properly distributing the nodes. Nodes associated with TSVs that are more likely to fail should be connected as close as possible to non-redundant nodes, and nodes that are connected together should have a probability of failure as independent as possible.

The results show that the micronetwork approach to the TSV failure problem is able to provide significant protection against multiple breaks while maintaining computation local. Moreover, most additional communication is on-plane (2D) and the latency or decoding delays are reasonably small. The micronetwork is a low complexity system that enhances the reliability of 3D integrated circuits.

5.2 Future Work

This research creates new opportunities for study. As it was mentioned in the introduction, research on how to deal with TSVs failure from an error correction perspective is practically inexistent. There is still a lot of uncertainty around many

aspects of 3D integrated circuits, and because of that, the micronetwork idea was developed on assumptions that need to be verified. For instance, it was assumed that the on-layer interconnections will not fail and that it is possible to determine when a TSV fails. If we allow the possibility for the on-layer interconnections to fail, it will be necessary to modify the nodes logic to account for this type of events. This should not be a major issue, but it will increase the complexity of the nodes. It will also require the introduction of redundant on-layer interconnections.

It is also necessary to find practical values for the number of TSV that a typical 3D IC will possess. In addition, it is required to have better estimates of the probability of failure for individual TSVs. A better knowledge of these parameters will allow a better study of the micronetwork, since its performance is dependent on those parameters. Moreover, this will be necessary for a better study of bursty breaks.

The simulations that were created to test the micronetwork were only behavioral. It will be useful to simulate the micronetwork at the transistor or gate level. This will make it possible to see the effect of the micronetwork on the data. The micronetwork creates new paths for the data to travel, and it is important to analyze the changes in parameters like resistance, capacitance, and inductance of the new path. It is necessary to assure that the micronetwork will not corrupt the information.

It may be the case that the number of redundant TSVs in a given 3D IC is large enough to make impractical the implementation of the micronetwork. The problem is that the intra-layer connections are proportional to the number of redundant TSV, and if they are too many they will end up occupying too much area on the silicon. This problem could be approached by dividing the micronetwork into several smaller and unrelated groups. This is not the optimal way to take advantage of all the redundancy, but it will allow for less complex nodes. For example, if there is a 3D IC with 300 non-redundant TSVs and 50 redundant TSVs, denote this arrangement as

300/50, there will be 50 intra-layer connections between any two nodes. The number of intra-layer connections could be reduced by dividing the TSV into two groups, one of size 180/30 and one of size 170/20. Note that in the group of size 170/20 it is only possible to tolerate 20 failures. Nevertheless, the size of the groups can be arranged in a way that the probability of failure is minimized by taking into consideration what the expected patterns of failure are.

The micronetwork only re-routes information in one direction on the layer, this is why the number of intra-layer connection has to be the same as the number of redundant TSVs. Nevertheless, this number could be cut in half by allowing the re-routing process to be in two directions. The node could be modified so that it routes in one direction by default and when it is not possible to route in that direction, it re-routes in the opposite direction. This modification will require an additional line of communication between adjacent nodes so that a node can notify to the adjacent node that there has been a change in the re-routing direction.

Figure 5.1 illustrates this idea. Part A it shows a section of the micronetwork that routes unidirectionally; this circuit can support/needs four redundant TSVs, but they are not shown. Part B and C show a modified version of circuit A; this version can support/needs eight redundant TSV, while it maintains the same number of intra-layer connections. It also requires an additional communication line, shown in green, that will be used to communicate the need of a change in the routing direction. When the fifth break occurs in circuit B, node NS6 is unable to handle all the information it is receiving. As a result, it signals node NS5, through the communication line (green line), that it cannot handle all the information it is receiving, and node NS5 signals node NS4 about the problem; the same behavior is repeated until the message reaches to node NS1. Node NS1 then realizes that it is the one causing the problem, since it is not receiving information from any other node. So, NS1 decides to send its

information in the other direction. Once circuit B has finished the decision process, the nodes will look as shown in part C.

Implementing this logic will be far more complex, and it will only be necessary if the number of intra-layer connections becomes a problem. The concept behind this logic is very similar to what we have described for the unidirectional communication, but the internal structure of the nodes will be different. The node will need more switches and a control logic, which is significantly more complex.

Fig. 5.1: Example of bidirectional routing.

References

- [1] B. R. S. Patti, "Three-dimensional integrated circuits and the future of system-on-chip designs," *Proceedings of the IEEE*, vol. 94, no. 6, 2006.
- [2] K. Nomura, K. Abe, S. Fujita, and A. DeHon, "Novel design of three-dimensional crossbar for future network on chip based on post-silicon devices," *2006 1st International Conference on Nano-Networks and Workshops*, pp. 1–5, Sept. 2006 [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4152809>.
- [3] C. Scheiring, H. Kostner, P. Lindner, S. Pargfrieder, and E. V. Group, "Advanced-chip-to-wafer technology: enabling technology for volume production of 3D system integration on wafer level," in *Integration The VLSI Journal*, 2004.
- [4] J.-q. Lu, K. Rose, and S. Vitkavage, "3D Integration: why, what, who, when?" pp. 25–27, 2007 [Online]. Available: <http://www.future-fab.com/>.
- [5] C. Ababei, B. Goplen, H. Mogal, K. Bazargan, and S. Sapatnekar, "Placement and routing in 3D integrated circuits," *IEEE Design and Test of Computers*, vol. 22, no. 6, pp. 520–531, June 2005 [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1541914>.
- [6] B. M. Motoyoshi, "Through-silicon via (TSV)," *Proceedings of the IEEE*, vol. 97, no. 1, pp. 43–48, 2009.
- [7] W. R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A. M. Sule, M. Steer, and P. D. Franzon, "Demystifying 3D ICs: the pros and cons of going vertical," *Integration The VLSI Journal*, pp. 498–510, 2005.
- [8] N. Ranganathan, K. Prasad, N. Balasubramanian, and K. L. Pey, "A study of thermo-mechanical stress and its impact on through-silicon vias," *Journal of Micromechanics and Microengineering*, vol. 18, no. 7, p. 075018, July 2008 [Online]. Available: <http://stacks.iop.org/0960-1317/18/i=7/a=075018?key=crossref.e761459d7744dca9701ca7243241e241>.
- [9] A.-M. Corley, "Chipmakers bet on a stacked deck," *IEEE Spectrum*, vol. 47, no. 4, pp. 14–14, Apr. 2010 [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5434837>.

- [10] J. U. Knickerbocker, P. S. Andry, B. Dang, R. R. Horton, M. J. Interrante, C. S. Patel, R. J. Polastre, K. Sakuma, R. Sirdeshmukh, E. J. Sprogis, A. M. Stephens, A. W. Topol, C. K. Tsang, B. C. Webb, and S. L. Wright, "Three-dimensional silicon integration," *International Business*, vol. 52, no. 6, pp. 553–569, 2008.
- [11] N. Khan, V. S. Rao, S. Lim, H. S. We, V. Lee, Z. X. Wu, Y. Rui, and L. Ebin, "Development of 3D silicon module with TSV for system in packaging," *Components*, pp. 550–555, 2008.
- [12] C. S. Selvanayagam, J. H. Lau, X. Zhang, S. K. W. Seah, K. Vaidyanathan, and T. C. Chai, "Nonlinear thermal stress / strain analyses of copper filled TSV (Through Silicon Via) and their flip-chip microbumps," *Components*, pp. 1073–1081, 2008.
- [13] H.-H. S. Lee and K. Chakrabarty, "Test challenges for 3D integrated circuits," *IEEE Design & Test of Computers*, vol. 26, no. 5, pp. 26–35, Sept. 2009 [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5286146>.
- [14] A. Contreras, T. Moon, A. Dasu, and J. Gunther, "Micronetworking: reliable communication on 3D integrated circuits," *Electronics Letters*, vol. 46, no. 4, p. 291, 2010 [Online]. Available: <http://link.aip.org/link/ELLEAK/v46/i4/p291/s1/&Agg=doi>.
- [15] Z. Li, X. Hong, Q. Zhou, S. Zeng, J. Bian, H. Yang, V. Pitchumani, and C.-K. Cheng, "Integrating dynamic thermal via planning with 3D floorplanning algorithm," *Proceedings of the 2006 International Symposium on Physical Design*, p. 178, 2006 [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1123008.1123048>.
- [16] S. Pasricha, "Exploring serial vertical interconnects for 3D ICs," *Proceedings of the 46th Annual Design Automation Conference*, p. 581, 2009 [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1629911.1630061>.
- [17] B. Goplen and S. Sapatnekar, "Thermal via placement in 3D ICs," *Proceedings of the 2005 International Symposium on Physical Design - ISPD '05*, p. 167, 2005 [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1055137.1055171>.
- [18] D. Sekar, C. King, B. Dang, T. Spencer, H. Thacker, and P, "A 3D-IC technology with integrated microchannel cooling," *Interconnect Technology*, pp. 13–15, June 2008 [Online]. Available: <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:A+3D-IC+Technology+with+Integrated+Microchannel+Cooling\#0>.

- [19] IBM, “Made in IBM labs: IBM cools 3-D chips with H_2O ,” June 2008 [Online]. Available: <http://www-03.ibm.com/press/us/en/pressrelease/24385.wss>.
- [20] U. Kang, H.-J. Chung, S. Heo, S.-H. Ahn, H. Lee, S.-H. Cha, J. Ahn, D. Kwon, J. H. Kim, J.-W. Lee, H.-S. Joo, W.-S. Kim, H.-K. Kim, E.-M. Lee, S.-R. Kim, K.-H. Ma, D.-H. Jang, N.-S. Kim, M.-S. Choi, S.-J. Oh, J.-B. Lee, T.-K. Jung, J.-H. Yoo, and C. Kim, “8Gb 3D DDR3 DRAM using through-silicon-via technology,” in *Solid-State Circuits Conference - Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*, pp. 130 –131,131a, 2009.
- [21] I. Loi, S. Mitra, T. H. Lee, S. Fujita, and L. Benini, “A low-overhead fault tolerance scheme for TSV-based 3D network on chip links,” *2008 IEEE/ACM International Conference on Computer-Aided Design*, pp. 598–602, Nov. 2008 [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4681638>.
- [22] P. P. Williamson, “Revisiting the classical occupancy problem,” *The American Statistician*, vol. 63, no. 4, pp. 356–360, Nov. 2009 [Online]. Available: <http://pubs.amstat.org/doi/abs/10.1198/tast.2009.08104>.

Appendix

Appendix

Code for the Nodes Processing Logic

```

import java.awt.*;
import javax.swing.*;

//Micronetworking: Reliable communication on 3D Integrated Circuits

//This class implements the processing logic for the nodes. There are
//four types of nodes: Non-Redundant Lower, Non-Redudnat Upper,
//Redundant Lower, and Redundant Upper. On the thesis this names were
//changed from Upper to Sender and from Lower to Receiver.

public class Node {

    int MAXINPUTS ;
    int MAXOUTPUTS;
    //These are the 4 different types of Nodes.
    public enum nodeType {RUPPER,NRUPPER,NRLOWER,RLOWER};
    //Each node conects to 4 neighbors.
    public enum nodeSide {UPPER,LOWER,LEFT,RIGHT};
    //Ports are the principal inputs and outputs of the nodes.
    public enum portStatus {OPEN,CLOSE}

```

```

private Node rightNode;
private Node leftNode;
private Node upperNode;
private Node lowerNode;
//Variable to store the node type.
private nodeType type;
private String name;
private int coordinateX;
private int coordinateY;
private boolean canIProcess;
private Color nodeColor;
private int nodeLenght;
private int nodeWidth;
//This is a helper class to draw the inputs and outputs of the nodes

//This is a class that has information concerning the status of
//the ports.
public class Port
{
    private int port;
    private Color color;
    private Node.portStatus status;
    private int vertex [][];
    public Counter count;
    int heigth ,width;
    Port()
    {

```



```

        this.status=portStatus.CLOSE;
        //We set port to -1 to symbolize that it does not have
        // information (0 or 1)
        this.port=-1;
        this.color=Color.GREEN;
        vertex = new int [1][2];
        count = new Counter();
    }

    //This class reset the port values. This will be used when the
    //node type is changed. This way we reset the values and let
    // the node know that it need to reconfigure.
    public void resetPort(int in){
        if(this.status==Node.portStatus.CLOSE);
        this.color=Color.GREEN;

        this.port=in; //in is either 1,0 or -1.
        this.count= new Counter();
    }
    public void setColor(Color c){
        this.color=c;
    }
    public void setPort(int val){
        this.port=val;
    }
    public void setStatus(Node.portStatus s){
        this.status=s;
    }

```

```

    }

    public Color getColor()
    {
        if( this.status==Node.portStatus.OPEN) return Color.RED;
        else if (this.port == -1) return Color.GREEN;
        return this.color;
    }

    public int [][] getVertex(){
        return this.vertex;
    }

    public int getHeigth(){
        return this.heigth;
    }

    public int getWitdh(){
        return this.width;
    }

};

//This class is used for the intra-layer ports
public class SidePort
{
    private int [] side;
    private Color [] sideColor;
    private int vertex [][];
    public Counter [] count;

```

```

public Node [] outputs;
int height,width;

SidePort(int maxOut)
{
    side= new int [maxOut];
    sideColor= new Color[maxOut];
    vertex= new int [maxOut][2];
    count = new Counter[maxOut];
    for (int i = 0; i < MAXINPUTS; i++) {
        this.side[i] = -1;
        this.sideColor[i] = Color.GREEN;
        count[i]=new Counter();
    }
}

public void resetSidePorts(){
    for (int i = 0; i < MAXINPUTS; i++) {
        this.side[i] = -1;
        this.sideColor[i] = Color.GREEN;
        this.count[i]= new Counter();
    }
}

public Color[] getColor()
{

```

```

        Color tempColor [] = new Color[this.sideColor.length];

        for(int i=0; i<this.sideColor.length; i++)
        {
            tempColor[i]=sideColor[i];
            if(this.side[i]==-1) tempColor[i]= Color.GREEN;
        }
        return tempColor;
    }

    public int [][] getVertex(){

        return this.vertex;
    }

    public int getWitdh()
    {
        return this.width;
    }
    public int getHeigth()
    {

        return this.heigth;
    }

};

```

```

//This class is used to keep track of the number
//of nodes a particular bit goes through
public class Counter
{
    private String counter [];

    Counter(){
        this.counter = new String [1];
        this.counter [0] = Node.this.name;
    }

    //this function set the inputs counters
    public void setCounter(String []inCount)
    {
        this.counter = new String [inCount.length];
        System.arraycopy(inCount, 0, this.counter, 0, inCount.length);
        if(this.counter[0]== null) this.counter[0] = Node.this.name;
    }

    //this functionr set the outputs counters
    public void setCounter(String []inCount, String add)
    {
        this.counter = new String [inCount.length+1];
        System.arraycopy(inCount, 0, this.counter, 0, inCount.length);
        counter[inCount.length]=add;
    }
}

```

```

        if (this.counter[0]== null) this.counter[0] = Node.this.name;
    }

    public String [] getCounter() {
        return this.counter;
    }

    public void setCounter(String inCount, int p){
        if(p<this.counter.length)
            this.counter[p]=inCount;
        if (this.counter[0]== null) this.counter[0] = Node.this.name;
    }

}

private SidePort sideInput ;
private SidePort sideOutput ;
private Port nodeInput;
private Port nodeOutput;

//This is the constructor to set the initial values.
public Node(int max)
{
    MAXINPUTS=max;
    MAXOUTPUTS=max;
    this.type = nodeType.RUPPER ;
}

```

```

        this.sideInput= new SidePort (MAXINPUTS);
        this.sideOutput= new SidePort (MAXINPUTS);
        this.leftNode= null;
        this.rightNode=null;
        this.upperNode=null;
        this.lowerNode=null;

        this.canIProcess=true;

        this.nodeInput  = new Port ();
        this.nodeOutput = new Port ();
        this.name = "NA";

    }

    public Node(nodeType nT,int maxout)
    {
        this(maxout);
        this.setType(nT);
    }

//This function attach a node to its neighbors. newNode is the node to be
//attached and x is the side were the node is going to be attached.
public void attachNode(Node newNode, nodeSide x )
{

    switch(x)
    {
        case LOWER:

```

```

        this.lowerNode=newNode;
    break;
    case UPPER:
        this.upperNode=newNode;
    break;
    case RIGHT:
        this.rightNode=newNode;
    break;
    case LEFT:
        this.leftNode=newNode;
    break;
}
}

//This function attached the neighbors nodes all at once. It starts at the
//upper node and goes clock wise until the left node.
public void attachNode(Node uN, Node rN,Node dN,Node lN)
{
    this.leftNode=lN;
    this.lowerNode=dN;
    this.rightNode=rN;
    this.upperNode=uN;
}

//This function returns an array containing the four neighbor nodes
public Node[] getAdjacentNodes()
{

```



```

        Node newNode[] = new Node[4];
        newNode[0] = this.upperNode;
        newNode[1] = this.rightNode;
        newNode[2] = this.leftNode;
        newNode[3] = this.lowerNode;
        return newNode;
    }

    public void setType(nodeType T){
        this.type = T;
    }

    public nodeType getType(){
        return this.type;
    }

    //Here is where every node process how to handle the information
    //at its inputs.
    public void processNode() {
        int i = MAXINPUTS - 1;

        synchronized (this) {
            while (i >= 0 && this.sideInput.side[i] == -1) {
                i--;
            }
        }
    }

```

```

switch (this.type) {
    case RUPPER:
        this.processRUpper(i);
        this.setUpperNodeIn(this.nodeOutput);
        this.setRightNodeSide(sideOutput);

        break;
    case RLOWER:
        this.processRLower(i);
        this.setLeftNodeSide(sideOutput);
        break;
    case NRUPPER:

        this.processNRUpper(i);
        this.setUpperNodeIn(this.nodeOutput);
        this.setRightNodeSide(sideOutput);

        break;
    case NRLOWER:
        this.processNRLower(i);
        this.setUpperNodeIn(this.nodeOutput);
        this.setLeftNodeSide(sideOutput);

        break;
}

}

```

```

    }

//revisted 1.0
public void processRUpper(int start)
{

    if(start!=-1)
    {
        if(this.nodeOutput.status==portStatus.CLOSE)
        {
            this.nodeOutput.port=this.sideInput.side[start];
            this.nodeOutput.color=this.sideInput.sideColor[start];
            this.nodeOutput.count.setCounter
                (this.sideInput.count[start].getCounter(),this.name);
        }

        else
        {
            this.sideOutput.side[start]=this.sideInput.side[start];
            this.sideOutput.sideColor[start]=this.sideInput.sideColor[start];
            this.sideOutput.count[start].setCounter
                (this.sideInput.count[start].getCounter(),this.name);
        }

        for(int k=0;k<start;k++)

```

```

    {
        this.sideOutput.side[k]=this.sideInput.side[k];
        this.sideOutput.sideColor[k]=this.sideInput.sideColor[k];
        this.sideOutput.count[k].setCounter
            (this.sideInput.count[k].getCounter(),this.name);
    }
}
}

```

```

public void processRLower(int start) {

    int j;
    if (this.nodeInput.status == portStatus.CLOSE) {
        this.sideOutput.side[0] = this.nodeInput.port;
        this.sideOutput.sideColor[0] = this.nodeInput.color;
        this.sideOutput.count[0].setCounter
            (this.nodeInput.count.getCounter(),this.name);

        j = 1;
    } else {
        j = 0;
    }
}

```

```

    }
    for (int k = 0; k <= start; k++) {
        this.sideOutput.side[k + j] = this.sideInput.side[k];
        this.sideOutput.sideColor[k + j] = this.sideInput.sideColor[k];
        this.sideOutput.count[k + j].setCounter
            (this.sideInput.count[k].getCounter(), this.name);
    }

}

public void processNRLower(int start)
{

    int j;
    if (start != -1)
    {
        this.nodeOutput.port = this.sideInput.side[start];
        this.nodeOutput.color = this.sideInput.sideColor[start];
        this.nodeOutput.count.setCounter
            (this.sideInput.count[start].getCounter(), this.name);
    }

    if (this.nodeInput.status == portStatus.CLOSE)

```

```

{
    if (start == -1)
    {
        this.nodeOutput.port = this.nodeInput.port;
        this.nodeOutput.color = this.nodeInput.color;
        this.nodeOutput.count.setCounter
            (this.nodeInput.count.getCounter(), this.name);

    }
    else
    {
        this.sideOutput.side[0] = this.nodeInput.port;
        this.sideOutput.sideColor[0] = this.nodeInput.color;
        this.sideOutput.count[0].setCounter
            (this.nodeInput.count.getCounter(), this.name);
    }
    j = 1;
}
else
{
    j = 0;
}

for (int k = 0; k < start; k++)
{
    this.sideOutput.side[k+j] = this.sideInput.side[k];
    this.sideOutput.sideColor[k+j] = this.sideInput.sideColor[k];
}

```

```

        this.sideOutput.count[k + j].setCounter
            (this.sideInput.count[k].getCounter(), this.name);
    }

}

public void processNRUpper(int start) {

    if (this.nodeOutput.status == portStatus.CLOSE) {
        if (start != -1) {
            this.nodeOutput.port = this.sideInput.side[start];
            this.nodeOutput.color = this.sideInput.sideColor[start];
            this.nodeOutput.count.setCounter
                (this.sideInput.count[start].getCounter(), this.name);

            this.sideOutput.side[0] = this.nodeInput.port;
            this.sideOutput.sideColor[0] = this.nodeInput.color;
            this.sideOutput.count[0].setCounter
                (this.nodeInput.count.getCounter(), this.name);

        } else {
            this.nodeOutput.port = this.nodeInput.port;
            this.nodeOutput.color = this.nodeInput.color;
            this.nodeOutput.count.setCounter
                (this.nodeInput.count.getCounter(), this.name);
        }
    }
}

```

```

    }

} else {
    this.sideOutput.side[0] = this.nodeInput.port;
    this.sideOutput.sideColor[0] = this.nodeInput.color;
    this.sideOutput.count[0].setCounter
        (this.nodeInput.count.getCounter(), this.name);

    start++;
}

for (int k = 0; k < start; k++) {
    this.sideOutput.side[k + 1] = this.sideInput.side[k];
    this.sideOutput.sideColor[k + 1] = this.sideInput.sideColor[k];
    this.sideOutput.count[k + 1].setCounter
        (this.sideInput.count[k].getCounter(), this.name);
}

}

public void setOutPortStatus(Port in) {
    this.nodeOutput.status = in.status;
    this.nodeOutput.color = in.color;

```



```

        if (in.status != Node.portStatus.CLOSE) {
            this.nodeOutput.port = -1;
            this.nodeOutput.color = Color.RED;
        }

    }

    public void setInPortStatus(Port in) {
        this.nodeInput.status = in.status;
        this.nodeInput.color = in.color;
        if (in.status != Node.portStatus.CLOSE) {
            this.nodeInput.port = -1;
            this.nodeInput.color = Color.RED;
        }
    }

    public Node.portStatus getMyInPortS()
    {
        return this.nodeInput.status;
    }

    public Node.portStatus getMyOutPortS()
    {
        return this.nodeOutput.status;
    }

    public synchronized void setUpperNodeIn(Port portValue)

```

```
{  
    if ( this . upperNode != null )  
        synchronized ( this . upperNode )  
        {  
            this . upperNode . setMyinput ( portValue );  
        }  
}  
  
public synchronized void setRightNodeSide( SidePort portValue )  
{  
  
    if ( this . rightNode != null )  
    {  
  
        synchronized ( this . rightNode )  
        {  
            this . rightNode . setMySidesIn ( portValue );  
        }  
  
    }  
}  
  
public synchronized void setLeftNodeSide( SidePort portValue )  
{  
    if ( this . leftNode != null )  
        synchronized ( this . leftNode )
```

```

        {
            this.leftNode.setMySidesIn(portValue);
        }
    }

    public synchronized void setMySidesIn(SidePort values)
    {
        for (int i=0;i<MAXINPUTS;i++)
        {
            this.sideInput.side[i]=values.side[i];
            this.sideInput.sideColor[i]=values.sideColor[i];
            this.sideInput.count[i].setCounter(values.count[i].getCounter());
        }
    }

    public int [] getMysidesIn()
    {
        return this.sideInput.side;
    }

    public synchronized void setMyinput(Port value)
    {
        synchronized(this)
        {
            this.nodeInput.port= value.port;
            this.nodeInput.color=value.color;
        }
    }

```

```
        this.nodeInput.count.setCounter(value.count.getCounter());  
    }  
}
```

```
public void setName(String name) {  
    this.name=name;  
}
```

```
public String getName(){  
    return this.name;  
}
```

```
public int getMaxInputs(){  
    return MAXINPUTS;  
}
```

```
public void setInputColor(Color c)  
{  
  
    this.nodeInput.color=c;  
    //    this.painter.drawInput(c);  
}
```

```
}

public Node.Port getPort()
{
    return this.nodeInput;
}

public int getMyOutput()
{
    return this.nodeOutput.port;
}

public String[] getMyOutputCounter()
{
    return this.nodeOutput.count.getCounter();
}

//Used to calculate the "delay"
public int getMyOutputCounterLength()
{
    return this.nodeOutput.count.getCounter().length;
}

public int getMyInput()
{
    return this.nodeInput.port;
}
```

```

public int [] getMysidesOut ()
{
    return this.sideOutput.side;
}

public void setInputV(int v[][],int w,int h)
{
    this.nodeInput.vertex[0][0]=v[0][0];
    this.nodeInput.vertex[0][1]=v[0][1];
    this.nodeInput.height=h;
    this.nodeInput.width=w;
}

public void setSideOutputV(int v[][], int w, int h) {

    this.sideOutput.height=h;
    this.sideOutput.width=w;
    for (int i = 0; i < sideOutput.vertex.length; i++) {
        sideOutput.vertex[i][0] = v[i][0];
        sideOutput.vertex[i][1] = v[i][1];
    }

}

public SidePort getSidePort ()
{

```

```

        return this.sideOutput;
    }

    //Set the position on the frame. This help to change the color
    public void setPosition(int x, int y)
    {
        coordinateX = x;
        coordinateY = y;
    }

    public void setCanIprocess(boolean f)
    {

        this.canIProcess=f;
    }

    public boolean getCanIprocess()
    {
        return this.canIProcess;
    }

    public int [] getPosition()
    {
        int c[]={coordinateX ,coordinateY};
        return c;
    }

    public void setNodeColor(Color c){

```

```
        this.nodeColor=c;

    }

    public Color getNodeColor() {
        return this.nodeColor;
    }

    public void setNodeWidth(int w){

        this.nodeWidth=w;
    }

    public int getNodeWidth()
    {
        return this.nodeWidth;
    }

    public void setNodeHeight(int l)
    {
        this.nodeLenght=l;
    }

    public int getNodeHeight()
    {
        return this.nodeLenght;
    }

    public void resetMyPorts(int in)
    {
```



```

    this.nodeInput.resetPort(in);
    this.nodeOutput.resetPort(in);
    this.sideInput.resetSidePorts();
    this.sideOutput.resetSidePorts();
}

//Set the node configuration to Redundant
void toRedundant()
{
    if (this.getType()==Node.nodeType.NRLOWER)
        this.setType(Node.nodeType.RLOWER);
    else if (this.getType()==Node.nodeType.NRUPPER)
        this.setType(Node.nodeType.RUPPER);

    setNodeColor(Color.BLACK);
    resetMyPorts(-1);
}

//Set the node configuration to Non-Redundant
void toNonRedundant()
{
    if (this.getType()==Node.nodeType.RLOWER)
        this.setType(Node.nodeType.NRLOWER);
    else if (this.getType()==Node.nodeType.RUPPER)
        this.setType(Node.nodeType.NRUPPER);
}

```

```
        setNodeColor(Color.BLUE);  
        resetMyPorts(0);  
  
    }  
  
} //end of class
```