11-13-2021

Asynchronous, Distributed Optical Mutual Exclusion and Applications

Ahmed Bahaael Mansour
Louisiana State University and Agricultural and Mechanical College

ASYNCHRONOUS, DISTRIBUTED OPTICAL MUTUAL EXCLUSION AND APPLICATIONS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Devision of Electrical and Computer Engineering

by
Ahmed Bahaa Mansour
B.S.,Cairo University, 2000
M.S., Louisiana State University, 2004
December 2021

This thesis is dedicated to all Optical Interconnects researchers.

And now for a novel approach for
Optical Interconnects

Acknowledgments

I have a lot to declare for all those who supported me.

First and foremost, I would like to express my sincere gratitude to my main advisor, Dr. R. Vaidyanathan, for his invaluable support and for his contributions of time and ideas, for all the discussion and inspiration. He never saved his time or effort when I needed him.

I also have to thank the whole committee, Dr. Trahan, Dr. Wei, Dr. Veronis, and Dr. Cabral for being a part of my achievement.

Second, I cannot hide my love and gratitude for my supporting wife; for the wonderful environment she added for me to stay with my work.

Third, I always declare that I owe my whole life to my wonderful dad, my wonderful mom, and my great brother and sister.

Table of Contents

List of Tables

List of Figures

Abstract

Silicon photonics have drawn much recent interest in the setting of intra-chip and module communication. In this dissertation, we address a fundamental computational problem, mutual exclusion, in the setting of optical interconnects. As a main result, we propose an optical network and an algorithm for it to distribute a token (shared resource) mutually exclusively among a set of n processing elements. Following a request, the token is granted in constant amortized time and $O(n)$ worst case time; this assumes constant propagation time for light within the chip. Additionally, the distribution of tokens is fair, ensuring that no token request is denied more than $n - 1$ times in succession; this is the best possible. The proposed algorithm is distributed (nodes operate without any centralized control) and asynchronous (nodes are temporally independent and do not rely on a common clock).

Additionally, we extend this work to distribute multiple tokens, that can be identical or distinct. With some modification, we can also employ the mutual exclusion network for broadcasting or multicasting data between processing elements. This in turn allows for the implementation of global Boolean operations spanning the processing elements.

Further, we identify some applications of our work. These include an optical queue, an essential element of many systems with optical interconnects; the optical queue can be used to implement a fast data transfer network.

Chapter 1. Introduction

Optics is considered one of the most promising alternatives to electrical interconnects that currently have become the bottleneck in many systems [1] (including for example, datacenters [2]). Optical systems have been used for inter-chip [1, 3], and intra-chip communications [2, 4, 5]. Several architectures with optical interconnects have been proposed in the last decade [6–10]. Recent work on packageless processors also make optical interconnects more attractive [11]. One common optical arrangement used for intra-chip communication employs silicon waveguides and microring resonators [12, 13]. External lasers are used to inject light into the waveguide, often many wavelengths multiplexed, each of which can currently drive a 3–5 GHz that can collectively deliver up to 25 Gbps. Such systems with multiple channels can potentially deliver in the order of a terabyte per second bandwidth [2]. In this work, we will employ an optical fabric with such silicon waveguides and microring resonators.

Researchers and industry have traditionally employed optics primarily for data communications domain because of its huge bandwidth. However, in our work, we will employ optics primarily in the coordination domain.We employ traditional waveguides and microring resonators for evanescence based switches. We rely on the idea of conservation of energy; a novel approach to the "mutual exclusion" problem.

Mutual Exclusion is problem of fundamental importance in concurrent systems, including traditional systems, and particularly in those with optical communication fabrics [6, 8]. In this dissertation we propose as a main result an optical mutual exclusion architecture consisting of (a) a network based on silicon waveguides and microring resonators (or simply microrings) that connect a set of n processing elements (or simply PEs), and (b) a

distributed algorithm to resolve contention among the PEs for a shared resource (Mutual Exclusion). Our solution is asynchronous in the sense that we will not need a centralized clock for the PEs. The solution is also completely distributed in the sense that there is no centralized controller and each PE will have no information other than what is local. These factors make the system more scalable in that clock skew and information locality are not major factors in expanding the system, both in size (number of processing elements) and spatial (the chip area over which the system spans). Further, the proposed optical network itself is not a big departure from networks typically used for data transport in an optical interconnect [6]; consequently, the hardware and communication overheads for our approach are quite minimal.

Contributions of this Work

- We propose an optical architecture (network + algorithm) for distributing a token with mutual exclusion. The architecture has the following properties:

- The architecture is asynchronous in that no assumption is made about the timings of the request and release of an acquired token. There is no underlying clock synchronizing the operation of the network.

- The architecture is fast in that every token request on an n-node system is granted in $\Theta(n)$ time in the worst case and constant amortized time over a large number of requests and grants; this assumes constant time propagation of light in the environment (chip).

- The architecture is fair in that no PE receives the token more than once before every pending request has been satisfied at least once. This results in each PE waiting for no more than $n-1$ token cycles.

- The algorithm is distributed, in that each PE i acts solely on the basis of local information. There is no centralized controller.

- The proposed network uses an optical fabric that is very close to the standard network components used with optical interconnects. Thus the proposed network should be viewed as a "minimum specification of" rather than "in addition to" the available network. This, coupled with the fundamental nature of mutual exclusion in a parallel or distributed environment, makes this work of wide applicability.

We extend the above results in the following directions:

- we propose architectures (network and algorithm) to extend single token distribution to multiple tokens. We consider cases where the tokens can be identical or distinguishable. We explore trade-offs between network costs and algorithm speeds in these settings.

- We propose networks to broadcast data from one PE to all others and to multicast data within groups of PEs.

- The broadcast idea is used to design a network for global Boolean operation (that span PEs). Barrier Synchronization is an application of such an operation. The

architectures of Chapters 5 and 6 are special cases of a Boolean function, priority encoder.

- We use Mutual Exclusion to simulate an optical queue in a manner that is much easier to implement than with a purely optical approach. The optical queue facilitates fast data transfer between PEs connected by an optical interconnect.

- We cast the Mutual Exclusion algorithm as an efficient Token Network (in which the token moves only through PEs that seek the network).

Organization of the Dissertation: In the next chapter, we go through some preliminary ideas needed for the reader to better appreciate the rest of the work. The following Chapter 3 includes a literature review. Then in Chapter 4 we discuss some optical background and introduce two building blocks for the networks we propose. Chapters 5, 6, and 7 develop mutual exclusion architectures that progressively provide more features. In Chapter 5, we introduce the Base Architecture that serves as a fundamental building block and to introduce ideas in a simple setting that carry forward to subsequent chapters. The Base Architecture distributes the token mutually exclusively but in it a PE cannot hold on to its token as it can be preempted by another PE. This problem is corrected by the Non-Preemptive Architecture of Chapter 6. This architecture, however, is not fair; PEs may wait indefinitely for the token, repeatedly being beaten out by higher priority PEs. In Chapter 7 we introduce the Fair Architecture, a main result of this dissertation.

Chapter 8 is devoted to multiple tokens, and we describe applications in Chap-

ter 9 (Broadcast, multicast, Boolean operations, optical queue and token network). Finally, we summarize our results and include some concluding remarks in Chapter 10.

Chapter 2. Preliminaries

In this Chapter we describe some preliminary ideas, including those related to a distributed architecture and mutual exclusion.

2.1. A Distributed Architecture

In this work, we will view a distributed architecture as consisting of two components (a) a distributed communication network connecting a set of n processing elements, and (b) a distributed algorithm running on the architecture that solves the problem in question (primarily mutual exclusion here).

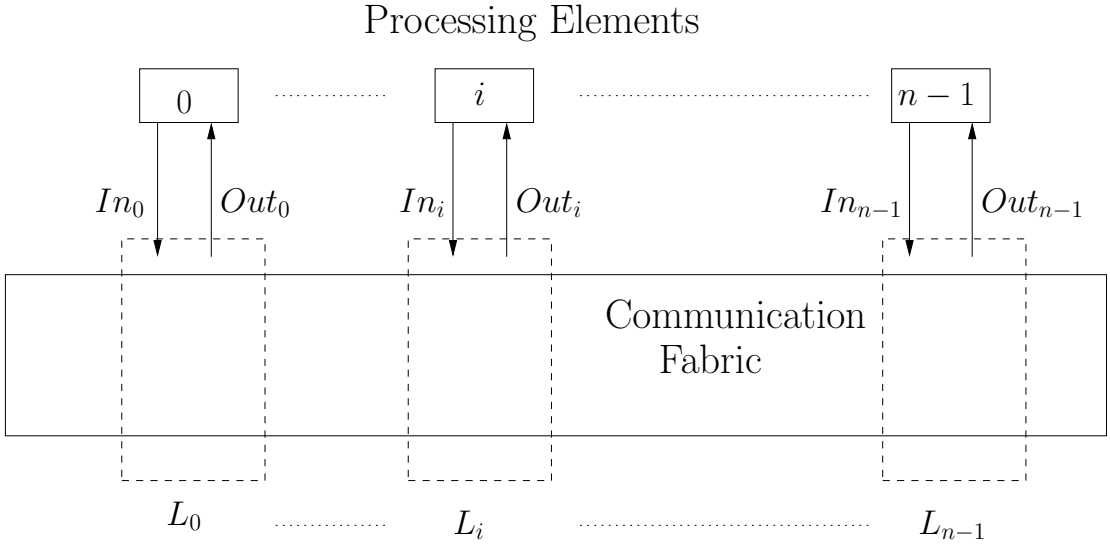

Figure 2.1. A general model of a distributed communication network.

Figure 2.1 shows an abstract view of a distributed network. Let $P = \{i \ : \ 0 \leq i < n\}$ be a set of processing elements (PEs) that communicate over a network through a set of local interfaces L_i, where $0 \leq i < n$. Each interface L_i has a set of signals In_i (resp., Out_i) that are input into (resp., output from) the communication fabric of PE i. For a conventional fabric, the local interface may be a set of input/output ports, and In_i, Out_i the signals for communication between PE i and the interface L_i. In the optical networks

in this dissertation, L_i consists of components such as waveguides, microrings and photodetectors, and Out_i, In_i are signals for controlling these components and those representing information received from these components. In both the conventional and optical networks, the communication fabric extends outside these local interfaces, for example, as wires and optical waveguides that span larger parts of the network.

2.2. Mutual Exclusion

Consider a set $P = \{i \ : \ 0 \leq i < n\}$ of n PEs, each of which has a flag R_i that indicates whether PE i is requesting the token (a shared resource). Mutual exclusion is method to assign to each PE i the value (0 or 1) of a flag G_i that indicates whether PE i has been granted the token. This assignment must be such that (a) if there is any PE with $R_i = 1$, then exactly one PE has $G_j = 1$ (i, j may be different), and if all PEs have $R_i = 0$ then all PEs have $G_i = 0$. We will relax this requirement in Chapter 7, where a PEi with $R_i = 0$ can still have $G_i = 1$. This is only for the purpose of the ease of Algorithmic expression, and does not alter the meaning of mutual exclusion definition here. In the setting of a distributed architecture, each PE i independently sets the value of R_i which is then input to the network as part of Out_i. On running the distributed algorithm, the network outputs information as part of In_i that PE i can use to update the value of G_i in a manner consistent with mutual exclusion.

Throughout this dissertation we will assume that a PE makes only one request at a time and that it does not rescind its request before it is granted the token. This property, termed *Request Persistence*, will be used later in the work. Thus every token grant corresponds to exactly one request and and each PE has at most one outstanding request

at any time.

We now define some performance measures for mutual exclusion.

System Response Time: This is also called synchronization delay. The system respose time is the time during which the system has at least one token request, but the token is yet to be granted. It represents the system idle time. This time depends, among other factors, on the times at which PEs request the token. We will use the worst system response time, and time amortized (over several requests and grants) in this dissertation. When liveness (see definition below) is guaranteed, the system response time is finite.

PE Response Time: The response time for PE i is the time from when PE i sets R_i to 1 to the time when the system allows it to set G_i to 1. It should be noted that a PE's request may never be granted (if the architecture does not guarantee fairness), in which case, the PE response time could be infinite. As in the system response time, we will express the PE response time as the worst case or as an amortized value over multiple request-grant pairs.

Since the system we propose operates asynchronously, its is convenient to describe the desired behaviour in time. For any time t, let $R_i(t)$ and $G_i(t)$ denote the values of flags R_i and G_i at time t.

Formally, the solution to mutual exclusion requires the following conditions to be satisfied:

Safety: For any t, there is at most one PE i with $G_i(t) = 1$; only one PE is granted the token at a time.

It is important to note that safety must be unconditionally satisfied (for example, regardless of how close to each other two requests may be, or how small the period of time for which $G_i = 1$).

Safe Assignment: For any time t and an arbitrarily small non-zero interval δ, let $t - \delta$ (resp., $t+\delta$) denote the time immediately before (resp., immediately after) t. If $G_i(t-\delta) = 0$ and $G_i(t + \delta) = 1$, then $R_i(t - \delta) = 1$; that is, the token can be granted only to a PE requesting it.

Note that this requirement is not always achieved by all approaches in the literature. As this requirement will be clearly maintained in all of the proposed solutions in this dissertation, we will not address it in any detail.

Non-Preemption: If $G_i(t) = 1$, then it remains 1 until PE i gives the token up by setting $R_i = 0$. That is, if PE i holds the token, then no other PE can cause PE i to give up the token.

Liveness: If there is at least one PE i such that $R_i(t) = 1$, then at some time $t' > t$, such that $t' - t$ is finite, there is a PE j such that $G_j(t') = 1$; that is the token is ultimately granted. This condition is often called *lock freedom*.

We go one step further and show that the system response time of an n-PE system is $\Theta(n)$ in the worst case, and constant, if amortized over many requests and grants; these results assume that the propagation time for the light in the computing environment (chip) is constant.

Fairness: If $R_i(t) = 1$, then at some $t'' > t$, such that $t'' - t$ is finite, $G_i(t'') = 1$; that is, every request is ultimately granted. This condition is often called *wait-freedom*. The term

we will use in this dissertation will be *starvation freedom*

We show that if a PE i requests the token, then no other PE j can be granted the token twice before PE i gets the token, resulting in the worst-case PE response time of $n-1$ cycles of token grants and releases. Each of these token grants is made in amortized constant time.

Request Persistence: This is a requirement imposed by our approach. In our approach, a PE that requests the token will not withdraw until it has the token granted.

2.3. Clock Synchronization and Asynchronous Systems

Temporal awareness is essential for coordination in all distributed systems. There are two main models for management of time in such systems, synchronous, asynchronous; a mixed system is also possible.

2.3.1. Synchronous Systems

In a synchronous system, a centralized clock is assumed to be active for all participating processing elements to rely on. synchronous behavior characterizes the system according to the function of mainly three components, the processes, the local clock and the communications channels. This means that there has to be bounds on the time a process takes to execute a step, and there have to be bounds on clock drifts and message delays. These are not easy to achieve. Moreover, the entire system typically has to account for the worst case in delays and drifts, there may also be wasted clock cycles in synchronous systems in waiting for a processing element to complete a task or send information.

Such factors result in a big loss of performance due to the worst case model that we have to follow to maintain safety.

2.3.2. Asynchronous Systems

An Asynchronous system simply has no assumption regarding the temporal behavior of the distributed system. Every component acts when it's ready. The work in this dissertation is an asynchronous model. We consider this feature as an important contribution of our work. As noted earlier, it offer great flexibility, scalability and reliability to the system. In our design, the system performance makes no assumption on the speed of the PEs, or their temporal behavior, or even the delays of system components. One exception is the assumption of a response time upperbound for a signal in Chapter 6 (See Algorithm 2). This assumption does not require a synchronous behavior however.

The asynchronous behavior also adds reliability and scalability in the sense that there are no constraints on the temporal behavior of PEs.

Chapter 3. Literature Review

In this chapter, we discuss the previous work related to the work in this dissertation. We discuss what was done in distributed algorithms, we discuss the token ring networks as there are some resemblances with our presented work, we also discuss Optical interconnects and some work that was done in this field, as we present opposite direction arbitration, we show some similar idea.

3.1. Mutual Exclusion

Distributed algorithms have attracted the researchers to find solutions for a multiprocessing element environment without the need for a centralized controller [14].

Mutual Exclusion (or Mutex) is an important problemin a distributed computing environment. There is a lot of work for different mutual exclusion algorithms [15]. Generally, distributed algorithms for mutex mainly categorized to message passing or shared memory.

In message passing algorithms, the PEs operate usually in a point-to-point network and send messages to each other with the information that they need to know to make their own decision. In such solution, the main performance metrics are the following, first is the message complexity, which is number of messages sent by the PEs for algorithm to reach a solution for the problem. The second metric is the synchronization delay (system response time), which is the time for the system to reach the next PE that will take access to the shared resource after a PE gives up access to it. The third metric is the PE response time, which is the time interval a request waits for it to have access to the shared resource after its request messages have been sent out [15], [16], [17]. Our

approach, although very different, has similar measures (system and PE response times). Our approach does not involve sending explicit messages, in the sense of having an explicit destination. If a rough sense of message complexity is used, then our approach requires a constant number of bits to be sent for each request.

The second category of distributed mutex algorithms is the shared memory approach [15], [16], [17], where each PEs share a memory (possibly a distributed shared memory) for communications. Although there is a lot of work in this direction, we will not discuss it further as our approach has little resemblance to the shared memory approach.

The other way for categorization of Mutex algorithms in distributed algorithms environment is time stamps, token based, or quorum based [14].

In timestamp based mutex solutions, each PE generates a request and adds a timestamp so that PEs can determine the earliest request to allow access to the shared resource [15], [16], [17]. A well known message passing Mutex Algorithm is the Lamport algorithm [15], [16], [17], which is a timestamp based algorithm that depends on a digital clock, and it employs message broadcasting for requests that gets access to the shared resource depending on the timestamps. A PE has access to shared resource when all other PE s send a reply with a larger time stamp and if there are other requests, its own request is on the top of the queue. At the end, it broadcasts a release message with again a timestamp and others remove its request from their queues so that they can go on with the other requests in the queue. There are so many modifications done by researchers for performance improvements in the same line like for example reducing the number of reply messages in case the two PE s have different messages that show there are no pending requests with lower timestamps [15], [16], [17].

The token based passes a tokenbetween PEs, and the one that has it has access to the shared resource till it's done and then it releases the token [15], [16], [17]. A well known token based algorithm is the Suzuki-Kasami's algorithm. It just relies on requesting PE s broadcast a request for the token to the other PE s. A PE that possess the token releases the token after it's done with the shared resource. A sequence number is being used instead of the time stamp. There is only one copy of the token in the system, so possession of the token ensures mutual exclusion. In this work, we will often refer to PEs obtaining the token. However, here the token is granted to a PE satisfying a set of conditions, based on signals it puts into and obtains from the network (See Figure 2.1). The quorum based approach uses a minimum number of votes (quorum) to perform consistent operations. The request process is performed on subsets of PEs, step by step, ensuring that only one request has exclusive access to the shared resource [15], [16], [17]. Quorum based Algorithms are quite different from our work.

In our work, we will do a token based approach. However, our work cannot be categorized as a message passing or a distributed shared memory approach. However, it's a bit analogous to message passing in a manner that it involves reading of Optical data, and this reading is destructive. Hence, when more than a PE is reading such data, at most one gets a Logic 1 and all others get a Logic 0. In this way, it's a kind of message passing based on destructive reading of information. Also, although we employ a token based approach, but it's not depending on a sequence number that has to be broadcasted. Also, token based algorithms sometimes may assign a token to a nonrequesting PE, which may never happen in our work. However, it resembles the sequencing concept in the sense that sometimes the system favors the PE that requests the token earlier, we will see this in

more details in Chapter 6 and Chapter 7.

3.2. Token Ring Networks

A lot of work was done on Network protocols [8, 18]. One well-known topology is the Token Ring. In this topology, the Nodes of the network are connected in a ring. A token is possessed by one of the nodes for sending messages, and when it's done, it just hands the token to its immediate neighbor regardless it needs to send or not. If a node receives the token and it does not need it, it just passes the token to its immediate neighbor. Although we do not propose a network protocol, yet this ring resembles our work in a sense that the token is moving in a circular framework with the advantage that the token doesn't necessarily go to the immediate neighbor, it goes to the closest requesting neighbor.

A lot of work has been done in Token Ring Protocols to try to solve the problem of passing the token to nonrequesting nodes, which always degrades performance [8, 18]. The work proposed by Vantrease *et al* [8, 18] addresses this problem and provides a solution, but still it has some differences from our work.

3.3. Mutual Exclusion Related Work in the Optical Domain

As noted earlier, optical interconnects have been attractive to researchers as they can reach much greater transmission rates [2], they have less power dissipation and do not introduce capacitive or inductive coupling that can reduce the bandwidth as in metal interconnects [2]. There is a lot of work in the optical environment with different approaches to contention resolution [6].

One example is the DCAF system proposed by Nitta et al. [6] that uses optical

queues to resolve contention among nodes using a given wavelength. They propose a completely connected network, an arbitration free environment, with no need for any two PE s to share a waveguide or a microring. However, the messages reaching a given PE have to share an optical detector at the end. To overcome this source of contention, they suggested the use of Optical queues; optical queues are difficult to implement, however [19]. In our work, we will show in section 9.3 that we can do the job of the optical queue without the need for a physical optical queue. In other words, this work will enable research such as DCAF.

Proietti *et al.* [10, 20] use arrayed waveguide grating routers to avoid centralized control and manage transmission permissions. Pan *et al.* [21] propose an optical arbitration scheme that is, based on quotas over an epoch that provides max-min fairness in a network with differentiated service; these approaches do not work as a semaphore to provide atomic mutual exclusion, however. There are many other results on Optical interconnects that employed mutex in the electric domain [22]. The literature in distributed computing itself (for instance, [16, 17, 23]) is replete with examples of distributed mutual exclusion, and symmetry breaking techniques such as leader election. Starvation-free mutual exclusion has also been studied before, for example [24–27].

The closest previous work that employs similar ideas to this work is that of Vantrese et al. [18] that presents two arbitration schemes oriented towards communication on a ring topology. A PE can read a message coming from the central node and takes the bandwidth needed and resends a message with the rest for the PE s downstream to utilize the rest of the bandwidth. It is mainly a communication network protocol, and it uses extra channels for the PEs that can be starved to have a higher priority and solve the

fairness problem with the center node.

The method does not appear to easily translate for general mutual exclusion use; more importantly, it is not asynchronous like the one we propose. Further, it employs a centralized controller to ensure fairness (that our network incorporates automatically and in a distributed fashion). The important resemblance between our work and Vantrease's work is that when a PE reads data from an optical waveguide, downstream PEs cannot read anything further.However, we cannot consider their work as based on passive message passing in a distributed environment as is the case in our work.

3.4. Opposite Direction Arbitration

Another related work is that of Kavi and Mehta [28] who propose optical arbitration using oppositely directed waveguides as we do for the Non-Preemptive Architecture (Section 6) and the Fair Architecture (Section 7). However, the model of Kavi and Mehta assumes pipelining of n signals in the waveguide (for a system with n elements) and this does not scale well. Our work, on the other hand, is limited primarily by the delay and attenuation on the waveguide, which is small for inter–, or even intra–chip communication.

Our approach, however, is fundamentally different from most in the literature. It is based on the simple idea that when light is drawn out of a waveguide by a PE, upstream in the waveguide, then it automatically prevents PEs downstream from accessing the light. This is the basis for the mutual exclusion. However, additional details are needed, particularly for safety, fairness and asynchronous operation.

It should also be noted that the standard performance metrics (space and message complexity) of conventional distributed mutual exclusion algorithms are not as directly re-

lated to the optical network and algorithm. To this end, we will show that each PE needs only 3 bits of information conveyed by other PEs through the network.

Chapter 4. Optical Background

Despite the advances in electronics, metal interconnects are lagging behind and they are becoming the bottleneck in the new technologies [6]. Optical interconnects are expected to be the alternative that will introduce the desired bandwidth for inter-processor communications [11] in a multiprocessing system.multiprocessor environments.

In this chapter, we will introduce the main optical components used in this work and provide some information about them that the reader would find useful to understand our work. All optical systems employ a well known component namedthe Waveguide.

A "waveguide" physically guides the light through the required path instead of letting it disperse (as in free space). One example of a waveguide is the optical fiber, that is very widely used in communications. The light can be injected to the waveguide or drawn out of it using different methods. In our work, we will focus on silicon waveguides and the evanescent field effect.

The optical fibers by themselves are one of the mature forms of optical waveguides that is widely used for long haul communications. One may think it can do a good job in intrachip or module communications, but there are technical difficulties. In the chips, another kind, rectangular waveguides are employed instead. One of the main difficulties for the Optical fibers is that adding a cylindrical component in a chip is very difficult, and hence, to have a fiber in a chip, it has to be inserted, which is technically very difficult and also, building other components around it will be almost impossible. Also, the Optical fibers have a large size compared to the Electronic components and this favors the rectangular waveguide as it has a smaller size being in the order of hundreds of nano-meters vs 1-1.5 micrometers for the optical fibers.

There are completely different properties for Optical components from those for Electrical. For example, despite that Optical components do not have capacitance or Inductance problems, still the Evanescent fields impose restrictions on the waveguide placements.

The waveguides must be a few micrometers away from each other so that they do not affect each other through Evanescent fields. This puts restrictions on the number of waveguides that can be in a chip, being much fewer than the Electronic components. However, with the best employment of the Optical components like for example using the same components for different purposes without burdening the system, better results can be achieved.

Optical components have different properties from Electrical ones. Waveguides can intersect without light splitting between the two waveguides, the extension of the waveguide and the intersecting one. There is a small loss of about 0.1 dB in energy, but we cannot withdraw the light from a waveguide by intersecting the waveguide holding light with another one. This can help having two dimensions of the waveguides in the same plane without the fear of problems with interferences. Of course, it cannot exceed a limit as the power loss should not be more than 3dB in most of the systems.

However, there is a physical property that exists in Optical environment that can withdraw light from an optical waveguide to another. The Evanescent fields. This happens when two Optical waveguides are placed within a short distance between them. This can introduce coupling of energy between them, and energy can be exchanged between the two. This can be used to transfer light and hence data between devices. This can be done using Optical microrings.

The Optical microring is a short circular waveguide that can be coupled to two waveguides to pick up light energy from one and transfers it to the other one with the same mechanism of Evanescent fields. In this way, so many propsed systems employ the microrings to transfer light energy between two flat waveguides.For light transfer to happen, there are restrictions for some physical properties like the relative Electrical permittivity (ϵ_r) of the material, together with the length of the microring, the distance between the microring and the flat waveguide.

The relative permittivity can be controlled Electrically. This property is being employed to tune the microrings to particular wavelengths to pick up some particular channel between the different processing elements. Due to some technical properties, the control is still limited and cannot pick up so many channels, and in most of the systems, it's designed to pick up a channel or remains inactive.

The Evanescent fields still imposes a restriction in our work, even if the microring is inactive, still there will be some leaked energy and a drop of around (0.07 dB) is there. Hence, there will be limits on the number of microrings a light path can tolerate without a significant drop of energy.

One of the differences between the Optical and Electrical connections is that in Electrical, there can be vertical connections, and there can be different levels of metal connections that are all connected together. But In Optical interconnects field, it's really difficult technically to have a vertical microring to couple two flat microrings. This actually puts more restrictions on the number of waveguides that can be used, but still, it's a very powerful tool that researchers have and still are working on.

Yet, in our system, we deal with the microring as an ideal or near ideal switch, ei-

ther active and hence light is withdrawn, or inactive and light just passes unchanged.

Now, we will discuss the usage of these elements in more detail.

4.1. Optical Building Blocks

Many proposed systems employ silicon waveguides [6–10] to transport the light and microring resonators as switches or taps on these waveguides [6–10]. We describe two optical building blocks, that we will call (a) Optical Switch and (b) Optical Detector. These building blocks will allow the rest of the dissertation to abstract away from details of optical devices and focus on the distributed computing aspect of this work.

4.1.1. Optical Switch

Figure 4.1 shows a schematic of the Optical Switch. It has two inputs m, x and two outputs y, z; m is an electrical signal and x, y, z are optical. Each of m, x, y, z can be viewed as a Boolean signal with values from $\{0, 1\}$; for the optical signal, a 0 (resp., 1) indicates the presence (resp., absence) of light.

The Optical Switch described here directs light x (under electrical control m) from an input waveguide to one of two outputs y or z. We first consider the behavior of the Optical Switch in the steady state (when the signals m, x, y, z are either 0 or 1 and nothing else). In Boolean Algebraic terms, $y = x \cdot m'$ and $z = x \cdot m$. The Optical Switch can be viewed as demultiplexer, noting that

$$(y, z) = \begin{cases} (x, 0), & \text{when } m = 0 \\ (0, x), & \text{when } m = 1 \end{cases} \tag{4.1}$$

Actually, the block in Figure 4.1 will be used in the rest of the dissertation, and it's the main building block of optical interconnects.

As mentioned earlier, in a multi processor environment, designing a system based

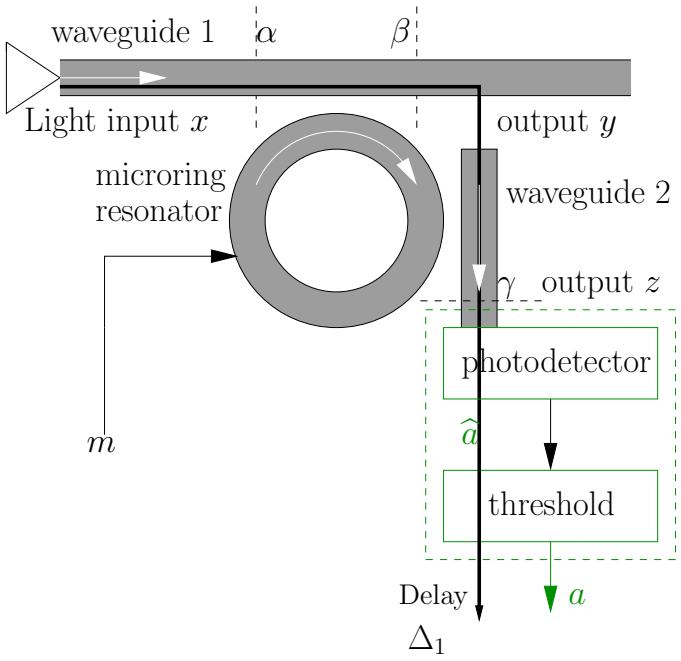

Figure 4.1. A schematic of the Optical Switch (and an Optical Detector with the green block).

on a centralized clock is really hard and there are technical and physical difficulties that will cause a lot of problems or impose a lot of restrictions. A big advantage of our work is that it's asynchronous, with no centralized clock that the processors rely on. Hence, we need to study the transient behavior of our Optical components that we will employ in our design.

We now consider the transient behavior of the Optical Switch when a change of value of m causes a change of the light in y, z. Consider the schematic of Figure 4.1, once again. For this analysis we assume that the value of m is appropriately latched, so that $m = \widehat{m}$ in Figure 4.1. We consider an inherent latch in the system so that when a microring is activated, it remains active even if the original input wasn't active any more, and remains active till it's cleared. Suppose that $m = 0$, then the light (if any) in waveguide 1 proceeds unimpeded from input x to output y. There is no light in the microring

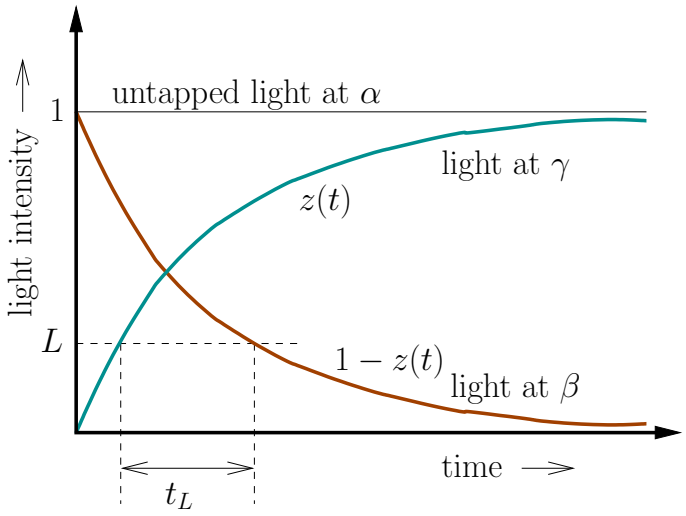

Figure 4.2. An Optical Switch's transient behavior.

resonator and waveguide 2. When m becomes Logic 1 (microring activated), then the light in the region $[\alpha, \beta]$ of waveguide 1 of Figure 4.1 gets transferred into the microring (over time), and then from the microring into waveguide 2.

The top horizontal line represents the unit amount of light at point α, when $m = 0$. When $m = 1$, the light gradually (in the order of picoseconds) moves from waveguide 1 to waveguide 2 (via the microring). This is shown by the reduction of the light at point β (brown curve) and the increase of the light at point γ (teal curve, representing output $z(i)$). Ignoring losses and delays the signal at β is $1 - z(t)$. Because the light is directional, we will call the point α upstream of point β or point β downstream of point α on waveguide 1.

The following result notes that there is a finite time within which the level of light downstream of an active switch falls below a threshold (see Figure 4.2); this relies on the fact that $z(t)$ is monotonically increasing.

Lemma 4.1.1 *Let $0 < L < 1$ be some value of light power. For any such L, there exists a*

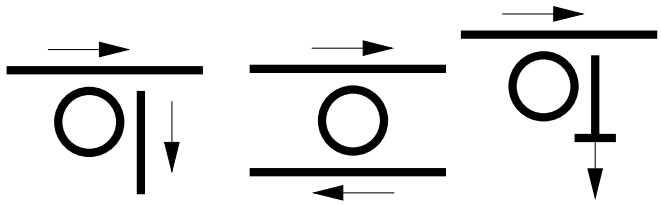

Figure 4.3. Depictions of an Optical Switch in two orientations (left, center), and an Optical Detector (right).

finite time $t_L > 0$ *such that for any time* $t > 0$*, if* $z(t) > L$*, then for any time* t' *such that* $t' > t + t_L$*, we have* $1 - z(t') < L$. ■

We note that in a non-ideal Optical Switch the signal $z(t)$ starts rising a little later than at time $t = 0$ (assuming the level of light at β starts falling at $t = 0$). So the level of light at β at time t is $1 - z(t') < 1 - z(t)$ where $t' > t$. Thus the above lemma holds for an non-ideal Optical Switch as well.

For brevity, we will depict an Optical Switch as shown in Figure 4.3(left). The arrows indicate the optical input and two outputs. For clarity, the electrical input (m in Figure 4.1) is not shown in Figure 4.3. Figure 4.3(center) shows a different orientation of the waveguides. The principle is still the same; namely redirection of any input light to one of the two outputs.

4.1.2. Optical Detector

We now use the Optical Switchof Section 4.1.1 to build a module that uses a Boolean electrical input m to detect light and generate a Boolean electrical output a based on certain thresholds. The Optical Detector is identical to the Optical Switch ,except for a photodetector-thresholding circuit at the output of waveguide 2, as shown in green in Figure 4.2. The Optical Detector produces a Boolean output $a = 1$ whenever the light at γ satisfies certain thresholding conditions explained later. Recall that the signals x, m are

Boolean inputs to the Optical Detector as well. In the steady state if light enters point γ in waveguide 2, then output $a = 1$ (as implied by the Boolean Equation (4.1)); the transient case, that is vital for the asynchronous behaviour considered in this paper, is discussed below.

The photodetector has the (analog) optical signal z as input and produces an analog electrical signal $\widehat{a}$ with the same characteristics as function $z(t)$ at point γ of Figure 4.2.

Specifically, for any given power $0 < L < 1$ of the light entering the photodetector, there is a corresponding electrical level $L$of signal $\widehat{a}$.

A thresholding circuit uses signal $\widehat{a}$ as input and produces a Boolean output a as follows:

$$a(t) = 1 \text{ iff } \widehat{a}(t') \geq L, \text{ for all } t - T < t' \leq t \tag{4.2}$$

Thus, $a = 1$ whenever $\widehat{a}$ exceeds the threshold Electrical Level of L.

continuously for T time. Here T is a time threshold and L is a power threshold for the Optical Detector. Assuming a 0-delay photo-detector-threshold circuit, the above definition can be viewed in terms of Figure 4.2; the 0-delay assumption is only to simplify the discussion and does not impact the ideas introduced.

An Optical Detector is denoted as shown in Figure 4.3 (right).

In the subsequent chapters, we will employ the above components to achieve mutual exclusion. We first introduce the very basic idea that we build our work on, light comes from one side, the closest active PE gets the light and gets the token (Chapter 5); liveness is guaranteed by flow of light, power and time thresholds are implemented to en-

sure safety and safe assignment.

Next, in Chapter 6, we modify the architecture to add non-preemption. Finally in Chapter 7, we introduce the main architecture of this work that incorporates fairness.

4.2. Practical Considerations

In the previous sections, we considered the Optical components as ideal. Although in reality, there are losses in coupling, and even if a microring is inactive, it introduces losses to the light in the waveguide, and even the light propagation in the waveguides is not ideal either, and there are losses associated with this propagation.

In this way, we need to consider the light levels that the inputs to the detectors will reach. Assuming unit power input, we have to address the lowest light level that an input may have as it is inputted to a light detector.

Consider the case where each PE in the system can have up to k microrings in the path of the light, loss per microring as x dB, total loss for the whole path of light as Q dB.

So the total loss T in dB is $k * x + Q$

Now, let's consider the total attenuation that a light detector can still tolerate as y dB In this case, $T \leq y$ Hence, $k * x + Q \leq y$

The quantities x, y are technology dependent, and also for Q, perhaps the only difference between Q and x, y is that it has a part of our design as the longer the waveguide the higher Q, but let's consider it as constant for simplicity. In this case, we can find the largest number of PEs we can have in an Optical system. Moreover, we can also change our layout to reduce the number of microrings that can attenuate the light in its path,

meaning it'll be a matter of increasing the number of possible PEs in the system at the cost of adding some more waveguides or having a an area for this modified path set.

Chapter 5. The Base Architecture

In this chapter, we introduce the main principle that we employ to achieve mutual exclusion. We use a network in which PEs are arranged linearly on a waveguide into which light enters from one end. Thus, a PE is upstream or downstream on this waveguide, depending on whether it's closer to the light source or not. The broad principle driving our approach to mutex is the following. If light is drawn by a PE i then no other downstream PE j can draw the light as PE i has depleted the waveguide of light. Thus if one PE draws the light, another cannot. This idea, based on conservation of energy, is the basis for safety in mutex (See section 2.2).

The network we propose here is not new (as a structure). However, its use with Optical Detectors (see subsection 4.1.2) with appropriate thresholds in the context of mutex is our contribution. The structure we introduce also functions as a base building block for other networks.

Despite its simple structure, the correct operation of the Base Architecture requires several details to be required and met. In Section 5.1 we describe the Base Network and its algorithm. Recall that an "Architecture" we have in this dissertation has a Network (minimum optical hardware specification) and an Algorithm (that runs distributedly on this network) associated with it. Next, in Section 5.2 we prove the correctness of the Base Architecture with respect to safety and liveness of mutex. In the process, we also derive the worst case and amortized system response times. Finally, in Section 5.3 we identify the limitations and uses of the Base Network.

5.1. Base Network and Algorithm

Here, we will use the Optical Detector of subsection 4.1.2 for the Base Architecture, which is simply a sequence of n Optical Detectors sharing a common waveguide (see Figure 5.1). The Base Network (the hardware component of the Base Architecture) represents

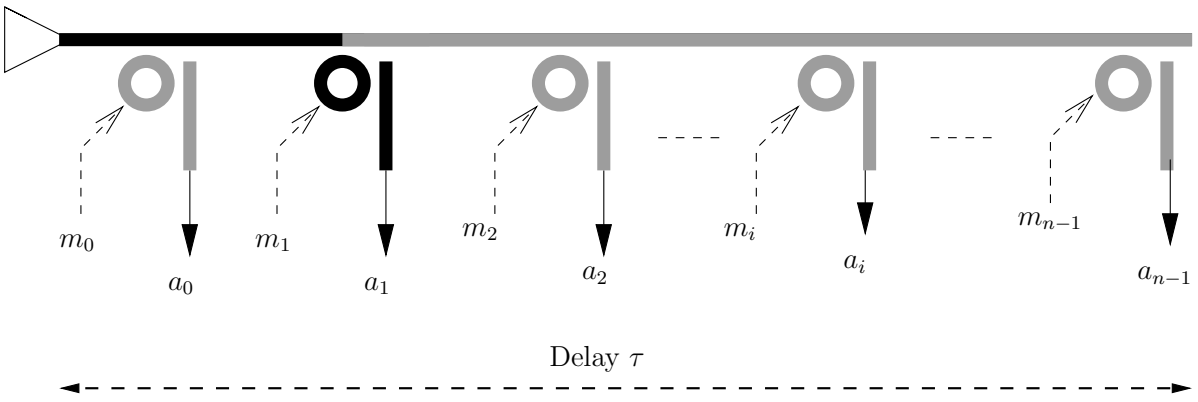

Figure 5.1. The Base Architecture. The Figure shows the network in the steady state with $m_0 = 0$ and $m_1 = m_2 = 1$. . The triangle on the left of the shared waveguide indicates a laser input, with the light flowing from the left to the right.

a fundamental building block of other optical networks proposed in this dissertation (and in many others in the literature). The network in Figure 5.1 shows competing PEs whose inputs m_i and outputs a_i correspond to m and a of Figure 4.1. Consider an example in which inputs $m_0 = 0$, $m_1 = m_2 = m_i = m_{n-1} = 1$ .Microring 0 does not draw the light out of the waveguide. Consequently output $a_0 = 0$. Further since $m_1 = 1$ and the waveguide carries light to PE 1, then $a_1 = 1$. However for PE 2, $a_2 = 0$ even though $m_2 = 1$, as the waveguide has been depleted of light before it reaches PE 2. This idea clearly generalizes to n PEs.

Recall flags R_i, G_i in the definition of the mutual exclusion problem in Chapter 2. If, in Figure 5.1, $m_i = R_i$ (indicating whether PE i requests the token) and $a_i = G_i$ (indicating whether PE i is granted the token), then in this example all PEs except for PE 0

request the token and PE 1 is granted the token. Thus to request a token, PE i simply sets $m_i = R_i = 1$ and it is holds the token as long as $G_i = a_i = 1$. To free the token, it simply sets $R_i = m_i = 0$, and (resets the latch i holding the value of m_i).

The Algorithm part of the Base Architecture implements this function.

Algorithm 1: Base Architecture Procedures

```
/* For the Base Architecture (that allows a token to be preempted),
   the flag G_i is identical to the output a_i of the
   Base Architecture                                              */

1 Procedure Get_B(m_i, a_i)  /* Compete for Token on Base Architecture */
2 |  Condition: R_i = 1
3 |  m_i ⟵ 1  /* activate microring                               */
4 |  wait until G_i = a_i = 1  /* Token granted                    */

5 Procedure Free_B(m_i)  /* Free Token */
6 |  Condition: R_i = 0
7 |  m_i ⟵ 0  /* not competing for token; this will cause a_i = G_i to
   |     go to 0                                                   */
```

In general, the Algorithm we use for each of the proposed architectures have one or more procedures, each executed when a set of conditions is satisfied. For example, Algorithm Get_Base$(i, \cdots)$ is executed by each PE i that satisfies the condition $R_i = 1$ (PE i is requesting the token). We assume that this condition is to be satisfied before the procedure is executed; however once execution of a procedure starts, it continues to completion regardless of whether the starting condition holds during the execution.

5.2. The Base Architecture Correctness and Performance

In the steady state, we can observe that in the Base Architecture at most one of its outputs $a_i = 1$. This follows from the observation that if light is drawn out by one microring, then it is not be available for any microring downstream. However, we must

adjust the Optical Detector thresholds to ensure that even under transient conditions two different PEs do not hold the token simultaneously (see safety condition in chapter 2).

Consider the Optical Detector in Figure 4.1. Recall that in response to m changing from 0 to 1, the levels of optical signal z, and subsequently the electrical signal $\widehat{a}$, begin to increase from 0 (see Lemma 4.1.1 and the remark following the lemma). Let Δ_1 be the worst-case delay for this response of signal $\widehat{a}$. That is if at time t, m goes from 0 to 1 then the latest time at which $\widehat{a}$ starts changing is $t + \Delta_1$. Let Δ_2 be the worst-case delay of the thresholding circuit; that is if $\widehat{a}(t)$ has just satisfied the level and time thresholds, then the latest time at which a changes from 0 to 1 is $t + \Delta_2$. Let $\Delta = \Delta_1 + \Delta_2$. In addition, let τ represent the optical delay of the shared waveguide of the Base Architecture (time for light to travel from PE 0 to PE $n-1$); that is, the time needed for the light to travel on waveguide 1 between the farthest pair of optical detectors in the Base Architecture is at most τ. For any level threshold $0 < L < 1$ of the Optical Detector, set its time threshold to be

$$T > t_L + \Delta + \tau \tag{5.1}$$

Some of these ideas are illustrated in Figure 5.2. Observe that if only one PE i requests the token, then the time from setting $m_i = 1$ to the time when a_i becomes 1 is $t_{max} = T + \Delta + t_L = 2(\Delta + t_L) + \tau$. All of these quantities can be quite small (in the order of a 100 picoseconds [2]).

Recall from Figure 4.2 that t_L is the time needed to deplete the light level entering any downstream Optical Switch to below L. Recall also that $a_i(t)$ denotes the value of digital output a of Optical Detector i at time t. Further, note that level threshold L of

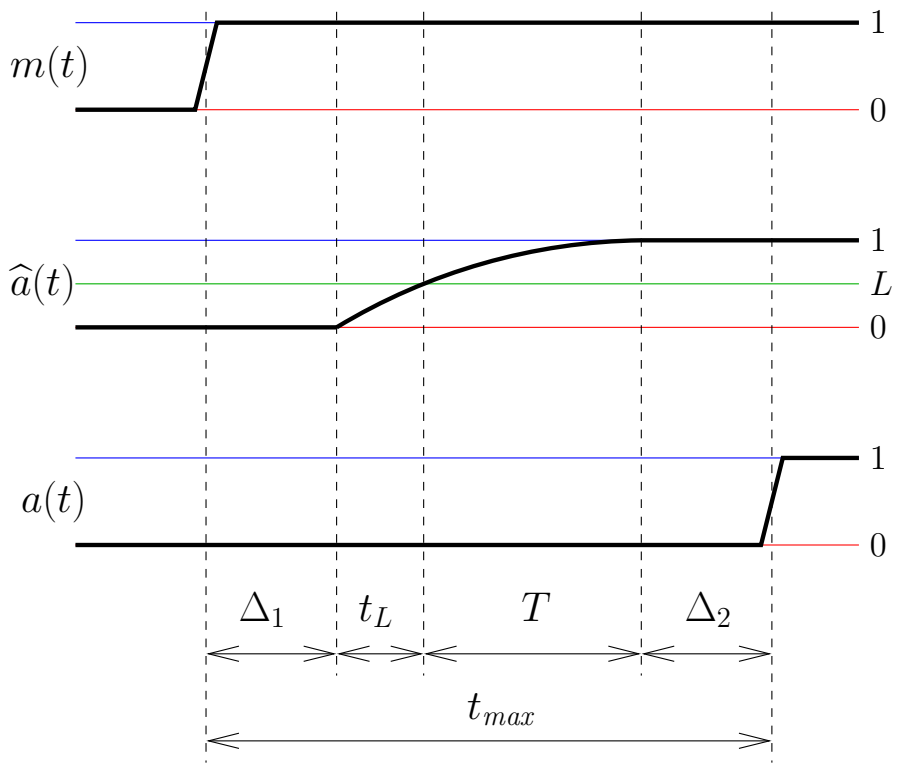

Figure 5.2. An illustration of the timing parameters of the Base Architecture.

optical signal z_i, corresponds to the level A_L of electrical signal $\widehat{a}_i$; that is, level $z_i = L$, causes level $\widehat{a}_i = A_L$.

Theorem 5.2.1 (*Mutual Exclusion*) *For any PEs* $0 \leq i, j < n$, *of the Base Architecture, if at any time* t, $\quad a_i(t) = a_j(t) = 1$, *then* $i = j$.

Proof: Let thresholds L, T and quantities A_L, t_L, Δ and τ be as described above. Suppose that at time t, $a_i(t) = a_j(t) = 1$ for some $i > j$. Then for all time t_1 satisfying $t - T \leq t_1 \leq t$, we have $\widehat{a}_i(t_1) > A_L$. Similarly, $\widehat{a}_j(t_1) > A_L$ for all $t - T \leq t_1 \leq t$.

Now consider any $t_2 \geq t_1 + t_L$. We have $z_i(t_2) < L$ where $z_i(t_2)$ is the light power level in the waveguide downstream at PE i, Hence for all $j > i$ light input $f(t_3) < L : \forall t_3 :$ $t_2 + \tau \leq t_3$, where $f(t_3)$ is the light power level at time t_3 at Optical Detector$i$. Hence, $\widehat{a}_j(t_4) < A$ for all $t_3 + \Delta \leq t_4$

Thus, $a_j(t_4) = 0$, but $t_3 + \Delta \leq t_4$:

Therefore: $t_2 + \tau + \Delta \leq t_4$:

Consequently: $t_1 + t_L + \tau + \Delta \leq t_4$

This results in, $t_1 + t_L + \tau + \Delta \leq t_4$

Therefore: $t - T + t_L + \tau + \Delta \leq t_4$ Hence, as $t_L + \tau + \Delta \leq T$: then, if we consider $t_5 = t + (t_L + \tau + \Delta) - T$: then $t_5 \leq t$

Therefore: $a_j(t_4) = 0 \forall t_4 \geq t_5$ which implies: $a_j(t) = 0$ Contradiction, in other words, with proper choice of L, A, T: safety is always maintained.

■

Remark: The key idea of the above proof is that (in addition to the time t_L needed to deny downstream nodes light above the level threshold L) any stray light passed downstream due to the delay Δ of $\widehat{a}$ in responding to m, or the delay τ of the waveguide in transporting the light is accounted for in the time threshold.

Let the system response time of the Base Architecture be T_B^S. The worst case PE response time for the Base Architecture is $T_B^P = \infty$ as in the architecture can cause a downstream PE to never get the token.

Theorem 5.2.2 *Let $t_{\max}$ be the worst case time for the token being granted when there is a single request in an n-PE Base Architecture. The system response time of the Base Architecture is at most $(n-1)t_{\max}$ for a single token grant and at most $2t_{\max}$ per token grant, when amortized over a large number of token requests. Here $t_{\max}$ represents the sum of maximum delays in a PE Optical components.*

Proof: Consider the situation where at time t PE i sets $R_i = m_i(t) = 1$ (requesting the token). Assuming there is no other request from PE $j < i$ at this time, PE i receives

the token when a_i be comes 1 at time $t + delays + threshold = t + t_{\max}$. However, suppose PE $i-1$ starts competing so that just before PE i receives the token the lights falls below level L and the token is not granted to PE i after a delay of almost $t_{\max}$. This process can be extended to have PE $n-1$ start first and each preceding PE making a request just in time to deny the token to downstream PEs. Thus the worst case delay in awarding the token can be $(n-1)t_{\max}$.

This extended delay cannot happen every time, however, due to the request persistence property that we assume (see Section 2.2). Suppose a set of $1 \leq x \leq n$ requests from PEs i_1, i_2, i_x (with $i_1 < i_2 < \cdots < i_x$), are made and the request times arranged so that $(x-1)t_{\max}$ time is spent before PE i_i receives the token. Because requests are persistent, the remaining $x-1$ PEs will not remove their request until they receive the token. That is, they cannot delay each each other as before. The only delay in their receiving the token, when no upstream PE has a request is the delay in moving the light from waveguide 1 to waveguide 2, detecting it and satisfying the threshold requirements; this quantity is upperbounded by $t_{\max}$. The remaining $n-x$ PEs can have cascading delays as the fisrt x PEs, but only once. Thus, over a sequence of $2n$ requests, there are $2n$ grants in at most $t_{\max}\left((n-1)+1+n\right)$ time. Thus the amortized time for each request is constant. ■

Let us consider the model as in Figure 5.3. In this case, when we need to compute the amortized system response time, we get a start and end points in time and compute number of requests and number of grants in this time span. But let's recall the rules that we get from the model, the requests can come within any small time, but the grants cannot be within less than T time, it can be more if a PE was just about to be granted the token and just a higher priority PE interrupts the operation (or addups of this process),

but those PEs that were interrupted or just outside from the beginning can be put in a set where at most one of them can be either granted the token or start the period T and gets interrupted without being granted ; and this is the key point for which the amortized system response time can be found to be at most $2 * T$. Again recall the model in Figure 5.3, for the number of requests from t_1 till t_2, let it be $Y = X + V$, we can have X granted and $V < n$ still waiting. For each PE being granted the token, either it was never in waiting mode, and it must wait a T period, and at most interrupted a PE with a lower priority amongst the set of waiting PEs. In this case, this PE grant contributed with at most $2 * T$ time to the total time. The other case is that the PE that gets the token granted was in the waiting mode; and again the grant process adds T period of time to the total time, but it might have taken another T period in another grant process to move to the waiting mode; meaning that the whole grant process added at most $2 * T$ period of time to the total time. Hence, if we have X grant processes, then the total grant process time is at most $2 * X * T$ time, and if we have V waiting, we have at most $V * T$ more.

In this way, the total time for X grants is at most $(2 * X + V) * T$.

In other words, if PE i is one of the X PEs contribute to the total time by T to get the token, and at most another T wasted. Reason for this wasted time either PE i was about to be granted the token and was rejected, or there was another waiting PE j that was about to get the token and PE i interrupted it. The other PEs waiting can add at most T time each if it was competing and about to get the token then got rejected, but it happens with them just once, and it keeps waiting till it gets the chance to be granted the token, meaning that there is no waste in the following cycles.

Thus, the maximum amortized time for system response is $(2 * X + V) * T/X$, recall

that $V < n$, then for large X, maximum amortized time is $2 * T$.

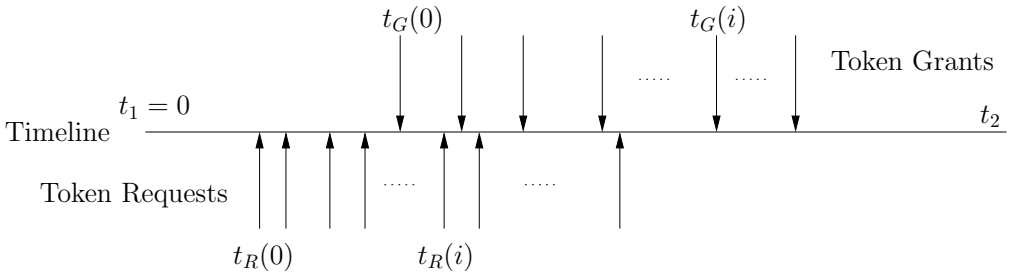

Figure 5.3. Requests and grants sequence in time

5.3. Limitations and Uses of the Base Architecture

In this section, we identify why the Base Architecture is not useful directly for Mutex. However, it serves very well as a building block for other networks that address various aspects of Mutex. Further, because of its simple structure and similarity to standard optical fabric for data communications, it is likely to be part of any optical interconnection.

Limitations Theorems 5.2.1,5.2.2 showed that the Base Architecture provides safety and liveness to Mutex. It is also clear from Algorithm 1 that safe assignment is also satisfied. However the network has limited practical direct use for Mutex as it does not guarantee non-preemption.

Take for instance the example of Figure 5.1. Suppose PE1 gets the token (by the hint of being the most upstream PE requesting the token). Regardless of whether PE1 being done or not with its use the token, PE0 can at any time preempt PE1 simply by setting $R_0 = m_0 = 1$. In this case, PE0 will draw the light off from the waveguide before PE1 has a chance to receive the light. This would make $a_1 = G_1 = 0$.

Uses of the Base Architecture While the Base Architecture is not useful as standalone structure for Mutex, it forms an important building block. In this section, we identify two situations where a Base Network is used. Before we proceed, we define the Characteristic Sequence of a Base Architecture.

Definition 1 For any Base Architecture connecting a set of n PEs, the *Characteristic Sequence* of the Base Architecture is a list of the PE ids in the order in which they receive the light on the waveguide. ■

Remark: For Figure 5.1 the Characteristic Sequence is $\langle 0, 1, 2 \ldots, n-1 \rangle$.

Definition 2 *A sequence S of length n is a sequence of n quantities denoted typically by:*

$$\langle S = S_1, S_2 \cdots S_{n-1} \rangle$$ ■

The Reverse of sequence S is another sequence $S^{-1} = \langle S_{n-1}, S_{n-2}, \cdots S_1, S_0 \rangle$

The Base Architecture is used in subsequent portions of this dissertation in two ways, both for which we use two copies of the Base Architecture with Characteristic Sequences S that are reverses of each other.

That is, the Base Architecture will be used in our thesis with the light flow (and the PE priorities) in opposite directions. In one case, the two Base Architecture structures are completely independent and this is called the Hand_Over Network in Section 7.1. In another case, one instance of the Base Architecture will drive the other as in the Non-Preemptive Network of Chapter 6.

Chapter 6. The Non-Preemptive Architecture

In Chapter 5, we introduced the Base Architecture that constitutes a main building block of our work.A main problem of the Base Architecture was that a PE holding the token may not keep holding the token without being preempted by a higher priority PE. Key to this preemptive behavior is that upstream PEs can preempt the flow of light to downstream PEs and hence wrest away the token at any time. In this Chapter, we introduce the Non-Preemptive Architecture that prevents preemption, in addition to preserving safety and response times (liveness).

6.1. The Non-Preemptive Network

Consider the Non-Preemptive Network shown in Figure 6.1. Observe that it consists of two Base Networks, one colored blue (the "Blue Network") and the other the "Red Network." The important difference between these two instances of the Base Network is that the light propagates in them in opposite directions in the corresponding waveguides. PE i is downstream of PE j in the Blue Network iff PE i is upstream of PE j in the Red Network. Thus, a PE with high priority in the Blue Network has a low priority in the Red Network and vise versa. The broad idea of the functioning of this network is as follows. The Blue Networks give a temporary token a_i to competing PEi exactly as in the Base Architecture. This temporary token is used as permission to compete on the Red Network. The winner of the Red Network obtains the actual token b_i. Once the token is obtained, say by PE i, another PE j_1 (with $j_1 > i$ that is downstream of PE i in the Blue network) cannot preempt PE i, as it cannot obtain a temporary token. On the other hand PE j_2 (with $j_2 < i$ that is downstream of PE i in the Red Network) can get a temporary

token but not the actual token held by PE i. We now discuss additional details.

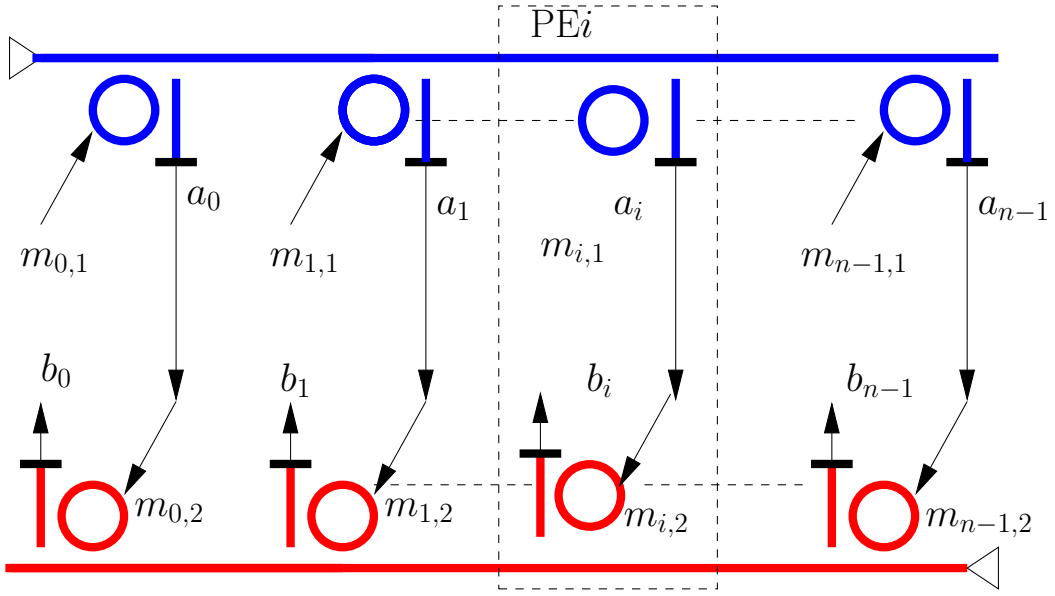

Figure 6.1. The Non-Preemptive Network. Note that output a_i of the Blue Network is directly input to the $m_{i,2}$ input of the Red Network (in a time bounded manner). As in the Base Network, the white triangle in the left (Resp. right) end of the Blue (Resp. Red) waveguide denotes a laser source and indicates that the light travels on the Blue (Resp. Red) waveguide from left to right (Resp. right to left).

Recall that the Optical Detector of Figure 4.1 has input m that is latched. For the sake of the explanation here, let $\widehat{m}$ be the latched value of m. That is, once latched, the value of $\widehat{m}$ does not change even if the value of m does. In Figure 6.1 the temporary token a_i output from the Blue Network is the same as signal $m_{i,2}$ whose latched counterpart $\widehat{m}_{i,2}$ does not change even if a_i does. Figure 6.2 with parts illustrates the basic idea of the Non-Preemptive Architecture; although this example is with PE 1 obtaining the token, the idea clearly extents to any PE. Suppose PE 1 obtains the temporary token a_1 as PE 0 was not in the competition at the time PE 1 set $m_{1,1}$ to 1; this also makes it impossible for PE i, for $2 \leq i < n$, (that are downstream of PE 1 in the Blue Network) to obtain the temporary token a_i. The temporary token a_1 (and the fact that $a_i = 0$, for all $2 \leq i < n$), allows

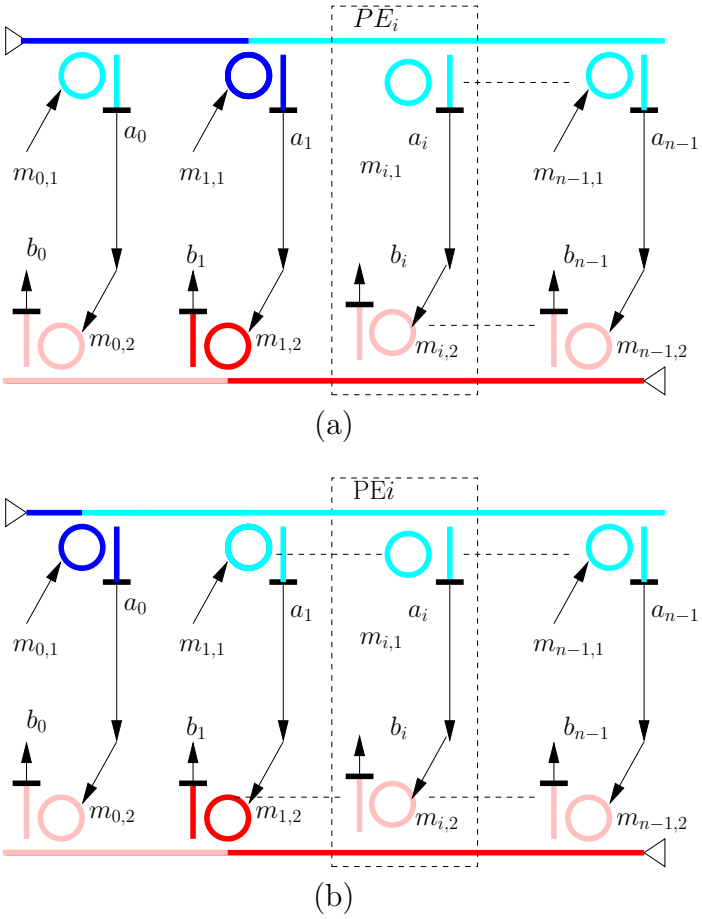

Figure 6.2. The Non-Preemptive Networkin a situation when PE 1 competes and gets the token, then after a while, PE 0 gets interested in the token.

PE 1 to redirect the light in the Red Network to produce $b_1 = 1$. After PE 1 has latched the value 1 into $m_{1,2}$, an upstream PE in the Blue Network (PE 0 here) cannot prevent PE 2 from obtaining the token, as a PE that is upstream of PE 1 in the Blue Network, is downstream of PE 1 in the Red Network. Note that in the example a_1 may become 0, but $\widehat{m}_{1,2}$ continues to be 1 until the latch is reset.

Before we proceed, we need to define the thresholds for the optical detectors of the Non-Preemptive Architecture. The Blue Network uses a level threshold L and a time threshold $T_{\text{blue}} = t_L + \Delta + \tau$ (see Equation (5.1)). After the Blue Network sets an a_i to 1, there is a delay before the latched value $\widehat{m}_{i,2}$ becomes 1. Let this delay be upper bounded

by δ. Then, for level threshold L, the time threshold of the Red Network is $T_{\text{red}} > T_{\text{blue}} + \delta$. This prevents preemption by a PE in the Blue Network (see Lemma 6.3.2). It should be noted at this point that there need not be a direct connection from a_i to $m_{i,1}$. PE i could, on receiving $a_i = 1$, separately set $m_{i,1} = 1$ and latch it; however, there must be an upper bound δ on the delay to do this.

6.2. The Non-Preemption Algorithm

Algorithm 2 with its procedures show the working of the Non-Preemptive Architecture.

Algorithm 2: Non-Preemptive Architecture Procedures

```
1  Procedure Get_N(m_{i,1}, m_{i,2}, a_i, b_i)  /* Compete for Token */
       /* PE i competes for the token with variables m_i, a_i, b_i      */
2      Condition: R_i = 1 and G_i = 0
3      m_{i,1} ⟵ 1  /* compete on blue Base Architecture               */
4      wait until a_i ⟵ Get_B(m_{i,1}, a_i) = 1  /* Wait for temporary token */
       /* a bounded time δ elapses after a_i = 1 until PE i begins
          competing in the red Base Architecture by setting m_{i,2} to 1
          (next statement)                                              */
5      m_{i,2} ⟵ 1  /* compete on red Base Architecture                */
6      wait until b_i ⟵ Get_B(m_{i,2}, b_i) = 1  /* wait for token     */
7      G_i ⟵ 1  /* Token granted                                       */

8  Procedure Free_N(m_{i,1}, m_{i,2})  /* Free token */
9      Condition: G_i = 1 and R_i = 0
10     G_i ⟵ 0  /* token given up                                      */
11     Reset latches m_{i,1} ⟵ m_{i,2} ⟵ 0
```

The Non-Preemptive Architecture Algorithm has one Procedure for obtaining the token and another one for releasing it, (see Algorithm 2). On entering Procedure $Get\_N(\cdots)$, PE i first activates its Blue Network microring $m_{i,1}$ (line 3). This allows it to compete on the Blue Base Network and obtains $a_i = 1$, if possible; this is reflected in the

wait of line 4. One could view $a_i = 1$ as an irrevocable permission for PE i to compete in the Red Network (move to line 5). In line 5, PE i activates its Red Network microring. If it succeeds in the competition in the Red Network, (obtains $b_i = 1$), then it grants itself the token $G_i \longleftarrow 1$ (line 7). To release the token, PE i deactivates its microrings as in line 11 ($m_{i,1} \longleftarrow 0, m_{i,2} \longleftarrow 0$).

A question may arise about the difference between the Non-Preemptive Architecture and the Base Architecture; why does it provide non-preemtion? The key idea here is that the granting of the temporary token is irrevocable. When a PEi gets permission to compete in the Red Network (gets $a_i = 1$), and then if flag a_i subsequently drops to 0, it still keeps competing for the token. The permission can only be withdrawn when the PE decides to deactivate its rings and this happens only when it gives up the token.

6.3. Correctness and Performance of the Non-Preemptive Network

Clearly the Algorithm guarantees safe assignment; that is, only a PEi that seeks the token (as $R_i = 1$) can be granted the token. The condition for executing Procedure Get_N by PEi includes $R_i = 1$.

Because the Non-Preemptive Architecture outputs through a Base Network, safety is guaranteed by Theorem 5.2.1.

Lemma 6.3.1 *The Non-Preemptive Architecture satisfies mutual exclusion safety.* ■

The steady state analysis is straightforward.

With reference to Theorem 5.2.1, if $a_i = 1$ then for all $j \neq i$ $a_j = 0$.

Second, again with same reference but concerning the Red Network, also $b_i = 1$ implies for all $j \neq i$, $b_j = 0$. These two work well in the steady state, but in our case,

transient analysis is crucial for safety. The keyword for the safety in our case is that we inherit the threshold time from the Base Architecture. In our case, we adjust two different thresholds for the Blue Networkand the Red Network.

Theorem 6.3.2 below establishes the Non-Preemption property of the Non-Preemptive Architecture.

Lemma 6.3.2 *In the Non-Preemptive Architecture if PE i is holding the token, then another PE $j \neq i$ cannot obtain the token until PEi gives up the token by executing Procedure Free_N.*

Proof: Let PE i hold the token. We now show that PE j ($j \neq i$) cannot cause PE i to lose the token.

Since PE i already holds the token, (having won it in the competition), we assume that PE j makes its request after PE i has already obtained the token. Because PE i has $G_i = 1$, $b_i = 1, m_{i,1} = 1$, and $m_{i,2} = 1$. We now have one of two cases:

Case 1: Here $j > i$ (PE j is downstream PE i in the Blue Network). PE j starts its request by setting $m_{j,1} = 1$, however a_j remains 0 as $m_{i,2} = 1$ and ensures that light in the Blue Network cannot reach PE j because it is downstream. Hence, PE j waits at line 3 of Get_N (Algorithm 2) for a_j to become 1, and does not compete in the Red Network.

Case 2: Here $j < i$ (PE j is upstream PE i in the Blue Network, but downstream PE i in the Red Network). PE j sets $m_{j,1} = 1$ and gets $a_j = 1$ and reaches line 5 of procedure Get_N and sets $m_{j,2} = 1$ but as PE i has $m_{i,2} = 1$, PE j does not receive the light in the Red Network and b_j remains 0. The key here is that the permission to compete in the Red Network is irrevocable. The additional time δ in the time threshold of the Optical Detectors of the Red Network, ensure that once, $\widehat{m}_{i,2}$ has been set, no other PE can

prevent PE i from getting and holding the token. ■

Theorem 6.3.3 *The Non-Preemptive Architecture provides mutual exclusion with safety, safe assignment, liveness and non-preemption.*

Safety in the steady state is guaranteed from the conservation of energy. Safe assignment is guaranteed from the fact that a PE may compete for the token only if it wishes for the token. Liveness is guaranteed in ideal operation of the algorithm from the propagation of light. Transient analysis is similar to the Base Architecture with the exception of showing that the extra term in the Red Network time threshold ensures nonpreemption in transient analysis. Non preemption in the steady state is shown in the above mentioned discussion proving Lemma 6.3.2.

The following observation (resulting from the fact that the most downstream PE of the Blue Network is the most upstream PE of the Red Network). For example, for the Non-Preemptive Architecture, PE $n-1$ has the lowest priority in the Blue Network and the highest priority in the Red Network. This observation is used in the next Chapter 7.

6.4. Fairness in Non-Preemptive Architecture

Before we close this section, observe that the sequence $\langle 0, 1, 2, \cdots, n-1 \rangle$ specifies the order in which PEs are upstream on the Blue Network. This sequence also specifies the PE priorities for obtaining the token. That is, if two PEs request the token simultaneously, the one that is more upstream in the Blue Network wins. In this backdrop consider the case where PEs 0 and 1, repeatedly request and release the token; here no PE i with $i > 1$ ever receives the token. While the system response time is still finite, the PE response times for $i > 1$ could be infinite. That is, the Non-Preemptive Architecture is not

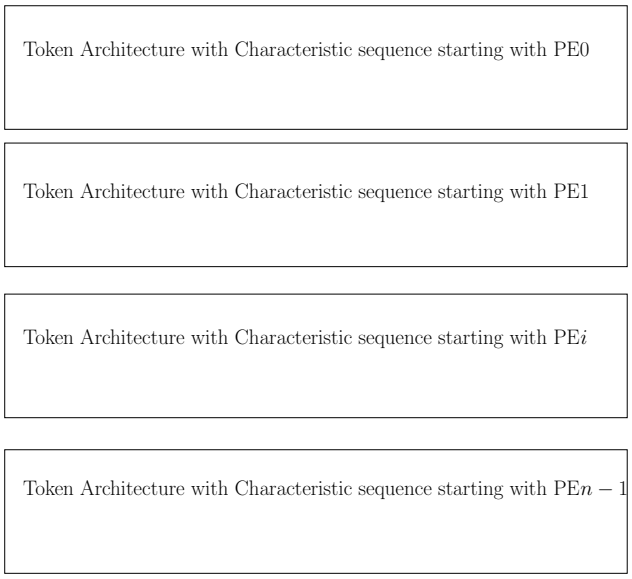

Figure 6.3. Multiple Non-Preemptive Network solution

fair.

To solve this problem, the priorities must be dynamic. The first idea that we can think about is having multiple Non-Preemptive Architecture structures, so that the competition moves between these structures, and hence, the priorities change and no PE can be starved as in Figure 6.3.

If we use such a solution, we can just modify the Algorithm as when a PE wins the competition, it lets other PEs compete in the Non-Preemptive Network structure that this former winner has the lowest priority. This guarantees fairness as no PE will have highest priority for ever, and each PE with low priority is guaranteed to have at least one step towards the light source after each token cycle, and hence, a better priority level.

However, this solution employs a large number of microrings as each PE will have an order of n microrings in the structure, which means that it will increase the cost. Also, it limits the scalability of the system as each PE has to know the number of competing

PEs and with the addition of a PE to the system, each PE will have to add two microrings and the system will have to have an extra Non-Preemptive Network structure which is not desirable.

It's also worth to mention that in the Base Architecture, the PE competing structure was that of an Optical Detector, and to solve the preemption problem, we had to modify the PE competing structure to that in Figure A.1. We still don't want to have huge modifications when moving from a structure to the next.

As we mentioned, the multiple Non-Preemptive Network idea is not feasible. Yet, we can still have this structure hypothetically without the need for all those structures using a rotating input source to the Non-Preemptive Network structure. In this solution, we will take advantage of the Optical benefits and circular structures that helps us have different start and end points for the light to and from the competing media. This work will be discussed with more details in Chapter 7, which is the main result of our work. We will show how we will solve the fairness problem with the Non-Preemptive Architecture.

Chapter 7. Fair Architecture

The requirements of a mutual exclusion algorithm enumerated in Chapter 2 include safety, safe-assignment, liveness and non-preemption, all of which are provided by the Non-Preemptive Architecture. In this chapter we introduce the *Fair Architecture* that provides fairness (in addition to the properties guaranteed by the Non-Preemptive Architecture of Chapter 6). Specifically, the Fair Architecture guarantees that while a PE waits on a token, no other PE receives the token more than once. The Non-Preemptive Architecture solved the preemption problem in the Base Architecture (Chapter 5). However, it uses a fixed set of PE priorities and as a result of which, a low priority PE can be starved; hence the Non-Preemptive Architecture is not fair (see Section 6.4).

The idea that we introduce in this chapter is that of a rotating priority, a PE with a low priority at one time will have a higher priority next time around. This is captured in the Fair Architecture, the main topic of this chapter. It implements fairness through a dynamic priority, as a result of which a PE requesting a token cannot be denied more than $n-1$ times in succession.

7.1. The Fair Network

The Fair Network is shown in Figure 7.1. The Blue Network in Figure 7.1 has an "external waveguide," B_e, into which an external laser injects unmodulated light. The Blue Network also has a "circular waveguide," B_c, that has no independent light source, but which carries light drawn from B_e. Similarly, the Red Network has external and circular waveguides R_e and R_c, respectively. In addition we have the handover waveguides, $(H_\ell\ ,\ H_r)$ that carry light from external sources in opposite directions. Recall that each

PE i interacts with the optical fabric through an interface L_i (see Figure 2.1); for the Fair Architecture, L_i consists of six microrings $m_{i,j}$, for $1 \leq j \leq 6$ (see Fig 7.1). The outputs of the Optical Detectors associated with the microrings $\{1, 2, 5, 6\}$ are denoted by $a_i, b_i, c_{i,\ell}, c_{i,r}$, respectively. These signals are outputs of interface L_i to PE i. The signals controlling the microrings $m_{i,j}$ are the inputs from PE i to the interface L_i of Fig 2.1 in Section 2.1. We will use $m_{i,j}$ to refer to both the microring j of PE i as well as the Boolean signal to control this microring. For any subset $S \subseteq \{j \ : \ 1 \leq j \leq 6\}$ we will denote the set of microrings in $\{m_{i,j} : j \in S\}$ as $m_{i,S}$.

Finally, we observe that the time thresholds of the Optical Detectors of the Red (microrings 2) and Blue (microrings 1) Networks are as in the case of the Non-Preemptive Network. Microrings $3, 4$ are Optical Switches and do not have Light Detectors and do not need time thresholds. Time thresholds for the handover network (in green) are as in the Base Network.

Each PE in the Fair Architecture uses six local microrings (numbered $1-6$ for PE i in Figure 7.2), this is in contrast with only one in the Base Architecture and just two in the Non-Preemptive Architecture. Figure 7.2 shows a detailed view of a PE and its local microrings. The Fair Architecture replaces each microring of the Non-Preemptive Architecture with two microrings, microrings $1, 3$ for microring 1, and microrings $2, 4$ for microring 2 as shown in Figure 7.2. This is to allow the PE priority (characteristic sequence) of the Non-Preemptive Network embedded in the Fair Network to be dynamic, we elaborate on this below. The last two microrings (numbered $5, 6$) are used in the Handover Network for coordination to allow this dynamic priority change.

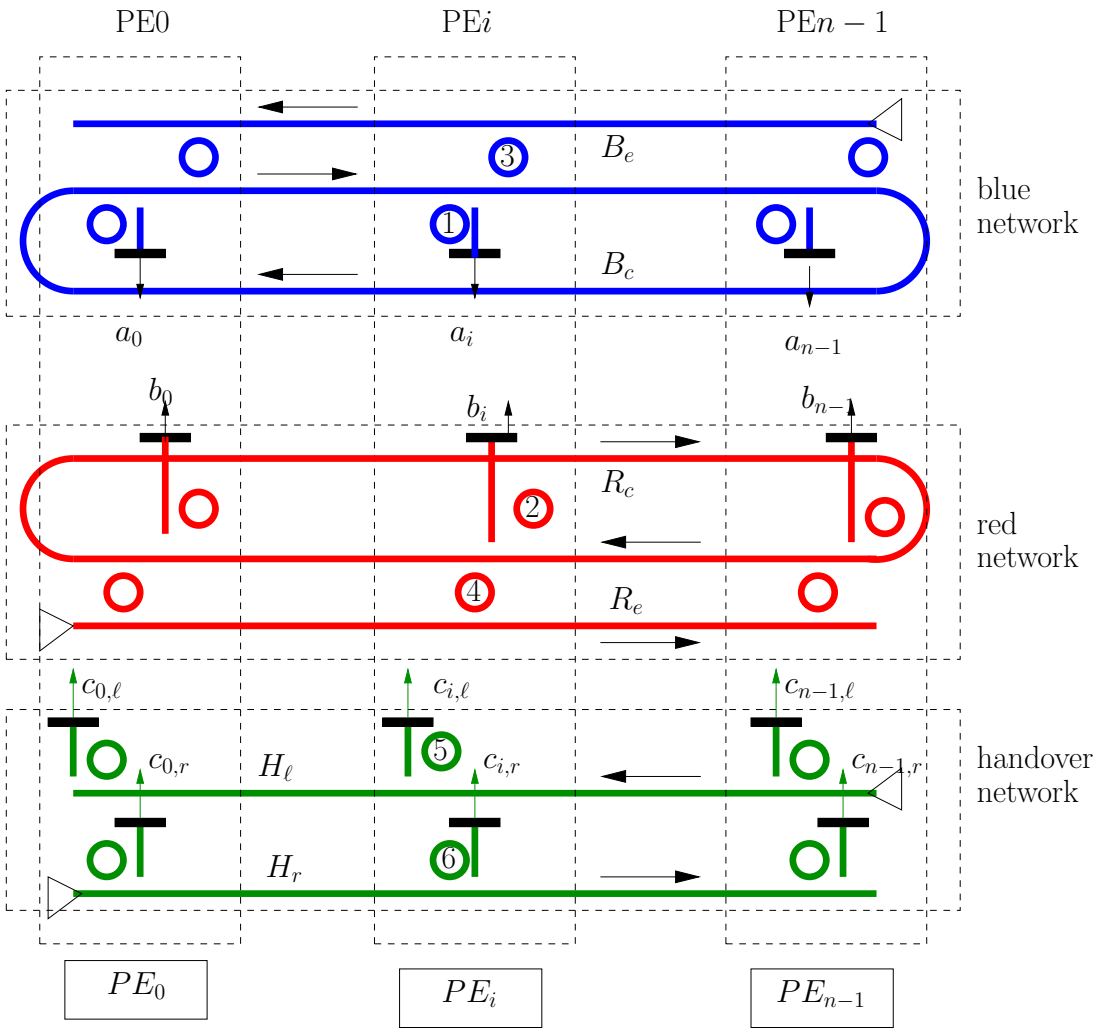

Figure 7.1. The Fair Network. As before the white triangles indicate outside source lasers and the arrows indicate the direction of light flow in waveguides.

7.2. Non-Preemptive Network Component of the Fair Architecture

As in the Non-Preemptive Network, the Fair Network has a Blue Network and a Red Network that distribute a temporary token to exactly one PE requesting it. As before, PE i in the Blue Network produces bit a_i (to activate its Red Network), which, in turn, produces a bit b_i; in the Fair Architecture, however getting $b_i = 1$ allows PE i to move further towards obtaining the token. The Fair Architecture also needs to perform additional coordination for fairness. The Red and Blue Networks of the Fair Architecture

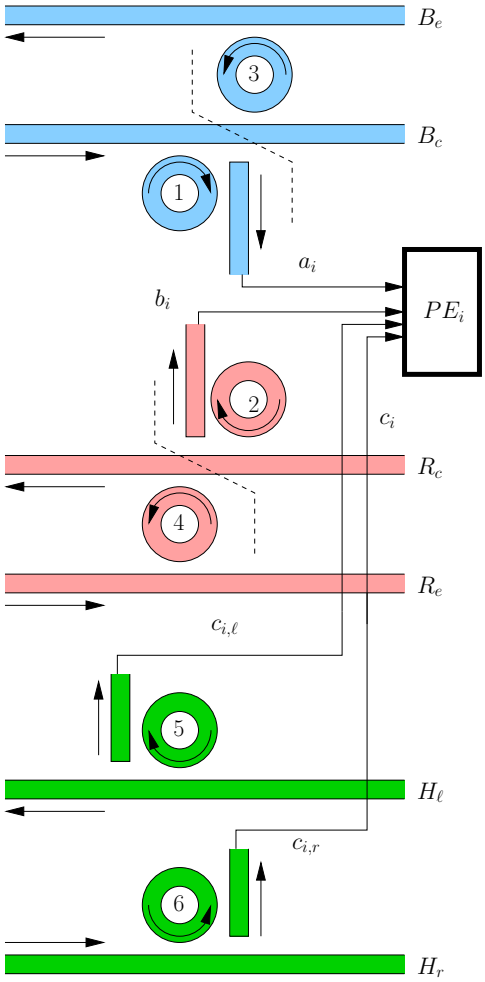

Figure 7.2. The Fair Network PE details.

are a little different from those of the Non-Preemptive Architecture in that they can alter the effective relative positions of PEs on the red and blue waveguides (relative PE priorities or Characteristic Sequence). This employs an additional coordination step when the token is released by one PE and picked up by another. The Handover Network (shown in green in Figure 7.1) is used for this purpose. Nevertheless at each competition, the logical structure of the Fair Network is the same as that of a Non-Preemptive Network. Algorithm 3 shows the procedures used in the Fair Architecture that we will elaborate on later.

In the Fair Network, microrings $m_{i,\{3,4\}}$, when activated, allow the light into the

Blue and Red circular networks. Note that if for every $0 \leq i < n$, we have $m_{i,\{3,4\}} = 0$, then circular waveguides B_c, R_c are both dark and $a_i = b_i = 0$ for each PE i. Each node also has two "sink microrings," $m_{i,1}$ and $m_{i,2}$ in B_c and R_c, respectively, that end the light path in the circular waveguides. (Microrings $m_{i,\{1,2\}}$ have two roles, one for detecting light for signals a_i, b_i and also act as sink if PE i is maintaining the shape of the Network.)

We illustrate how the Fair Architecture reconfigures itself as the Non-Preemptive Architecture of Figure 6.1. The primary difference between the two architectures is that one has linear Blue and Red waveguides, and the other has sets of linear and circular waveguides (R_e, R_c) and (B_e, B_c). Two possible configurations of these Fair Network waveguides are shown in Fig 7.3.Here, for some PE i_0, $m_{i_0,\{1,2,3,4\}} = 1$ (that is microrings $1, 2, 3, 4$ of PE i_0 are set). All other PEs $i \neq i_0$ have $m_{i,\{1,2,3,4\}} = 0$. This results in the light from the Blue (linear) external waveguide B_e entering the circular waveguide B_c right past PE i_0 going around other PEs $i_0 + 1, i_0 + 2, \cdots, 0, 1 \cdots$ in that order and returning to PE i_0. The corresponding configuration of the Red Waveguide R_c starts at PE $i_0 - 1$ and goes around to PE $i_0 + 1$ and exits at PE i_0. Figure 7.2 part (a) shows a generic PE i with $i < i_0$ and part (b) shows another case with $i > i_0$. The PE i_0 in the above illustrations is named the Anchor (it anchors the end of the Blue Network and begins the Red Network in the following PE).

Just as the Non-Preemptive Network of Figure 6.1 was characterized by the sequence $\langle k \ : \ 0 \leq k < n \rangle$, the Non-Preemptive Network induced in the Fair Network by the "Anchor" (that holds the current Non-Preemptive Network Architecture shape) PE i_0 is characterized by $\langle (i_0 + k) \bmod n \ : \ 1 \leq k \leq n \rangle$, in which PE $i_0 + 1$ has the highest priority and i_0 has the lowest. The Fair Architecture ensures that once PE i obtains the token

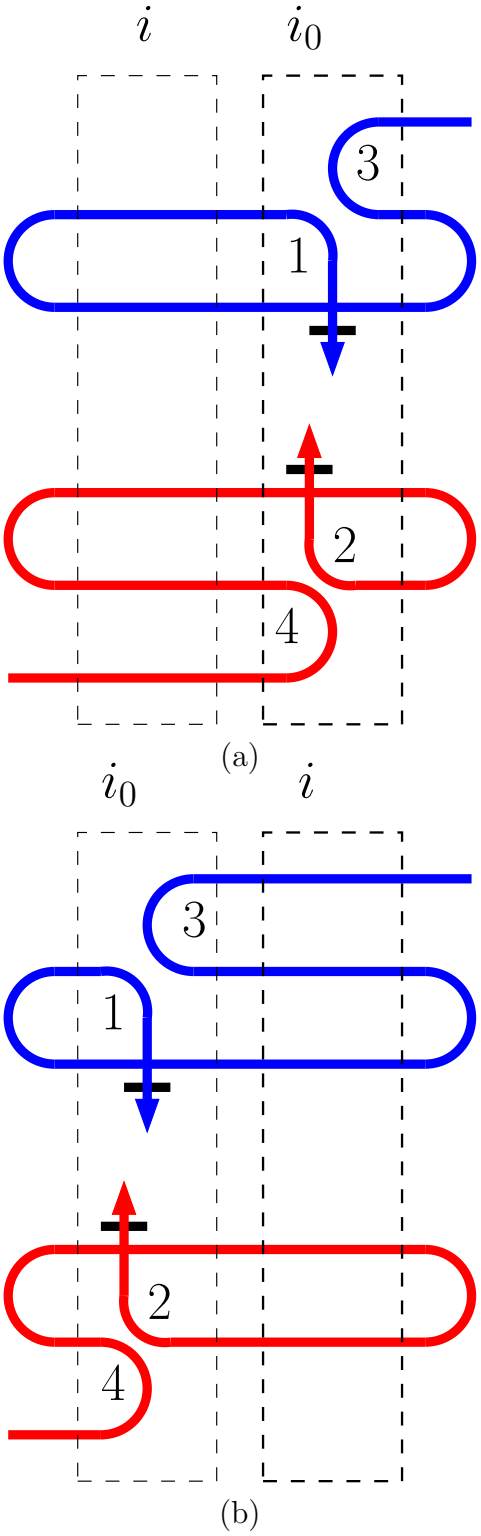

Figure 7.3. The Fair Network as a Non-Preemptive Network when light is available for competing PEs

it configures to place itself as Anchor with lowest priority for the next competition. The suite of procedures in Algorithm 3 ensure that the coordination required is properly performed.

The Handover Network is necessary for changing between current and new "Anchors".

Before we close the section, we observe that what happens during competition is still the same as in the Non-Preemptive Architecture of Chapter 6 and its induced Non-Preemptive Network of the Fair Architecture. In Figure 6.2, we showed an example where competition happened, PE 1 won the competition, and then PE 0 got interested, won the competition in the Blue Network, got the temporary token to compete in the Red Network, but found no light as PE 1 has higher priority in the Red Network. We show that this also happens in the Fair Architecture during competition. This is illustrated in Figure 7.4 for two different cases, observe that PE i_0 is the Anchor, PE i_1 has a higher priority that PE i_2; PE i_2 won the competition and then PE i_1 got interested, again won the competition in the Blue Network but couldn't win the competition in the Red Network as PE i_2 has higher priority in the Red Network. This is illustrated in two cases, in part (a), $i_0 < i_1 < i_2$ and in part (b), $i_1 < i_2 < i_0$.

The following lemma captures the above ideas.

Lemma 7.2.1 *For any $1 \leq i \leq n$, n-PE Fair Architecture with $m_{i,\{1,2,3,4\}} = 1$ and for all $j \neq i$, $m_{j,\{3,4\}} = 0$, the circular waveguides B_c, R_c and microrings $m_{i,\{1,2,3,4\}}$ behave as a Non-Preemptive Network characterized by the sequence $\langle (i+k) \bmod n \ : 1 \leq k \leq n \rangle$.*

■

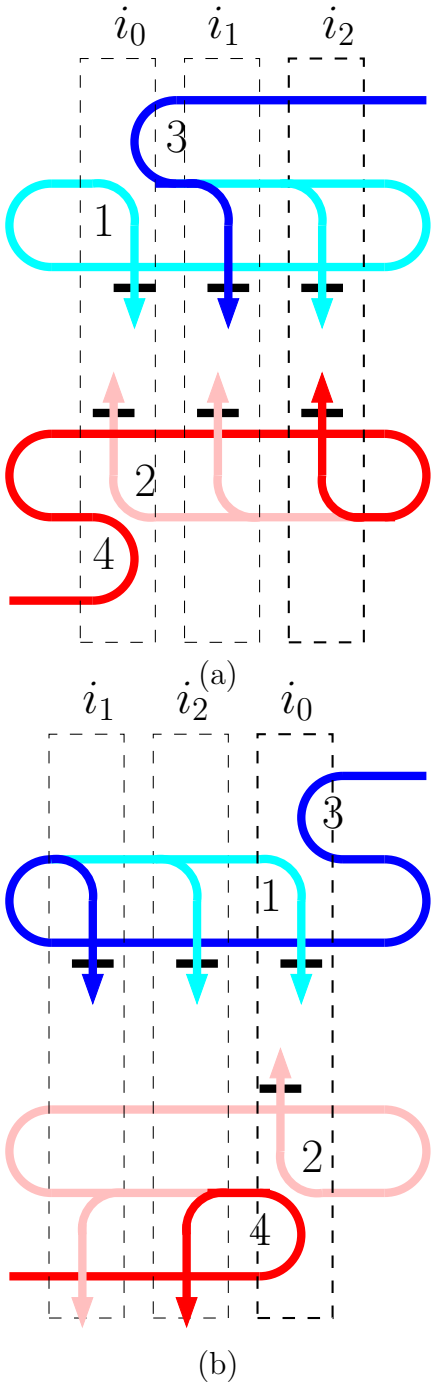

Figure 7.4. An illustration of the induced Non-Preemptive Network of the Fair Architecture. Part (a) is for $i_0 < i_1 < i_2$ and part (b) is for $i_1 < i_2 < i_0$

7.3. Fair Architecture Algorithm

Algorithm 3 lists the procedures for the Fair Architecture.

Algorithm 3: Fair Architecture Procedures `/* Compete for token */`

1 **Inital State:** for some PE i_0, $m_{i_0\{1,2,3,4,5,6\}} = 1$, $R_{i_0} = 1$, $G_{i_0} = 1$.
2 For all PE $i \neq i_0$, $m_{i\{2,3,4,5,6\}} = 0$, $G_i = 0$.
3 **Procedure** $Get\_F(m_{i,\{1,2,\cdots,6\}}, a_i, b_i, c_{i,\ell}, c_{i,r})$ `/* Get Token (for all but the anchor) */`
4 *Condition:* $R_i = 1$ and $G_i = c_{i,\ell} \cdot c_{i,r} = c_i = m_{i,\{2,3,4\}} = 0$
5 wait until $[Get\_N(i, m_{i,1}, a_i, b_i)]$ `/* Compete in current Non-Preemptive Network. Sets` $m_{i,\{1,2\}} = 1$ `*/`
6 $m_{i,\{5,6\}} \longleftarrow 1$ `/* seek handover */`
7 wait until $([c_{i,\ell} \longleftarrow (Get\_B(m_{i,5}, c_{i,\ell})$ AND $c_{i,r} \longleftarrow Get\_B(m_{i,6}, c_{i,r})] = 1)$ `/*` $c_i = 1$ `implies that handover is complete */`
8 $G_i \longleftarrow 1$ `/* token obtained */`
9 $m_{i,\{3,4\}} \longleftarrow 1$ Non-Preemptive Architecture is operational with PE i t the end of the Base Network

10 **Procedure** $Handover(m_{i,\{5,6\}}, c_{i,\ell}, c_{i,r})$ `/* Free Token and Handover to new Anchor */`
11 *Condition:* $G_i = R_i = c_i = 0$ and $m_{i,\{2,3,4\}} = 1$
12 $G_i \longleftarrow 0$
13 $m_{i,\{1,2,3,4\}} \longleftarrow 0$ `/* PE` i `removes itself from the terminal positions on the Non-Preemptive Network */`
14 $m_{i,\{5,6\}} \longleftarrow 0$ `/* PE` i `releases the handover signal */`

Before we proceed, recall that PE i sets flag $R_i = 1$ iff it is currently requesting the token and that a request is not withdrawn before the token is received, then PE i gets the token and makes use of it and when it's done, PE i sets flag $R_i = 0$. Note that for all the proposed architectures, the algorithms have no control on flag R_i. The Fair Architecture grants the token to PE i by allowing it, under algorithmic control, to set flag $G_i = 1$ (line 8 of Procedure *Get_F* of Algorithm 3) . The procedures determine the way

in which PE i controls its microrings and the network provides PE i with values of flags $a_i, b_i, c_{i,\ell}, c_{i,r}$ to guide its progress.

As in the Base Architecture and Non-Preemptive Architecture cases, a PE begins execution of a procedure iff its starting condition is satisfied. However once started, the execution continues to completion, even if the starting condition no longer holds. We also note that conditions for starting different procedures are disjoint. Hence, at any given point in time, a PE executes at most one of the procedures listed in Algorithm 3.

7.3.1. An illustration of the Fair Architecture Algorithm

We now show the idea of the Fair Architecture Algorithm through an example that we illustrate in Figures 7.5, 7.6, 7.7.

Suppose that PE i_0 currently has the token. Subsequently, there is an idle time with no competitors. Following which PE 0 wins the competition. Then PE 0 wants PE i_0 to hand over the control to PE 0. Look at this in detail.

In Figure 7.5, we show the shape of network when PE i_0 just set the flag c_{i_0} to 1, and finished line 7 of Procedure *Get_F* of Algorithm 3. At this point, notice that $m_{i_0,\{3,4\}} = 0$ (Observe conditions to Execute Procedure Get_F). It also turns out that for other PEs $i \neq i_0$, microrings $3, 4$ are unactivated, that it $m_{i,\{3,4\}} = 0$. Consequently, there is no light in the circular networks B_c, R_c. Consequently, no other PE i with $i \neq i_0$ can have $a_i = 1$ or $b_i = 1$. At this point, PE i_0 grants itself the token (line 8) and proceeds to open the Blue Network and the Red Network to competition by setting $m_{i_0,\{3,4\}} = 1$ (line 9). This is illustrated in Figure 7.6.

At this point, suppose that PE i_0 holds the token and no other PE is competing for

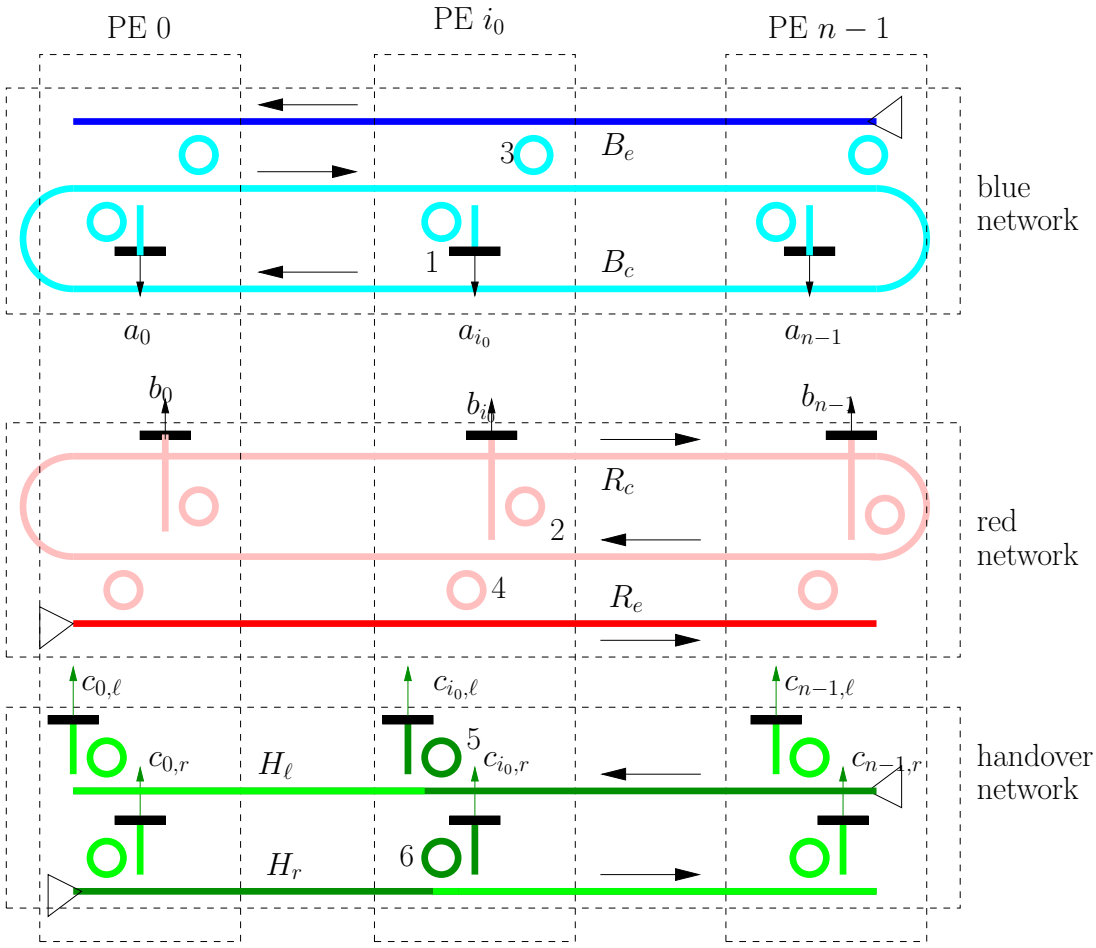

Figure 7.5. The Fair Network when PE i holds the token and has not yet allowed competition.

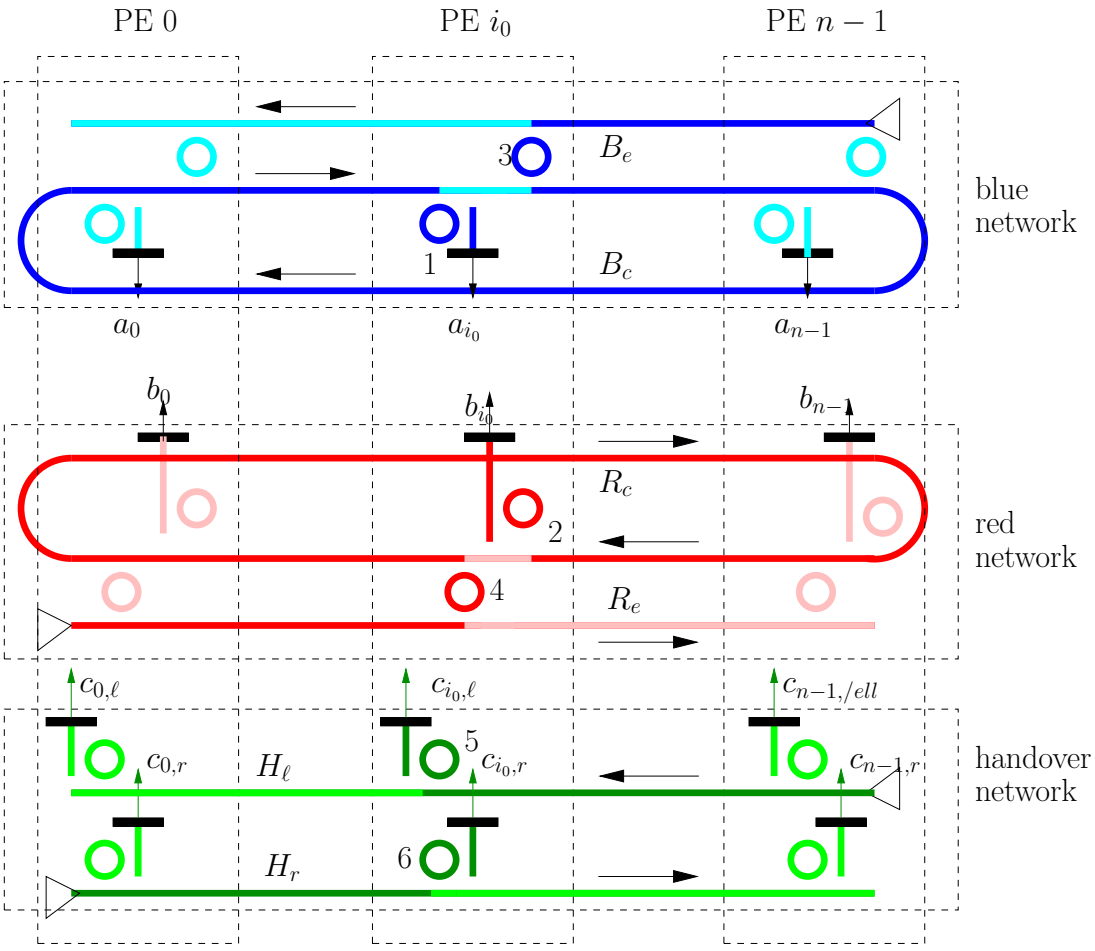

Figure 7.6. Fair Network when PE i allows competition and no PE is interested in obtaining the token.

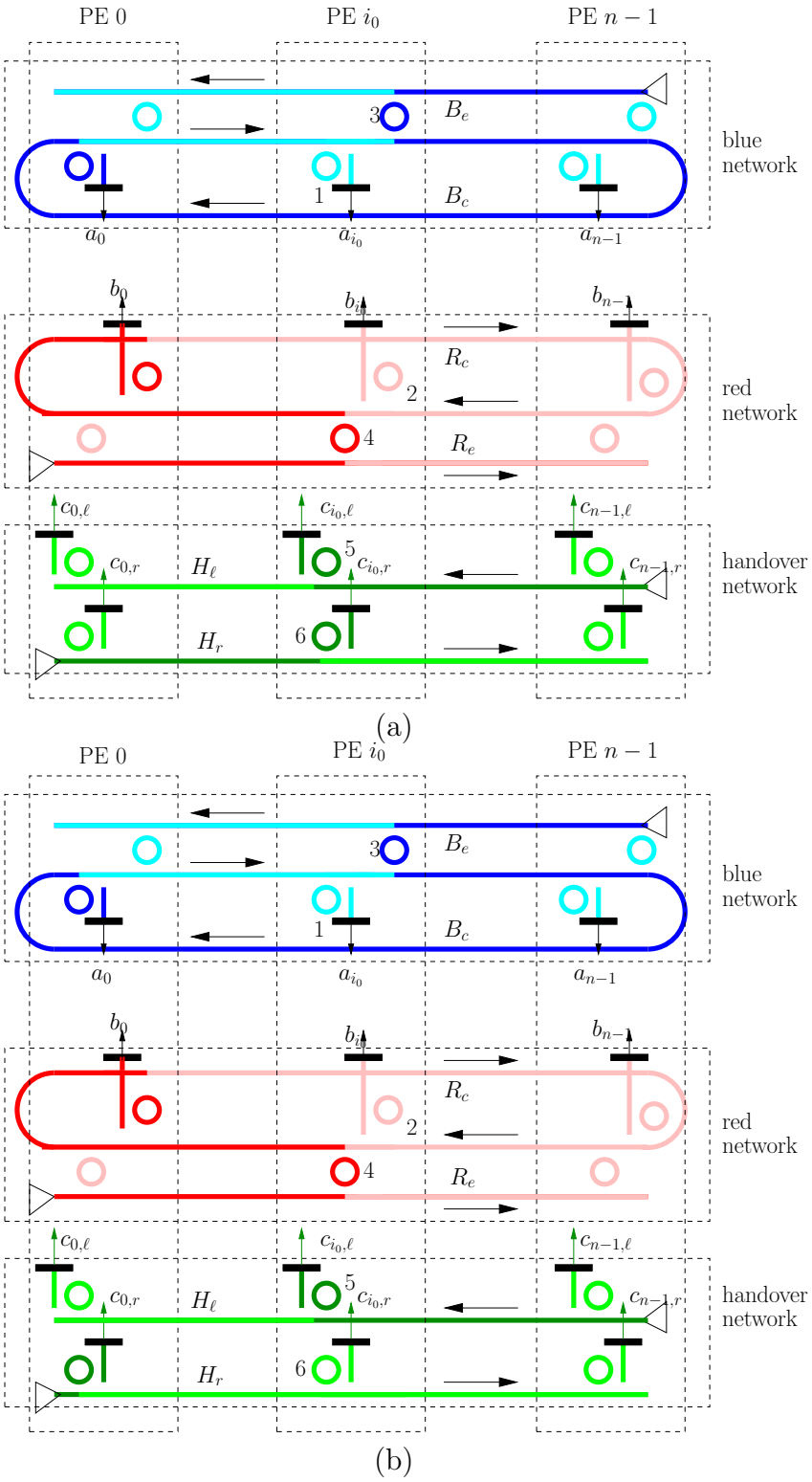

Figure 7.7. The Fair Network during the Handover of the Anchor portion from PE i_0 to PE 0 .

the token. Figure 7.6 clearly shows the induced Non-Preemptive Architecture with Characteristic Sequence $< (i_0 + j) mod\ n; 1 \leq j \leq n >$ with no PE i attempting to get flags a_i or b_i.

At a subsequent point, suppose that PE 0 enters and wins the competition for the token as in Figure 7.7 part (a), in the process executing up to line 6 of Procedure Get_F. At this point, PE 0 sets $m_{0,\{5,6\}} = 1$ as shown in Figure 7.7 part (b). Consequently $c_{0,\ell} = c_{i_0,r} = 0$ and $c_{0,r} = c_{i_0,\ell} = 1$. Hence for both PE 0 and PE i_0, $c_0 = c_{0,\ell}\text{AND}c_{0,r} = 0$ and $c_{i_0} = c_{i_0,\ell}\text{AND}c_{i_0,r} = 0$. PE 0 waits at line 7 for c_0 to become 1. PE i_0, once it has finished using the token and sets $R_{i_0} = 0$, now begins to execute Procedure Handover; initially sets $G_{i_0} = 0$ (gives up the token), and then sets $m_{i_0,\{1,2,3,4\}} = 0$, and this disconnects the light from the circular waveguides B_c, R_c as PE i_0 is not maintaining the shape of the network any more. Then at this time sets $m_{i_0,\{5,6\}} = 0$ to let another PE that in this case it's PE 0 to promote to Busy, then maintains the shape of the Network, and this happens next as this allows $c_{i_0,r} = c_{i_0,\ell} = 0$ and allows $c_{0,r}$ and $c_{0,\ell}$ to promote to 1 and consequently c_0. At this point, PE 0 will proceed to line 8 in Procedure Get_F and obtains the token, and at this point, it takes the place of PE i_0 at the start of this illustration as in Figure 7.5.

Lemma 7.3.1 *Let $c_{i,\ell}$ and $c_{i,r}$ be the Boolean outputs of the handover network. Let $c_i = c_{i,\ell}$ AND $c_{i,r}$ then at most one PE i can have flag $c_i = 1$.* ■

7.4. Fair Architecture Operation

We now describe the operation of the Fair Architecture. Figure 7.8 show the different states a competing PE in the Fair Architecture during execution of the Algorithm.

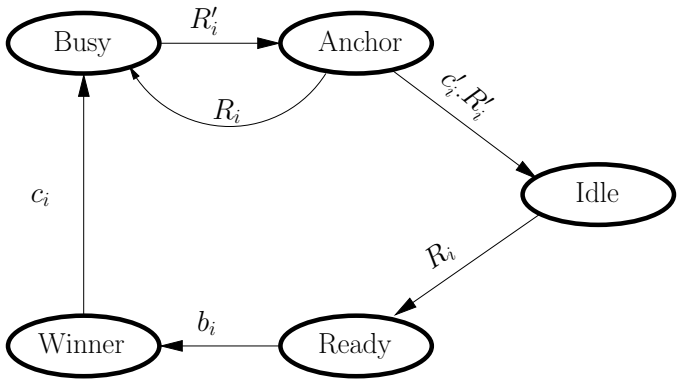

Figure 7.8. Element states in the Fair Network. Logical AND, OR and NOT are indicated by $\cdot$, $+$ and $'$. For brevity, we use the variable $c_i = c_{i,r} \cdot c_{i,\ell}$. For clarity, the state diagram does not show transitions from a state to itself.

In Table 7.1, we provide a formal definition of the states in terms of flags and PE to interface inputs.

Table 7.1. PE states in terms of flags and microrings

State	G	R	$m_{i,1}$	$m_{i,2}$	$m_{i,\{3,4\}}$	$m_{i,\{5,6\}}$
Busy	1	1	1	1	–	1
Anchor	1	0	1	1	1	1
Idle	0	–	0	0	0	0
Ready	0	1	1	–	0	0
Winner	0	1	1	1	0	1

Recall that the Fair Network can be configured to act as a Non-Preemptive Network (Lemma 7.2.1) with variable characteristic sequence. The Fair Architecture places the previous token holder as the last PE in this characteristic sequence, without changing the relative positions of the PE s. The operation of the Fair Architecture can be intuitively understood in terms of five possible states for each of the n PE s (see Figure 7.8 and table 7.1). A PE moves between states based on five flags: G_i (holds token), R_i (requests to obtain the token), a_i (output of Blue Network), b_i (output of Red Network) and c_i (handover signal). Each edge in the state diagram is labeled with a transition expression that has to be satisfied for the transition to occur; if no transition expression is sat-

isfied, then we assume that there is no state change. For example from state `Busy`, PE i moves to state `Anchor` when $R_i = 0$ (or as indicated R_i' (equals 1)). When $G_i = R_i = 1$, no transition expression out of state `Busy` is satisfied, so PE i remains in state `Busy`. In the transition expressions we have used $\cdot$, $+$ and $'$ to indicate logical AND, OR and NOT. We will show that at any given point in time, there can be at most one PE in the `Busy`, `Winner` or `Anchor` states. However, several PEs may simultaneously be in the `Idle` or `Ready` states.

There are two procedures that transition the elements through the five states (see Algorithm 3).

These procedures are stated using the same convention as in the Non-Preemptive Architecture case, the procedures are condition driven and serve to specify the sequence of actions performed by the PE; some of the actions and the sequencing between them is performed within bounded time, for example, through hardware.

Recall the example in subsection 7.3.1. In Figs. 7.5,7.6, PE i is in state Busy, other PEs are in state Idle. In the general case, one or more can be in state Ready, but just for our example, we assume they are in Idle. In Figure 7.7 parts (a) and (b), PE 0 is in state Winner. For brevity, we didn't add more figures and examples but the example shows the transition of the shape of the Fair Network and we now discuss some transition of PEs from a state to another.

We now informally describe the operation of the Fair Architecture. For correct operation, we initially assume the Fair Architecture to satisfy the following conditions: An element i_0 (say) is in state `ABusy`, holds the token, and has $G_{i_0} = R_{i_0} = 1$, $m_{i_0,\{1,2,5,6\}} = 1$ and $m_{i_0,\{3,4\}} = 0$. All other elements $i \neq i_0$, initially have $G_i = 0$ and $m_{i,\{1,2,3,4,5,6\}} = 0$ and

are in state `Idle`. Because, microrings $3, 4$ are turned off for all elements, there is no light in the circular waveguides B_c and R_c of the Blue and Red networks (see Figure 7.1). This prevents any `Ready` PE from progressing beyond line 3 in Procedure Get_F .

In initialization, for PE i_0, it starts to allow light in both the Blue and Red loops to allow the other PE s to compete.

Observe also that the only set of conditions for executing the procedures in Algorithm 3 are satisfied by PE i_0 are those for Handover, and this can happen only when it's not interested any more in the token, and none is satisfied by idle PEs. Therefore, once competition is allowed, if there exists one or more ready states, exactly one of the competing PE s will become a Winner. However, this Winner will wait until the c flag becomes a 1, and this can not happen unless PE i_0 executes procedure Handover, and this cannot happen until i_0 finishes using the token and sets $R_{i_0} = 0$.

At this point i_0 executes Procedure Handover and changes state from `Anchor` to `Idle` after the completion of the procedure Handover. When the light enters the B_c AND R_c, the PE just after i_0 has a chance to tap into it. Thus the order of elements on the blue network is $(i_0 + k)(\text{mod } n)$, for $k = 1, 2, \cdots, n$. Similarly, i_0 is first on the red network whose order is $(i_0 - k)(\text{mod} n)$ for $k = 0, 1, \cdots, n-1$. Thus, we have a Non-Preemptive Architecture structure, and when PEs execute the procedure Get_Non-Preemptive, it's a Non-Preemptive Architecture with the anchor i_0 with lowest priority. In this way, the blue light passes by $(i_0 + 1)(\text{mod} n)$, then $(i_0 + 2)(\text{mod} n)$ .... and so on till $n - 1$ and then 0 till i_0 and the other way around in the red side.

If there are any `Ready` elements executing Procedure Get_F and waiting at line 5, then (by Theorem 6.3.3) exactly one of these (say i_1) receives $b_{i_1} = 1$ and moves on

to the `Winner` state, waiting at line 7 for c_{i_1} to become 1. Because i_1 has activated microrings $5, 6$, the current Busy PE i_0, when becomes not interested in Token any more, PE i_0 finds that $c_{i_0} = 0$ (notice that if two different PEs i_0 and i_1 each activates microrings $5, 6$, they both have $c_{i_0} = c_{i_1} = 0$). With $c_{i_0} = 0$, the Busy PE i_0 executes Procedure Handover(i_0) that first releases the token, and then removes the anchor from its position at the end of the logical Non-Preemptive Network, and then releases the handover signal (line 14) that disconnects PE i_0 from the handover waveguides and allows c_{i_1} to become 1; element i_0 now moves to the `idle` state. This allows a winner (possibly PE i_1) to complete line 7 of Get_Fair, obtain the token (line 8) and move to the `busy` state.

If the busy PE i at some point in time loses interest in token ($R_{i)} = 0$), and conditions for Handover are met, if there was a winner, then $c_i = 0$ for both the winner and the Busy, and will be forced to handover for flag R it will keep executing Procedure HandOver till the end even if it gains back interest as we mentioned before that once a PE starts execution of a procedure, it completes it till the end even if during execution, conditions change and are not met any more. On the other hand, if a PE in state Anchor meaning he was busy and lost interest in the token, and then gained interest back before conditions for Procedure Handover were met, meaning there is no winner, PE gets back to the Busy state as flag G is still a 1 and flag R is back to 1.

Hence, if Handover is completed, the next PE that will allow competition moves at least one step.

In this way, if a competing node i is far from the current anchor, the Fair Architecture ensures that the next anchor will come closer to i by at least by 1, thereby increasing its priority for the next competition.

The main results of this section are the following theorems. The Fair Network performs mutual exclusion with safety, safe assignment, liveness, non–preemption and starvation-freedom. ■

The safety, safe assignment, liveness and non–preemption are inherited from the Non-Preemptive Architecture properties. The starvation–freedom is gained from the rotation of the relative position of the elements in the Fair Network, meaning that the anchor comes closer to competing nodes in each token cycle.

Lemma 7.4.1 *For any competing node PE i, if the Anchor is PE i_0, then the most number of waiting cycles for PE i is $(i - i_0) mod\ n$.* ■

The above lemma comes from the fact that after each competition, the Anchor moves at least one step towards any competing PE.

Theorem 7.4.2 *A PE that is seeking the token receives it within $n - 1$ token cycles.* ■

In the above theorem, a token cycle is the time encompassing the competition for the token and its use, again this is because of the rotation of the nodes relative priorities.

Theorem 7.4.3 *Assuming constant propagation delay for light in the chip (or all waveguides have constant optical length), the response time of the Fair Network is at most $O(n)$ for a single competition, and constant per competition, when amortized over a large number of competitions.* ■

This is inherited from properties shown in the Base and Non-Preemptive Networks.

With these set of theorems, we can start proving that our main work, the Fair Architecture, provides safety, liveness, safe assignment, nonpreemption and guarantee that a requesting node does not wait for more than $n - 1$ token cycles before being granted the token, which provides fairness. A detailed proof will be provided in the next section.

7.5. Proof of Correctness of the Fair Architecture

Now, we show that our work provides all Mutex requirements. We will show that the initialization of the system, together with the procedures can guarantee all requirements mentioned in Chapter 2, Safety, safe assignment, liveness, nonpreemption, starvation freedom.

One of the key requirements for proper operation is the initial conditions, that exactly one PE is in state Busy or Anchor, and other PEs are in state ready or idle.

First requirement is safety. Safety is guaranteed as for a PE i to hold the token, the only statement that has $G_i \longleftarrow 1$ exists in Procedure Get_F, and this one means that $c_i = 1$ and this means that the former busy PE deactivated his mircorings $5, 6$ and this happens only in Procedure Handover, and before this statement is executed, another statement $G \longleftarrow 0$ is executed.

Lemma 7.5.1 *The execution of procedures Get_Fair and Handover guarantees that if one PE i such that $G_i = 1$, then for all $j \neq i$ then $G_j = 0$.*

Proof: We need to show that no two PEs can have the token or flag G active at the same time:

The statement $G_i \longleftarrow 1$ exists only in Procedure Get_Fair in line 8, and this implies that $c_i = 1$ (line 7) which can happen only for at most one PE as indicates Lemma 7.3.1. Also, for another PE i_0 if it was holding the token, then to let another PE i have $c_i = 1$ it has to reset its microrings $5, 6$ which is in line 14 of Procedure Handover and this means it executed line 12, $G_i \longleftarrow 0$.

In other words, the statement of token assignment implies the activation of the c

flag, which cannot be active for more than one. The other case, that a PE had the token, it has to give it away before it lets another PE get the c flag. ■

Another proof: Let's also prove safety by contradiction:

Suppose we have two distinct PEs, PE i and PE j such that $i \neq j$ have the token assigned, in other words, suppose $G_i = G_j = 1$ at some point in time.

Now let's see what this means in our Algorithm: This means that PE i executed Procedure Get_F and just executed line 8. This implies that $c_i = 1$. Also, in the same manner, we will reach that $c_j = 1$. But according to Lemma 7.3.1, this cannot happen.

Hence, there is a contradiction.

■

Second requirement is safe assignment. Safe assignment is guaranteed as for a PE to compete for the token, it has to have flag R active as a condition if Procedure Get_F. Hence, only a requesting PE can get access to the token.

Third requirement is liveness. We will show that conditions and statements in procedures will guarantee that one PE wins the competition and promotes to Busy state. This is guaranteed as a PE in busy state, after it is no longer interested in the token, conditions in procedure Handover will guarantee that it handovers the token and lets a winner promote to state Busy. Liveness of the system happens because of conditions on the procedures, we can add that a busy PE i is assumed to have an interrupt driven action when $R_i = 0$ so that in case another PE j has won the competition and activated $m_{j,\{5,6\}} = 1$ as a means to signal PE i that PE j is waiting for handover, PE i executes procedure Handover and lets PE j promote to state Busy. The existence of a winner in

case of the existence of at least one PE requesting the token is guaranteed by the properties of the Non-Preemptive Architecture and the mapping of the Fair Architecture to the Non-Preemptive Architecture (see Section 7.2).

NonPreemption is guaranteed through the conditions of the procedure Handover, as it requires a PE i in state Busy to lose interest in the token ($R_i = 0$) to change the value of flag ($G_i = 0$).

Now let's start the proof of starvation freedom :

As mentioned earlier, at any instant of competition, there is a Characteristic sequence that determines the current PE with the highest priority, and as we go up with numbers, the priority is decreased. In other words, the priority of a PE is a function of the absolute value of the difference in number between the index of this PE and the index of the highest priority PE (of course *mod* n).

Yet, when a PE i wins a competition, and promotes to busy, it puts itself at the end of the sequence, and the new Characteristic sequence starts with PE $i+1$. This means that the priority set is changed, as the addition of 1 indicates that the number at which the sequence starts has a better priority, or in other words, the general case is that the higher the number is, the lower the priority is, and with addition of a 1 to the starting PE in the sequence, this means that the lowest priority PE in a competition is going at least a step towards the starting PE in the sequence.

We first define Z as a count of the number of times a PE gets the token assignment.

Theorem 7.5.2 *Variable Z can only be increased by* 1.

Proof:

In the set of Procedures in Algorithm 3, the statement $G_i \longleftarrow 1$ exists only in the Procedure Get_F.

For the initial condition, at $t = 0$, PE i_0 gets $G_{i_0} = 1$ and all other PE s, for $i \neq i_0$, the initial conditions have $G_i = 0$.

Hence, in initialization, it's just one.

For $t > 0$, this can only happen in Procedure Get_F. In the Procedure Get_F, the statement exists in line 8, and this can happen only if $c_{i,r} = c_{i,\ell} = 1$ in line 7 which can happen only for at most one PE as shown in Theorem 7.3.1, and this also implies that $b_i = 1$ which can also happen for at most one PE as in reference to Theorem 6.3.1 and Lemma 7.2.1.

Hence, for $t > 0$, the set of PE s can have at most one PE get the token assignment.

We now define t_v for v is an integer, as the time at which $Z = v - aatt_v - \delta$ and at $t = t_v$, $Z = v$

Recall first that we noted that transition of a PE from a state to another that has effect on other PE s is mainly done by the network interface flags a, b, c, and a state is defined in terms of its own flags and activated microrings.

Proof by induction:

In our proof, we will mainly rely on proving validity of our work based on induction on Z, and the idea is that we have control for initialization, and the designed procedures can direct the PE s and also the system from a state to another.

Statement: For any value of Z=v, at $t = t_v$ exactly one PE i (say) has $G_i = 1$ and has microrings $m_{i,/1,2,5,6/} = 1$, and all other PE s have microrings $3, 4, 5, 6$ inactive, and for

$t_v \leq t < t_{v+1}$ at most one PE i say can have $G_i = 1$ and microrings $3, 4$ activated during the interval.

For $Z = 0$: at t_0, this is the initial state, and $t_0 = 0$ and this is according to Algorithm 3, initial conditions guarantees that base statement is true at t_0. Now, let's see the rest of the interval:

Interval proof: at $t = 0$, only PE i_0 has $m_{i,/1,2,3,4,5,6/} = 1$ and also $G_{i_0} = 1$ and all other PE s have all the microrings deactivated. Now, let's see what happens during the interval:

The first case is trivial, when all PE s are not interested in the token, and in this case for $i \neq i_0$, $R_i = 0$ and conditions for all procedures will be false and either $R_{i_0} = 1$ and in this case PE i_0 keeps the token, and at some time $R_{i_0} = 0$ and still nothing happens.

The second case is some PE is interested in the token at this or some time, for any PE i where $i \neq i_0$, the only Procedure that it may be allowed to execute is Get_F, as $R_i = 1$, and hence, competition can take place as there exists some PE that is interested in the token. Hence, a PE j say will win the competition and have flag $b_j = 1$ and cannot be preempted according to Theorems in section 6 and Lemma 7.2.1. Hence, PE j will proceed in the execution of Procedure Get_F to line 6 that activates microrings $5, 6$ setting $c_{i_0} = 0$ and by some means signals the busy PE i_0 that a Winner exists, waiting for PE i_0 to finish using the token and have flag $R_{i_0} = 0$ and hence, conditions for Procedure Handover become true for PE i_0 and execution starts.

Before checking what will happen, let's recall that till this time, only PE i_0 has flag $G_{i_0} = 1$ and has microrings $3, 4$ active, meaning that there is only one PE i_0 that has the

token and holds the shape of the network and priorities. All other PE s can have either at most microrings $1, 2$ active, or PE j has microrings $1, 2, 5, 6$ active.

Now, let's go step by step with the execution of PE i_0 of Procedure Handover. First statement is in line 12, and PE i_0 resets flag G, still this is its own flag and doesn't affect other PE s' flags. But this can keep the system safe when another PE sets its G flag. Now, let's check what will happen in the next statement in line 13 where PE i_0 resets microrings $1, 2, 3, 4$ and thus, both loop waveguides B_c and R_c become dark, this can make flags a and b for other PE s a 0. This can only affect a competing PE that hasn't reached yet line 5 in Procedure Get_F, but for PE j, it's still waiting for flag c_j to be a 1. Now, let's proceed to the final statement executed by PE i_0 in line 14 in Algorithm 3, PE i_0 deactivates microrings $5, 6$; and this will let the only other PE j that has $m_{j,\{5,6\}} = 1$ to get $c_j = 1$ and hence, PE j proceeds to line 8 and gets the token; and here $t = t_1$. The last statement in line 9 will activate microrings $3, 4$ for PE j. This will be done in the following interval, but this shows that only one PE can have microrings $3, 4$ active, and we reach a situation very similar to those at $t = t_0$, the only difference is that some PE s may have microrings $1, 2$ active.

Note that interval proof is very similar for all intervals.

Note that for a general case, meaning for $t = t_v$ for any integer $v \neq 0$, only microrings $1, 2$ can be active for any other PE other than the one in state Busy.

Still with the initial conditions above, any PE having microrings $1, 2$ active cannot proceed beyond line 5 in Procedure Get_F and will have to wait for the busy PE to execute Procedure Handover. The rest of the proof is just the same.

In other words: we will just go now to the induction step and show it's just the

same as the base step.

For $Z \geq 0$: Assume statement is true for $Z = k$, we need to prove that the statement is true for the interval $t_k \leq t < t_{k+1}$ and t_{k+1} at $Z = k + 1$.

Proof: now, we assumed that at $t = t_k$, there exists one Busy PE that has flag G active, and has microrings $1, 2, 5, 6$ active, and all other PE s have microrings $3, 4, 5, 6$ inactive.

Now, let's see what happens in the interval:

$t = t_k$ implies that statement $G_i \longleftarrow 1$ was executed by PE i for some i.

This can happen only in Procedure Get_F in line 8, and this means that the next statement in line 9 will be executed where PE i activates microrings $3, 4$ and allows competition for the token for the other PE s that can have at most microrings $1, 2$ active.

The interval proof is the same as in the base.

We can show that the worst case for the synchronization delay is: $\mathrm{T}_{instruction}$ for line 9 in Algorithm 3 executed by the PE in state Busy + Tcompetition(which is the same as in the Non-Preemptive Architecture case) + $\mathrm{T}_{instruction}$ for line 6 in Algorithm 3 + executed by the Winner + $3 * T_{instruction}$ for the Anchor to execute the Procedure *Handover* + $\mathrm{T}_{instruction}$ for line 8 in Algorithm 3 + executed by the Winner

Instructions can be grouped to either executed in constant times or can be executed in $O(n)$ time in worst case situations like the competition in Non-Preemptive Architecture structures. In other words, it can be shown that generally, the worst case for PE and system response times can not be more than $O(n)$ times.

Chapter 8. Multiple Tokens Problem

In Chapters 5, 6, 7, we introduced our solution for the Mutual Exclusion problem with a group of n PEs competing for a shared resource, we introduce a token based solution where at most one PE can have access to the shared resource through holding a unique token. In this chapter, we will extend the concept to solve a more general problem, we have a pool of n competing PEs, that compete for a pool of k tokens in a pool of competing media. This is the case when there is a server that may have for example multiple ports, and the competing set of PEs may request access to that set of ports so that each PE can just setup a connection with this server through one of those ports. The set of shared resources will be dealt with again as tokens. These tokens can be either indistinguishable, distinguishable, or a mix.

Problem Definition: To precisely define the problem, we consider the case of having a set of parameters:

- n PE s, competing for k tokens
- k tokens that can be partitioned into q disjoint subsets $Tokens_i$ for $(1 \leq i \leq q \leq k)$ such that each $Tokens_i$ has $k_i \geq 1$ elements. $Tokens_i$ is a set of k_i indistinguishable tokens (all tokens of $Tokens_i$ are identical). However, for $i_1 \neq i_2$, tokens of $Tokens_{i_1}$ can be distinguished from tokens of $Tokens_{i_2}$.

In the multi tokens problem, a solution needs to satisfy the same requirements mentioned before in Chapter2.

In Chapter 7, we solved the Mutual Exclusion problem through a network that

has two loops and a couple of flat networks. In this chapter, we will refer to the couple of loops as competing medium, and the couple of linear networks named the Handover network.

For solving the new problem, we will employ a set of competing networks and another set of handover networks. We will show with more details the roles of each of them.

We also have another Definition for the items we will have in our solution:

Solution Parameters: The solution we propose will not be a big deviation from the Fair Architecture we proposed in Chapter 7. However, we will have some modifications to suit the multi token problem.

- C sets of 2 loops as competing media for the PE s to compete for tokens
- H number of flat network sets for Holding a token, or the network shape

Each PE competing for a token may select a block $Tokens_c$ of tokens to compete for, $Multi\_Token(n, k_1, k_2, \cdots, k_q)$. There are special cases like for example $q = 1$ when all the tokens are identical, and also $q = k$, meaning that all the tokens are distinguishable. These two special cases will be discussed in detail as we will see in this chapter.

8.1. Introduction

In solving the multiple token problem, a solution has to satisfy the same requirements as in single token problem; Safety, Liveness, Safe_Assignment, Non_Preemption and Fairness. We build on the results we achieved in Chapter 7 so that we have such properties and start working on getting better results. We will consider the competing envi-

ronment as separated from the Hand_Over environment. As showin in Figure 8.1, we will have two different sets, one for competition having the blue and red loops, and one for control having the green flat networks. There is coordination between the two sets of networks, but we will show that they will be dealt with in a different manner.

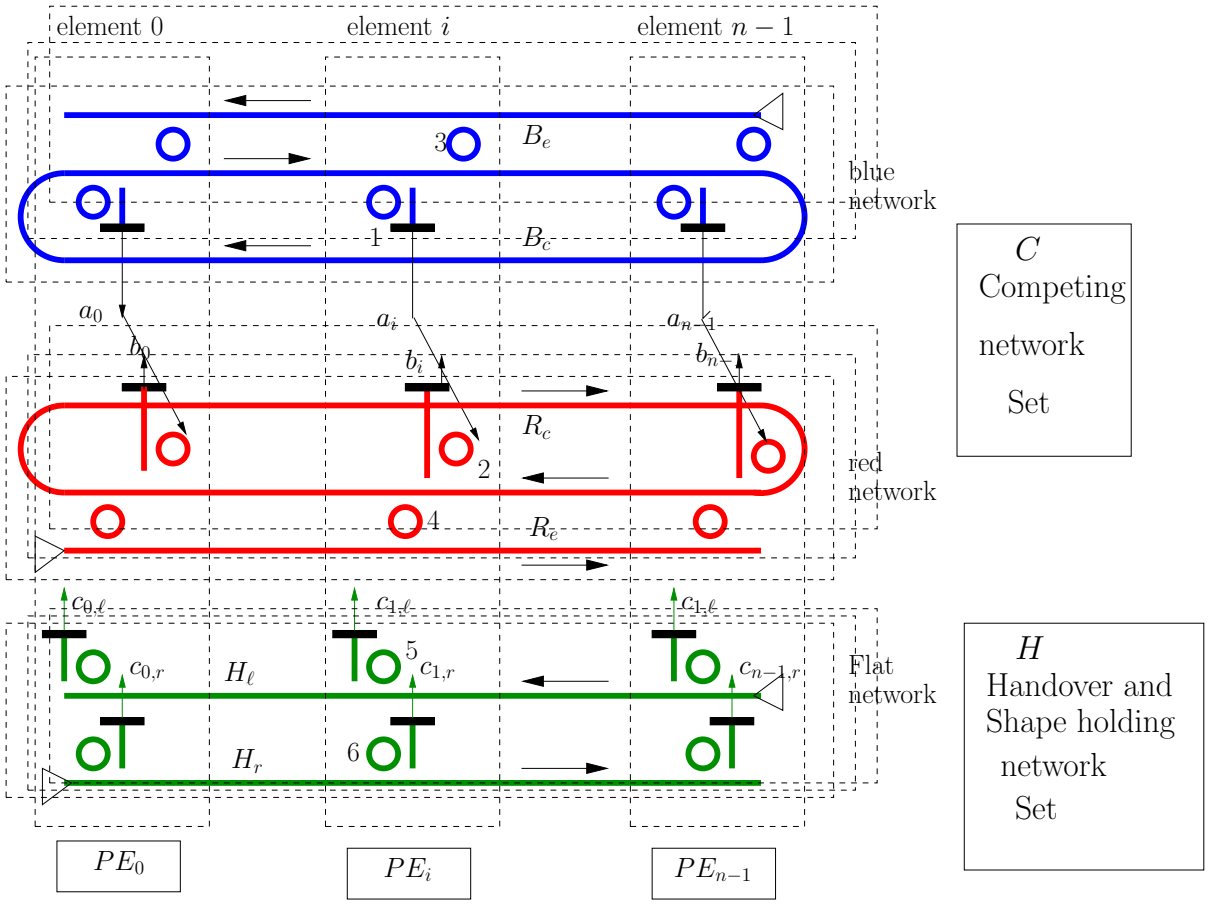

Figure 8.1. The Fair Network as a general Token Problem Solution network: Note that we left general numbers C and H for the competing loops and flat networks respectively.

In the studied case in Chapter 7, n PE s are competing for just 1 token, meaning that the set $T(i, i_k)$ has only one element, with value 1, and also C and H have values of 1. In this case, the flat network is used for both holding the token and holding the shape of the network as mentioned in Chapter 7 as the Fair Architectureis considered a rotating Non-Preemptive Network with a Characteristic sequence, and hence, there always has

to exist a PE that holds the start of the characteristic sequence. Next we will discuss the case for general values for $T(i, i_k)$, C and H.

In the next case, where $i = 1$, and it'll be just one set of k identical tokens. In such a case, we will choose particularly one of the flat networks to hold both a token and the network shape for one of the competing loop sets. We can solve the problem with general C, but in the first Algorithm presented, we will present the solution for $C = 1$ in Algorithms 4,5 .

The following presented in this Chapter will be the case when $T(i, i_k)$ has k distinct sets of tokens with each set has only 1 token. In this case, we cannot just have a flat network that can hold both a token and the network shape, there has to be one flat for each one of the competing loop sets. We will present a simple solution for $C = 1$.

One simple way is to have flat Handover networks as the same number of tokens, meaning that for if K is the total sum for all sets of tokens, then $H = K$.

We now discuss the idea. Once the PE wins, if he finds a free token, it can just grab it and let other PEs compete.

For the simple set of one set of identical token, the presented idea is having the top flat network holds the shape of the competing network and can hold a token too, and it's used for handover if there exists one or more free tokens, and in case a PE wins a token, it goes to the top, and lets other PEs compete, and lets the winner have the token once it gets hold of another token released by some other PE. This happens as all PEs holding any token keeps looking for the next one so that it frees its place to the upper PE, meaning with lower order.

Another approach is that a winning PE will have to check a large number of flat

networks, to find which one is free and once it finds a free token, the idea is the same, but we just don't want to burden the PE s as much as we can.

In the other case, where the tokens are distinct, a PE seeking a token will just go to the desired flat Hand_Over network, which is simpler, but the harder part is that it can either make other PE s looking for other tokens wait or perhaps will rely on another way to be the next PE for this particular token. We will discuss the solutions in more details in the next sections.

8.2. Indistinguishable Tokens Problem Solution

In this section, we will discuss the case if there are multi tokens and they are all indistinct, that can be the case if there are multiple shared resources and they are all identical. In other words, we have n PE s, $i = 1$, k identical tokens, and k flat networks.

To make it easier, we define a new variable named Max which represents the number of tokens that are allowed in the system. In a single token system, $Max = 1$, and in our case, $Max = k$.

We note that there will be multiple busy PE s holding tokens. Recall that in the first case that we will discuss now, there is just one set of loop networks for competition to take place, there has to be only one anchor, which in our case will be the last one who won a token, and in this case, this PE will allow the other PE s to compete for a token until there is a signalling between the two letting them know there is a winner and if the last finds a token, he just hands over the token shape and let him hold a token. In the next paragraph, we will discuss the first solution for this problem, where the very first flat network will do the job, and each PE holding a token will always try to check the next if

it's free so that the upper one figures out it's free.

In this case, we will also define a new variable for each PE i named Held(i) for the order at which PE i knows which token it's holding. Also, as there are more than one flat network for holding the tokens, we add an extra label for the microrings which PE i is activating as there are multiple microrings $5, 6$. The only disadvantage of this approach is that it may cause a bit more delay as in case the last Hand_Over network is releases, a winner will have to wait till all the other PE s holding a token keep moving till the end until the former winner finds the next fee to let him take Hand_Over network 0. However, it's a bit simpler and a PE doesn't have to activate and deactivate so many microrings and check again a lot of signals.

There is another approach that we could employ as instead of relying on the Hand_Over network 0, we can just make it easier for us as to make it for any Hand_Over network, and the PE s don't have to look for the higher order Hand_Over network to move the held token, but in this case, a competing PE will have to activate all its Hand_Over microrings and check if any of these networks is free and just pick one up. The disadvantage of this approach is that it'll make a PE, specially winning one has to activate and deactivate a larger number of rings (k), and check a larger of signals($2 * k$), and execute more instructions, but it will be a bit faster.

In all cases, we will have to have a different set of definitions for the PE states, as there are multiple Busy, but just one `Busy_Anchor`, `Ready` and `Idle` will be the same.

Analysis In the presented solution, we break the solution to two steps ,a PE works exactly as in the single token fair solution in the sense that it just competes for a particu-

Algorithm 4: Get_One_Of_Multi_Tokens_Part_1 `/* Compete for one among multi Indistinguishable tokens Part1 */`

```
1  Procedure Get_Multiple(m_{i,1}, a_i, b_i)  /* i */
2      Get Token (for all but the anchor)
3  Condition: G_i = 0 and c_i = 0 and R_i = 1 and m_{i,{3,4}} = 0
4  wait until [b_i ⟵ Get_N(i, m_{i,1}, a_i, b_i) = 1]   /* Compete in current
       Non-Preemptive Network                                          */
5  m_{i,{5,6},0} ⟵ 1   /* seek handover                                 */
6  wait until [c_{i,0} ⟵ (Get_B(m_{i,5}, c_{i,ℓ,0}) AND Get_B(m_{i,6}, c_{i,r,0})] = 1) /* c_{i,0} = 1
       implies that handover is complete                               */
7  G_i ⟵ 1   /* token obtained                                          */
8  held(i) ⟵ 0
9  m_{i,{3,4}} ⟵ 1

10 Procedure Move_Holder(Movedownthetokenheld)  /* i */
11     Condition:  G_i = 1
12       /* PE i moves down and lets other PE s move down              */
13     ℓ ⟵ held(i)
14     if ℓ < Max − 1 then
15         m_{i,{5,6},ℓ+1} ⟵ 1;
16         if c_{i,ℓ+1} = 1 then
               /* The previous line allows light into the original token
                  held so that another PE can move to it               */
17             held(i) ⟵ ℓ + 1
18         if ℓ ≠ 0 then
19             m_{i,{5,6},ℓ} ⟵ 0;
```

lar token, and when it wins, it just goes to the Hand_Over network, waits till the busy releases this token. In this way, it's just the same. What we really add in our solution is that the winner looks for other PE s that have won before if any of them has released his token, and to make it simpler, each one looks for the next token, and if any token $j \neq 0$ is released, the PE holding token $j - 1$ takes it and allows the one on top to take it and so on. Till token 0 is released and the Busy_Anchor lets the winner promote to Busy_Anchor.

Algorithm 5: Get_One_Of_Multi_Tokens_Part_2 `/* Compete for one among multi Indistinguishable tokens Part2 */`

```
1  Procedure Release_Multi(Release)  /* i */
2      Condition:  R_i = 0 AND G_i = 1 AND m_{i,{3,4}} = 0
3        /* PE i releases the token held for Busy and not Busy_Anchor */
4      G_i ⟵ 0
5      ℓ ⟵ held(i)
6      m_{i,{5,6},ℓ} ⟵ 0;
7  Procedure Handover(i)  /* Handover to new Anchor */
8      Condition: held(i) ≠ 0 AND c_{i,0} = 0 and m_{i,{3,4}} = 1 OR R_i = 0 AND
       G_i = 1 AND held(i) = 0 AND c_{i,0} = 0 AND m_{i,{3,4}} = 1
9      if held(i) = 0 then
10         G_i ⟵ 0
11     m_{i,{1,2,3,4}} ⟵ 0  /* PE i removes itself from the terminal
           positions on the Non-Preemptive Network                    */
12     m_{i,{5,6},0} ⟵ 0  /* PE i releases the handover signal        */
```

proof of correctness The correctness here is not very different from the Fair in the sense that the Process is in two steps, the first is the same as the Fair single token architecture, as in reality the PE s compete for token 0, with the everything being the same.

We first note that an Anchor is maintaining the shape of the network has microrings $3, 4$ active, and has also microrings $5, 6$ active till another winner shows up and takes the leadership and holds the shape of the network, provided of course that either this busy moved forward because some other busy gave his token, or it's just not interested any more although the other busy PE s didn't give any of their tokens.

Now, we need to prove that for any PE (i) holding the token, either the Busy_Anchor or just any other Busy will move on. In other words, there will always be competition once a Busy releases the token:

The first condition, if a non Anchor Busy releases the token, a non anchor im-

plies that it's not a PE that holds the shape of the network, meaning that a PE j is non Busy_Anchor and Busy if $G_j = 1$ and $m_{j,\{3,4\}} = 0$, this means that when it became busy, it's allowed to do two procedures, either Move or Allow, and if it does Move, it can keep doing it until it has the very last

Now, the Busy_Anchor will release the token in one of two cases: Either it already holds a token somewhere other than token 0, and not interested any more, and no winner has shown up yet, so it will wait until some winner comes and takes the leadership , and will wait as the shape of the network has to be maintained by Exactly One PE. At some point in time, a Winner will show up and activates microrings $5, 6$ in Hand_Over network 0, and in this case, the busy will give the shape of the network and leaves the winner to be the busy anchor, then conditions for Release will be met, and the former busy anchor will give up the token and the access to the Hand_Over network it used to hold. The other case is that it's still holding token 0 and this means that it just decided to give up the token after a very short time before the other busy ones gave any of their tokens. In this case, a Winner will come and conditions for Handover will be met, and the former busy anchor doesn't have to do anything other than giving the token, the shape of the network, and access to the Hand_Over network 0, and this will take place in procedure Handover.

To make it easier, we will use some terms for simplicity like for example in Table 8.1, we will use $m_{i,\{5,6\}}$ to indicate at least one set of microrings $5, 6$ other than the microrings $\{5, 6\}, 0$ whose holder has the shape of the network or does the handover as a winner and will hold it next.

Note that in multiple tokens state, there can be more than a busy PE, but for the

Table 8.1. For Multiple indistinguishable tokens: PE states in terms of flags and microrings

State	G	R	$m_{i,1}$	$m_{i,2}$	$m_{i,\{3,4\}}$	$m_{i,\{5,6\}}$	$m_{i,\{5,6\},0}$
Busy	1	1	1	1	–	1	0
Busy_Anchor	1	1	1	1	1	1	1
Anchor	0	0	1	1	1	0	1
idle	0	–	0	0	0	0	0
ready	0	1	1	–	0	0	0
Winner	0	1	1	1	0	0	1

Anchor, or Busy_Anchor, he has to be at most one.

8.3. Distinct Tokens Problem Solution

In this case, we act in a bit a different way, the easy part about it is that the PE holding a token doesn't have to move so that the Anchor will just rely on the upper one, or the winner will have to look in more than a Hand_Over network. In this case, a winner will go to the specific token it's looking for, but the question will be if it's not free, what will he do? The first solution is so easy, it'll just get the handover and wait for the desired token to be released, but this may delay other tokens that are looking for other ones. But it's just easy and so simple. The solution we rely on in our work will just give limits, meaning that in case the PE wins the competition for a number of rounds without being able to win the token, it'll wait and deprive other PE s from competition until it gets the chance to win its desired token and make use of it.

We will have to define a bit more parameters for our flags, like for example $G_{i,j}$, and variables, like for example $Reject_{i,j}$ where for both, i is the order of the PE and j is the order of the token, which is a count for the number of rejections the PE has got for this particular token.

Also, for the initialization, we will have all $Reject$ parameters initialized to 0

The main differences will be the addition for more parameters in each procedure, and there will be some more procedures for our case, which can still be the general for the mixed case of more than a set of indistinct tokens.

Also, there will be a Hand_Over network just for handover between PE s, and there will be a set of

Now let's discuss Algorithms 6,7

Algorithm 6: Get_Particular_One_Of_Multi_Tokens_Part1 `/* Compete for one among multi distinct tokens part2 */`

1 **Procedure** *Get_Multiple_Distinct*$(m_{i,1}, a_i, b_i, j)$ `/* Get Token j (for all but the anchor) */`
2 *Condition:* $G_i = 0$ and $c_i = 0$ and $R_i = 1$ and $m_{i,\{3,4\}} = 0$
3 wait until $[b_i \longleftarrow Get\_N(i, m_{i,1}, a_i, b_i) = 1]$ `/* Compete in current Non-Preemptive Network */`
4 $m_{i,\{5,6\},0} \longleftarrow 1$ `/* seek handover */`
5 wait for $[c_{i,0} \longleftarrow (Get\_B(1, \text{left}, m_{i,5}, c_{i,\ell,0})$ AND $Get\_B(1, \text{left}, m_{i,5}, c_{i,r,0})] = 1)$ `/*` $c_{i,0} = 1$ `implies that the network shape handover is complete */`
6 $m_{i,\{5,6\},j} \longleftarrow 1$ `/* seek handover */`
7 wait long enough to allow for handover to happen `/* this is used only for efficiency. It does not impact algorithm correctness */`
`/* In the following,` $c_{i,j} = 1$ `implies that desired token handover is complete */`
8 **if** $[c_{i,0} \longleftarrow (Get\_B(m_{i,5}, c_{i,\ell,j})\ AND\ Get\_B(m_{i,6}, c_{i,r,j})] = 1)$ **then**
9 $G_i \longleftarrow 1$ `/* token obtained */`
10 $Reject_{i,j} = 0$
11 **else if** $Reject_{i,j} < limit$ **then**
12 $Reject_{i,j}++$
13 **if** $Reject_{i,j} = limit$ **then**
14 go line 7

Some notes: we put the number 5 instruction time slots in Procedure Get_Multi_Distinct just as a parameter that can be changed by the user depending on the conditions, al-

Algorithm 7: Get_Particular_One_Of_Multi_Tokens_part2 `/* Compete for one among multi distinct tokens part2 */`

```
1  Procedure Release_Multi_Distinct(i)  /* Release */
2      Condition:  R_{i,j} = 0 AND G_{i,j} = 1
3      G_{i,j} ⟵ 0
4      m_{i,{5,6},j} ⟵ 0;
5  Procedure Allow_Multi_Distinct(i)  /* Allow competition:  */
6      Condition: c_{i,0} = 1 AND m_{i,{3,4}} = 0
7        /* PE i has the token shape and allows competition for other
            PE s                                                    */
8      m_{i,{3,4}} ⟵ 1
9  Procedure Handover(i)  /* Handover to new shape holder */
10     Condition: c_{i,0} = 0 and m_{i,{3,4}} = 1
11     m_{i,{1,2,3,4}} ⟵ 0  /* PE i removes itself from the terminal
           positions on the Non-Preemptive Network                  */
12     m_{i,{5,6},0} ⟵ 0  /* PE i releases the handover signal      */
```

though the Hand_Over procedure has only 3 instructions, we consider an interrupt to force the PE holding the shape of the network to handover the network.

Also, there is another procedure for the network shape holder.

Table 8.2. For Multiple distinct tokens: PE states in terms of flags and microrings

State	G	R	$m_{i,1}$	$m_{i,2}$	$m_{i,\{3,4\}}$	some $m_{i,\{5,6\}}$	$m_{i,\{5,6\},main}$
Busy	1	1	1	1	–	1	0
Busy_Anchor	1	1	1	0	1	1	1
Anchor	0	0	1	1	1	0	1
idle	0	–	0	0	0	0	0
ready	0	1	1	–	0	0	0
Winner	0	1	1	1	0	1	0

proof of correctness Again we build our proof on the Fair Architecture in Section 7.5. In other words, what happens in Algorithms 6,7 is not different from what happens in Algorithm 3. The only difference is that there is an extra step for the winner to check the

Token it's looking for, and either it gets it if it's free, or it gets rejected and in this case, it lets other PE s compete for their desired tokens. No other change can affect the correctness of the work. Correctness is again that the Architecture provides needed metrics: Safety, liveness, fairness, safe assignment, nonpreemption, request persistence.

Safe assignment, nonpreemtion, and request persistence are ensured from conditions on Procedures to start and also that a PE when starting a procedure,and it keeps executing till the end even if conditions are not true any more. Conditions we talk about are like for example that a PE cannot get a token unless it's requesting, a PE once has started competing, the Procedure ends with the PE granted the token.

For formal proof of safety, liveness and fairness:

Safety and liveness can be proved like in section 7.5. We just don't want to repeat it.

The only factor that's different in our case from Fair(n,1,1,1) is fairness. Fairness is guaranteed as in our case, we put a limit on rejections, and instead of guaranteeing a requesting PE will be granted his token in a round and at most $n - 1$ token cycles, we just multiply it by the rejection limit, which is a constant defined by users. In other words, a PE i that wins the competition will ultimately get the token granted after the limit of rejections. Although this may slow the process, but we have to stop letting other PEs compete to guarantee safety.

As mentioned earlier, we keep moving step by step towards the more general case.

8.4. Multiple Competing Loops

In this case, we will have more than a competing medium, still having total flat networks as the total number of tokens that the PE s compete for. There are more than a case to study in such case, like if the PE s can compete in all competing loops or just a subset, or a mix having some particular PE s have more access to competing media than others.

In the general case, there will be an index for the competing microrings as the index introduced earlier for the flat networks microrings. Also, there has to be ways on which to choose and if it's staric or dynamic choice.

One of the main modifications is the initialization procedures, we first modify the initialization for PE s having more than one PE initialized in Busy_Anchor state. Second there will be a set of solutions for the PE assigned to which competing medium, depending on the solution we choose.

As we go step by step, we first present the simple case where there are C competing media, and each PE has access to only one of them. In such a case, for each competing network, each one will have roughly n/C PE s in each of them. Everything will be just the same as the earlier sections except that each PE will be competing in its assigned competing medium. In such a case, for the first problem of Indistinguishable tokens, it'll not introduce speeding benefits to the system, as it's nothing but just breaking the size of competing loops. However, it can introduce layout ease benefits to the network. We recall that technical constraints always introduce restrictions that apply in our work, but we just try to work to modify our work to meet such constraints as much as possible without affect-

ing performance. The second case of distinct tokens will differ, as there has to be ways for a rejected PE competing for a particular token to stop other PE s in other competing loops from attempting to access this particular token. The solution we provide is adding an extra step for the winner and has to just activate and deactivate its two rings to check if the token is available, and if it's not available, it just waits for a random time and check it again as in the CSMA/CD approach. For non rejected PE s, we provide no change.

We now provide the modified procedure to match our problem.

Algorithm 8: Get_One_Of_Multi_Tokens_in_multiple_competing_networks `/* Compete in a multiple competing networks environment */`

1 **Procedure** *Get_Multiple_Distinct*$(m_{i,1}, a_i, b_i, j)$ `/* Get Token j (for all but the anchor) */`
2 *Condition:* $G_i = 0$ and $c_i = 0$ and $R_i = 1$ and $m_{i,\{3,4\}} = 0$
3 wait until $[b_i \longleftarrow Get\_N(i, m_{i,1}, a_i, b_i) = 1]$ `/* Compete in current Non-Preemptive Network */`
4 $m_{i,\{5,6\},0} \longleftarrow 1$ `/* seek handover for network shape */`
5 wait for $[c_{i,0} \longleftarrow (Get\_B(1, \text{left}, m_{i,5}, c_{i,\ell,0})$ AND $Get\_B(1, \text{left}, m_{i,5}, c_{i,r,0})] = 1)$ `/*` $c_{i,0} = 1$ `implies that the network shape handover is complete */`
6 $m_{i,\{5,6\},j} \longleftarrow 1$ `/* seek handover for token */`
7 wait long enough to allow for handover to happen `/* this is used only for efficiency. It does not impact algorithm correctness */`
`/* In the following,` $c_{i,j} = 1$ `implies that desired token handover is complete */`
8 **if** $[c_{i,0} \longleftarrow (Get\_B(m_{i,5}, c_{i,\ell,j})$ *AND* $Get\_B(m_{i,6}, c_{i,r,j})] = 1)$ **then**
9 $G_i \longleftarrow 1$ `/* token obtained */`
10 $Reject_{i,j} = 0$
11 **else if** $Reject_{i,j} < limit$ **then**
12 $Reject_{i,j}++$
13 **if** $Reject_{i,j} = limit$ **then**
14 go line 7

Note that we didn't show modified initialization for brevity.

Also, for initialization, it's just the same except that there is more than PE i_0 to be initialized as Busy_Anchor, it'll be a set C of PEs in state Busy_Anchor.

Finally, if there are totally n competing PEs, k total number of tokens for different sets, C loops, there will be $k + C$ flat networks.

Correctness Recall that we build on the correctness in section 7.5 and again check modifications. In this case, the solution is just the same as in section 8.2 and section 8.3. The only difference is that a multiple rejected PE s competing for the same token may collide, but CSMA/CD is proved to be deadlock free.

8.5. General Multitoken Case and Single Token Case and Benefits Employment

For the general multitoken case, with general number of distinct set of tokens, and with general number for each set of identical tokens, and with general number of competing loops, we will solve it as a build up of the cases mentioned above.

First, initialization, it's same like in the above case in section 8.4, meaning a number of C PEs in state Busy_Anchor. Competing for one of a set of Indistinguishable tokens, they will have the upper one index as the one that the PEs check for in holding the token and move down as in section 8.2. There will be a HandOver flat network per each competing set of loops.

In other words, for brevity, we just don't write the whole algorithm for each case, we just write the ones that we add for each, or else, we just mention which one we employ.

Access to multiple loops Another strategy for competing in multiple loops if we have chance to let PEs have access to competing to more than a competing loop set. In this

case, we can have more than a strategy to choose which loop a PE competes in. We have to have an initialization any way, which can be even like before, but moving can be according to different strategies like for example if a PE finds that competition in this loop is taking long, then after winning in a competition, it can start moving to compete in another one to try to get a token faster. This kind of strategies may enhance performance and reduce idle times, but in all cases not that much, as it's always a matter of a few instructions, but the main target in such a case is to improve the PE response time as PEs may not have to wait for a higher average of competition rounds to be able to get its target token.

Special case to General case and vise versa Last we note that the main result of our work, Chapter 7 has the case for n PEs, 1 token, 1 competing set of loops, and 1 flat networks.

In this case, the closest one is the Indistinguishable multi tokens as there in section 8.2 with $k = 1$. As mentioned earlier, there is no need for an extra flat network to hold the shape of network, so the PE that holds the token is the same PE that holds the shape of the network.

In all the cases, the synchronization delay is just the execution of a number of statements in a procedure. The only case that it may take longer is when there are a set of multiple distinct tokens, when a PE may wait a random time and there is a CSMA/CD environment.

generalization In this chapter, we had our solution provided to different problems relying on the existence of multiple Handover networks either to organize the transfer of leadership, or for holding a token, or both. However, with the employment of some components like a counter which can be in the Electronic or Optical domain, in such a case, a PE can hold a token and increment the counter until it gets its max, and in such a case, competition can stop. In other words, the solution provided introduced an $n-$counter that has n Optical bits, but it can be generalized to any other form. It's just simpler in the form we showed, but we can modify it to reduce the number of Handover networks.

Chapter 9. Applications

The Mutex problem is famous as it always has wide applications in computer media. In Our work, we mainly focus on employing our solution for some famous problems, which are the Broadcast and Multicast problem, the Boolean Optical Gates, and the Optical Queue.

The Broadcast problem is defined as a particular PE needs to send a package of data to all other PEs. The Multicast problem is defined as a particular PE needs to send a package of data to a particular group of PEs and they all share a particular I.D that covers this particular group.

The second problem, the Boolean Optical gate is defined as performing Boolean Logic using Optical components, which can be done by either building a particular set of gates like AND, OR and NOT, or one of two Universal gates that all kinds of Logic can be done using them, which are the NOR and the NAND gates. We will show that we can build a NOR gate using our system.

The third problem is the Optical Queue. The Queue is defined as a data structure that consists of a list of records such that records are added at one end and removed from the other. The challenge is how to construct such structure in the Optical domain, or in other words, how to perform such a job when sending and receiving data in the Optical domain. We will show that we can employ our system to achieve this goal.

9.1. Broadcast and Multicast

Broadcast is an operation where a PE sends data to all neighbors. In our case, we can use the same Mutex competing loops network to achieve this goal by just employing a

simple algorithm.

Algorithm 9: Broadcast Procedures

```
/* For the broadcast of a bit, just for illustratin the flag Get_i
   is active for all PEs                                          */

1 Procedure Get_Broadcast(id, m_i, a_i) /* get broadcasted bit in the blue
   loop B_c */
2   Condition: R_i = 1
3   m_{i,1} ⟵ 1  /* activate microring to receive data            */
4   wait until a_i = 1  /* Broadcasted bit received               */
5   m_{i,1} ⟵ 0  /* deactivate microring to let other PEs receive data
       */
6   m_{i,2} ⟵ 1  /* activate microring to receive acknowledge     */
7   wait until a_i = 1  /* Acknowledge bit received               */
8   m_{i,1} ⟵ 0  /* deactivate microring to let other PEs receive
       Acknowledge bit                                            */
```

Definitely, when a PE broadcasts data, it's not just a bit, but this can be done by so many different ways, like for example sending a bit sequence in the start and another sequence or even the same to end the broadcasted data. We just provide the simplest way to send a bit. Depending on the application, the algorithm can be modified.

The acknowledge is just another bit only, and this can be done to show that all other PEs that got the data also got the acknowledge bit.

This algorithm assumes the PEs are all trying to get the message from the sender, that again there can be format for the sender, but just we wanted to skip these details for brevity.

This idea can be generalized for a $k - ary$ torus where one sends and each receiver just broadcast it, and when all the PEs get the acknowledge from those they resend the data to, they start acknowledging the sender. The modification in the procedures will be

so simple.

multicast Another application we have is multicast, where a PE tries to send data to a particular set that has an I.D, which can be easier for the $k - ary$ torus, which in this case, the header can be there for some particular areas like a particular side of a torus, that each PE knows which header it's responsible for, so that when a PE sends data, all the neighbors pick the first, but just a portion or even one can resend the data and others do nothing with it.

9.2. Global Boolean Operations

In section 4.1.1 we showed that a detector can be considered a digital demultiplexer, and we can actually extend the concept when having more components. Even for the single Token Base or NonPreemptive or Fair Architectures, or Multi Token or even Multi Token Distinct Architectures, we can map all the components to Boolean Logic components and see how it looks like in this environment.

9.2.1. Priority Encoder

We can easily show that all our work is nothing but priority encoders in all our Architectures, whatever the single or different cases for Multi tokens. The easiest way to prove statements in Boolean logic is the truth tables as in Table 9.1. One detector has a Boolean equivalent as in Figure 9.1.

preemptive Priority Encoder

The simplest structure we introduced in our work is the Base Networkand if we check its Logic Equivalent, it's an N-input/N-Output Priority Encoder, but still again an active output can be preempted. In other words, if a PE i has output $a_i = 1$, a higher pri-

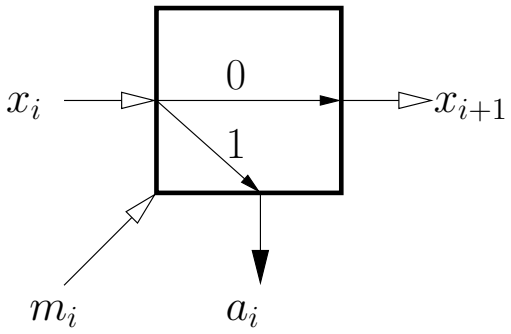

Figure 9.1. The switch as a demultiplexer: where x_i is the input, m_i is the select line, and a_i and x_{i+1} are the outputs as select $1, 0$ respectively

Table 9.1. Truth table for a Base Architecture with a sink signal x_3

x_0	m_0	m_1	m_2	a_0	x_1	a_1	x_2	a_2	x_3
1	1	–	–	1	0	0	0	0	0
1	0	1	–	0	1	1	0	0	0
1	0	0	1	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	1

ority PE j with $j < i$ can cause $a_i = 0$. This problem can be solved in the nonpreemptive Priority Encoder.

NonPreemptive Priority Encoder

Again we solve the preemption problem in this section, but again it's a fixed priority encoder. Here, the truth table is the same as the preemptive priority encoder, the only difference is that once a PE i has an output $b_i = 1$, a higher priority PE j with $j < i$ cannot preempt it or cause b_i to be 0.

The problem that's still there is that the priorities are fixed, which means that it's not fair, and a PE can be starved. Thus, a rotating priority encoder can solve this problem.

Rotating Priority Encoder

In the case of the Fair Architecture, we just have a nonpreemptive priority encoder with a variable start, or a rotating priority encoder. In other words, a PE i that has a low priority in a competition, is guaranteed to get a better priority by at least 1 in the following competition. This introduces fairness

9.2.2. NOR gate

With reference to table 9.1, we can observe that output x_3 is 1 meaning light exists if and only if all the inputs m_0, m_1 and m_2 are all Logic 0 which means they are all inactive, which resembles the truth table of a NOR gate for inputs m_0, m_1, m_2, although the switches can be done for more applications, but if we put a sink after the light input passes by a number of switches, the output in this sink is their Boolean NOR.

In other words, on top of the outputs related to each PE, we have the sink at the end of the waveguides. The only case that light goes to the sink is if all the PE s have inactive rings, which is analogous to a NOR gate, having the inputs as the activation inputs for the microrings and the output is taken from a Detector at the sink.

There are so many applications for such gate. True that it's not that easy to extend the Boolean gates concepts as it's a bit bulky compared to the Electronic gates, but still a NOR gate output for all or a subset of the inputs has its applications.

Applications for Mutual exclusion are so many as mentioned earlier, but we will focus in this chapter about how we can solve some of the famous problems that ares still open for researchers and we will show how we solve those problems employing our system.

The first problem is that of the Optical Queue.

Optical Queue Problem Definition: The Queue is generally defined as a group of information packages that can be of equal or different sizes that are stored in a medium and that there is a way to access those packages in a manner that's related to their arrival times. The problem is now how to get it in the Optical Domain. Nitta *et al* proposed the DCAF paper where they proposed a completely connected network so that there is no arbitration for accessing the communication media [6]. However, there was still the problem of whose package will be detected by the Optical detector. Meaning that there is still some sharing and this still may need some arbitration; however, he proposed the use of an Optical Queue that has an input that gets the Optical Data from senders, puts them in order and outputs them to the detector. However, till now, this Optical Queue is a challenge for researchers, and we propose a Logical Simulation for the Optical Queue using our system.

9.3. An Application–Simulating an Optical Queue

The DCAF On-Chip Optical Interconnect [6] constructs a completely connected optical network in which each processor can receive data on a fixed set of wavelengths but can transmit data in all n sets of wavelengths. Specifically if Λ_i is a set of wavelengths (one per bit in the word or phit transmitted) associated with PE i, then any PE j wishing to transmit to PE i uses the wavelengths in Λ_i. This could result in multiple simultaneous transmissions to PE i (all on the same set of wavelengths, albeit using different waveguides). DCAF uses an optical queue to automatically resolve and buffer these contending packets. Optical queues are difficult to implement [19], however.

The Fair Network can substitute for the optical queue (see Figure 9.2). The receiv-

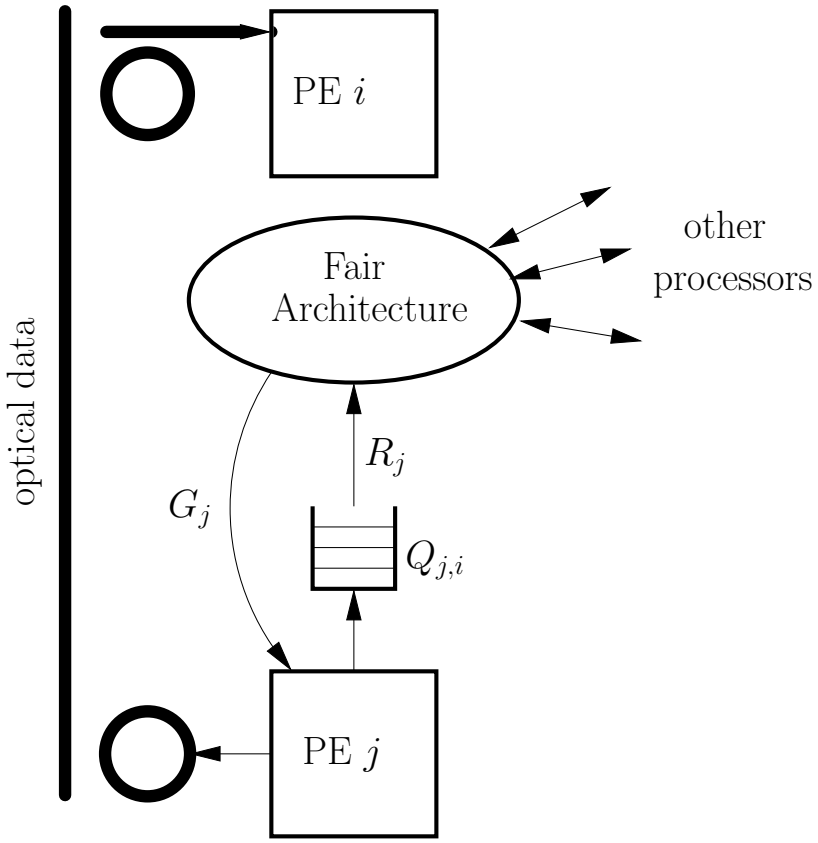

Figure 9.2. Using the Fair Network for processor-to-processor communication.

ing PE i has an n-element Fair Network associated with it (instead of an n-input optical queue). Each PE j with a packet or flit to send to PE i keeps this information on this packet (just the identifying information without the payload) in a conventional output queue $Q_{j,i}$. When $Q_{j,i}$ has enough information to warrant sending optically (queue size exceeds a threshold or queue data has been waiting for a while), p_j sends a signal W_j to compete for the token to access Λ_i. After it receives the token ($T_j = 1$) it generates the data corresponding to the information in the queue $Q_{j,i}$ and sends it optically using wavelengths in Λ_i exclusively.

Thus this communication is bufferless in the optical domain. While local storage is still electrical, data transport is performed optically (which is where the primary advan-

tage of optics over conventional communication lies).

Our approach is not overly expensive either. For n-processors and a data width of w, DCAF uses n^2w microrings (n optical queues). Adding the Fair Network introduces (at most) $6n^2$ rings over the network. This is just a $\frac{6}{w}$ increase over the original network, not counting the cost of the optical queue. For example with $w = 64$ this increase is about 11% for avoiding optical queues.

Another advantage of the proposed method is that no synchronization is needed among the contenders for the token. The wires for W_j and T_j could be inexpensive without encountering problems due to skew or signal rate. Placing PEs i and j far apart would not pose a problem.

We also observe that in the general case of a degree d network, each Fair Network would arbitrate a d-way contention. With a reasonably small degree certain topologies (such as hypercubes and multidimensional meshes) can be used to build large networks.

9.4. Token Ring Topology

As mentioned earlier in Section 3.2, our work can be generalized to the general Mutex. However, we also solve the problem that Token Ring Topology researchers have of sending the token to nonrequesting nodes. It's not always totally a Token Ring Topology with token moving only through requesting Nodes, but it's either as mentioned, or at most a node will get it in the next round, but still moving only through the requesting nodes. It's still by itself a great achievement, specially that it doesn't require any synchronisation amongst nodes.

Chapter 10. Conclusions and Future Work

We have proposed an optical Architecture (the Fair Network and associated Algorithm) that acts as a distributed semaphore to mutually exclusively grant a token to elements requesting it. We developed the Fair Network through a progression of simpler networks (the Base and Non-Preemptive Networks). The Fair Network provides a starvation-free, fair framework for distributing the token exclusively to a single element at a time. Further, the proposed networks are distributed (with no centralized control) and operate asynchronously (without a common clock).

The Networks have been proposed as if each waveguide carries only one channel. With WDM, the number of waveguides used can be reduced. For example, the Fair Network of Figure 7.1 shows four linear waveguides. By transmitting two wavelengths per waveguide, this number can be reduced to two; we still need separate waveguides for different directions. In situations requiring multiple arbitration (such as in routing), many of them could be rolled into a pair of waveguides. It may also be possible to reduce the number of rings in the architecture to 6 per element.

This work is based on ideal optical components. Practical components will exhibit losses even when the lights traverses a deactivated microring; the waveguide itself will also have attenuation losses. Additionally, there may be issues related to the range of light intensities that a microring has to operate correctly across. All these have the effect of reducing the value of n that can we supported on the Fair Network. Nevertheless, these restrictions are due to technological factors, rather than on account of a fundamental limitation.

10.1. Choice of Parameters

One of the main parameters that we will discuss its choice and effects of increasing or decreasing of this parameter is ℓ which is the level of light that the detector considers a 1. Generally speaking we have to choose it around $5dB$ or more above the noise level. Of course, the lower we choose, the longer the time we have to wait for the signal level to decrease in the light carrier waveguide after we withdraw the light through the microring, and also, the more we choose, it will take longer for the light level in the detecting waveguide to reach the required level, specially if the leader is at the far end from the outside source and the level decreased due to attenuation. Hence, it's a tradeoff between two main factors.

Also, there are some technological constraints that makes a choice of $3dB$ to be the largest difference between the levels the detector should get.

The exact decaying constant data is not available but it's in the picoseconds order. However, the only added factor is the maximum propagation time which may be in the order of 10s of picoseconds or larger for a larger chip. This factor will be the governing factor in our system, but with the proper choice of ℓ, which is around 0.36 (assuming the entering level as unity). With the proper choice of ℓ, the time thresholds will be well chosen.

There are two approaches for the choices of ℓ and the time thresholds, the first approach is to choose them even for all the PE s, and this makes it easier. The second approach is to check for each PE and its position and how and what's the highest and lowest levels of light it can get, and hence, try to chose it as the best fit possible for each PE rel-

ative to where the light enters the system.

It's also worth to note that some modifications can change constants, like for example in the Non-Preemptive Network, we don't have to have the Base Network parameters and time thresholds, we can even have a time threshold only in the Red Network and have a little larger time threshold, and in this case, the choice will take into account both the delays of the Blue and Red Networks. The reason behind it is that safety is needed in the Red Network but not crucial in the Blue Network.

10.2. Future Work: Some Modifications

We can do some modifications to reduce the number of used microrings, like for example we can rely on the linear waveguides that have microrings $3, 4$ for handover and just do some signals that allow light in different paths so that we can just have the same detectors used for microrings $1, 2$ for the handover.

Another future direction is that in Multi token solution problem, we proposed a solution having a number of linear waveguide sets equal to the number of tokens and we added a number of the competing media sets. An interesting solution for researchers to modify the solution to rely on a smaller number of linear waveguides.

Another direction for future research is that as mentioned before the microrings and the waveguides are not different from the communications elements. So it'll be interesting to think about a design to use the microrings and waveguides for more tasks than competing and holding tokens.

Appendix A. Non-Preemptive Architecture: PE Details and Timing Analysis:

In Figure A.1, details are shown for the Non-Preemptive Architecturecompeting node. As we mentioned earlier, the key word for a Base Architectureor the Non-Preemptive Architectureis the thresholds to maintain transient safety.

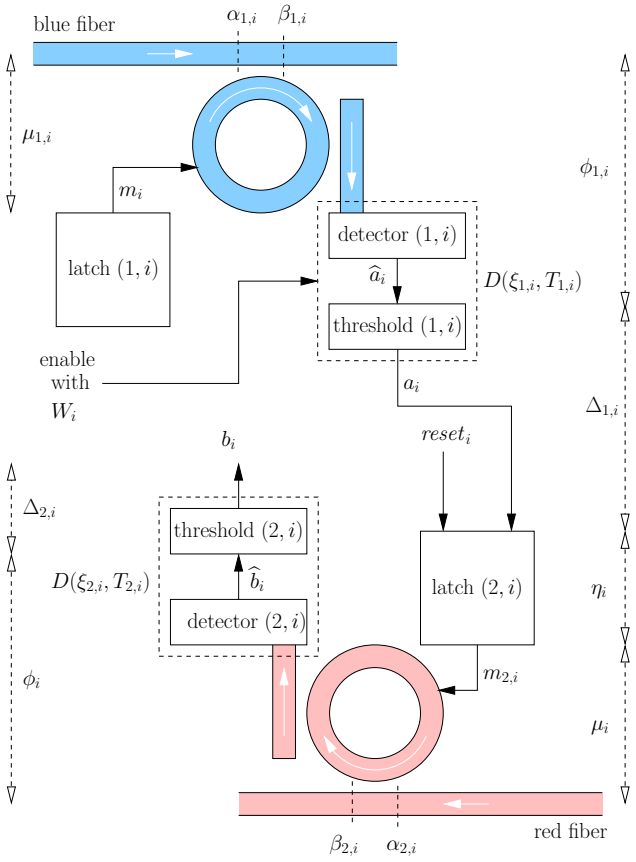

Figure A.1. The Non-Preemptive Network node details

Observe delays in Figure A.1. Let the delay of the inner latch or the time bound for the PE i to execute the next instruction be η_i and let $\eta_{\max} = \max\{\eta_i \ : \ 0 \leq i < n\}$. Other quantities such as $\mu_{1,\max}, \phi_{1,\max}$ etc. are defined similarly. All of these quantities, except $\Delta_{1,\max}$ and $\Delta_{2,\max}$, are independent of design parameters ($T_{i,1}$ and $T_{i,2}$) and depend only factors such as manufacturing tolerances and physical constants.

Therefore, we will not distinguish them between the red and blue networks and let $\mu_{\max} = \max\{\mu_{1,\max}, \mu_{2,\max}\}$.

As before, the level thresholds satisfy $\xi_{i,1} > a_{1/2}$ and $\xi_{i,2} > b_{1/2}$. The time thresholds are set to satisfy:

$$\Delta_{1,\min} > \tau_{\max} + \phi_{\max} + \delta_{\max} \qquad \text{and} \qquad \Delta_{2,\min} > \eta_{\max} + \tau_{\max} + \phi_{\max} + \delta_{\max} + \mu_{\max} \quad \text{(A.1)}$$

At time t, let $m_{2,i}$ change from 0 to 1. If after t_{reset} time output $b_i(t + t_{reset}) = 0$, then $reset_i$ is asserted. Observe that in the absence on any competition for element i, after $m_{i,2}$ changes to 1, at most t_{reset} is needed for b_i to change to a 1. If it has not yet changed, then an upstream element $j > i$ of the red network is drawing the light before it gets to i. In this case, resetting the latch causes i to compete all over again in the next competition cycle (without getting credit for setting $m_{i,2}$ to 1). This feature is important for the fair architecture of Section 7. Also note that the flag W_i indicating that element i wishes to obtain the token is used to enable the detector output to microring $(i, 2)$. While this is not needed for the token architecture, it is essential for the Fair Architecture in which a "leader" could otherwise cause a deadlock by tying up a_i even when it does not seek the token.

Appendix B. Multicast/Broadcast For Multi-Dimensional Torus:

The presented work, we showed how to do Broadcast in one dimension loop. To extend the idea to a multi dimensional torus, we broadcast the opcode for broadcast. Again, when a PE wants to resend the message after receiving it, it can do another token request in another dimension, which is a Multi-Token Problem as presented in Chapter 8. In other words, we just introduce the idea here and all tools for solving it.

Bibliography

[1] *Future directions in silicon photonics*, First edition. ed., ser. Semiconductors and semimetals, volume 101. Cambridge, MA: Academic Press, 2019.

[2] S. Bernabé, Q. Wilmart, K. Hasharoni, K. Hassan, Y. Thonnart, P. Tissier, Y. Désières, S. Olivier, T. Tekin, and B. Szelag, "Silicon photonics for terabit/s communication in data centers and exascale computers," *Solid-State Electronics*, vol. 179, p. 107928, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038110120303956

[3] P. Koka, M. O. McCracken, H. Schwetman, X. Zheng, R. Ho, and A. V. Krishnamoorthy, "Silicon-photonic network architectures for scalable, power-efficient multi-chip systems," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 117–128, Jun. 2010.

[4] M. A. Taubenblatt, "Optical interconnects for high-performance computing," *J. Lightwave Technol.*, vol. 30, no. 4, pp. 448–457, Feb 2012.

[5] D. Stevenson and R. Conn, "Bridging the interconnection density gap for exascale computation," *Computer*, vol. 44, no. 1, pp. 49–57, Jan 2011.

[6] C. Nitta, M. Farrens, and V. Akella, "Dcaf - a directly connected arbitration-free photonic crossbar for energy-efficient high performance computing," in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, May 2012, pp. 1144–1155.

[7] Y. Pan, P. Kumar, J. Kim, G. Memik, Y. Zhang, and A. Choudhary, "Firefly: Illuminating future network-on-chip with nanophotonics," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 429–440, Jun. 2009.

[8] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. P. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. G. Beausoleil, and J. H. Ahn, "Corona: System implications of emerging nanophotonic technology," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 153–164.

[9] A. Shacham, B. Lee, A. Biberman, K. Bergman, and L. Carloni, "Photonic noc for dma communications in chip multiprocessors," in *High-Performance Interconnects, 2007. HOTI 2007. 15th Annual IEEE Symposium on*, Aug 2007, pp. 29–38.

[10] R. Proietti, Y. Yin, R. Yu, C. J. Nitta, V. Akella, C. Mineo, and S. J. B. Yoo, "Scalable optical interconnect architecture using awgr-based tonak lion switch with limited number of wavelengths," *J. Lightwave Technol.*, vol. 31, no. 24, pp. 4087–4097, Dec 2013.

[11] S. Pal, D. Petrisko, A. A. Bajwa, P. Gupta, S. S. Iyer, and R. Kumar, "A case for packageless processors," 2017.

[12] A. Malacarne, F. Gambini, S. Faralli, J. Klamkin, and L. Poti, "High-speed silicon electro-optic microring modulator for optical interconnects," *Photonics Technology Letters, IEEE*, vol. 26, no. 10, pp. 1042–1044, May 2014.

[13] D. Rabus, *Integrated Ring Resonators: The Compendium*, ser. Springer Series in Optical Sciences. Springer Berlin Heidelberg, 2007.

[14] P. Flocchini, G. Prencipe, and N. Santoro, *Distributed Computing by Oblivious Mobile Robots*, ser. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012.

[15] A. D. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*, 1st ed. USA: Cambridge University Press, 2008.

[16] H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*. John Wiley Interscience, March 2004.

[17] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[18] D. Vantrease, N. Binkert, R. Schreiber, and M. Lipasti, "Light speed arbitration and flow control for nanophotonic interconnects," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, Dec 2009, pp. 304–315.

[19] X. Wang, X. Jiang, and A. Pattavina, "Constructing n -to-n shared optical queues with switches and fiber delay lines," *IEEE Trans. Inf. Theor.*, vol. 58, no. 6, pp. 3836–3842, Jun. 2012.

[20] R. Proietti, Y. Yin, R. Yu, C. Nitta, V. Akella, and S. Yoo, "An all-optical token technique enabling a fully-distributed control plane in awgr-based optical interconnects," *Lightwave Technology, Journal of*, vol. 31, no. 3, pp. 414–422, Feb 2013.

[21] Y. Pan, J. Kim, and G. Memik, "Featherweight: low-cost optical arbitration with qos support," in *44rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2011, 3-7 December 2011, Porto Alegre, Brazil*, 2011, pp. 105–116. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155633

[22] L. Wang, J. Jayabalan, M. Ahn, H. Gu, K. H. Yum, and E. J. Kim, "A case for handshake in nanophotonic interconnects," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, 2013, pp. 177–188.

[23] Y. He, K. Gopalakrishnan, and E. Gafni, "Group mutual exclusion in linear time and

space," in *Proceedings of the 17th International Conference on Distributed Computing and Networking*, ser. ICDCN '16. New York, NY, USA: ACM, 2016, pp. 22:1–22:10.

[24] F. Cicirelli and L. Nigro, "Modelling and verification of starvation-free mutual exclusion algorithms based on weak semaphores," in *Computer Science and Information Systems (FedCSIS), 2015 Federated Conference on*, Sept 2015, pp. 773–779.

[25] W. Hesselink and M. IJbema, "Starvation-free mutual exclusion with semaphores," *Formal Aspects of Computing*, vol. 25, no. 6, pp. 947–969, 2013.

[26] A. Luo, W. Wu, J. Cao, and M. Raynal, "A generalized mutual exclusion problem and its algorithm," in *Parallel Processing (ICPP), 2013 42nd International Conference on*, Oct 2013, pp. 300–309.

[27] M. Naimi and O. Thiare, "A distributed deadlock free quorum based algorithm for mutual exclusion," *CoRR*, vol. abs/1312.3347, 2013. [Online]. Available: http://arxiv.org/abs/1312.3347

[28] K. M. Kavi and D. P. Mehta, "Mutual exclusion on optical buses," *Parallel Processing Letters*, vol. 12, no. 3-4, pp. 341–358, 2002. [Online]. Available: http://dx.doi.org/10.1142/S0129626402001038

Vita

Ahmed Mansour

Education

Master of Science in Computer Engineering at Louisiana State University, Spring 2004.

Bachelor of Science in Electronics and Communications, Cairo University, Egypt. July 2000

ACADEMIC EMPLOYMENT

Graduate Teaching Assistant, Department of Electrical and Computer Engineering, Louisiana State University, Fall 2001 -Fall 2003, Fall 2013-Spring 2021. Responsibilities include: assisting professors with the preparation and presentation of undergraduate courses, grading, and tutoring. Sometimes I did give lectures.

Graduate Teaching assistant in German University in Cairo. Fall 2006-Spring 2012.

PUBLICATIONS

Asynchronous, Distributed, Optical Mutual Exclusion, October 2017, DOI: 10.1007/978-3-319-69084-1_29 Conference: International Symposium on Stabilization, Safety, and Security of Distributed Systems

Ahmed B. Mansour, Ramachandran Vaidyanathan, Shuangqing Wei

Patent

US9977206B2 United States