Technische Universität München
TUM Department of Electrical and Computer Engineering
Chair of Electronic Design Automation

Router Port Assignment for Wavelength-Routed Optical Network-on-Chip

Bachelor Thesis

Xinyu Zhang

Technische Universität München
TUM Department of Electrical and Computer Engineering
Chair of Electronic Design Automation

Router Port Assignment for Wavelength-Routed Optical Network-on-Chip

Bachelor Thesis

Xinyu Zhang

Supervisor :	Dr. -Ing. Tsun-Ming Tseng
Supervising Professor :	Prof. Dr.-Ing. Ulf Schlichtmann
Topic issued :	27.07.2020
Date of submission :	03.12.2020

Xinyu Zhang
Arcisstraße 21
D-80333 Munich Germany

Abstract

Wavelength-routed optical networks-on-chip (WRONoC) is a promising platform for communication in multiprocessor systems-on-chips. A proposed method called CustomTopo (Li et al. 2018) is used to lower resource usage by reducing add-drop filters (ADFs) in a WRONoC topology. However, the ADF-reduction may lead to more crossing loss in the topology. In this work, I propose an algorithm to mitigate the drawbacks of CustomTopo. This algorithm can reassign the router ports of the topology to reduce the number of on-chip crossings resulting from the concept of CustomTopo. Moreover, this algorithm can also represent the change of off-chip connection caused by port reassignment. Experimental results show that this algorithm is applicable in various cases and can reduce the number of on-chip crossings up to 57.14%.

Acknowledgements

I would like to say a special thank you to my supervisor Mr. Tsun-Ming Tseng. His trust, support and encouragement have made this an inspiring experience for me. I would also like to thank my family for all the support, love and trust they have shown me. Finally, I would like to thank Mr. Alexander Paul Prommesberger, for all his love and support.

Contents

List of Figures

List of Tables

1. Introduction

Optical networks-on-chips (ONoCs) have arisen as an promising next-generation platform, with the rapid development of on-chip communication in multiprocessor systems-on-chips (MP-SoCs) (Li et al. 2018). ONoCs rely on waveguides to transmit signals. With waveguides, the conveyed electrical signals can be transmitted as optical signals and the optical signals can be guided from the laser source up to the receivers.

ONoCs can be classified into two categories based on their routing mechanisms: (1) *active networks* and (2) *passive networks*, the later is termed wavelength-routed optical networks-on-chips (WRONoCs) (Truppel et al. 2020). Active networks route the signals by using a real-time switching mechanism. In a passive network, the routing path of every master-slave pair is determined at its design time. Signals with different wavelengths can be transmitted through the same waveguide until they are demultiplexed by add-drop-filters (ADFs).

For WRONoCs, reducing resource usage is the first main goal to achieve during design time. Two important metrics for resource usage in a WRONoC topology are introduced: *the number of wavelengths* and *the number of ADFs* (Peano et al. 2016). Nowadays, most of the WRONoC topology generation requires full-connectivity (i.e. all masters need to connect to all slaves (Briere et al. 2007)). However, the masters are not always necessarily communicating with all slaves. Thus, the assumption of full-connectivity can lead to a high wastage of resource usage. To avoid this wastage, (Li et al. 2018) has proposed a new method named *CustomTopo*, which can generate customized WRONoC topologies without the assumption of full-connectivity, and the ADF usage can be reduced by using this method.

Moreover, lowering power consumption is the second main goal to achieve for the WRONoC topology during design time. *Insertion loss* is the main factor of power consumption (Tseng 2013). When light is inserted into an optical link, some power is taken away by photonic devices and optical paths, and this amount of power is called insertion loss. There are five components of insertion loss: (1) *propagation loss*, (2) *crossing loss*, (4) *drop loss*, (5) *through loss* and (5) *bending loss* (Nikdast et al. 2015). Propagation loss is the energy decrease along waveguides.

Crossing loss is caused by crossings between waveguides. Drop loss occurs at the drop points of microring resonators (MRRs). Through loss occurs when a signal directly goes through an MRR. Bending loss is the power loss that occurs for changing signal direction at waveguide bending points. Considering crossing loss causes more power consumption than the propagation loss with existing WRONoC technology parameters (Tseng et al. 2019)„ keeping the number of crossings as small as possible is the main goal to achieve to lower power consumption. Figure 1.1 has illustrated the concept of ADF reduction for a one-column matrix of CustomTopo. As we can see, after an ADF$_{m,s_j}$ with resonance frequency Λ_{m,s_j} is removed in the column, a corresponding external crossing is generated in the layout. The external crossing is marked in red in Figure 1.1. This external crossing can result in more crossing loss to the topology and lead to more power consumption. Thus, there is a pressing need for an optimization method that can reduce the crossing loss which CustomTopo produces.

The major contributions of this work include an algorithm focusing on the reassignment of router ports based on symmetric communication matrices representing the communication between ports to reduce the number of on-chip crossings. Nevertheless, port reassignment can lead to off-chip crossings. Therefore, this work also proposes a method that can calculate the number of off-chip crossings that port reassignment results in.

This thesis is structured as follows: this chapter is succeeded by the preliminaries, chapter 2, where the necessary knowledge and notations are introduced. The proposed algorithm of port reassignment and the method for calculating the number of off-chip crossings are in chapter 3. The results by solving nine test cases are presented in chapter 4. At last, the conclusion is given in chapter 5.

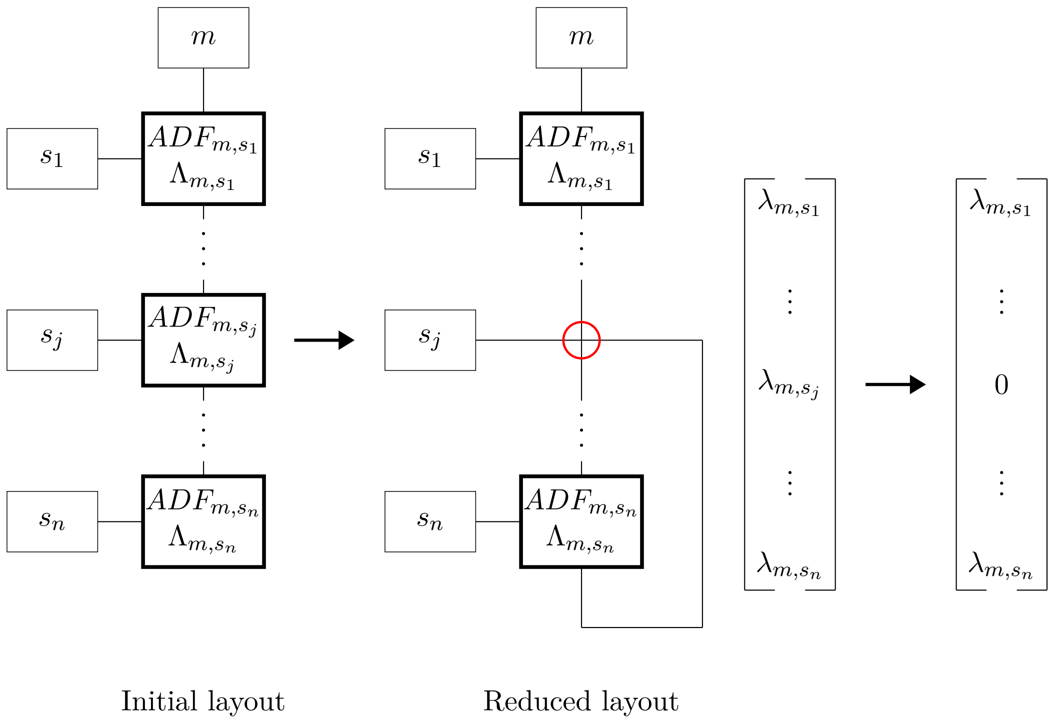

m
s_1
ADF_{m,s_1} Λ_{m,s_1}
s_j
ADF_{m,s_j} Λ_{m,s_j}
s_n
ADF_{m,s_n} Λ_{m,s_n}
Initial layout
Reduced layout
λ_{m,s_1} λ_{m,s_j} λ_{m,s_n}
0

Figure 1.1.: ADF reduction for a one-column matrix. (Li et al. 2018)

2. Preliminaries

This chapter introduces preliminary knowledge and concepts which are necessary to understand in the later chapters. At first, the 3D architecture for WRONoC applications is presented, in which the overview of the WRONoC structure is given. In section 2.2, the basic units on the photonic layer of the 3D architecture are discussed, which are memory controller, hub, and router. The function of the router is completed by using add-drop filters (ADFs). The ADFs are discussed in 2.3. Reducing the number of ADFs can be beneficial, this is discussed in 2.4. To model the communication between different components in WRONoC systems, communication matrix and logical topology are used. They are presented in 2.5. The on-chip crossings can occur in the logical layout of the communicative components, the concept of the generation of on-chip crossings is described in 2.6. Moreover, the components also have off-chip connections outside the logical topology. The off-chip crossings can occur in the off-chip layout, the concept of the generation of the off-chip crossings is described in 2.7. Finally, the routing problem of this thesis, which is solved in the following chapters, is formulated in 2.8.

2.1. 3D architecture for multicore processors

The 3D architecture for multicore processors is a typical setting for WRONoC applications. As shown in Figure 2.1, this structure consists of a vertically stacked photonic layer and an electronic layer (Ramini et al. 2013). Through-silicon vias (TSVs) connect the clusters of the processors on the electronic layer and their corresponding hubs on the photonic layer. The electrical signals travelling along the TSVs can be converted into or from optical signals by the laser sources providing on the photonic layer.

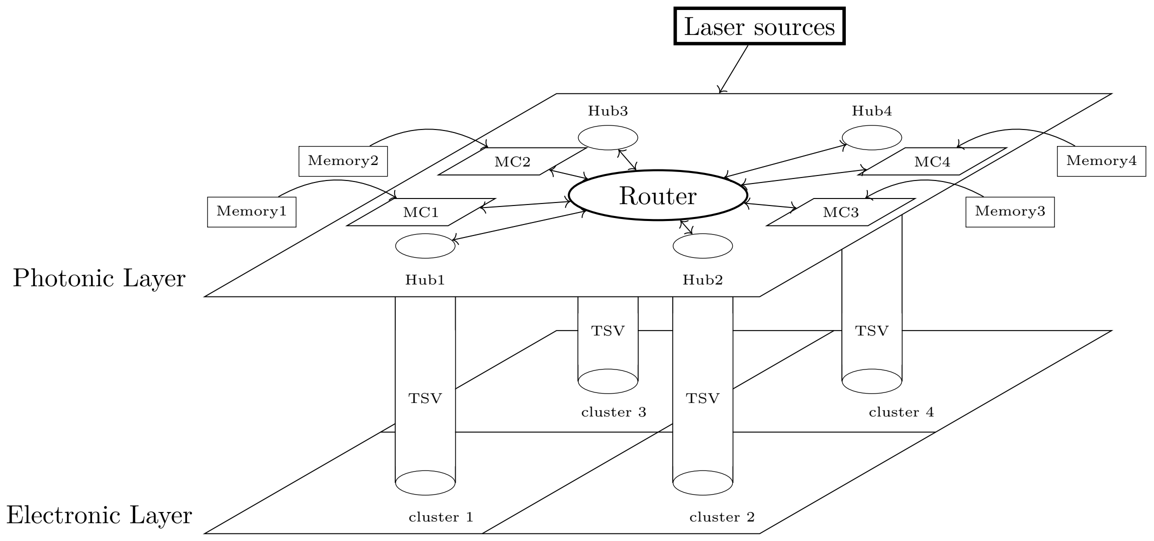


Figure 2.1.: A typical WRONoC setting on a 3D-stacked chip. (Tseng et al. 2019)

2.2. Basic units on photonic layer

As shown in Figure 2.1, the photonic layer consists of three parts: (1) *memory controllers (MCs)*, (2) *hubs*, and (3) *router*. The MCs control the flow of data going to and from the memory. The hubs are the interfaces where the conversion between optical signals and electrical signals takes place. The router can drive data transmission between different communication-pairs. There are three types of communications: (1) communication among clusters, (2) communication from a cluster to an MC, (3) communication from an MC to a cluster (Ramini et al. 2013). Thus, a cluster or an MC can either send or receive signals. If we denote *master* as the message sender and *slave* as the message receiver, then a component on the photonic layer can either be a master or a slave.

2.3. Add-drop filter (ADF)

If we simultaneously send signals with different wavelengths through a waveguide and want to direct them to different targets, we can use ADFs to demultiplex the signals and allow them to go to their targets. Figure 2.2 (a) illustrates a basic 2-*input* $\times$ 2-*output* ADF structure containing two crossing waveguides and two microring resonators (MRRs). An MRR has a circular silicon structure, and its radius defines its resonance frequency (Xiao et al. 2007). As shown in Figure 2.2 (a), different colors represent different wavelengths of the signals. The

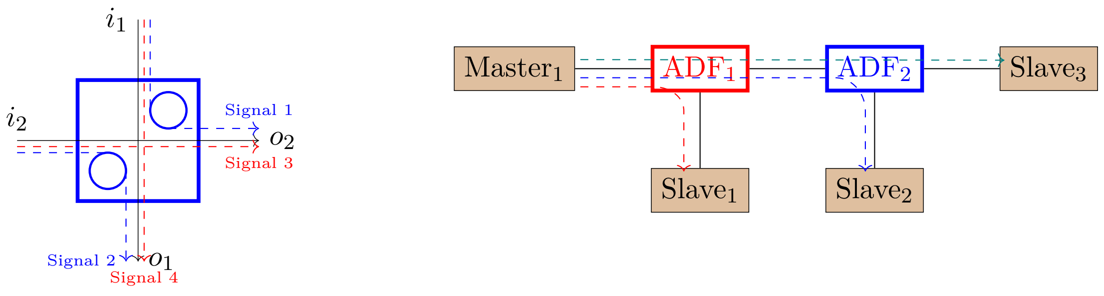


Figure 2.2.: Left: (a) basic structure of an ADF. Right: (b) functionality of ADFs.

resonance frequency of the ADF is marked in blue. The wavelengths of signal 1 and signal 2 are correspondent to the resonance frequency of the ADF. Therefore, signal 1, which enters input i_1, changes its direction along the MRR and is routed to output o_2. Similar to signal 1, signal 2, which enters input i_2, is routed to output o_1. In contrast to the blue signals, the red signals, signal 3 and signal 4 do not resonate with the ADF and maintain their directions.

Figure 2.2 (b) shows a simple WRONoC topology. The master and slaves denote output and input ports of the external modules, respectively. Suppose a signal group of three signals with different wavelengths is transmitted along the waveguide simultaneously. The red and blue signals are resonated with the resonance frequency of ADF$_1$ and ADF$_2$. Therefore, these two signals can be demultiplexed at ADF$_1$ and ADF$_2$ correspondingly and arrive at Slave$_1$ and Slave$_2$. The green signal, which does not resonate with any ADF, travels along the waveguide and arrives at Slave$_3$.

2.4. ADF reduction

A topology with fewer ADFs than its master-slave pairs is called a topology with an ADF reduction structure (Li et al. 2018). As shown in Figure 2.2 (b), the master communicates with three slaves by two ADFs. Even though the number of ADFs is smaller than the number of slaves, the signals with different wavelengths are routed successfully.

The default path and default slave of an ADF reduction structure are introduced in 2.4.1. The constraints which need to be put onto the default paths are introduced in 2.4.2.

2.4.1. Default path and default slave

In Figure 1.1, all slaves communicate with one master by dedicated ADFs in the initial layout. This 1-$master \times n$-$slaves$ topology consists of n ADFs. However, as shown in the reduced layout of Figure 1.1, $n-1$ ADFs are already sufficient to complete the data transmission among the master and the slaves. A randomly chosen ADF_{m,s_j} is replaced by a regular waveguide, and a new waveguide is used to connect the ADF_{m,s_n} and slave s_j. The signals which are resonated with Λ_{m,s_j} travel along the waveguide and across all the ADFs. Finally, these signals arrive at s_j without resonating with any of the ADFs since $\lambda_{m,s_j} \neq \Lambda_{m,s_i}$ for all $1 \leq i \leq n, i \neq j$. Therefore, s_j is defined as the *default slave* of m, and the signal path from m to s_j is defined as the *default path* of master m. In this case, λ_{m,s_j} is denoted as 0, indicating m and s_j does not need ADFs to communicate with each other.

2.4.2. Constraints on default path

As described in section 2.4.1, the default slave of a master m_i only exists while the other slaves connect with this master through ADFs. This statement can also be formulated as follows: every master can only possess one default slave.

As shown in Figure 2.2 (b), the green signal reaches its target Slave_3 by using the default path of Master_1. In this case, the green signal is the remaining signal. The remaining signal is defined as the signal which has passed all ADFs on the default path but did not resonate with any of the ADFs. If a signal is a remaining signal to communicate with a slave by using the default path of a master, it needs first to pass all the ADFs to reach its target. Therefore, the default path between m_i and s_j needs to consist of all ADFs in the i-th column and the j-th row of the logical topology (Li et al. 2018). The concept of the logical topology is described in the next section.

In conclusion, there are two constraints that need to be put onto default paths:

1. *Every master can only possess one default slave.*
2. *The default path between master m_i and slave s_j needs to consist of all ADFs in the i-th column and the j-th row of the logical topology.*

2.5. From communication matrix to logical topology

A *communication matrix* can model the entire communication behavior between all master-slave pairs (Truppel et al. 2019). The number of masters is denoted as n_m; the number of slaves is denoted as n_s. Each entry λ_{m_i,s_j} indicates the wavelength of the signal, which is travelling between master m_i and slave s_j for all $1 \leq i \leq n_m, 1 \leq j \leq n_s$. Then the communication matrix takes the form:

$$\begin{array}{c c}
 & \begin{array}{cccc} m_1 & m_2 & \cdots & m_{n_m} \end{array} \\
\begin{array}{c} s_1 \\ s_2 \\ \vdots \\ s_{n_s} \end{array} & \begin{pmatrix} \lambda_{m_1,s_1} & \lambda_{m_2,s_1} & \cdots & \lambda_{m_{n_m},s_1} \\ \lambda_{m_1,s_2} & \lambda_{m_2,s_2} & \cdots & \lambda_{m_{n_m},s_2} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{m_1,s_{n_s}} & \lambda_{m_2,s_{n_s}} & \cdots & \lambda_{m_{n_m},s_{n_s}} \end{pmatrix}
\end{array}$$

If there is no communication between m_i and s_j, the corresponding entry λ_{m_i,s_j} will then be set as NA; otherwise if the communication exists, the entry λ_{m_i,s_j} will be set as the wavelength of the signal which connects m_i and s_j, and this entry is denoted as * (Li et al. 2018). For the master-slave pairs which use default paths to communicate with each other, their entries are set as 0s.

A communication matrix can be illustrated by a logical topology. In the logical topology of an existing communication matrix, an ADF indicates a signal path between a master and a slave which is not directly connected by a waveguide. As shown in Figure 2.3, every * denoted connection is translated into ADF. Considering there is no communication between m_2 and s_2, there is no connection between this master-slave pair in the logical topology. There is a 0 in the communication matrix representing the default path between m_3 and s_3. The signal, which does not resonate with any other ADF on the path from m_3 and s_3, will travel along the waveguide without being affected by the ADFs.

Here I want to emphasize that as described in section 2.4.2, every master can only possess one default slave. Therefore, there could only be one 0 in each row and each column of the communication matrix.

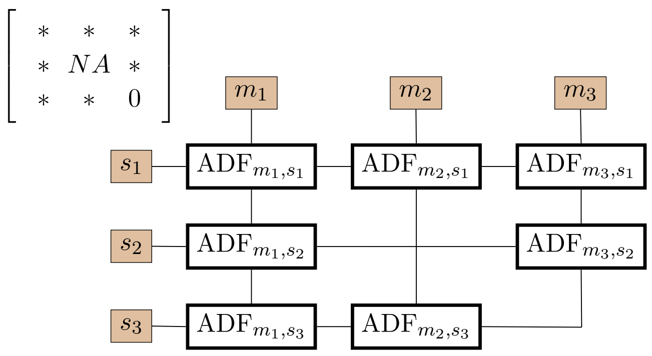


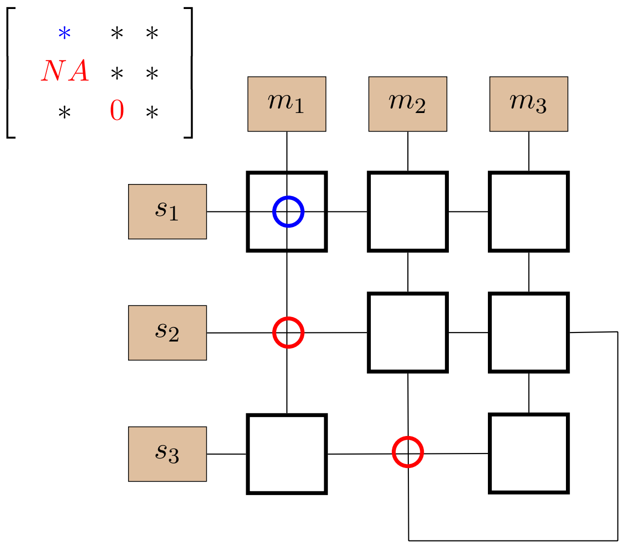
Figure 2.3.: Transformation from communication matrix into logical topology.

2.6. Generation of on-chip crossings

As illustrated in Figure 2.4, the type of the on-chip crossings can be classified into two categories: (1) *internal crossings*, which mean the crossings inside the ADFs; this crossing is marked in blue in the figure. (2) *external crossings*, which identify the crossings outside the ADFs; they are marked in red in the figure. The internal crossings result from the internal structure of ADFs, and they are mostly unavoidable. The commonly discussed crossing loss results from the external crossings. As we can see, after an ADF is replaced by a default path, an external crossing is generated. If a master-slave pair is a no-communicative pair, it can also generate a crossing in the topology. Therefore, the external crossing can be generated by 0 (representing default path) or NA (representing no-communicative pair) in the communication matrix. In the later chapters, on-chip crossings refer to external crossings in a topology.

Moreover, the location of the 0s and NAs also plays a role in generating on-chip crossings. As shown in Figure 2.5 and Figure 2.6, if the 0 or the NA is surrounded by *s (representing communication-pairs) in a communication matrix, there is a crossing in its corresponding logical topology. However, if the NAs locate in the n_m-th column or the n_s-th row, and the 0 locates at position (n_s, n_m) of the communication matrix, these NAs and this 0 cannot cause crossing to the logical topology.

Besides, if there are multiple 0s in a communication matrix, their location may also result in more crossings. As shown in Figure 2.7, if the 0s are located in a "downstairs" order (i.e. there exists a 0 which has a smaller row index and a smaller column index compared to one of the other 0s), there are only two crossings in the topology. On the contrary, if the 0s are located in an "upstairs" order (i.e. there exists a 0 which has a smaller row index and a larger column



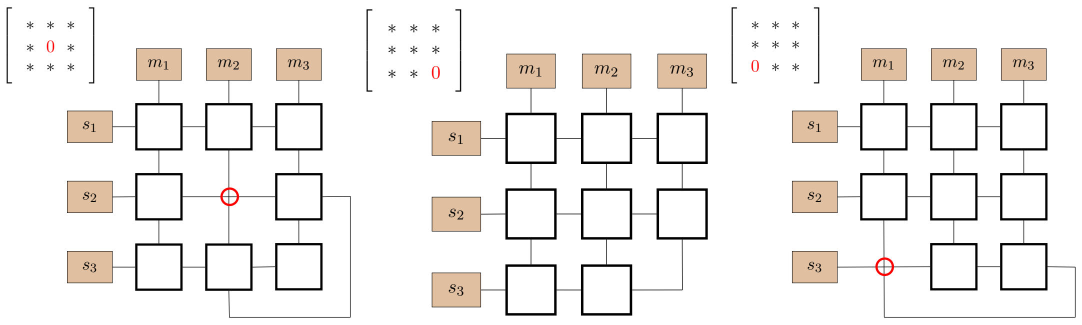
$\begin{bmatrix} * & * & * \\ NA & * & * \\ * & 0 & * \end{bmatrix}$
m_1
m_2
m_3
s_1
s_2
s_3

Figure 2.4.: Internal crossing and external crossing in a logical topology.

index compared to one of the other 0s), there are six crossings in total. The default path of (s_3, m_1) is used as an example to explain the mechanism of more crossings in an *upstairs zero structure*. It is clear to see in the figure that if the default path wants to travel through all ADFs in the first column and the third row, it has to cross the waveguides of (s_2, m_2) and (s_1, m_3), which can bring two more crossings to the topology. In a communication matrix, there can exist upstairs zero structure and downstairs zero structure at the same time, as shown in Figure 2.8. The number of on-chip crossings, which the upstairs zero structure causes, can be calculated with this equation:

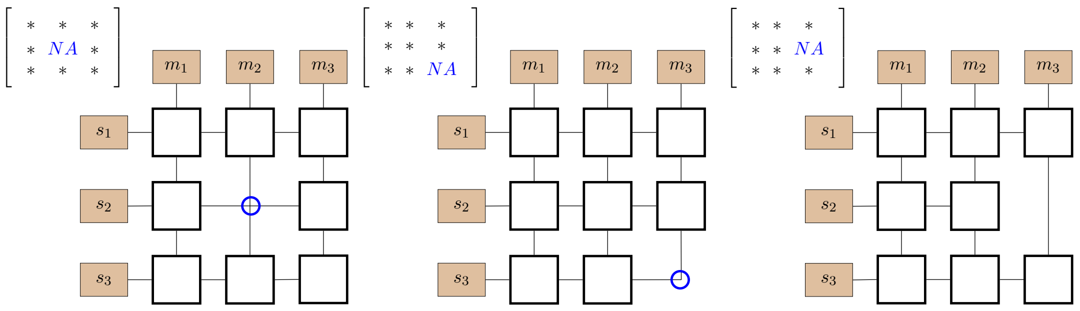
$$\begin{aligned} \#upstairs\ crossings = \#zeros(s_o, m_p)\ for\ zero(s_i, m_j),\ for\ o < i\ \&\ p\ > j, \\ for\ 1 \leq o, i \leq n_s,\quad 1 \leq j, p \leq n_m \end{aligned} \tag{2.1}$$

If a communication matrix has both the upstairs zero structure and the downstairs zero structure, this equation will only be applicable to calculate the number of on-chip crossings that the upstairs zero structure causes.



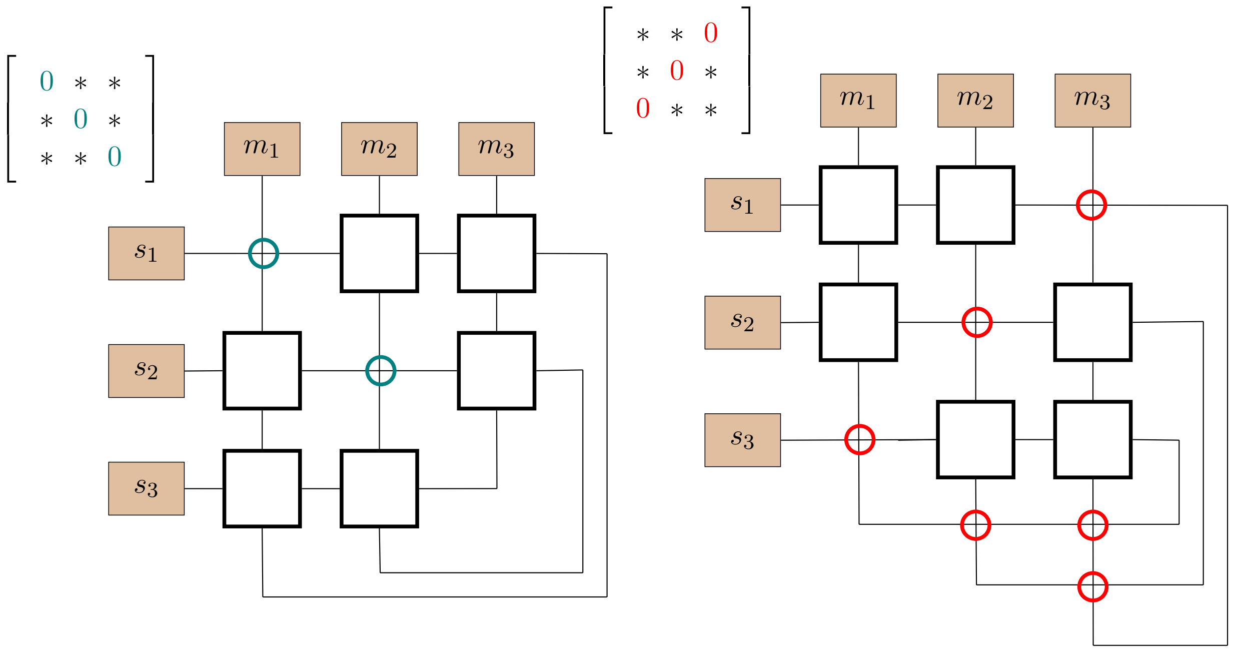
$\begin{bmatrix} * & * & * \\ * & 0 & * \\ * & * & * \end{bmatrix}$
$\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & 0 \end{bmatrix}$
$\begin{bmatrix} * & * & * \\ * & * & * \\ 0 & * & * \end{bmatrix}$
m_1
m_2
m_3
s_1
s_2
s_3

Figure 2.5.: Location of 0 can impact the number of on-chip crossing in a logical topology.



$\begin{bmatrix} * & * & * \\ * & NA & * \\ * & * & * \end{bmatrix}$
$\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & NA \end{bmatrix}$
$\begin{bmatrix} * & * & * \\ * & * & NA \\ * & * & * \end{bmatrix}$
m_1
m_2
m_3
s_1
s_2
s_3

Figure 2.6.: Location of NA can impact the number of on-chip crossing in a logical topology.



$\begin{bmatrix} 0 & * & * \\ * & 0 & * \\ * & * & 0 \end{bmatrix}$
$\begin{bmatrix} * & * & 0 \\ * & 0 & * \\ 0 & * & * \end{bmatrix}$
m_1
m_2
m_3
s_1
s_2
s_3

Figure 2.7.: Different structures of 0s in topology. Left: (a) downstairs zero structure. Right: (b) upstairs zero structure.

$$\begin{bmatrix} * & * & * & 0 \\ * & 0 & * & * \\ * & * & 0 & * \\ 0 & * & * & * \end{bmatrix}$$

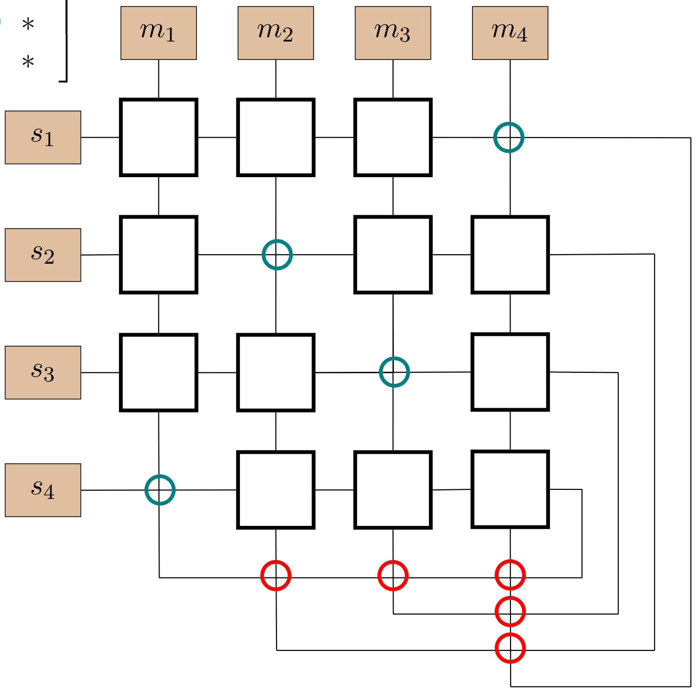


Figure 2.8.: A communication matrix with both the upstairs zero structure and the downstairs zero structure.

2.7. Generation of off-chip crossings

As discussed in section 2.2, a hub and an MC can be both the master and the slave. The master- and slave-ports in a logical topology are the pins of their corresponding hub or MC. This thesis assumes that the master-ports are located in the north of the communication matrix and the logical topology. The slave-ports are located in the west of the communication matrix and the logical topology. The same index of a master and a slave indicates one hub or one MC (e.g. $Master_1$ and $Slave_1$ are two pins of Hub_1, and $Master_2$ and $Slave_2$ are two pins of MC_1). The master-slave pair of the same hub or MC is connected off-chip.

This thesis defines *ports switch* as two ports switching their locations with each other. It is assumed that the ports switch can only happen among masters or slaves, and this switch cannot happen between a master and a slave. However, Figure 2.9 shows three kinds of ports switch. The three scenarios are: (b) switch master-ports, (c) switch slave-ports, and (d) switch both kinds of ports at the same time. Figure 2.10 models the off-chip connection of the ports switch in Figure 2.9. Ports 1 are marked in blue, ports 2 are in red, ports 3 are in orange, and ports 4 are in teal. It is noticeable, that there are five off-chip crossings in Figure 2.10 (b), and there are three off-chip crossings in Figure 2.10 (c). In the contrast, there is no off-chip crossing in Figure 2.10 (d). Therefore, if there are only ports switches among masters or slaves, there occur off-chip crossings in the off-chip layout. If two slaves and their off-chip connected masters are switched at the same time, as shown in Figure 2.10 (d), s_2 switches with s_4, and m_2 switches with m_4, there is no off-chip crossing. Therefore, if two master-ports are switched, and their off-chip connected slave-ports are also switched, the off-chip crossings that the master-ports switch has caused can compensate the off-chip crossings that the slave-ports switch has caused.

2.8. Routing problem formulation

In a WRONoC system, the location of the components, including communication-pair (denoted by *), default path (denoted by 0), and no-communicative pair (denoted by NA) in the communication matrix is the main factor in producing on-chip crossings in the logical topology. To reduce the number of on-chip crossings, I can change the location of the components by reassigning their master- and slave-ports. The port reassignment is done by performing a sequence of ports switches. The reassignment needs to be based on the existing communicative

$$\begin{array}{c|cccc} & m_1 & m_2 & m_3 & m_4 \\ \hline s_1 & * & 0 & * & * \\ s_2 & * & * & NA & * \\ s_3 & * & * & * & * \\ s_4 & NA & * & 0 & * \end{array} \qquad \begin{array}{c|cccc} & m_4 & m_2 & m_3 & m_1 \\ \hline s_1 & * & 0 & * & * \\ s_2 & * & * & NA & * \\ s_3 & * & * & * & * \\ s_4 & * & * & 0 & NA \end{array}$$

$$\begin{array}{c|cccc} & m_1 & m_2 & m_3 & m_4 \\ \hline s_3 & * & * & * & * \\ s_2 & * & * & NA & * \\ s_1 & * & 0 & * & * \\ s_4 & NA & * & 0 & * \end{array} \qquad \begin{array}{c|cccc} & m_1 & m_4 & m_3 & m_2 \\ \hline s_1 & * & * & * & 0 \\ s_4 & NA & * & 0 & * \\ s_3 & * & * & * & * \\ s_2 & * & * & NA & * \end{array}$$

Figure 2.9.: Different types of ports switch. Upper left: (a) the original communication matrix. Upper right: (b) the communication matrix after the master-ports switch between m_1 and m_4. Bottom left: (c) the communication matrix after the slave-ports switch between s_1 and s_3. Bottom right: (d) the communication matrix after both the master- and the slave-ports switch between m_2 and m_4, s_2 and s_4.

relations between masters and slaves. Moreover, the reassignment of ports can impact the off-chip connection between masters and slaves. Therefore, the change of off-chip connection caused by port reassignment is also important to be taken into consideration.

In this thesis, I take communication matrices as the input matrix. As described in section 2.7, I constrain the master-ports on the north of the communication matrix, and the slave-ports on the west of the communication matrix. The logical topology of the communication matrix is used to illustrate the on-chip connection between the master-slave pairs. The off-chip layout of the communication matrix is used to illustrate the off-chip connection of the master-slave pairs.

The first goal is an optimization algorithm to reduce the number of on-chip crossings of the input matrix by reassigning the ports of the input matrix. Moreover, another goal is a method to analyze the off-chip connection after the port reassignment.

Many factors can result in on-chip crossings. Three questions are essential to the optimization algorithm: It is worth discussing which structure can cause more on-chip crossings than other structures. If this structure is optimizable, is also a meaningful question that needs to be discussed. From this question, how to optimize this structure becomes another considerable question. Therefore, analyzing and solving these questions are also important to achieve my goals.

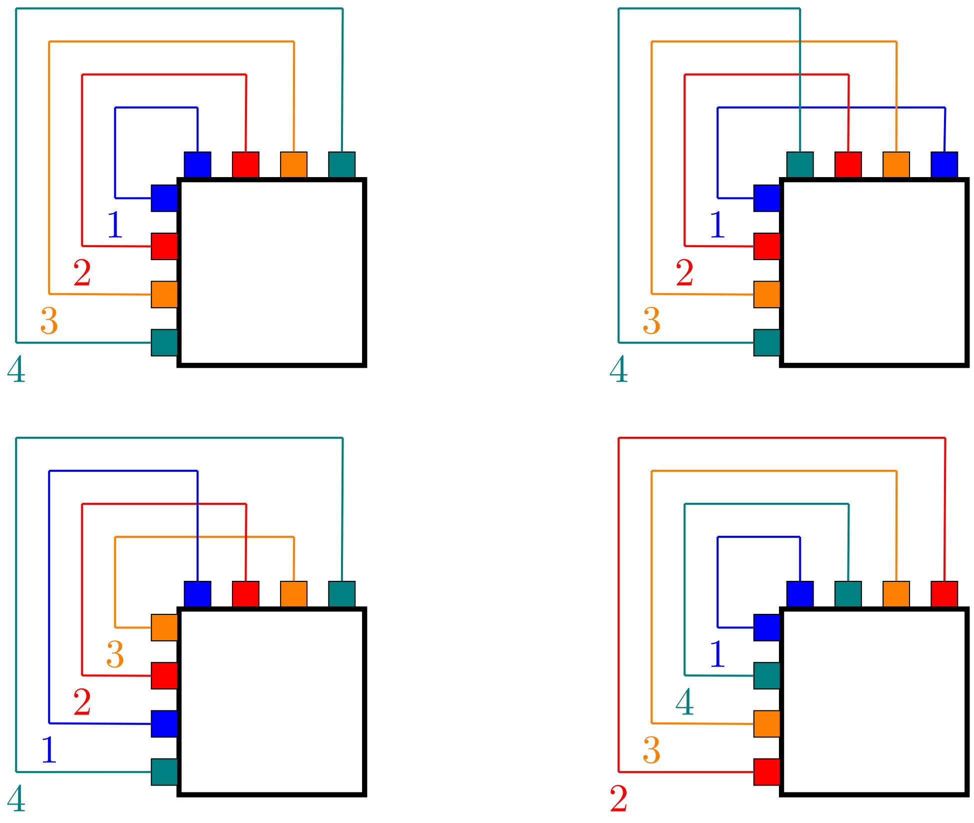


Figure 2.10.: Off-chip layouts of communication matrices in Figure 2.9. Upper left: (a) the original off-chip layout. Upper right: (b) the off-chip layout after the master-ports switch between m_1 and m_4. Bottom left: (c) the off-chip layout after the slave-ports switch between s_1 and s_3. Bottom right:(d) the off-chip layout after the master- and slave-ports switch between m_2 and m_4, s_2 and s_4.

If I conclude the inputs, constraints, and goals, the routing problem of this thesis can be formulated as follows:

reduce:

    Number of on-chip crossings

subject to:

    Port reassignment (2.2)

& Analyzing the off-chip connection

given:

    Communication matrix, its logical topology, and its off-chip layout

3. Methodology

In the proposed method of optimizing WRONoC systems by reassigning their ports, the number of on-chip crossings is among the most important factors. However, before optimizing a system, the initial status of this system needs to be analyzed. After analyzing the initial status of the system, the most suitable optimization process can be applied on the system. To identify the improvement after the optimization, a comparison between initial status and optimized status needs to be done.

Based on above steps, this chapter is structured as follows: I propose an equation in 3.1 to calculate the on-chip crossings of a logical topology from its communication matrix. I then analyze the structure of the input matrix in 3.2. To reduce the number of the on-chip crossings of the input matrix, I propose a method to optimize this input matrix in 3.3. Finally, a finding about how the off-chip connection changes caused by the optimization is presented in 3.4.

3.1. Count on-chip crossings of the input communication matrix

As described in chapter 1, the crossing loss is the major component of the insertion loss of a WRONoC system. Before starting to optimize the input matrix, there is a need to count the number of on-chip crossings. As it was explained in section 2.6, in a communication matrix, 0s and NAs are the cause of on-chip crossings in the corresponding logical topology. Moreover, as shown in Figure 2.5 and 2.6, the position of 0s and NAs has a significant impact on the total number of on-chip crossings in a logical topology. After analyzing the position of 0s and NAs individually and carefully, I then propose the following equation to calculate the number of on-chip crossings:

$$\begin{array}{rl}
 & \#zeros + \#NAs \quad (0) \\
- & \#zero\ at\ position\ (n_s, n_m) - \#NA\ at\ position\ (n_s, n_m) \quad (1) \\
- & \#NA\ in\ the\ n_s - th\ row - \#NA\ in\ the\ n_m - th\ column\ except\ for\ position\ (n_s, n_m) \quad (2) \\
+ & \#zeros(s_o, m_p)\ for\ zero(s_i, m_j),\ for\ o < i\ \&\ p > j,\ for\ 1 \le o,\ i \le n_s,\ 1 \le j,\ p \le n_m \quad (3) \\
- & \#NAs\ in\ sequence\ with\ NA(s_i, m_{n_m}),\ for\ 1 \le i \le n_s\ on\ column\ i \quad (4) \\
- & \#NAs\ in\ sequence\ with\ NA(s_{n_s}, m_j),\ for\ 1 \le j \le n_m\ on\ row\ j \quad (5) \\
+ & \#NAs(s_a, m_{n_m} - e + 1),\ for\ 1 \le e \le x,\ if\ zero(s_a, m_b)\ \&\ NA(s_a, m_{n_m}) \quad (6) \\
 & for\ 1 \le a \le n_s,\ 1 \le b \le n_m \\
 & (x\ is\ number\ of\ NAs\ in\ sequence\ with\ NA(s_a, m_{n_m})) \\
+ & \#NAs(s_{n_m} - f + 1, m_b),\ for\ 1 \le f \le y,\ if\ zero(s_a, m_b)\ \&\ NA(s_{n_s}, m_b) \quad (7) \\
 & for\ 1 \le a \le n_s,\ 1 \le b \le n_m \\
 & (y\ is\ number\ of\ NAs\ in\ sequence\ with\ NA(s_{n_s}, m_b)) \\
- & \#zero(s_o, m_{n_m})\ if\ NA(s_i, m_{n_m}),\ for\ o \le i \le n_s \quad (8) \\
- & \#zero(s_{n_s}, m_p)\ if\ NA(s_{n_s}, m_j),\ for\ p \le j \le n_m \quad (9) \\
\hline
= & \#on - chip\ crossings
\end{array} \tag{3.1}$$

This equation can be explained as follows:

(0) As default, I assume that every 0 and NA in the input matrix can result in one crossing.

(1) If there is a 0 or an NA at position (n_s, n_m) of the communication matrix, as shown in

Figure 2.5 and 2.6, this 0 or NA cannot cause on-chip crossing in the logical topology. Therefore the number of this 0 or NA should be subtracted from the total number of on-chip crossings.

(2) As proposed in section 2.6, if the NAs place in the n_s-th row or the n_m-th column of the communication matrix, they can not cause any crossing in the logical topology. Therefore, they need to be subtracted from the total number of on-chip crossings. Considering the 0 or NA at position (n_s, n_m) is already calculated with equation (1). Therefore, this 0 or NA needs to be excluded.

(3) If there is an upstairs zero structure in the communication matrix, the number of on-chip crossings, which this structure can cause, can be calculated with equation (3). This number needs to be added to the total number of on-chip crossings.

(4)(5) From Figure 3.1, it is clear to see, that if there is an NA$_\alpha$ at position (s_i, m_{n_m}), or an NA at position (s_{n_s}, m_j) of the communication matrix, for $1 \leq i < n_s$, $1 \leq j < n_m$, and there exist NAs, which are in sequence with this NA, these sequencing NAs cannot result in crossings. Therefore, the number of these sequencing NAs of NA$_\alpha$ needs to be subtracted from the total number of on-chip crossings.

(6)(7) Figure 3.2 reveals a fact that if there is a 0 which is located in the s_i-th row or the m_j-th column in cases (4) and (5), the default path of this 0 needs to travel through the path, where the NAs in cases (2), (4) and (5) locate. Therefore, these NAs can cause new crossings, and the number of these NAs needs to be added to the total number of on-chip crossings.

(8)(9) Figure 3.3 illustrates a critical situation of cases (6) and (7): there is a 0 in the n_s-th row or the n_m-th column of the communication matrix, and the elements on its right or below it are all NAs. In this case, the default path which this 0 causes does not cross with any other path. Therefore, this 0 cannot cause any crossing in the logical topology, and this 0 needs to be subtracted from the total number of on-chip crossings.

3.2. Analyze the input communication matrix

After counting the total number of on-chip crossings of the input matrix, I now focus on the structure of the input matrix and comprehensively analyze the matrix structure.

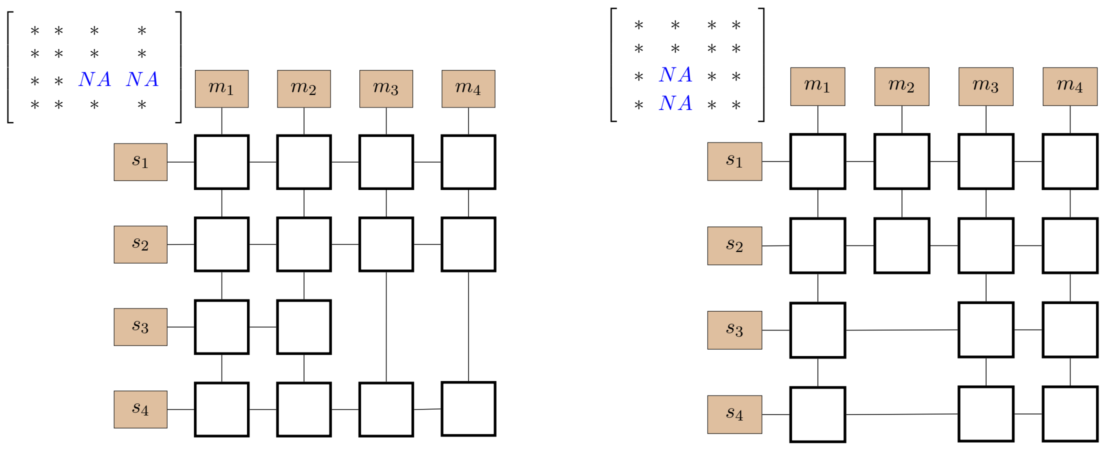


Figure 3.1.: NAs in sequence in the column and the row.

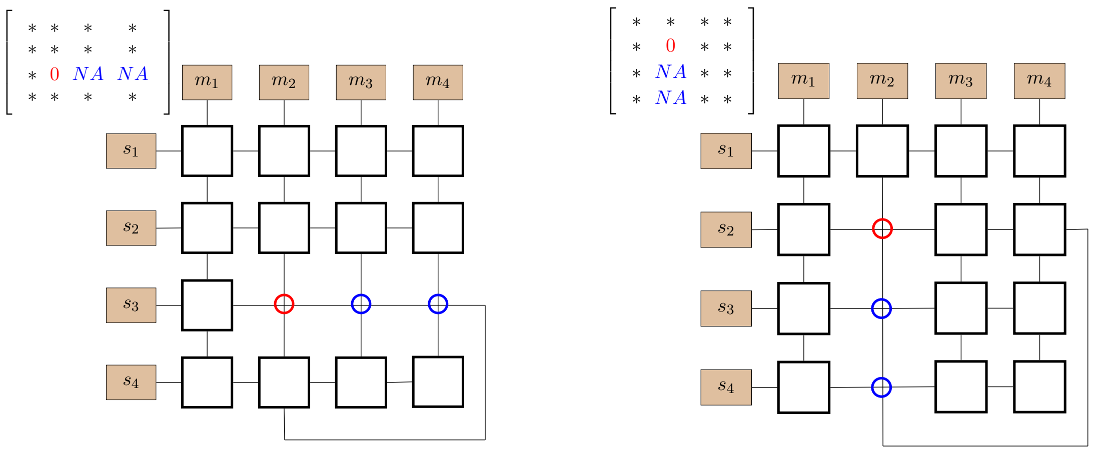


Figure 3.2.: NAs in sequence in the column and the row with 0.

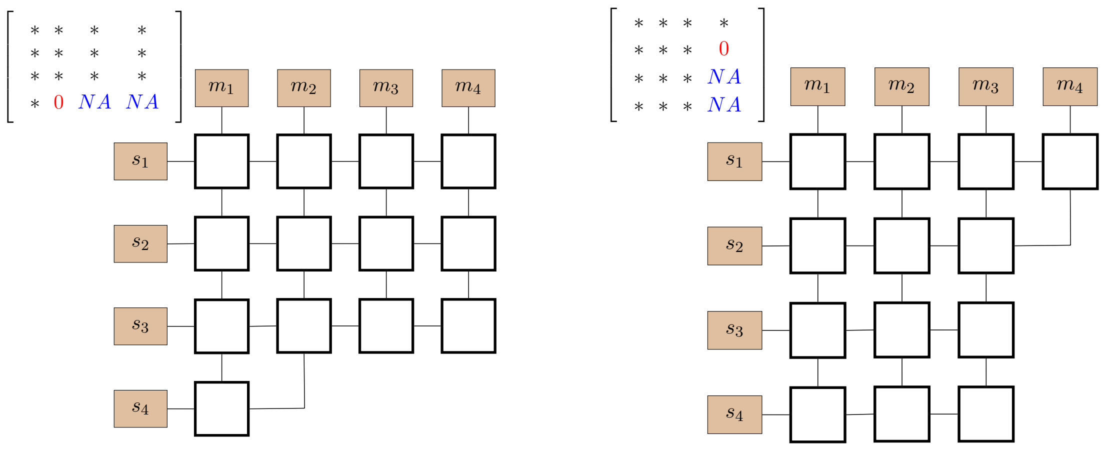


Figure 3.3.: NAs in sequence in n_m-th column and n_s-th row with 0.

$$\begin{array}{ccccccccc}
 & \multicolumn{2}{c}{C_{1,1}} & & & & \multicolumn{2}{c}{C_{1,n}} & \\
 & a_{1,1} & a_{1,2} & \cdots & & & a_{1,2n-1} & a_{1,2n} & \\
 & a_{2,1} & a_{2,2} & \cdots & & & a_{2,2n-1} & a_{2,2n} & \\
C_{2,1} & a_{3,1} & a_{3,2} & \cdots & & & & & \\
 & a_{4,1} & a_{4,2} & \cdots & & & & & \\
 & \vdots & \vdots & \ddots & & & \vdots & \vdots & \\
 & a_{2n-1,1} & a_{2n-1,2} & \cdots & a_{2n-1,2n-3} & a_{2n-1,2n-2} & a_{2n-1,2n-1} & a_{2n-1,2n} & C_{n,n} \\
 & a_{2n,1} & a_{2n,2} & \cdots & a_{2n,2n-3} & a_{2n,2n-2} & a_{2n,2n-1} & a_{2n,2n} & \\
 & \multicolumn{2}{c}{C_{n,1}} & & \multicolumn{2}{c}{C_{n,n-1}} & & &
\end{array}$$

Figure 3.4.: A divided $2 \cdot n \times 2 \cdot n$ matrix.

In this section, I first divide the input matrix into smaller units in section 3.2.1, and then check the optimizability and efficiency of these units in section 3.2.2 and 3.2.3.

3.2.1. Divide the input matrix

The author of (Ramini et al. 2013) has proposed a restriction of network partition: at most four masters and four slaves are used to interconnect with each other in each network partition. Therefore, I constrain the size of the input matrix to $2 \cdot n \times 2 \cdot n$ and observe 2-*masters*$\times$2-*slaves* grid as the fundamental structure of a $2 \cdot n \times 2 \cdot n$ topology. Each $2 \cdot n \times 2 \cdot n$ input matrix is then divided into multiple 2×2 grids, and they are defined as *cells* of the input matrix. As shown in Figure 3.4, the matrix is divided into n^2 cells, and each of them is denoted as $C_{i,j}$, for all $1 \leq i, j \leq n$, individually.

3.2.2. Check the optimizability of the cells

As shown in equation (3.1), the location of the 0s and NAs in a communication matrix is an important factor to determine the total number of on-chip crossings. Therefore, if I want to reduce the number of on-chip crossing in a logical topology, I can change the location of the 0s and NAs in its communication matrix by reassigning the ports. As illustrated in Figure 3.5 (a), there are four crossings resulting from the location of the NA and 0s in this communication matrix. However, if I switch master-ports m_3 and m_4, slave-ports s_3 and s_4, the number of on-chip crossing in the optimized topology in Figure 3.5 (b) is reduced to zero. In this example, the cell $C_{2,2}$ can be optimized by switching its master- and slave-ports at the same time to reduce the number of on-chip crossings that this cell causes.

After analyzing every possible cell structure, I have formulated these categories about optimizabilty of a cell: (1) *Cell cannot be optimized.* (2) *Cell can be optimized by switching its master-ports.* (3) *Cell can be optimized by switching its slave-ports.* (4) *Cell can be optimized by switching its master- and slave-ports at the same time.* (5) *Cell does not need to be optimized.*

If I classify the cells into the above categories according to their structures, the different structure with different attributes of optimizability results in seven different cases:

Case 1 **Cell cannot be optimized:** If there are two non-ADF elements in the cell, and they place either in the diagonal or in the anti-diagonal of the cell, and at least one of these two elements is zero, then this kind of cells cannot be optimized by switching the ports:

$$\begin{pmatrix} 0 & * \\ * & NA \end{pmatrix}, \begin{pmatrix} 0 & * \\ * & 0 \end{pmatrix}, \begin{pmatrix} * & 0 \\ NA & * \end{pmatrix}, \begin{pmatrix} * & NA \\ 0 & * \end{pmatrix}, \begin{pmatrix} NA & * \\ * & 0 \end{pmatrix}, \begin{pmatrix} * & 0 \\ 0 & * \end{pmatrix}.$$

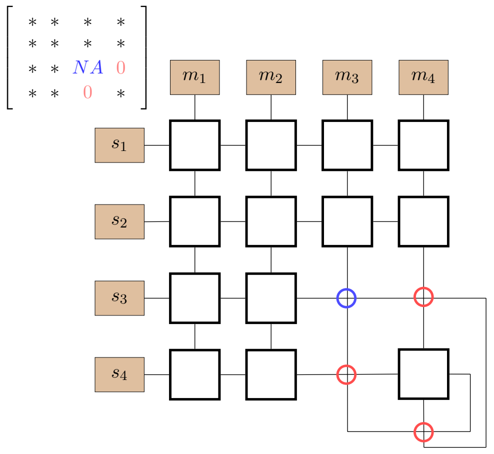
Case 2 **Cell can be optimized by switching its master- or slave-ports:** If there are only two NAs in the cell, and they place in the diagonal of the cell, then this kind of cell can be optimized by switching either its master- or slave-ports. To simplify the algorithm, I consider that this kind of cells can be optimized by switching its master-ports:

$$\begin{pmatrix} NA & * \\ 0 & NA \end{pmatrix}, \begin{pmatrix} NA & 0 \\ * & NA \end{pmatrix}, \begin{pmatrix} NA & * \\ * & NA \end{pmatrix}, \begin{pmatrix} NA & 0 \\ 0 & NA \end{pmatrix}.$$

Case 3 **Cell can be optimized by switching its master- or slave-ports:** If there is only one 0 or one NA in the cell, the rest of the elements are all ADFs, and this 0 or this NA is located in the anti-diagonal of the cell; or if there is only one ADF in the cell, and this ADF positions at the bottom left corner of the cell, and the rest of the elements are all NAs, then this kind of cell can be optimized by switching its master- or slave-ports. To simplify the algorithm, I consider that this kind of cell can be optimized by switching its master-ports:

$$\begin{pmatrix} * & * \\ 0 & * \end{pmatrix}, \begin{pmatrix} * & 0 \\ * & * \end{pmatrix}, \begin{pmatrix} * & * \\ NA & * \end{pmatrix}, \begin{pmatrix} * & NA \\ * & * \end{pmatrix}, \begin{pmatrix} NA & NA \\ * & NA \end{pmatrix}.$$

Case 4 **Cell can only be optimized by switching its slave-ports:** If the 0 in the cell has a neighboring NA and there is an ADF under this 0, this kind of cell can only be optimized by switching its slave-ports:



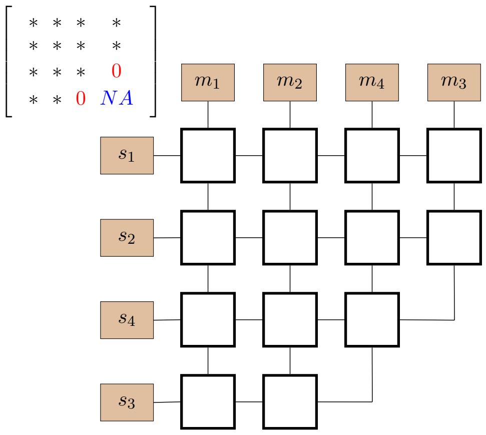


Figure 3.5.: Optimize the communication matrix by switching its ports. Left: (a) original communication matrix and its logical topology. Right: (b) optimized communication matrix and its logical topology.

$$\begin{pmatrix} 0 & NA \\ * & NA \end{pmatrix}, \begin{pmatrix} 0 & NA \\ * & 0 \end{pmatrix}, \begin{pmatrix} NA & 0 \\ NA & * \end{pmatrix}, \begin{pmatrix} 0 & NA \\ * & * \end{pmatrix}, \begin{pmatrix} NA & 0 \\ * & * \end{pmatrix}.$$

Case 5 Cell can only be optimized by switching its master-ports: If the 0 in the cell has a neighboring NA and there is an ADF on the right of the 0; or if there are two ADFs and two NAs in the cell, and the NAs are all on the ADFs' left side, this kind of cell can only be optimized by switching its master-ports:

$$\begin{pmatrix} 0 & * \\ NA & NA \end{pmatrix}, \begin{pmatrix} 0 & * \\ NA & 0 \end{pmatrix}, \begin{pmatrix} NA & NA \\ 0 & * \end{pmatrix}, \begin{pmatrix} NA & * \\ 0 & * \end{pmatrix}, \begin{pmatrix} 0 & * \\ NA & * \end{pmatrix}, \begin{pmatrix} NA & * \\ NA & * \end{pmatrix}.$$

Case 6 Cell can be optimized by switching its master- and slave-ports at the same time: In this case, the presenting cells are the cells whose master- and slave-ports both need to be switched:

$$\begin{pmatrix} NA & 0 \\ 0 & * \end{pmatrix}, \begin{pmatrix} 0 & * \\ * & * \end{pmatrix}, \begin{pmatrix} NA & * \\ * & * \end{pmatrix}.$$

Case 7 Cell does not need to be optimized: The not presented cells are the cells, which do not cause any crossing to the topology. Therefore, this kind of cell can be categorized as does not need to be optimized.

3.2.3. Check the efficiency of the cells

To explore more properties of cells, I focus on the efficiency of the cells. The efficiency of a cell is related to the number of non-ADF elements in the cell and shows the relation between the number of on-chip crossings, which the cell can produce, and the number of non-ADF elements in the cell. The categories of cell efficiency are: *inefficient*, *middle efficient*, and *efficient*. Different structure of cell with different cell efficiency is as below:

Case 1 Inefficient cells: Inefficient cells are the cells with multiple non-ADF elements inside the cell, and every non-ADF element can cause an on-chip crossing in the topology. Therefore, the number of on-chip crossings in the topology of an inefficient cell is equal to the number of non-ADF elements in the cell, despite the cells with upstairs zero structure. The cells with an upstairs zero structure can cause one more crossing than its number of non-ADF elements.

- The cells with upstairs zero structure are inefficient: $\begin{pmatrix} * & 0 \\ 0 & * \end{pmatrix}, \begin{pmatrix} NA & 0 \\ 0 & * \end{pmatrix}$.
- If there is only one 0 in the cell, and ADFs occupy the other positions, the cell is inefficient: $\begin{pmatrix} * & * \\ 0 & * \end{pmatrix}, \begin{pmatrix} 0 & * \\ * & * \end{pmatrix}, \begin{pmatrix} * & 0 \\ * & * \end{pmatrix}$.
- If the element at position (1,1) of the cell is a non-ADF element, and it neighbors with a non-ADF elements (this neighbor is different from the element at position (1,1)), and the rest of the elements are ADFs, then the cell is inefficient:
$$\begin{pmatrix} NA & * \\ 0 & * \end{pmatrix}, \begin{pmatrix} 0 & * \\ NA & * \end{pmatrix}, \begin{pmatrix} NA & 0 \\ * & * \end{pmatrix}, \begin{pmatrix} 0 & NA \\ * & * \end{pmatrix}.$$
- If there is one NA at position (1,1) of the cells, and the rest of the elements are all ADFs, this cell is inefficient: $\begin{pmatrix} NA & * \\ * & * \end{pmatrix}$.

Case 2 Middle efficient cells: Middle efficient cells are the cells with multiple non-ADF elements inside the cell, but not all of the non-ADF elements can cause on-chip crossing in the topology. Therefore, the number of on-chip crossings in the topology of a middle efficient cell is smaller than the number of non-ADF elements in the cell.

- If there are two NAs in the cell and they are located in the diagonal of the cell, this kind of cell is middle efficient: $\begin{pmatrix} NA & * \\ 0 & NA \end{pmatrix}, \begin{pmatrix} NA & 0 \\ * & NA \end{pmatrix}, \begin{pmatrix} NA & * \\ * & NA \end{pmatrix}, \begin{pmatrix} NA & 0 \\ 0 & NA \end{pmatrix}$.

- If there is a 0 at position (1,1) of the cell, and a non-ADF element is located at position (2,2), and there is at least one ADF in the cell, this kind of cell is middle efficient:

$$\begin{pmatrix} 0 & * \\ NA & NA \end{pmatrix}, \begin{pmatrix} 0 & NA \\ * & NA \end{pmatrix}, \begin{pmatrix} 0 & * \\ * & NA \end{pmatrix}, \begin{pmatrix} 0 & * \\ NA & 0 \end{pmatrix}, \begin{pmatrix} 0 & NA \\ * & 0 \end{pmatrix}, \begin{pmatrix} 0 & * \\ * & 0 \end{pmatrix}.$$

- If there is at least one NA in the cell, and there is one ADF at position (2,2), and there is one 0 at position (1,2) or (2,1), this kind of cell is middle efficient:

$$\begin{pmatrix} NA & 0 \\ NA & * \end{pmatrix}, \begin{pmatrix} * & 0 \\ NA & * \end{pmatrix}, \begin{pmatrix} NA & NA \\ 0 & * \end{pmatrix}, \begin{pmatrix} * & NA \\ 0 & * \end{pmatrix}.$$

- If there is one NA at position (1,1) of the cell, one 0 at position (2,2), and the rest of the elements are ADFs, this kind of cell is middle efficient:

$$\begin{pmatrix} NA & * \\ * & 0 \end{pmatrix}.$$

Case 3 Efficient cells: Efficient cells can possess multiple non-ADF elements inside the cell or do not possess any of the non-ADF elements inside the cell. If there are non-ADF elements in this kind of cell, none of these elements can cause on-chip crossings to the topology. Then, these cells are efficient. The cells which I did not enumerate here are all efficient.

3.3. Optimize the input matrix

To recap, the last two sections have discussed the cause of on-chip crossings and the optimizability and efficiency of each cell. Now I discuss the optimization algorithm to reduce the number of on-chip crossings in the logical topology of the input matrix.

The optimization process has two steps: the relocation of the input communication matrix in section 3.3.1 and the elimination of the upstairs zero structure in section 3.3.2.

3.3.1. Relocate the input matrix

In section 2.6, it was shown that the number of on-chip crossings in the logical topology could be efficiently reduced, if the 0s and NAs are located in the n_m-th column or the n_s-th row of the communication matrix. Moreover, if a cell is efficient, it cannot result in on-chip crossings in the logical topology; or if a cell is optimizable, it cannot result in on-chip crossings in the logical topology after the ports switch. Therefore, the concept of relocating the input matrix is to switch the efficient or the optimizable cell with the cell $C_{n,n}$ to reduce the number of on-chip crossings in the logical topology.

As discussed in section 2.7, if two master-ports are switched, and their off-chip connected slave-ports are also switched, the off-chip crossings that the master-ports switch has caused can compensate the off-chip crossings that the slave-ports switch has caused. Therefore, if I want to switch the efficient or optimizable cell with the cell $C_{n,n}$ and prevent extra off-chip crossings, I need to first consider the cells in the diagonal of the input matrix. If there is no efficient or optimizable cell in the diagonal of the input matrix, I need to focus on the neighbors of cell $C_{n,n}$. The off-chip crossings which the relocation of the input matrix produces, are discussed in section 3.4.

Algorithm 1 shows the complete concept of the relocation. The relocation can be put into the following steps:

- From line 13 to line 19, it is first checked, if the edge matrix cell $C_{n,n}$ belongs to the nonoptimizable cell group. If the edge matrix is optimizable and belongs to the efficient cells group, the algorithm takes the input matrix as the output matrix. If the edge matrix is not efficient but optimizable, the algorithm optimizes this edge matrix by using the function switch_ports to switch the ports of this edge matrix depending on its optimizability. If the edge matrix does not fulfill any of these two conditions, to prevent extra off-chip crossings, the algorithm moves on to the diagonal matrices $C_{i,i}$, for $1 \leq i < n$.

- The algorithm executes the same procedure as for the edge matrix for the diagonal matrices from line 21 to line 30, until one of the efficient or optimizable diagonal matrix is switched with edge matrix $C_{n,n}$ and is relocated to the edge of the input matrix. The ports switches between diagonal matrices and edge matrix are switching two master-ports and their off-chip connected slave-ports. Therefore, this kind of ports switches cannot result in any off-chip crossings.

- If the edge matrix and the diagonal matrices cannot fulfill all conditions (these matrices are neither efficient nor optimizable), the algorithm is then implemented for the left matrix $C_{n,n-1}$ of the edge matrix, see line 32 to line 40. Even though ports switches between left matrix and edge matrix can reduce the total number of on-chip crossings in the logical topology, switching the left matrix with the edge matrix only requires master-ports switches. Therefore, no slave-ports switch can compensate the off-chip crossings that the master-ports switches causes. In this case, off-chip crossings are generated.

- If the conditions for the edge matrix, the diagonal matrices, and the left matrix fail, this algorithm will then be implemented for the upper matrix $C_{n,n-1}$ of the edge matrix, as shown from line 41 to line 49. Similar to switching the left matrix and the edge matrix, switching the upper matrix and the edge matrix only requires slave-ports switches. Therefore, off-chip crossings are generated by the slave-ports switches.

If the number of off-chip crossings that this algorithm causes is bigger than the number of on-chip crossings that this algorithm reduces, this step is skipped.

3.3.2. Eliminate the upstairs zero structure of input matrix

In section 2.6, I proposed a new idea, namely upstairs zero structure. This structure is a communication matrix with multiple 0s, and there exists at least one 0, which has a smaller row index and a larger column index compared to one of the other 0s. Moreover, as shown in equation (3.1), this upstairs zero structure is one of the main factors of producing many crossings in a topology. Therefore, this section aims to relocate the master-ports of the entire communication matrix to reduce the crossings, which the upstairs zero structure causes.

This is the concept of eliminating the upstairs zero structure: I browse every row elements from the first row to the n_s-th row of the communication matrix. If there exists a 0 at position (o, p), and it fulfills the condition of the upstairs zero structure with a 0 at position (i, j), for $1 \leq o,\ i \leq n_s$, and $1 \leq p,\ j \leq n_m$, then I simply switch the o-th column and the p-th column of the communication matrix, to relocate the 0 at position (o, p) to the position (o, o). Hence this 0 locates in the diagonal of the communication matrix.

The implementation of eliminating the upstairs zero structure is shown in listing 3.1. The

Algorithm 1: Algorithm of relocating the input communication matrix

Input : divided Matrix, optimizability of cells, efficiency of cells, input matrix $\mathcal{A}$

Output : relocated Matrix $\mathcal{R}$

```
Parameters:
$\mathcal{E}$: edge matrix, (i.e. cell $C_{n,n}$ of $\mathcal{A}$);
$\mathcal{U}$: upper matrix, (i.e. cell $C_{n-1,n}$ of $\mathcal{A}$);
$\mathcal{L}$: left matrix, (i.e. cell $C_{n,n-1}$ of $\mathcal{A}$);
$\mathcal{D}$: diagonal matrix except for edge matrix, (i.e. all cell in the diagonal of $\mathcal{A}$ except for edge
  matrix);
$\mathcal{N}$: number of division in one dimension;
$\mathcal{O}_n$: group of cannot be optimized cells;
$\mathcal{G}$: group of efficient cells;
$\mathcal{V}_p$=read_given_information($\mathcal{A}$);
$\mathcal{P}$=switch_ports($\mathcal{A}$);
$\mathcal{C}$=switch_cells($\mathcal{A}$);
if $\mathcal{E} \notin \mathcal{O}_n$ then
    if $\mathcal{E} \in \mathcal{G}$ then
        $\mathcal{R}$=$\mathcal{A}$;
        return $\mathcal{R}$;
    else
        $\mathcal{R}$=$\mathcal{P}(\mathcal{A})$;
        return $\mathcal{R}$;
    end
else
    if $\mathcal{D} \in \mathcal{G}$ then
        $\mathcal{A}$=$\mathcal{C}(\mathcal{D}, \mathcal{E})$;
        $\mathcal{R}$=$\mathcal{A}$;
        return $\mathcal{R}$;
    else
        if $\mathcal{D} \notin \mathcal{O}_n$ then
            $\mathcal{A}$=$\mathcal{C}(\mathcal{D},\mathcal{E})$;
            $\mathcal{A}$=$\mathcal{P}(\mathcal{A})$;
            $\mathcal{R}$=$\mathcal{A}$;
            return $\mathcal{R}$
        else
            if $\mathcal{L} \in \mathcal{G}$ then
                $\mathcal{A}$=$\mathcal{C}(\mathcal{A})$;
                $\mathcal{R}$=$\mathcal{A}$;
                return $\mathcal{R}$;
            else if $\mathcal{L} \notin \mathcal{O}_n$ AND $\mathcal{L} \notin \mathcal{G}$ then
                $\mathcal{A}$=$\mathcal{C}(\mathcal{L})$;
                $\mathcal{A}$=$\mathcal{P}(\mathcal{A})$;
                $\mathcal{R}$=$\mathcal{A}$;
                return $\mathcal{R}$;
            else if $\mathcal{U} \in \mathcal{G}$ then
                $\mathcal{A}$=$\mathcal{C}(\mathcal{A})$;
                $\mathcal{R}$=$\mathcal{A}$;
                return $\mathcal{R}$;
            else if $\mathcal{U} \notin \mathcal{O}_n$ AND $\mathcal{U} \notin \mathcal{G}$ then
                $\mathcal{A}$=$\mathcal{C}(\mathcal{L})$;
                $\mathcal{A}$=$\mathcal{P}(\mathcal{A})$;
                $\mathcal{R}$=$\mathcal{A}$;
                return $\mathcal{R}$;
        end
    end
end
```

algorithm go through every 0 in the input matrix A and compare them pairwise. If 0 at position $(o, indexZero)$ fulfills the condition of upstairs zero structure, the algorithm switches master-ports $m_{indexZero}$ and m_o. Then this 0 is relocated into the diagonal of the input matrix A. After this algorithm runs for every element of the upstairs zero structure, those 0s are all relocated into the diagonal of the input matrix A. Therefore, the upstairs zero structure is eliminated, and it is substituted with the downstairs zero structure. In this way, the total number of on-chip crossings in the logical topology of the input matrix A is efficiently reduced.

Listing 3.1: Elimination of the upstairs zero structure

```
    %A: input matrix A
    %rows: number of slave-ports of communication matrix;
    %columns: number of master-ports of communication matrix;
    %zeroOnRow: The row of communication matrix, which includes 0

for o=1:rows
    for i=1:rows
        for p=1:columns
            for j=1:columns
                if isequal(A(o,p),0) && isequal(A(i,j),0)
                    if (o<i && p>j) || (o>i && p<j)
                        if ismember(0,A([o],:))
                            zeroOnRow=A([o],:);
                            indexZero=find(zeroOnRow==0);
                            A(:,[indexZero o])=A(:,[o indexZero]);
                        end
                    end
                end
            end
        end
    end
end
```

3.4. Count off-chip crossings number of the optimized matrix

After optimizing the input matrix by reassigning its ports, off-chip crossings can be generated. This section uses two perspectives to analyze and calculate the number of generated off-chip crossings. Section 3.4.1 proposes a method to calculate the number of off-chip crossings based on a layout of a determined off-chip connection. Section 3.4.2 represents the change in the number of off-chip crossings during the process of port reassignment.

3.4.1. Stable calculation of off-chip crossings number

Section 3.3 has demonstrated that the optimization of an input matrix is in principle switching the ports of the input matrix. As illustrated in Figure 2.10, the number of off-chip crossings has increased, resulting from the ports switch. However, the number of off-chip crossings can be calculated with this equation:

• $\#off-chip\ crossings$	Number of off-chip crossings
• $\#e_1$	Number of elements in group 1
• $\#e_2$	Number of elements in group 2
• $\#o$	Offset of group 1

$$\#off-chip\ crossings = (\#e_1 + \#e_2) \times \#o + (\#e_2 - 1) \times \#e_1 - \#e_2 \tag{3.2}$$

I use two examples to illustrate this equation. In Figure 3.6, ports p_a are marked in blue, ports p_b are in red, ports p_c are in orange, ports p_d are in teal, ports p_e are in purple, ports p_f are in gray, and ports p_g are in pink.

I first look at Figure 3.6 (a). There are 16 off-chip crossings in this layout. To verify equation (3.2) I proposed, I take the following steps:

(1) I use slave-ports as reference ports and initial status, and index the location of the slave-ports s_a to s_g from 1 to 7.

(2) As shown in the figure, the locations of master-ports are different from the initial status. The locations of the master-ports m_a to m_g are 6, 7, 4, 5, 1, 2, and 3.

After analyzing the current master-ports location, I assume that the sequence of changes in location of the master-ports from the initial status is in this way:

(3) I consider port p_a groups with port p_b, I call this group G_α, with $\{p_a,\ p_b\}$. Ports p_e, p_f, and p_g are in another group, called group G_β, with $\{p_e,\ p_f, p_g\}$.

(4) Now I observe G_α and G_β as two new elements, and I re-index the ports. For the slave-ports, the location of group G_α is 1, slave-port s_c is in location 2, slave-port s_d is in location 3, and group G_β is in location 4.

(5) Considering the master-ports use the slave-ports as reference ports, the location of the elements of master-ports is: group G_β is in location 1, master-port G_α is in location 2, master-port m_d is in location 3, and group G_α is in location 4.

(6) If I now compare the location indices of master- and slave-ports, it can be seen that only the location of master-ports groups G_α and G_β is switched, and the rest of the location remains.

Now I combine the analysis and the equation (3.2):

(7) In group G_α there are two elements: p_a and p_b. Therefore, $\#e_1$ in the equation is equal to 2. In group G_β there are three elements: p_e, p_f, and p_g. Therefore, $\#e_2$ in the equation is equal to 3.

(8) If I compare the location of G_α in master- and slave-ports, it can be seen that in the master-ports, G_α has changed its location from 1 to 4. Therefore, $\#o$ in the equation is equal to 3.

Now equation (3.2) can be calculated as:

$$(2+3)\times 3 + (3-1)\times 2 - 3 = 16 \tag{3.3}$$

The result coincides with the number of off-chip crossings in the layout.

Now I look at Figure 3.6 (b). There are five off-chip crossings in total. I use the slave-ports as reference ports. In this case, the location of master-ports is the same as in (a). However, the location of slave-ports has changed: from top to bottom, the location of slave-ports s_a to s_g has now change into 7, 5, 6, 4, 2, 3, and 1. However, in this case, the ports switch is not as straightforward as in the last case, and I need to decompose the master-ports switch based on their slave-ports as follows:

(1) I now compare master- and slave-ports in (b): At beginning, Group of master-ports m_e and m_f switches its location with master-port m_g compared to slave-ports.

(2) Meanwhile, master-port m_c switches its location with group of master-ports m_d and m_b.

(3) After step (2), the master-port m_a switches its location with master-port m_b.

The total number of off-chip crossings is the sum of the number of off-chip crossings, which is caused by these three steps of the ports switch, individually:

$$(2+1)\times 1+(1-1)\times 2-1+(2+1)\times 1+(1-1)\times 2-1+(1+1)\times 1+(1-1)\times 1-1=5 \quad (3.4)$$

Using this equation to calculate the number of off-chip crossings has its difficulties and drawbacks:

- This calculation is based on assumption. I assume the process of the port reassignment based on the final off-chip layout. However, the assumption is sometimes different from the real port reassignment. If I use the real port reassignment to calculate the number of off-chip crossings, it can easily lead to controversy with this equation.

- Because this equation is based on assumption, it is challenging to implement this assumptive calculation in code.

For these reasons, I propose an improved method to calculate the number of off-chip crossings in next section.

3.4.2. Dynamic calculation of off-chip crossings number

Because of the difficulties and drawbacks of using the stable calculation for calculating the number of off-chip crossings, there is an urge to solve these problems. The improved dynamic

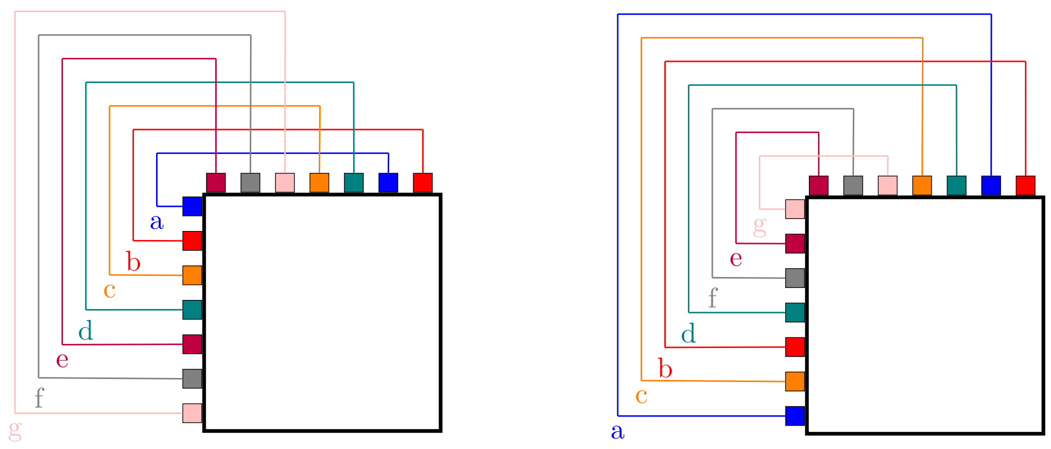


Figure 3.6.: Two examples that illustrate the calculation of the number of off-chip crossings.

calculation has achieved the following three goals:

- The calculation acts dynamically and reflects every step of the ports switch.
- The calculation is generalized and can be used in every critical and typical case.
- The calculation is implementable in code.

To propose this dynamic calculation, I re-analyze how off-chip crossings are generated. Then I implement the dynamic calculation of the number of off-chip crossings in code. Finally, I propose an algorithm to calculate the total number of off-chip crossings of an optimized input communication matrix based on the optimization algorithm.

Logic of off-chip crossings generation

To understand the logic of generating off-chip crossings, I compare the location of ports in Figure 3.6 (a) and focus on each off-chip connect master-slave pair individually. I continue using slave-ports as reference ports. If I look at ports p_a, it is clear that the connector of ports p_a crosses with the connectors of ports p_e, p_f, p_g, p_c, and p_d. Before the master-ports switch, the location of master-port m_a is the same as the location of slave-port s_a, which is location 1. However, after the port reassignment, the master-ports m_e, m_f, m_g, m_c, and m_d, which were initially located on the right side of the master-port m_a, are now located on the left side of the master-port m_a. In order to connect the master-port m_a and the slave-port s_a, the connector of ports p_a needs to travel through all the connectors of the ports, which now has moved from the right side of master-port m_a to its left side.

Therefore, by focusing on the ports p_a in the current layout, I can group the ports in the following way: *Group* G_γ: the master-ports locating on the right side of master-port m_a. *Group* G_δ: the slave-ports locating below the slave-port s_a. G_γ only contains master-port m_b. G_δ contains slave-ports s_b, s_c, s_d, s_e, s_f, and s_g. The intersection of these two groups is p_b, and it means that only master-port m_b remains on the right side of the master-port m_a after the master-ports switch. Therefore, there is no off-chip crossing between connectors of ports p_a and p_b. The off-chip crossings, which the reassignment of master-port m_a causes, are equal to the number of elements in Group G_δ minus the number of elements in the intersection.

Implementation of calculating off-chip crossings number

Now the cause of the off-chip crossings is discussed. The implementation of the dynamic calculation of the number of off-chip crossings is shown in listing 3.2. In this implementation, the algorithm uses master-ports as reference ports. As shown in the code, the algorithm goes through n_m-1 master-ports and n_s slave-ports and compare their locations pairwise until the algorithm finds the master- and slave-ports with the same index. Then the algorithm puts all the elements on the right side of this master-port into group rightOfMasterElement, and all the elements below this slave-port into another group belowOfSlaveElement, and find the intersection of these two groups. The number of off-chip crossings of each port is equal to the number of elements in group rightOfMasterElement minus the number of elements in the intersection group.

Listing 3.2: Implementation of dynamic calculation to count the number of off-chip crossings

```

% offChipCrossing: number of off-chip crossing;
% masterPortsIndex: Indices of master ports;
% slavePortsIndex: number of master-ports of communication matrix;
% rightOfMasterElement: group of elements, which are on the right of a master port;
% belowOfSlaveElement: group of elements, which are below of a slave port;
% intersection: group of elements, which are intersection of rightOfMasterElement and
    belowOfSlaveElement;


```

```
offChipCrossing=0
for i=1:length(masterPortsIndex)-1
    for j=1:length(slavePortsIndex)
        if isequal(masterPortsIndex(i),slavePortsIndex(j))
            rightOfMasterElement=masterPortsIndex([i+1:length(
                masterPortsIndex)]);
            belowOfSlaveElement=slavePortsIndex([j+1:length(
                masterPortsIndex)]);
            intersection=intersect(rightOfMasterElement,
                belowOfSlaveElement);
            if isempty(intersection) || ~isequal(intersection,
                rightOfMasterElement)
                offChipCrossing=offChipCrossing+length(
                    rightOfMasterElement)-length(intersection);
            end
        end
    end
end
```

Suppose I use this implementation to calculate the number of off-chip crossings of Figure 3.6 (b). Table 3.1 illustrates the calculation process, and the terms are explained as follows:

P_m: location of master-ports.

M: notation of master-ports.

G_1: group of master-ports, which are located on the right side of this master-port.

P_s: location of slave-ports.

S: notation of slave-ports.

G_2: group of slave-ports, which are located below this slave-port.

I: intersected ports of G_1 and G_2.

Table 3.1.: Results of using dynamic calculation to count the number of off-chip crossings.

L_m	M	G_1	L_s	S	G_2	I	D
1	m_e	m_f, m_g, m_c, m_d, m_a, m_b	2	s_e	s_f, s_d, s_b, s_c, s_a	p_f, p_c, p_d, p_a, m_b	1
2	m_f	m_g, m_c, m_d, m_a, m_b	3	s_f	s_d, s_b, s_c, s_a	p_c, p_d, p_a, p_b	1
3	m_g	m_c, m_d, m_a, m_b	1	s_g	s_e, s_f, s_d, s_b, s_c, s_a	p_c, p_d, p_a, p_b	0
4	m_c	m_d, m_a, m_b	6	s_c	s_a	p_a	2
5	m_d	m_a, m_b	4	s_d	s_b, s_c, s_a	p_a, p_b	0
6	m_a	m_b	7	s_a			1
7	m_b		5	s_b	s_c, s_a		0
C_{off}							5

D**:** difference between G_1 and I.

However, the dynamic calculation comes to the same result as illustrated in Figure 3.6 (b).

Algorithm of calculating total number of off-chip crossings of an optimized input matrix

In section 3.3, it was shown that there are two steps for optimizing an input matrix: the relocation of the input matrix and the elimination of the upstairs zero structure. The relocation step can switch not only the slave-ports but also the master-ports. In comparison to the relocation step, the elimination step only switches the master-ports. Therefore, if there only occurs the elimination step, the off-chip crossings which the master-ports switch in elimination step produce cannot be compensated. However, if there occurs a slave-ports switch in the relocation step, and this switch can compensate the master-ports switch in the elimination slave, some off-chip crossings can be compensated. Therefore, it is worth discussing, if the slave-ports switch occurs during the relocation step.

Algorithm 2 provides a method to calculate the total number of off-chip crossings after the optimization step, in order to clarify how the ports switch in the relocation step and elimination step changes the total number of off-chip crossings:

- as shown from line 8 to line 10, if the original slave-ports indices are equal to the relocated slave-ports indices, no slave-ports switch of the input matrix is performed (master-ports switch could be performed). In this case, the total number of off-chip crossings is equal to the number of off-chip crossings resulting from the master-ports switch in the elimination step.

- Line 11 to line 13 demonstrates the end result of the off-chip connected master- and slave-ports switch in the relocation step. In this case, the number of off-chip crossings resulting from the slave-ports switch is compensated by the number of off-chip crossings resulting from the master-ports switch. Hence the total number of off-chip crossings is equal to the number of off-chip crossings produced in the elimination step.

- If the relocated slave-ports indices are not equal to the relocated master-ports indices, and the relocated master-ports indices are not equal to the eliminated master-ports indices, it shows that the relocation of the master- and slave-ports does not happen to the off-chip connected master- and slave-ports. Moreover, there also occurs an elimination step. Therefore, the algorithm needs to count the number of off-chip crossing, which is caused by the master- and slave-ports relocation, then add this value with the number of off-chip crossings after the elimination step, and this sum is equal to the total number of off-chip crossings, as shown from line 14 to line 17.

- For the rest of the cases, the total number of off-chip crossings is equal to the sum of number of off-chip crossings which is generated in the relocation step, and the number of off-chip crossings which is generated in the elimination step, as shown from line 18 to line 21.

Algorithm 2: Counting the total number of off-chip crossings

Input : master-ports indices, slave-ports indices, off chip crossing number after relocation $\mathcal{C}_r$, off chip crossing number after elimination $\mathcal{C}_e$

Output : total off-chip crossing number $\mathcal{C}_t$

```
Parameters:
$\mathcal{I}_{m,o}$: original master-ports indices;
$\mathcal{I}_{m,r}$: master-ports indices after relocation;
$\mathcal{I}_{m,e}$: master-ports indices after elimination;
$\mathcal{I}_{s,o}$: original slave-ports indices;
$\mathcal{I}_{s,r}$: slave-ports indices after relocation;
$\mathcal{I}_{s,e}$: slave-ports indices after elimination;
if $\mathcal{I}_{s,o}$=$\mathcal{I}_{s,r}$ then
    $\mathcal{C}_t$=$\mathcal{C}_e$;
    return $\mathcal{C}_t$;
else if $\mathcal{I}_{s,r}$=$\mathcal{I}_{m,r}$ then
    $\mathcal{C}_t$=$\mathcal{C}_e$;
    return $\mathcal{C}_t$;
else if $\mathcal{I}_{s,r} \neq I_{m,r}$ AND $\mathcal{I}_{m,r} \neq I_{m,e}$ then
    $\mathcal{C}_r$ = count_off_chip($\mathcal{I}_{m,r}$,$\mathcal{I}_{s,r}$);
    $\mathcal{C}_t$=$\mathcal{C}_r$+$\mathcal{C}_e$;
    return $\mathcal{C}_t$;
else
    $\mathcal{C}_t$=$\mathcal{C}_r$+$\mathcal{C}_e$;
    return $\mathcal{C}_t$;
end
```

4. Experimental Results

This chapter shows experimental results that are generated by applying the proposed port reassignment algorithm on some WRONoC designs. The proposed methodology was implemented in MATLAB. Experiments were performed using a computer with a 2.9 GHz CPU and 8 GB of RAM. The test cases are generated under the constraints on the default path in section 2.4.2 with different communication matrices, logical topologies and off-chip layouts. I tested the algorithm on nine test cases:

- Cases 1, 2, 3: small-sized cases with 4×4 communication matrices.
- Cases 4, 5, 6: medium-sized cases with 6×6 communication matrices.
- Cases 7, 8, 9: large-sized cases with 8×8 communication matrices.

4.1. Execution of the algorithm on test case 1

I use test case 1 to illustrate the results of each step. The original logical topology and off-chip layout of test case 1 are illustrated in Figure 4.1.

After the algorithm takes the communication matrix of test case 1 as input, this matrix is divided into four cells. The test of the optimizability of the cells is executed at first, and their optimizability is as follows:

$C_{1,1} = \begin{pmatrix} * & * \\ * & NA \end{pmatrix}$ does not need to be optimized.

$C_{1,2} = \begin{pmatrix} 0 & * \\ * & 0 \end{pmatrix}$ cannot be optimized.

$C_{2,1} = \begin{pmatrix} NA & * \\ * & 0 \end{pmatrix}$ does not need to be optimized.

$C_{2,2} = \begin{pmatrix} 0 & * \\ * & NA \end{pmatrix}$ can be optimized by exchanging its master- and slave-ports.

After analyzing the optimizability of the cells, the algorithm checks the efficiency of each cell, which is:

$C_{1,1} = \begin{pmatrix} * & * \\ * & NA \end{pmatrix}$ is an efficient cell.

$C_{1,2} = \begin{pmatrix} 0 & * \\ * & 0 \end{pmatrix}$ is a middle efficient cell.

$C_{2,1} = \begin{pmatrix} * & 0 \\ 0 & NA \end{pmatrix}$ is an efficient cell.

$C_{2,2} = \begin{pmatrix} NA & * \\ * & * \end{pmatrix}$ is an inefficient cell.

According to the optimizability and efficiency of the cells of test case 1, the algorithm then starts to relocate the cells inside the communication matrix. The relocated communication matrix, its corresponding logical topology, and its off-chip layout are shown in Figure 4.2. Considering there is still an upstairs zero structure in the relocated communication matrix of test case 1, the algorithm continues operating on this matrix to eliminate the upstairs zero structure. The result of this step is shown in Figure 4.3.

In Figure 4.1, there are 12 crossings in total, including 12 on-chip crossings and 0 off-chip crossings. However, in Figure 4.3, the number of on-chip crossings is reduced to 6, and the increase in the number of off-chip crossings is 5, and there are 11 crossings in total.

The algorithm takes 0.169 seconds to complete the calculation of the number of on-chip crossings, the optimization of the input matrix, and the calculation of the number of off-chip crossings.

4.2. Comparison and analysis

In Table 4.1, results are shown for the communication matrices in appendix A respectively. The meaning of the columns in Table 4.1 is described as follows:

$$\begin{bmatrix} * & * & 0 & * \\ * & NA & * & 0 \\ * & 0 & NA & * \\ 0 & NA & * & * \end{bmatrix}$$

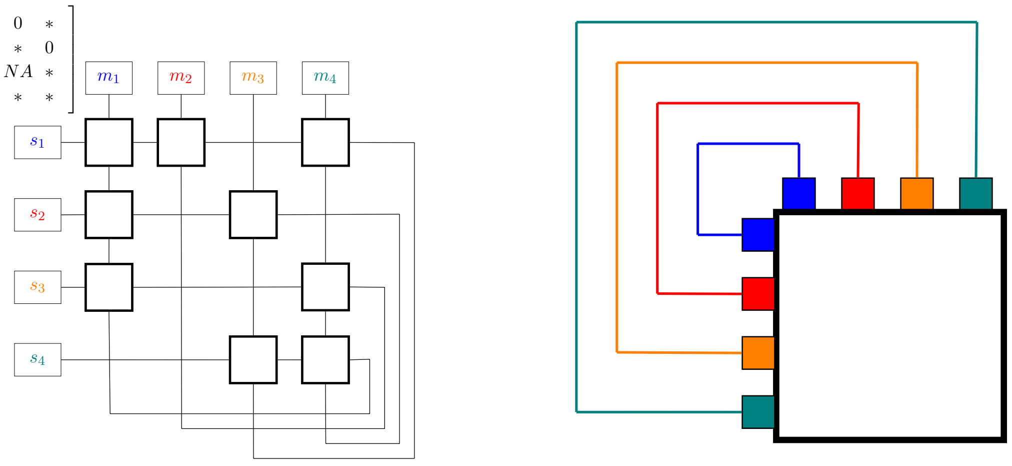


Figure 4.1.: The original communication matrix of test case 1, its logical topology and its off-chip layout.

$$\begin{bmatrix} * & * & * & 0 \\ * & NA & 0 & * \\ 0 & NA & * & * \\ * & 0 & * & NA \end{bmatrix}$$

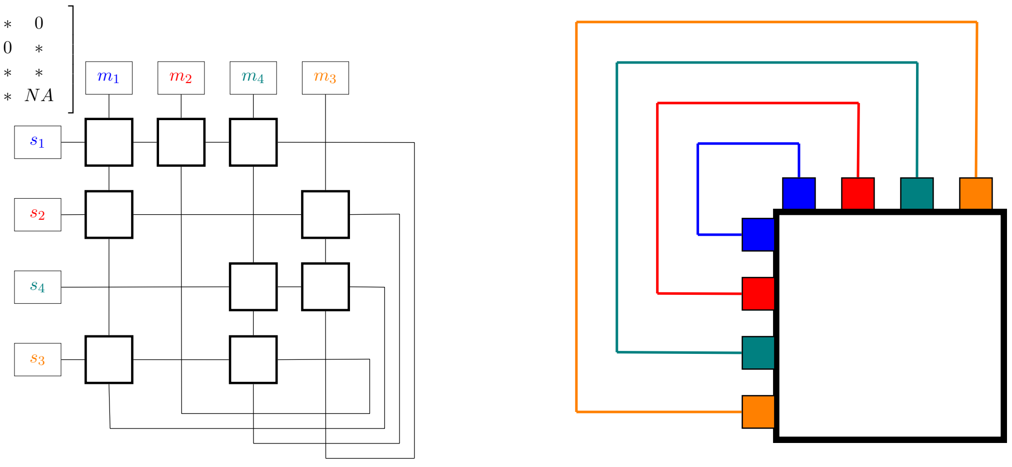


Figure 4.2.: The communication matrix of test case 1, its logical topology and its off-chip layout after relocating the cells.

$$\begin{bmatrix} 0 & * & * & * \\ * & 0 & * & NA \\ * & * & 0 & NA \\ NA & * & * & 0 \end{bmatrix}$$

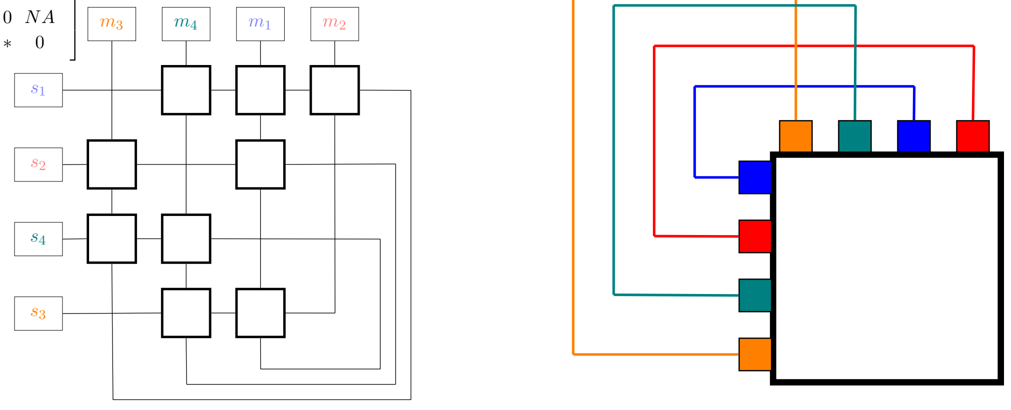


Figure 4.3.: The communication matrix of test case 1, its logical topology and its off-chip layout after eliminating the upstairs zero structure.

Table 4.1.: The results of the optimization algorithm.

	$C_{original,on}$	$C_{optimized,on}$	C_{off}	C_t	R_{on}	R_{total}	T
Test case 1	12	6	5	11	50.00%	8.33%	0.169
Test case 2	7	3	0	3	57.14%	57.14%	0.031
Test case 3	12	8	2	10	33.33%	16.67%	0.078
Test case 4	16	10	7	17	37.50%	-6.25%	0.044
Test case 5	15	11	3	14	26.67%	6.67%	0.127
Test case 6	15	10	5	10	33.33%	0%	0.014
Test case 7	34	25	11	36	26.4%	-5.88%	0.044
Test case 8	60	27	26	53	55.00%	11.67%	0.049
Test case 9	17	16	5	21	5.88%	-23.52%	0.047

$C_{original,on}$: on-chip crossings in the original logical topology.

$C_{optimized,on}$: on-chip crossings in the logical topology after port reassignment.

C_{off}: increased off-chip crossings after the port reassignment.

C_t: total crossing number after the port reassignment.

R_{on}: rate of the reduction of the number of on-chip crossings.

R_{total}: rate of the reduction of the total crossing number.

T: program runtime.

As shown in the experimental results, the algorithm achieves these goals:

1. This algorithm can successfully reduce the number of on-chip crossings in a logical topology through the port reassignment. In the test cases, the rate of on-chip crossings reduction is between 5.88% and 57.14%. The average on-chip crossings reduction rate is 36.14%. However, this algorithm performs a useful functionality in reducing the number of on-chip crossings in a logical topology.

2. The algorithm of calculating the number of on- and off-chip crossings is applicable in various cases. The algorithm of calculating the number of on-chip crossings can calculate not only the number of original on-chip crossings but also the number of on-chip crossings after port reassignment. The algorithm for calculating the number of off-chip crossings can represent the ports switch dynamically.

3. Table 4.1 shows the rate of the total crossing reduction is between -23.52% and 57.14% in the test cases. Furthermore, the algorithm for reducing total crossing number performs positive function in 5 test cases, neutral function in 1 test case, and negative function in 3 test cases. The average total crossing reduction rate is 7.20%. However, this algorithm performs an overall positive function in reducing the total number of crossings of an input communication matrix.

4. The runtime of this algorithm is between 0.031 seconds and 0.169 seconds, and the average runtime is 0.067. Therefore, this algorithm is relatively feasible and efficient.

I suggest future works in these three fields:

1. This algorithm perceives the default path as 0, no-communicative pair as -1, ADF usage as 1. However, this algorithm does not consider the different wavelengths usage of ADFs. The different wavelength usage of ADFs can sometimes redefine the location of the components in a logical topology. The future works can take the wavelengths of the ADFs into consideration.

2. Before performing this algorithm, I assumed that the master-ports are located on the north of the communication matrix and the slave-ports on the west. If the location of ports is too specific, the performance of this algorithm is also limited. In future works, the location of the ports can be more generalized than in my assumption. (e.g. Master-ports on the north, slave ports on the south.)

3. This algorithm only takes $2 \cdot n \times 2 \cdot n$ communication matrix as the input matrix. However, even though the matrices with full-connectivity are symmetric, there are still cases that the number of master-ports of these communication matrices does not equal to the multiples of 2. I suggest that the future works do not constrain the size of the input matrix, to perform in a wider range of communication matrices.

5. Conclusion

In this algorithm, a method to calculate the number of on-chip crossings is first presented. Based on this calculation and the communication matrix, an optimization algorithm is introduced to reassign the router ports of the communication matrix to reduce the number of on-chip crossings in the logical topology. Finally, a dynamic method is formulated to calculate the number of off-chip crossings generated by the optimization algorithm.

For future work, the algorithm can take different wavelengths of different master-slave pairs into consideration to analyze every individual communication. The location of ports can be refined to be more general so that this algorithm can handle a wider range of port reassignment. The algorithm can be improved by taking communication matrices with different sizes as input so that every possible communication matrix can be optimized by using this algorithm.

A. Test cases

The matrices, which I used to test the proposed algorithm, are the following:

$$\text{Test case 1}=\begin{array}{c|cccc} & m_1 & m_2 & m_3 & m_4 \\ \hline s_1 & \ast & \ast & 0 & \ast \\ s_2 & \ast & NA & \ast & 0 \\ s_3 & \ast & 0 & NA & \ast \\ s_4 & 0 & NA & \ast & \ast \end{array}$$

$$\text{Test case 2}=\begin{array}{c|cccc} & m_1 & m_2 & m_3 & m_4 \\ \hline s_1 & \ast & \ast & \ast & NA \\ s_2 & \ast & 0 & \ast & NA \\ s_3 & NA & \ast & 0 & \ast \\ s_4 & 0 & \ast & \ast & NA \end{array}$$

$$\text{Test case 3}=\begin{array}{c|cccc} & m_1 & m_2 & m_3 & m_4 \\ \hline s_1 & \ast & NA & NA & 0 \\ s_2 & 0 & \ast & NA & \ast \\ s_3 & NA & \ast & 0 & \ast \\ s_4 & \ast & 0 & \ast & NA \end{array}$$

$$\text{Test case 4}=\begin{array}{c|cccccc} & m_1 & m_2 & m_3 & m_4 & m_5 & m_6 \\ \hline s_1 & \ast & \ast & \ast & \ast & NA & \ast \\ s_2 & \ast & \ast & 0 & \ast & NA & \ast \\ s_3 & \ast & \ast & NA & NA & \ast & 0 \\ s_4 & NA & \ast & \ast & NA & \ast & \ast \\ s_5 & \ast & \ast & \ast & NA & 0 & \ast \\ s_6 & 0 & \ast & \ast & \ast & NA & \ast \end{array}$$

$$\text{Test case 5}=\begin{array}{c|cccccc} & m_1 & m_2 & m_3 & m_4 & m_5 & m_6 \\ \hline s_1 & \ast & NA & 0 & \ast & NA & \ast \\ s_2 & \ast & \ast & NA & \ast & \ast & \ast \\ s_3 & \ast & \ast & \ast & NA & \ast & 0 \\ s_4 & NA & 0 & \ast & \ast & NA & NA \\ s_5 & \ast & \ast & \ast & \ast & 0 & \ast \\ s_6 & \ast & \ast & NA & NA & \ast & NA \end{array}$$

$$\text{Test case 6=}\begin{array}{c|cccccc} & m_1 & m_2 & m_3 & m_4 & m_5 & m_6 \\ \hline s_1 & \ast & NA & 0 & \ast & \ast & \ast \\ s_2 & 0 & \ast & NA & \ast & \ast & NA \\ s_3 & \ast & \ast & \ast & NA & 0 & \ast \\ s_4 & \ast & \ast & NA & \ast & \ast & NA \\ s_5 & \ast & \ast & \ast & 0 & \ast & \ast \\ s_6 & \ast & 0 & \ast & \ast & \ast & NA \end{array}$$

$$\text{Test case 7=}\begin{array}{c|cccccccc} & m_1 & m_2 & m_3 & m_4 & m_5 & m_6 & m_7 & m_8 \\ \hline s_1 & \ast & 0 & \ast & \ast & NA & \ast & \ast & \ast \\ s_2 & 0 & \ast & \ast & \ast & NA & NA & NA & NA \\ s_3 & \ast & NA & \ast & \ast & NA & \ast & \ast & 0 \\ s_4 & \ast & \ast & NA & 0 & \ast & \ast & \ast & \ast \\ s_5 & NA & \ast & \ast & NA & \ast & \ast & 0 & \ast \\ s_6 & NA & \ast & \ast & \ast & 0 & \ast & NA & \ast \\ s_7 & NA & \ast & NA & NA & \ast & \ast & \ast & \ast \\ s_8 & \ast & \ast & 0 & NA & NA & \ast & NA & \ast \end{array}$$

$$\text{Test case 8=}\begin{array}{c|cccccccc} & m_1 & m_2 & m_3 & m_4 & m_5 & m_6 & m_7 & m_8 \\ \hline s_1 & \ast & \ast & \ast & \ast & NA & \ast & NA & 0 \\ s_2 & \ast & \ast & \ast & \ast & NA & NA & 0 & \ast \\ s_3 & NA & \ast & NA & \ast & \ast & 0 & NA & NA \\ s_4 & \ast & \ast & \ast & \ast & 0 & \ast & \ast & NA \\ s_5 & \ast & NA & \ast & 0 & \ast & \ast & NA & \ast \\ s_6 & \ast & NA & 0 & \ast & NA & \ast & \ast & \ast \\ s_7 & NA & 0 & \ast & NA & \ast & NA & \ast & NA \\ s_8 & 0 & \ast & NA & NA & NA & \ast & \ast & \ast \end{array}$$

$$\text{Test case 9=}\begin{array}{c|cccccccc} & m_1 & m_2 & m_3 & m_4 & m_5 & m_6 & m_7 & m_8 \\ \hline s_1 & NA & \ast & \ast & \ast & \ast & NA & \ast & \ast \\ s_2 & \ast & \ast & 0 & NA & NA & \ast & \ast & \ast \\ s_3 & NA & \ast & \ast & \ast & \ast & \ast & \ast & NA \\ s_4 & \ast & NA & \ast & \ast & \ast & NA & 0 & \ast \\ s_5 & \ast & \ast & NA & \ast & \ast & \ast & \ast & NA \\ s_6 & \ast & \ast & \ast & NA & 0 & \ast & NA & \ast \\ s_7 & \ast & \ast & NA & \ast & \ast & \ast & \ast & \ast \\ s_8 & \ast & NA & \ast & \ast & NA & \ast & NA & 0 \end{array}$$

Bibliography

Briere, M., Girodias, B., Bouchebaba, Y., Nicolescu, G., Mieyeville, F., Gaffiot, F. & O'Connor, I. (2007): System level assessment of an optical noc in an mpsoc platform, 2007 Design, Automation Test in Europe Conference Exhibition, S. 1–6.

Li, M., Tseng, T., Bertozzi, D., Tala, M. & Schlichtmann, U. (2018): Customtopo: A topology generation method for application-specific wavelength-routed optical nocs, 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), S. 1–8.

Nikdast, M., Xu, J., Duong, L. H. K., Wu, X., Wang, X., Wang, Z., Wang, Z., Yang, P., Ye, Y. & Hao, Q. (2015): Crosstalk noise in wdm-based optical networks-on-chip: A formal study and comparison, IEEE Transactions on Very Large Scale Integration (VLSI) Systems **23**(11): 2552–2565.

Peano, Andrea, Ramini, Luca, Gavanelli, Marco, Nonato, Maddalena & Bertozzi, Davide (2016): Design technology for fault-free and maximally-parallel wavelength-routed optical networks-on-chip, S. 1–8.

Ramini, L., Grani, P., Bartolini, S. & Bertozzi, D. (2013): Contrasting wavelength-routed optical noc topologies for power-efficient 3d-stacked multicore processors using physical-layer analysis, 2013 Design, Automation Test in Europe Conference Exhibition (DATE), S. 1589–1594.

Truppel, A., Tseng, T., Bertozzi, D., Alves, J. C. & Schlichtmann, U. (2020): Psion+: Combining logical topology and physical layout optimization for wavelength-routed onocs, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems S. 1–1.

Truppel, Alexandre, Tseng, Tsun-Ming, Bertozzi, Davide, Alves, José Carlos & Schlichtmann, Ulf (2019): Psion: Combining logical topology and physical layout optimization for wavelength-routed onocs, Proceedings of the 2019 International Symposium on Physical Design, ISPD '19, Association for Computing Machinery, New York, NY, USA, S. 49–56. `https://doi.org/10.1145/3299902.3309747`

Tseng, T. (2013): Routing of optical networks-on-chip.

Tseng, T., Truppel, A., Li, M., Nikdast, M. & Schlichtmann, U. (2019): Wavelength-routed optical nocs: Design and eda — state of the art and future directions: Invited paper, 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), S. 1–6.

Xiao, Shijun, Khan, Maroof H., Shen, Hao & Qi, Minghao (2007): Modeling and measurement of losses in silicon-on-insulator resonators and bends, Opt. Express **15**(17): 10553–10561. `http://www.opticsexpress.org/abstract.cfm?URI=oe-15-17-10553`