National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

Embeddings of a Cray T3D Supercomputer onto a Free-Space Optical Backplane

Palash Desai
(B.Eng 1993)

Microelectronics and Computer Systems Laboratory
Department of Electrical Engineering
McGill University, Montréal, PQ Canada

McGill

October 1995

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of

Master of Engineering

ISBN 0-612-12115-1

Canada

Abstract

The rapid increase in the demands for high bandwidth systems has motivated research in optoelectronic technologies and architectures. At McGill University, a five year five Major Project in Photonic Devices and Systems has been undertaken, with funding from the Canadian Institute for Telecommunications Research. One of the main goals of the project is to develop an optical backplane architecture capable of interconnecting several electronic printed circuit boards with an aggregate bandwidth on the order of 1 Terabit per second. Currently, the project is in its third year in which a representative subset of a Terabit Photonic Backplane is under development.

The objectives of this thesis are three fold. First, we motivate the study of optical backplanes by demonstrating that the interconnection network of a Cray T3D Supercomputer can be embedded onto the optical backplane. In particular, we demonstrate that the 3D mesh interconnect of the Cray T3D can be embedded into the "Dual Stream Linear HyperPlane" [9]. Secondly, having established a motivation we then provide a detailed review of the functional specifications of an optical backplane. In particular, we provide a detailed review of the *June 1995 backplane* design [31]. Thirdly, having established a motivation and a detailed design we then develop a executable software model of the June 1995 backplane using the VHDL hardware description language. The VHDL model is used to establish the functional correctness of the design. In addition, the VHDL model is used to develop a real-time simulator for the photonic backplane using 4013 Field Programmable Gate Arrays (FPGAs). The real time simulator can operate at a 4 MHz clock rate and can be used to test other electronic components such as the Message-Processors at realistic clock rates. The real-time simulator developed in this thesis will be used for real-time simulations of the associated high-speed electronic printed circuit boards which are currently under development.

Résumé

La demande croissante pour des débits améliorés de transferts de données a stimulé la recherche dans le domaine de l'optoélectronique et les architectures des systèmes qui y sont reliés. À l'université McGill, un plan de cinq ans a été mis en marche pour l'étude de systèmes optoélectroniques grâce à une subvention de l'Institut Canadien pour la Recherche en Télécommunication (CITR). L'un des objectifs principaux du projet est de développer un système d'interconnexions optique capable de relier plusieurs cartes électroniques à un régime de transfert de l'ordre de 1 Terabit/seconde. Le projet en est à sa troisième année et une portion significative est déjà en développement.

Les objectifs de ce mémoire se distinguent de trois façons. Premièrement, nous justifions l'étude des systèmes d'interconnexions optiques en système d'interconnexions d'un super-ordinateur puissant tel le Cray T3D peut se faire sur un système d'interconnexion optique. Plus particulièrement, nous démontrons que l'interconnexion tridimensionnelle en mailles du Cray T3D se réalise sur l'architecture optique d'un hyperplan linéaire à double flot. Ensuite, ayant établi une justification pratique, nous procédons à une analyse des spécifications d'un système d'interconnexions optique. Plus précisément, une analyse détaillée est faite pour l'architecture du *Système d'Interconnexions de Juin 1995 (June 1995 backplane).* Finalement, à la suite de la justification pratique et de la description détaillée d'un système d'interconnexion optique, nous procédons au développement d'un logiciel qui modèle le concept du *Système d'Interconnexions de Juin 1995* en code VHDL (langage de description des matériels VHSIC). Le modèle VHDL vérifie le bon fonctionnement du concept par le biais de simulations. En outre, la synthèse du modèle VHDL sur matrice de portes logiques programmables (FPGA) procure un simulateur temporel précis du système d'interconnexions. Le simulateur du système d'interconnexions est réalisé sur des circuit 4013 de Xilinx, opère a 4 MHz et se prête à des tests réalistes d'autres composantes du système tels les processeurs de messages. Le simulateur sur matrice de portes qui est développé dans ce mémoire sera utilisé pour des simulations temporelles exactes des cartes électroniques à haute performance reliées au projet et qui sont en voie de développement.

Acknowledgments

When I think of all the people who have directly and indirectly aided me during these past two years, I wonder whether a simple acknowledgment would do them any justice. I would like to begin by expressing my gratitude to my supervisor Ted Szymanski for his guidance and for providing me with an excellent environment to explore new and somewhat esoteric areas of electrical engineering. I thank H. Scott Hinton for teaching all of us to strive for personal excellence and to David V. Plant for his dynamism and kindness.

When one looks at the inner workings of any major project, it is usually the support staff that keep things operating smoothly. Jacek Slaboszewicz is gratefully acknowledged for keeping the MACS Lab computing network running under the most difficult of situations. I am also greatly indebted to Sylviane Duval and Bess LeBlanc who have always cheerfully answered my questions and helped me with university procedures.

I salute the volunteers of the WalkSafe Network and Foot Patrol of McGill University. I doubt I will ever meet such a fantastic group of dedicated and selfless individuals. If I could, I would thank each and every one of them for making my stay at McGill so pleasant and fulfilling. Among all the people that I have met through this organization, I would like to express my deepest warmth and affection for Amynah, Aous, Brandon, Denny, Faisal, Jocelyn, Karim, Kathy, Liza, Nadine, Tanya and Tracy.

I have many fond memories of my association with the MACS Lab. I am very privileged to call all the students who work there my friends. I especially salute Boonchuay, Eric, Michael, Sherif and Stéphane for being such wonderful people to know and to work with. I am greatly indebted to Nilanjan Mukherjee for spending countless hours answering my questions and to Ara Hajjar for helping me with Xilinx technology.

Finally, and most importantly, I would like to thank my parents and my sister for their years of unwavering support and understanding. Without them, the past two years

would not have been possible.

This thesis is dedicated to the memory of Vijayaben Desai. One of the greatest souls I have ever known.

Funding Acknowledgements

The Canadian Institute for Telecommunications Research (CITR) is a member of the Canadian Networks of Centres of Excellence (NCE). It is currently funding a Major Project in Photonic Devices and Systems [1]. The Major Project comprises of four projects located in three centres throughout Canada. These projects include the *Optoelectronic Devices, Optoelectronic Packaging Concepts, Optical and Optomechanical Hardware* and *Large ATM Architectures* [1].

One of the goals of this Major Project is to construct a photonic backplane based on free-space optics using smart pixel arrays. A photonic backplane architecture has been proposed to meet this end. The backplane architecture is currently under development at McGill University in which printed circuit boards will be optically interconnected.

The research presented herein was funded by CITR through project number 94-3-4 entitled *Large ATM Architectures* and by the Natural Science and Engineering Research Council of Canada grant number OGP0121601. This research was carried out in the Microelectronics and Computer Systems Laboratory (MACS) at McGill University. The computing resources are on loan to the MACS Laboratory from the Canadian Microelectronics Corporation through the 1993-1995 Equipment Loans.

Table of Contents

Chapter 5

Chapter 6

Chapter 7

List of Figures

Chapter 1
Introduction

1.1 Motivation

In recent years, a considerable amount of research has focused on integrating optics with electronics. The impetus for such research arises from the physical limitations inherent in all electronic devices. The ever increasing demand for high speed computers and high throughput telecommunication systems has pushed present day logic to reach unprecedented speeds [2][3]. Computers are advancing from parallel to massively parallel architectures operating on multiple data streams while telecommunication systems are rapidly progressing from circuit switching to packet based ATM switching systems thereby integrating both voice and data transport [4]. Historically, the telecommunication and computer industry were considered to serve orthogonal interests. However, as the telecommunication industry migrates to packet switching based technology, the switching networks will require switching nodes that are comparable to processing elements in a computer application. In addition, as computers move towards massively parallel architectures, the interconnection of potentially thousands of processors will be achieved using switching networks similar to those used in the telecommunication systems. It is on these systems that high performance demands are being placed on the most [2]. Figure 1.1 shows some memory and bandwidth requirements for past, present and future applications [5]. In the last decade, most applications such as oil reservoir modelling and 48-hour weather forecasting required aggregate bandwidths in the Gigabits/sec. range. During the 1990s and well into the next century, applications such as the human genome project, pharmaceutical design projects and less esoteric applications such as video conferencing will require systems whose aggregate bandwidths are in the Terabits/sec. range.

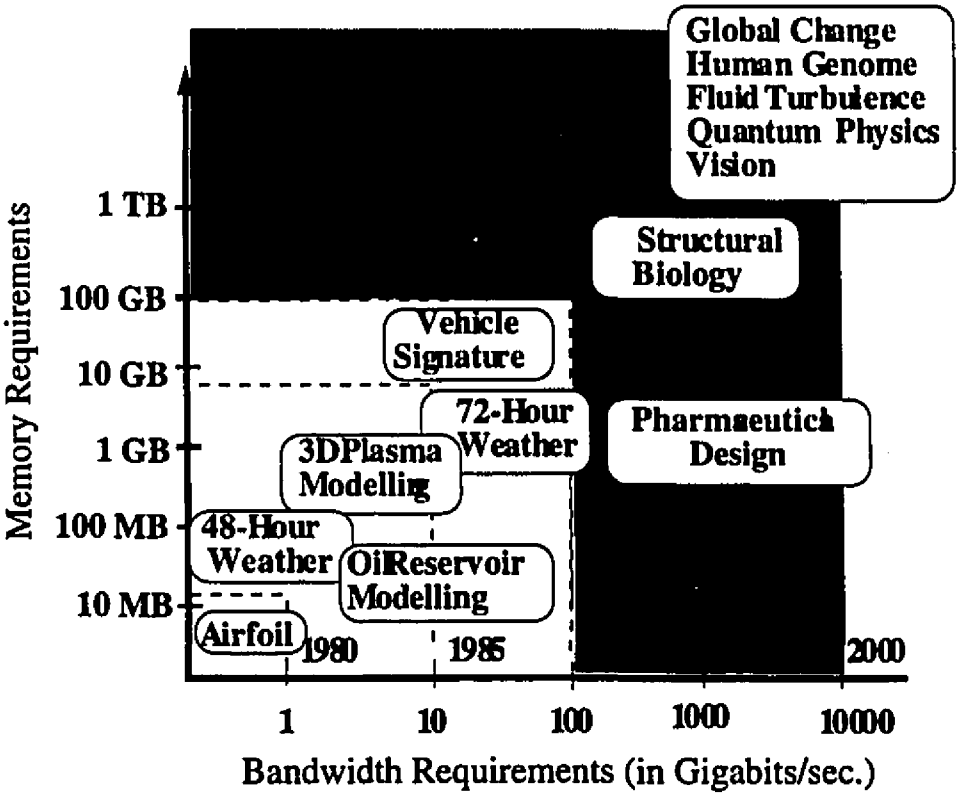

Figure 1.1 Memory and bandwidth requirements for past, present and future systems.

For many years, researchers have anticipated the need for optics as a means to implement these high bandwidth systems. Although logic gate switching times of less than 1 ns is presently achievable, it is unlikely that a purely electronic system as a whole could switch at that rate. The limitation primarily arises from the two-dimensional nature of these systems leading to long interconnection lines [6]. High bandwidth systems normally imply that the signals internal to the system itself propagate at high frequencies. The high frequency transmission causes the capacitive and inductive effects of the interconnection lines to become significant and the signal loss rate can be as high as .1 dB/cm [4] . These effects cause a distortion of signals travelling through these lines [7] leading to an increase in the bit error rate and necessitating error correction codes. The error correcting codes

increase the overall complexity of the system. In addition, adjacent interconnection lines that are parallel to one another inductively interact at higher frequencies causing electromagnetic crosstalk thereby degrading the signal as it propagates to its destination [2][3][6]. Furthermore, as $v\tau$ (where v is the speed of the clock signal and τ is its rise time) becomes comparable to the interconnection distances between modules, transmission lines will be necessary to distribute the clock [4].

A more interesting problem arises as the gate density of integrated circuits begin to rise. As the number of gates on the die increases, the number of I/O pins required will also increase. Empirical formulae such as *Rent's Rule* [2][8] given by

$$Number\ of\ Pins = k(Gates)^{\frac{1}{c}}$$

attempt to quantify the number of pins required as the number of gates on an integrated circuit (IC) increases. Here, k is a constant representing the degree of multiplexing or signal line sharing and c is a weighting factor whose value ranges from 1.0 to 3.0 (c is typically 1.79 for high performance systems).

Figure 1.2 graphically represents Rent's Rule for various values of k with c fixed at 1.79. The increase in gate density necessitates a greater number of input/output pins on each IC thereby requiring higher degrees of connectivity. The increase in connectivity leads to a growth in the density of signal traces surrounding the chip [2][3][8]. To compensate, the cross-sectional area of the interconnect must be decreased. However, the resistance of interconnects whose cross-sectional area have been reduced will increase due to the skin effect that occurs at high frequencies [4][9][10].

Given these limitations, the need for new paradigms in which electronic modules are interconnected becomes clear. One such paradigm is the use of optical signals as a means of interconnecting these electronic modules. Optical signals are non-interactive in nature and hence do not suffer from any of the physical effects that plague electronics. Optical signals also naturally propagate at high frequencies without any of the

aforementioned signal distortions and can be very small in diameter (on the order of 1 μm). Therefore, high connectivity is achievable [1]. Given that the optical signals have these qualities, there is a tremendous potential for high bandwidth systems using optical interconnects making them an ideal alternative to electronic interconnects [6] [11].

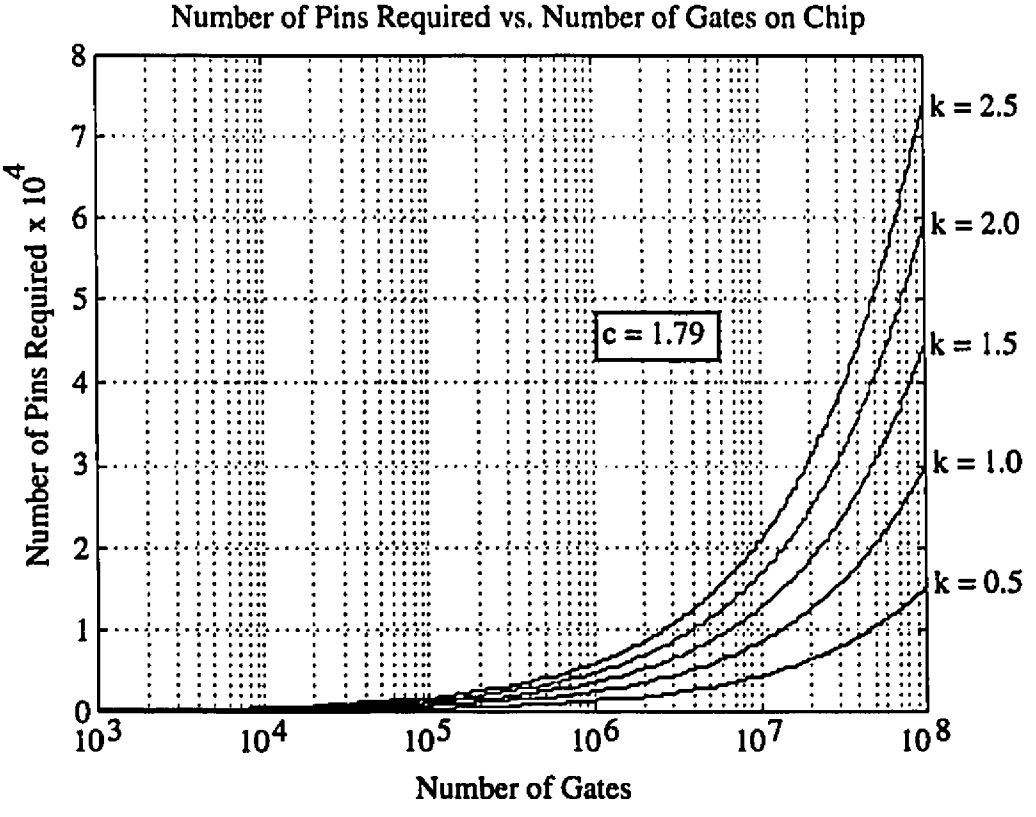

Figure 1.2 Graphical representation of Rent's Rule (*c=1.79*).

Recently, there have been many proposals and projects that attempt to capitalize on the potential benefits of optics. The Canadian Institute for Telecommunications Research (CITR) has undertaken a five year/five phase program to develop a backplane based on free space optical interconnect technology. The backplane connects electronic printed circuit boards (PCBs) via optical communication channels (OCCs). The OCCs are created by optically interconnecting Smart Pixel Arrays (SPAs). The SPAs are optoelectronic devices that have optical I/O connected to a semiconducting substrate consisting of processing

electronics [12][13]. The backplane has a potential connectivity of up to ten thousand high performance connections per PCB each capable of accommodating about 1 Gbits/sec. The aggregate throughput is therefore estimated to be on the order of 1 Terabit per second [12][14].

1.2 Author's Contributions

The versatility of the Photonic Backplane (often called the *Hyperplane*) arises from its ability to function in various applications. From the onset, the development of the Photonic Backplane arose from the vision of a high speed packet switching centre following the *Asynchronous Transfer Mode* (ATM) protocol for telecommunication purposes or as an interconnecting mechanism for *Massively Parallel Processing* (MPP) applications. It is the application to MPP systems that this thesis will concentrate on.

The contributions of this thesis are four fold. The first demonstrates the applicability of the backplane in MPP systems by embedding the network topology of a Cray T3D Supercomputer on to the Linear Hyperplane (the precise definition of the Linear Hyperplane will be given in a later chapter). It is shown that the embedding process is achievable through graph theory. Specifically, a method termed graph contraction is used in the embedding process. Various embedding schemes are shown and some comparative analyses will be made. The second contribution thoroughly documents the architecture of the optical hardware that forms the basis of the backplane. The optical hardware devices are termed CMOS SEED Smart Pixel Arrays (SPAs) and a complete physical and architectural description of these devices is presented. The third contribution is the development of the VHDL code that will describe the design of the SPAs (and hence, the backplane). The VHDL code is then read by the Synopsys CAD tool which will simulate the backplane design in software (a complete copy of the VHDL code can be found in [40]). The fourth and final contribution is the development of a hardware simulator of the design. The hardware simulator is achieved by synthesizing the VHDL code so that it can be downloaded on to a Field Programmable Gate Array (FPGA) through which real time simulations of the backplane can be undertaken. The resulting hardware simulator is an invaluable tool allowing rigorous testing of any electronic device that will eventually be

interfaced with the optical backplane at realistic clock rates.

1.3 Terminology

When participating in a project of this magnitude, it is vital that the definitions of the terms used are clearly defined. In this thesis we will repeatedly make references to several terms. Two of these terms are the Terabit Photonic Backplane and the Hyperplane. As we noted earlier, CITR is undertaking a five year, five phase program to develop an optical backplane that will have an aggregate throughput on the order of 1 Terabit per second [1]. In each phase, a demonstrator is designed and constructed. The demonstrators are used to show the progress of the project and highlight all the milestones achieved. Currently, McGill is in its third year of the five year program in which the design of the 1995/96 demonstrator is under development. The resulting 1995/1996 demonstrator will be a backplane constituting a representative subset of the Terabit Photonic Backplane. The design presented in Chapters 5 and 6 (which also can be found in [13][31]) of this thesis represents a *possible* design for the 1995/1996 demonstrator. Because the nature of this research leads to a constant evolution of designs, we have elected to refer to designs using time stamps. Therefore, the design presented and discussed in Chapter 5 and 6 is called the June 1995 backplane. The smart pixel arrays comprising the June 1995 backplane are appropriately called the June 1995 smart pixel arrays.

1.4 Overview

The research presented here revolves around the Terabit Photonic Backplane project in which a novel backplane architecture called the *Hyperplane* has been introduced. For several years, many different proposals for optical systems have been made. Chapter 2 will review some of this work and discuss the practicality of these systems.

Chapter 3 is devoted to the Terabit Photonic Backplane project. The design of the system will be described in detail as well as some of its functional specifications. The operation of the system hinges on the smart pixel arrays alluded to in the previous section. The optical aspect of the Smart Pixel Array is achieved using multiple quantum well

technology. The physical nature of the multiple quantum well technology and its integration in smart pixel array technology are briefly examined.

Next, the embedding of Cray T3D Supercomputer will be undertaken. Chapter 4 commences by formally describing the method of parallel graph contraction. It then proceeds by reviewing the architecture of the Cray T3D and its embedding on to the Hyperplane using parallel graph contraction. A comparative analysis of the various embedding schemes is then made.

Chapter 5 will describe the functional specifications of the June 1995 Hyperplane and the June 1995 smart pixel array to be used in the 1995/96 demonstrator. This will outline, in essence, how the backplane will function if and when this particular design is implemented.

Given the functional specifications of the June 1995 optical backplane and smart pixel arrays, our next objective is to describe them using VHDL and simulate them using Synopsys. The VHDL description is then synthesized for FPGA technology.[1] Chapter 6 commences by describing in some detail the FPGA technology as well as the hardware description language-VHDL. The VHDL description of the system is described and the simulation results are shown. The synthesis process is then outlined in detail and some statistical results of the synthesis are subsequently given. The result of the synthesis process is a hardware simulator for the backplane which can operate at realistic clock rates (in the MHz range).

Finally, in Chapter 7, the concluding statements are presented with some additional remarks on future work that could likely arise from this research.

---

[1] The design of the June *1994* optical backplane was also described using VHDL and synthesized for FPGA technology by this author. A complete discussion of this research can be found in [3].

Chapter 2
Evolution of Optical Technology

2.1 Introduction

With the foresight that the maximum speed of electronics will eventually reach a physical maximum, the search for new solutions using optical signals had begun. Over the past several years, there were many proposals for various optical devices and systems. From this research, two schools of thought in optical technology emerged. The first deals primarily with replacing entire electronic modules with their optical counterparts. The second uses optical signals as a means to interconnect electronic modules. In both contexts, a module could be an integrated circuit, a printed circuit board or an entire system such as a network or computer. In this chapter, we will closely examine both approaches selecting representative examples from each. The chapter commences by describing the systems using only optical devices, which shall hereafter be referred to as *all optical devices or systems,* by citing specific examples. Some practical considerations of all optical devices are then described including some remarks and observations made by researchers at Honeywell who tried to implement a massively parallel processing system using only optical devices. We will subsequently discuss systems that blend optics with present day electronic systems. These hybrid systems are referred to as *optoelectronic devices or systems* [11]. The advantages and disadvantages of optoelectronic devices are then reviewed.

2.2 All Optical Systems

In this section systems built entirely of optical systems are described. The objective of all optical devices is to create systems similar to electronic systems without the use of any electronic device, including control logic. The main motivation for this approach is that none of the components will suffer from the physical constraints that adversely affect electronic components. Theoretically, such systems would be capable of reaching speeds unattainable by electronic systems.

2.2.1 All Optical Programmable Logic Gate

We begin by describing an all-optical programmable logic gate developed at Heriot-Watt University [15]. The programmable gate, shown in Figure 2.1, is capable of implementing eight two-input Boolean functions (ON, OFF, OR, NOR, AND, NAND, XOR, XNOR). The programmable gate consists of two refractive and optically bistable components.

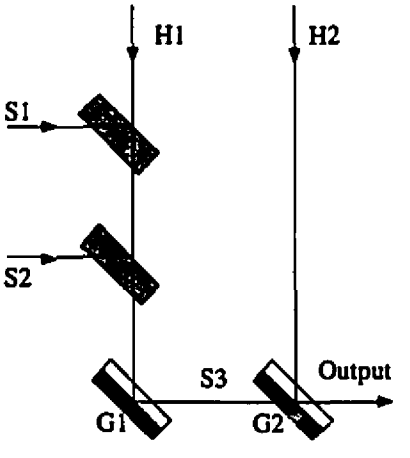

Figure 2.1 Schematic of all optical programmable logic gate. S_1 and S_2 are the optical input signals while H_1 and H_2 are the optical bias beams.

The bulk of the logic processing is performed by G_1 while G_2 acts as a logic level discriminator to the output of G_1. The inputs to the G_1 consists of three optical signals. S_1 and S_2 are the optical binary inputs while H_1 is a bias signal set to power levels labelled *a,b,or c* (Figure 2. 2a).

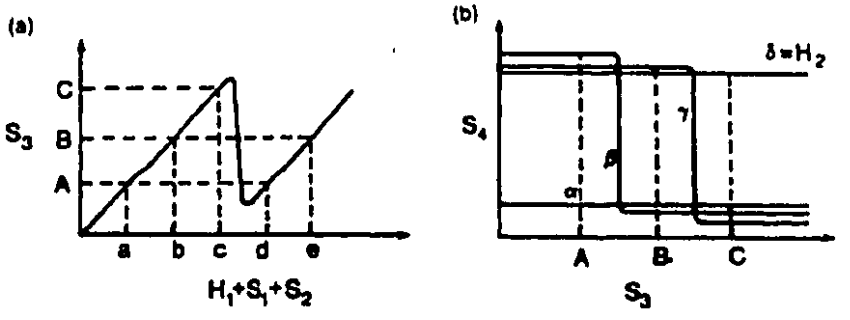

Figure 2.2 Idealized characteristics of the optical elements used.

These signals, either acting alone or combined, form the nonlinear output characteristic of G_1 (shown in Figure 2.2a). This has been termed *on axis* operation. The output of G_1, labelled S_3, is incident on to G_2 with H_2 as the bias beam yielding a characteristic response shown in Figure 2.2b.

By choosing the appropriate bias beams H_1 and H_2, the gate can be programmed to implement any of the previously mentioned logic gates. For example, if we want to implement an XOR gate, H_1 should be set to the intensity level *c* and H_2 to β. If S_1 and S_2 are both 0, then S_3 is C and if S_1 and S_2 are both one, then S_3 is B. When either of these power levels is incident on to G_2, the output will be 0 since H_2 is set to β. If either S_1 or S_2 is set high, then the intensity of S_3 will be set to A. With S_3 set to A and H_2 to set to β, the output S_4 will be high. This is exactly the functionality of an XOR gate.

The functionality of the programmable logic gate hinges on the nonlinear responses shown in Figure 2.2. Given that G_1 and G_2 are solely responsible for performing the logic

functions, they must be fabricated from optically bistable materials that exhibit such characteristics [15].

One such device is the ZnSe based nonlinear interference filters. These interference filters have similar characteristics as a Fabry-Perot Etalon with a nonlinear material in its cavity [15]. They are constructed by two outer stacks consisting of alternating layers of high and low refractive indices and each with a thickness of a quarter of the operating wavelength. These stacks perform as the mirrors found in the Fabry Perot systems. The cavity is filled with layers of absorbing materials preferably with a thickness of a half wavelength each [8]. This forms the nonlinear interference filter with an (in the case of ZnSe) operating wavelength of 834 nm.

The maximum measured speed of this device is on the order of 1 KHz and the reconfiguration speed is about 1 MHz. However, to ensure that the response of G_1 is identical for inputs S_1, S_2 and H_1, the wavelengths of these inputs have to be within 1 Å of each other. The only way to achieve this is to beam split one laser to form all three inputs. However, as the authors outlined, this can be difficult to accomplish in practice and is one of the major drawbacks of this design [15].

2.2.2 Stored Program Optical Computer (SPOC)

The *Stored Program Optical Computer* [16] is probably the first computer based on the Von Neumann model that is made entirely of optical devices. It parts from other all optical systems in that it is a general purpose device. Almost all other optical systems are application specific. SPOC comprises of several modules that form the instruction register, memory block, ALU, accumulator, address comparator and state control. The modules are interconnected using optical fibre.

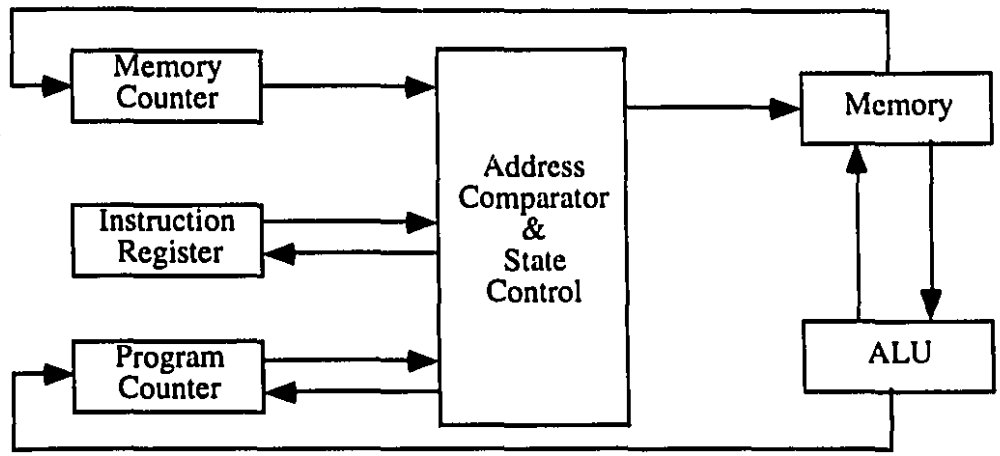

Figure 2.3 Modular design of the Stored Program Optical Computer.

Figure 2.3 shows a modular design description of the system. Previous implementations of any optical computer have relied on an external electronic host to provide control. SPOC has both code and data in its optical memory and can manipulate the input data using it's own built-in instruction set. The instruction set includes load/store instructions, ALU operations and branch statements.

In any computer, it is essential that all signals are synchronized. In conventional electronic computers, synchronization is maintained with the aid of flip-flops and a global clock. SPOC, however, does not use such an approach. Instead, synchronization is maintained by carefully determining the length of each fibre to control when signals arrive at a gate. This method is termed *time of flight* synchronization.

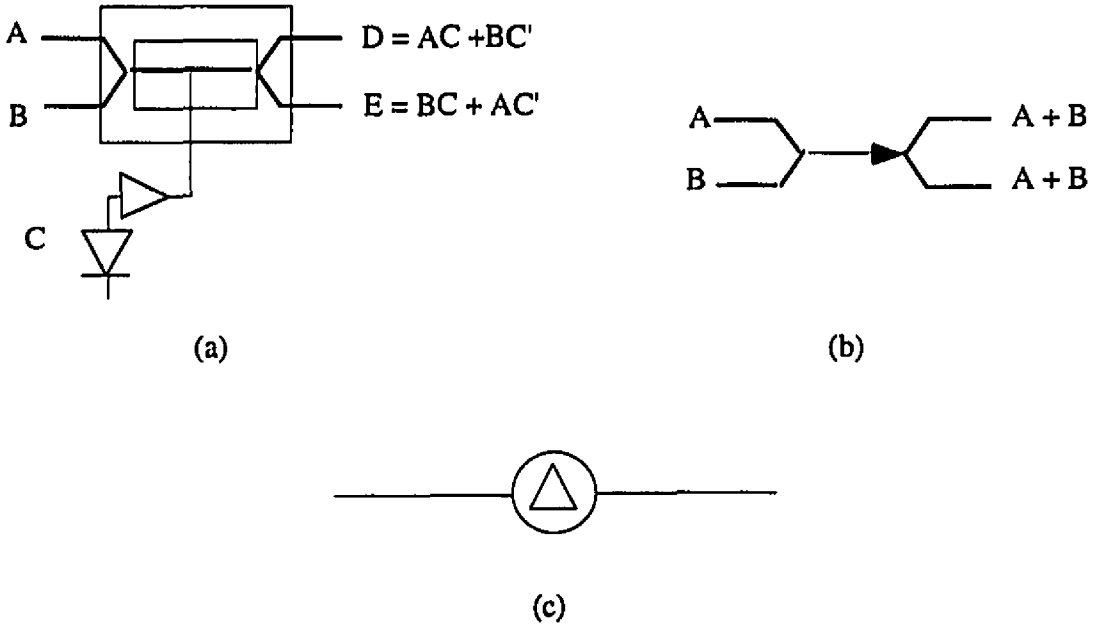

Figure 2.4 Optical components used to implement SPOC (a) directional coupler (b) splitter/combiner (c) optical fibre.

The optical components used to build SPOC are shown in Figure 2.4. The components include the optical switch, the splitter/combiner and the optical fibre. The optical switch is a directional coupler fabricated on a Lithium Niobate substrate [16]. Essentially, the directional coupler is a waveguide that realizes a 2x2 switch whose states (*cross* and *bar*) are determined by placing a voltage across it. In the absence of a voltage, the coupler is in the cross state and when a voltage is applied, the switch put into the bar state. The voltage across the switch is controlled by an optical signal C through a photodetector. If the inputs are Boolean variables, then the directional coupler realizes the boolean expressions shown in the figure.

The designers of SPOC claim that 4.5 V across the directional coupler is sufficient to perform switching. This is comparable to the voltage levels found in CMOS devices. However, in [8], it is shown that anywhere between 10-50 volts is required in order to avoid any optical crosscoupling between waveguides. The rate at which the source voltage

can reach such magnitudes is slew rate limited. This will limit the reconfiguration rate of the system.

The splitter/combiner is a straightforward device. It merely combines two input signals and then splits the combined signal back into two (logical OR operation). This device however has one significant drawback. In [8][17] it is noted that up to 75 percent of signal power can be lost due to the coupling of signals in the splitter/combiner device resulting in significant signal loss and highlighting the need for regeneration. Regeneration is achieved using optical amplifiers which introduce further delay into the system thereby reducing the maximum overall operating speed and increasing the overall cost.

The optical memory is achieved by the use of an optical fibre loop. The length of the fibre is

$$l = nl_c$$

Here n is the number of bits to be stored in the optical fibre and the l_c is the distance one bit travels in one clock period. As an example, 16 bit-words require 33 km of fibre if the clock speed is 100 MHz [16][18]. Notice, however, that this memory unit is not addressable. That is, memory is accessed in a serial manner which is considerably slower than conventional random access memory.

ARITHMETIC LOGIC UNIT

In this section, the ALU of the SPOC system is described as a representative example of how the various sub-modules operate. Figure 2.5 shows the configuration of the ALU module of the SPOC. It consists of several directional couplers that implements a bit serial addition function as follows:

$$S_n = A_nB_nC_{n-1} + A'_nB_nC'_{n-1} + A'_nB'_nC_{n-1} + A_nB'_nC'_{n-1}$$

S_n is the sum of A_n, B_n and the carry from the addition from the previous bit, *n-1*. Observe that the ALU also performs logical AND and OR functions when performing addition.

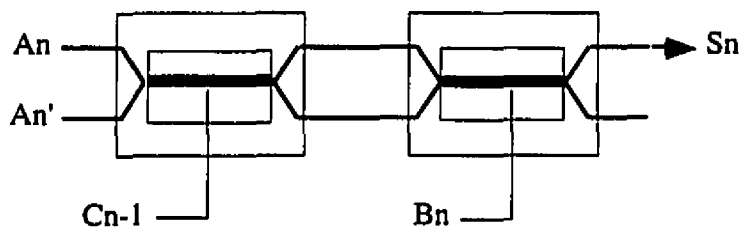

Figure 2.5 Arithmetic logic unit used in SPOC.

To save hardware, the ALU shown in Figure 2.5 was modified to simultaneously calculate seven functions in addition to the *Add* function. Using the current memory word and the accumulator as inputs, a total of eight functions are performed. Only one of these eight operations is required at a given time and so an 8 by 1 optical multiplexor (constructed using multiple directional couplers) is added to the output of the ALU to route the required solution to the proper destination [16].

All modules in the SPOC were interconnected using precisely cut fibre and operated at speeds of about 50 MHz. However, as the speed of the computer increases, the signals are more susceptible to changes in temperature. Such changes can cause signals to skew. Since there are no latches to maintain synchronization, this type of skew can have disastrous effects. In addition, the designers of SPOC reported that any physical disturbance of the optical fibre such as casual contact can cause shifts in the polarization of the light [16].

2.3 Practical Considerations of All Optical Computing

In the introduction, it was mentioned that the main motivation of optical computing was to find the means to implement faster, more powerful computers without the capacitive and resistive effects that afflict electronic signal paths. The previous two sections described

what the author believes to be representative examples of current optical computing. Indeed almost all implemented optical computers are variations of the previous two examples.

In both cases, the devices were implemented using almost no electronic devices. It is clear that there will be no problems with capacitive or resistive effects and there will be virtually no crosstalk between adjacent signals. At first glance, this might seem encouraging. However, it is unlikely that such devices could be used to satisfy present day demands and requirements. Although the programmable logic gate based solely on optical devices is truly novel in nature, its maximum speed is limited to about 1 KHz. Presently, electronic gates can operate in hundreds of MHz and in some cases up to 1 GHz [6].

In the case of SPOC, the instruction set was similar to the Von Neumann computer invented fifty years ago. There is no practical application of a computer with such limited power. In addition, [16] notes that although SPOC does not suffer from crosstalk, casual contact with the optical fibre will cause changes in polarization in the signals travelling through the fibre. Furthermore, SPOC can only operate at speeds that are comparable to electronic computers. There are plans to significantly increase the processing speed of SPOC. This, however, may not be possible since SPOC is synchronized using the time of flight approach. As the speed of SPOC increases, the optical fibres will become more sensitive to temperature changes making the system more difficult to synchronize.

A more serious problem with optical computing is memory. There is no efficient way to store data. Recall that over 30 kilometres of fibre was required to store a 16 bit word. Present day computers are equipped with several megabytes of memory to implement a vast range of algorithms. It is not possible to implement such large amounts of memory using optical fibre. Moreover, even if optical fibre could be used, the memory would have to be implemented using loops of fibre. This implementation would not allow random memory addressing and would degrade system performance.

Despite the many virtues that are possible with all optical computing, present technology is still too immature to create any marketable system. Consequently, a new approach to optical computing is needed.

In 1990 researchers at Honeywell proposed a design for a massively parallel optical computer. Although this system was never implemented, their analysis showed that it was presently impractical to make all-optical computers. It was suggested that, although great strides in optical computing had been made, the near term goals should research optics as a means of interconnect between processors only and let all computational devices be handled using current state of the art electronics [19]. In this manner, the capacitive effects of signal paths could be avoided while maintaining a high degree of computational power that has developed over the past fifty years. To the extent of the author's knowledge this was the first paper that recommended a departure from all optical computing. Since then, much of the research in optical computing has encompassed *optics in computing*. During this time, the notion of optics competing with electronics began to die out and the prospect of optics complementing electronics became increasingly popular [11][20].

2.4 Systems Based on Optoelectronic Devices

The second approach to using optics in present day technology is to blend optics with existing electronic technology. In these systems, optical signals are generally used as a means of interconnecting electronic modules. These modules could be individual chips, multichip modules or printed circuit boards. The interconnection of these modules could also be accomplished using lithium niobate fibres (optical fibre) or using free space optics. The main advantage of using optical fibre is that the alignment constraints imposed on the free space systems are relaxed to a certain degree. However, free space optics offers the potential of higher degrees of connectivity and parallelism since the optical signals are much smaller in diameter than optical fibre (optical signals can be 10 μm in diameter while optical fibre can be up to 70 μm in diameter). In this section, a systems level description of two optoelectronic demonstrators will be examined. The first of these demonstrators is the System V project conducted at AT&T Bell Labs. The second is the OPTOBUS product designed and developed by Motorola, Inc. The description of these systems will lead to the

description of the Terabit Photonic Backplane project which will be described in great detail in the next chapter.

2.4.1 System V Project

In this section, the System V project undertaken at AT&T's Bell Labs is reviewed [21]. The system is a 32 x 16 switching fabric as shown in Figure 2.6. The system is a five stage, 32 fibre input, 16 fibre output Extended Generalized Shuffle Network [21]. All interconnects between stages are achieved using free-space optics in which optical signals propagate orthogonally to the plane of the device substrate.

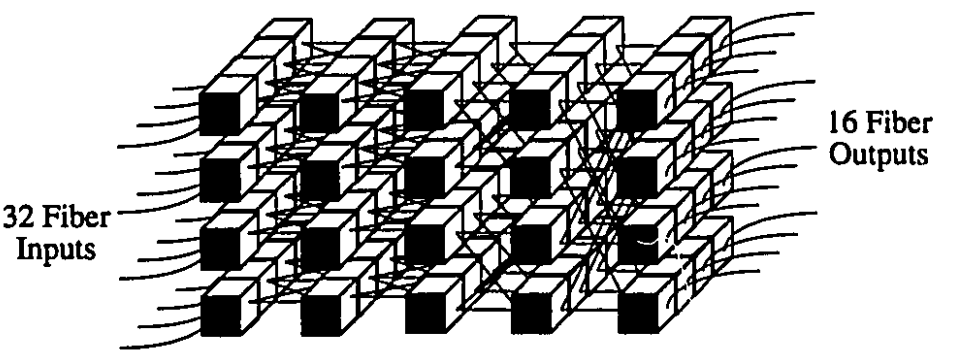

Figure 2.6 Architecture of the System V project (EGS network).

The network comprises the following. Each switching stage consists of a 4x4 array of 2 by 1 multiplexors or (2,1,1) nodes. Each node has two active inputs, one active output and an active capacity of 1. The switching stages are interconnected in a Banyan fashion to realize the extended generalized shuffle topology. The Banyan interconnect requires that each output of every node be imaged onto two inputs on the next stage. This is achieved by placing Binary Phase Gratings (BPG) between stages. The BPG uses Fourier Computer Generated Holograms (CGH) to perform a 1 to 3 split on each output signal and is used to steer the output to the required inputs of the next stage (Figure 2.7). Since only two input signals need to be imaged (in accordance with the Banyan

interconnection scheme) on the next stage, one of the three spots generated by the BPG must be blocked or masked [21].

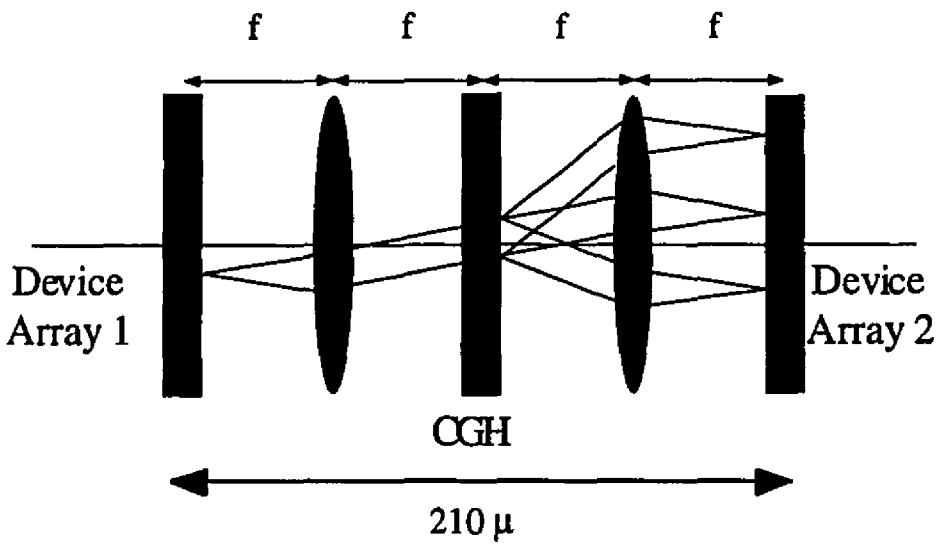

Figure 2.7 Optomechanical setup for the System V demonstrator.

Each of the nodes is implemented using a smart pixel. Essentially, a smart pixel is a device that has an optical input and an optical output (or modulator) embedded on a semiconducting substrate. The substrate can have logic etched on to it for on-chip processing. The optical input window will receive an optical input signal and convert it to an equivalent electronic signal. The logic on the semiconducting substrate will process the signal. The optical output window will convert an electronic signal from the substrate to an optical signal by modulating a continuous wave of light. The System V network was implemented using FET-SEED technology consisting of dual rail optical inputs and outputs which will be discussed in detail in the next chapter. The inputs consists of a pair of quantum well p-i-n photodiodes and a pair of clamping diodes to restrict the voltage swing of the input. The input/outputs were embedded on buffered FET logic which implemented the 2 by 1 multiplexor and a control latch. In all, 25 FET's per pixel were required. The nodes were separated by a 210 μm pitch in all directions and the optical input and output windows were each 11 μm x 11μm in dimensions [21].

The network operated as follows. Upon startup, the control signals were clocked into the nodes from the optical inputs and stored in the latches. The latches are enabled using external electronic latch enable signals. Once the control latches are set, the network configuration has been established. The network is then capable of routing data from the input ports to the appropriate output ports [21].

The System V project was a significant achievement because there are immediate applications of this system. If buffering is added to the system and a fast path-hunt algorithm is implemented, then the network can be operated in a packet switching mode. In addition, the use of optical signals as a means of interconnect allows dense interconnection intensive systems to be readily implemented [21].

However, there are still many factors that need to be considered in any interconnection scheme achieved using free space optics. The foremost difficulty is the alignment of the free-space optical signals. Recall that the area of each input and output window is about 11 μm x 11 μm. Physical conditions such as thermal expansion could cause the beams to misalign and miss the input window.

2.4.2 OPTOBUS Optical Link

The OPTOBUS package is an optical link that represents one of the first marketed products that utilizes optical interconnect through optoelectronic devices. It consists of a one dimensional array of bi-directional optical interconnects with an aggregate throughput of 1.5 Gbits/sec in each direction [22]. The first model of the OPTOBUS package comprises 10 transmit and 10 receive channels in a single 96 pin grid array package. The optical signals are transmitted and received through guided wave multimode optical fibre lines that can be of various lengths. The optical fibre lines alleviate some of the many alignment issues associated with free space optical transmission. Figure 2.8 shows the architecture of the system which contains twenty optical fibre lines each with a core diameter of 62 μm. Ten of these lines are used to transmit signals from Module 1 to Module 2 and ten are used to receive signals from Module 2. Each optical fibre has data

transfer capacity of 150 Mbits/sec leading to an aggregate data transfer rate of 1.5 Gbits/sec in each direction [22].

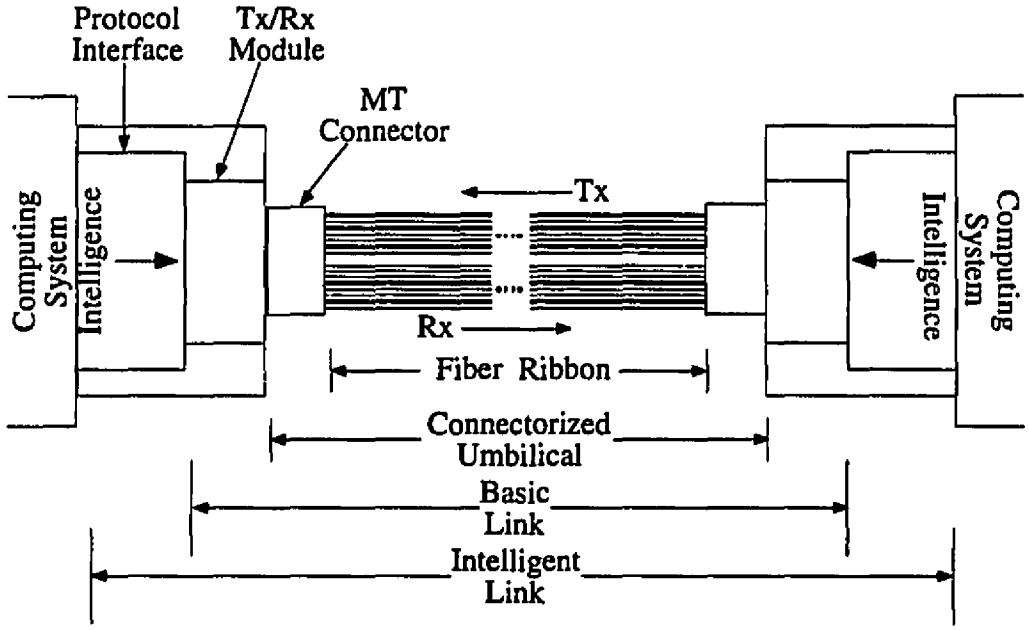

Figure 2.8 Functional layout of the OPTOBUS system.

The transmission of data is achieved through a one dimensional array of ten vertical cavity surface emitting lasers (VCSELs). The precise nature of VCSELs is rather detailed and is beyond the scope of this thesis. However, it suffices to say that VCSELs are miniature lasers that receive electronic data from its computer host through a laser driving integrated circuit and converts the data into optical signals. The laser driving integrated circuit must receive the data from the computer host in dual rail form. This means that all data transmitted from the computer host must also be complemented. The light emanating from the VCSELs will travel along the optical fibre where it is eventually received by the receiver on the receiving module. The receiver consists of an array of 10 photodetectors connected to a transimpedance amplifier. In essence, the photodetectors will convert

optical signals into electronic signals which can then be transmitted to the computer host on the receiving board [22].

The high bandwidth capabilities of the OPTOBUS system allows designers to delve into various applications such as high definition television (HDTV), data storage and file servers [22].

2.5 Chapter Summary

In this chapter, a survey of optical technology was discussed. We began by citing examples of systems that comprised only optical devices. We saw that although great strides had been made in this area, optical technology was still too immature to completely replace electronic technology. We then examined systems that blended optics with electronics. These hybrid systems called optoelectronic devices offered the benefits of using optics while still maintaining a high degree of computational power. We cited two specific examples of systems based on optoelectronic devices.

In the next chapter, we will describe the Terabit Photonic Backplane which is also a system based on optoelectronic devices. Once the specifics of the Terabit Photonic Backplane have been discussed, we will demonstrate how it can be used in real world applications (namely, the Cray T3D Supercomputer).

Chapter 3
Overview of the Terabit Photonic Backplane

3.1 Introduction

This chapter will overview the Terabit Photonic Backplane project mentioned in the first chapter. The development of the Terabit Photonic Backplane is one of the main objectives of the Major Project in Photonic Devices and Systems which is a five year endeavour funded by the Canadian Institute for Telecommunications Research [1]. The Major Project is subdivided into four projects. These projects conduct research in *Optoelectronic Devices, Optoelectronic Packaging Concepts, Optical and Optomechanical Hardware* and *Large ATM Architectures.* The research for these projects are carried out in three research centres across Canada [1]. At the end of the five year endeavour, a free space optical backplane with an aggregate throughput on the order of 1 Terabit/sec. will be constructed. We refer to this backplane as the Terabit Photonic Backplane. In this chapter, we review the architecture of the Terabit Photonic Backplane.

3.2 Terabit Photonic Backplane

3.2.1 Overview of the Backplane

Figure 3.1 shows a conceptual view of a free-space optical backplane [12][13] which could represent the Terabit Photonic Backplane. The backplane consists of a large number of optically connected channels (OCCs). The OCCs are created by Smart Pixel Arrays (SPAs) which are optically interconnected. SPAs are optoelectronic devices consisting of optical inputs and/or optical outputs and a semiconducting substrate consisting of electronic processing circuitry. The printed circuit boards (PCBs) shown in Figure 3.1 can transfer data from one another by injecting signals into the OCCs via the SPAs. The signals are then transferred to the destination PCBs where the SPAs will receive and extract the signals and convert them back into electronic form. One of the main advantages of using optical signals is the high degree of connectivity they offer. The backplane can offer up to ten thousand high performance optical signals per PCB leading to an aggregate throughput on the order of 1 Terabit per second [12][13].

Figure 3.1 shows that each PCB hosts a series of integrated circuits (ICs) . These ICs could be high performance switching nodes in which case the entire backplane could be used in a telecommunications application. Alternatively, the ICs could represent processing elements in which case the backplane could be used in a massively parallel processing application.

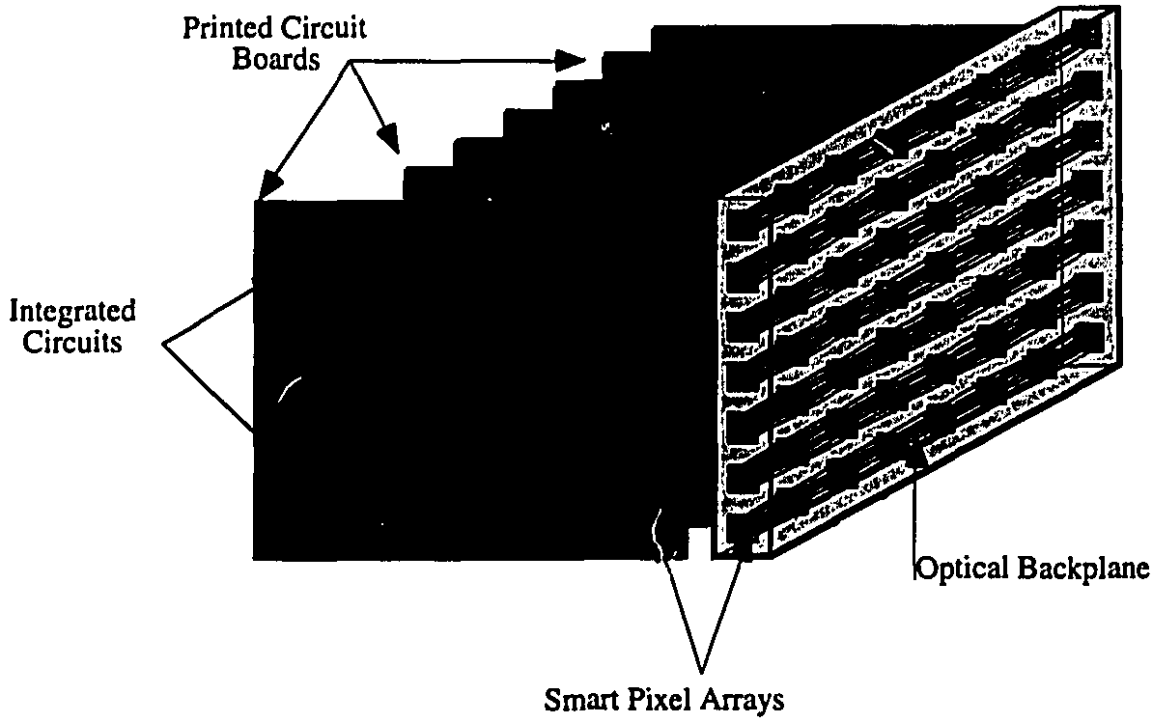

Figure 3.1 Conceptual depiction of the Terabit Photonic Backplane (Hyperplane)

3.2.2 Smart Pixel Array Technology

In the previous section, we mentioned that data to be transferred between PCBs is injected into the OCCs via the SPAs [12]. As its name implies, a smart pixel array is simply an array of smart pixels. A smart pixel is an optoelectronic device that has an optical input and/or an optical output with electronic processing circuitry and has the ability to be integrated into two-dimensional arrays as shown in Figure 3.2 [12]. In Figure 3.1, the SPAs are packaged optoelectronic chips that are mounted on a PCB. The SPA can communicate with the ICs mounted on the PCB through the electrical channels.

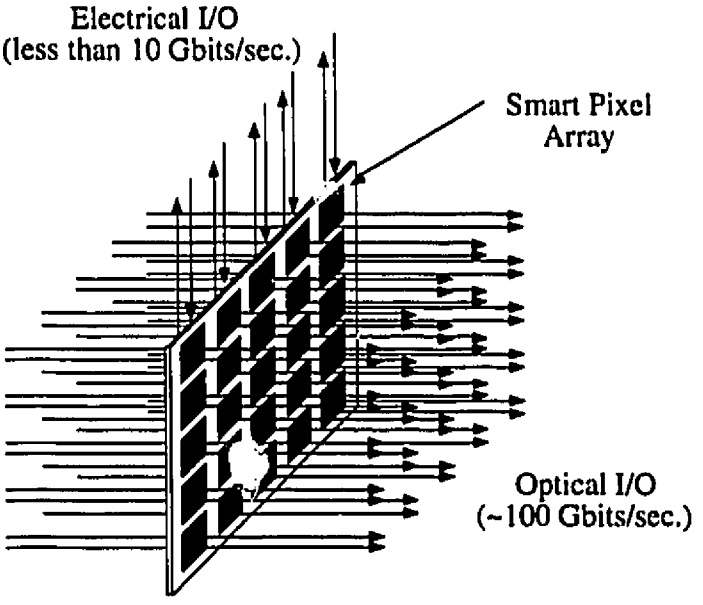

Figure 3.2 Smart pixel array.

In addition, each SPA can communicate with other SPAs mounted on other PCBs by using a two dimensional array of free-space optical signals [12]. The SPA shown in Figure 3.2 is a representative diagram of smart pixel array technology. There are many different technologies that can implement SPAs that are presently evolving. We will concern ourselves with only two of these technologies. The first is the smart pixel array based on VCSEL/MSM technology [12][23] while the second is smart pixel arrays based on CMOS SEED technology [12]. The long term goals of the Major Project is to develop monolithic smart-pixel arrays based on vertical cavity surface emitting lasers (VCSELs), metal-semiconductor-metal (MSM) photo-detectors and heterojunction field effect transistors (HFETs) [1]. The VCSELs are the optical outputs, the MSMs are the optical inputs and the HFET is the substrate on which the processing electronics is etched. Currently, the integration of HFETs with MSM detectors in Indium Phosphide based materials is under development. However, as we mentioned in the Introduction, demonstrators are built approximately once every year to highlight the milestones achieved. Since Indium Phosphide technology is still evolving, the SPAs used in the demonstrators thus far have been implemented using CMOS SEED technology. The CMOS SEED SPAs are based on

multiple quantum well (MQW) technology. A full description of MQW technology will be given next.

3.2.3 Multiple Quantum Wells

The main purpose of any optoelectronic device is to receive data in optical form and convert it into an electronic form. Once in this state, any desired processing can be performed on the data. Upon completion, the data can then be converted back into an optical signal for transmission. This process is also known as optoelectronic conversion. In the following, we will describe physical attributes of the *Multiple Quantum Well* which forms the basis of the CMOS SEED Smart Pixel Array.

The efficacy of SEEDs as optoelectronic devices hinges on the changes in optical absorbtion that are achieved by placing and external electric field across it. The optoelectronic conversion capability is accomplished using multiple quantum well (MQW) technology. Essentially an MQW is a device with a highly periodic structure consisting of layers of dissimilar materials [6][7][8] formed on a substrate such as GaAs. Each quantum well consists of a thin layer called a well surrounded on each side by a thicker layer called a barrier. Each layer is generally about 100 Angstroms thick and are grown using a molecular beam epitaxy process. The well is usually fabricated with GaAs while the barrier is made from AlGaAs. About 75 to 100 of these quantum wells are then stacked upon one another to form the multiple quantum well as shown in Figure 3.3.

The MQW operates as follows. Associated with any material is an absorbtion spectrum. The absorbtion spectrum gives an indication of how absorptive (that is how much light incident on a given material will be absorbed) a material is as the energy (and hence the frequency) of the light incident upon it varies. In bulk materials, this spectrum is smooth and rises smoothly as the frequency (and hence the energy) of the photons increases. In quantum well materials, the absorbtion spectrum contains discrete steps. These steps are present because the electrons and holes within quantum wells reside in discrete energy levels. The absorbtion spectrum shifts as electric fields are applied perpendicular to the quantum wells.

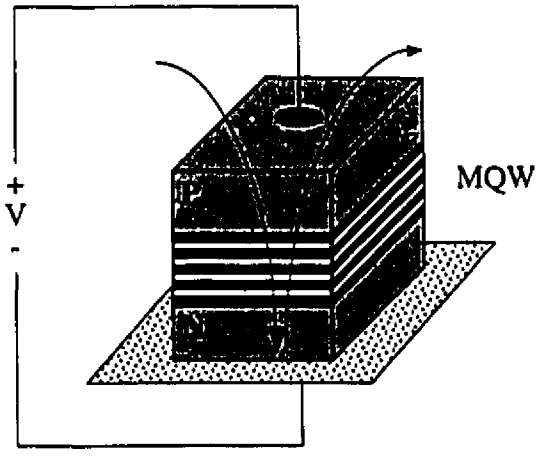

Figure 3.3 Physical structure of an MQW.

In quantum wells, the coulombic interaction between the electron and hole causes the absorbtion spectrum to exhibit peaks called excitonic peaks. These peaks occur at discrete energy levels associated with the incident photon. When a photon is incident to the quantum well at these peaks, a bound electron-hole pair, called an exciton, is created. This pair does not separate into a separate electron and hole but rather remains loosely coupled similar to that of a loosely coupled hydrogen atom. In a bulk material, the excitons are much larger and therefore are short lived except at low temperatures. However, in quantum well materials, the excitons are confined by the wells because of the barriers surrounding them. Therefore, even at room temperatures, the absorbtion spectrum exhibits excitonic peaks [6][7][8].

If an electric field is applied the electrons and holes would be pulled apart, to some degree, towards opposite sides of the well. If the material were a bulk material then the electrons and holes would be torn apart into distinct electrons and holes. However, in the case of the quantum well, the electrons and holes are prevented from being torn apart because of the barriers surrounding the well. Thus the electrons and holes would continue to orbit one another. Since the electrons and holes are already pulled apart, less energy is

required to form an exciton. Thus a red shifted absorbtion spectrum (an effect known as the Quantum Confined Stark Effect or QCSE) is observed when an electric field is applied as in Figure 3.4. In quantum wells where the band gap energy of the barrier is much higher than that of the well, then each well acts independently and the absorption multiplies by the number of wells.

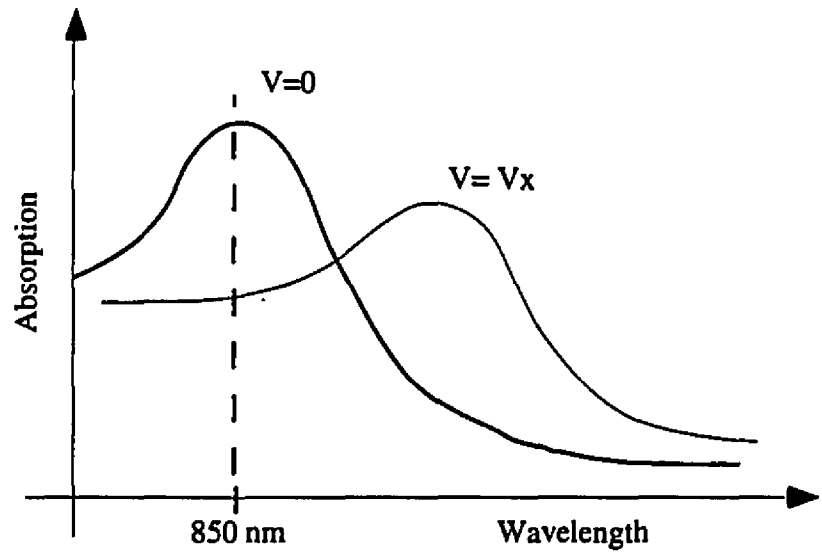

Figure 3.4 Quantum Confined Stark Effect (QCSE).

Given these properties, it is possible to construct an optical receiver and an optical "transmitter" using the multiple quantum wells. The word transmitter is somewhat deceptive because the MQW cannot produce light itself. However, using the QCSE, the multiple quantum well can modulate a continuous wave of light. Essentially, the MQW as a transmitter works as follows. A continuous wave of light is applied to the MQW. If the there is no voltage across the MQW then most of the CW will be absorbed. If however, a voltage is applied the absorption peak shifts and some of the CW will be reflected. That reflected wave of light will be received by a SEED on another Smart Pixel Array (and hence received by another printed circuit board) [8][7][24].

The MQW can also act as an optical receiver. In the absence of any voltage, the MQW behaves as a heavily reversed biased pin diode. Subjecting the reversed biased MQW to light, induces a photocurrent through the MQW. The presence of a photocurrent corresponds to a logical 1 where the absence corresponds to a logical 0. The MQW has several advantages. It can act as both a modulator and a receiver, it can be integrated in a VLSI system to embed logic on the die itself.

3.3 Operation of the Terabit Photonic Backplane

One of the advantages of the Terabit Photonic Backplane is its ability to embed various interconnection networks. The result is a multidimensional backplane which provides PCB to PCB connectivity. The MP's are capable of dynamically reconfiguring its access to the backplane channels establishing an interconnection network which is dynamically reconfigurable. The resulting three dimensional interconnection design space is often referred to as the *Hyperplane* [13].

So far, we have described the basic architecture of the Hyperplane and the technology used to implement the optical channels. However, the diagram shown in Figure 3.1 is only a conceptual depiction. There are several structures which can implement the Hyperplane [9]. In the following, we will review two of these structures-namely the *Dual Stream Linear Hyperplane* and the *Circular Hyperplane.* We will then discuss the specifics of transmitted data between boards.

3.3.1 Dual Stream Linear Hyperplane

Figure 3.5 shows the structure for the dual stream linear Hyperplane consisting of four PCBs [9]. The OCCs are organized into two uni-directional data streams called the *upstream* and the *downstream.*

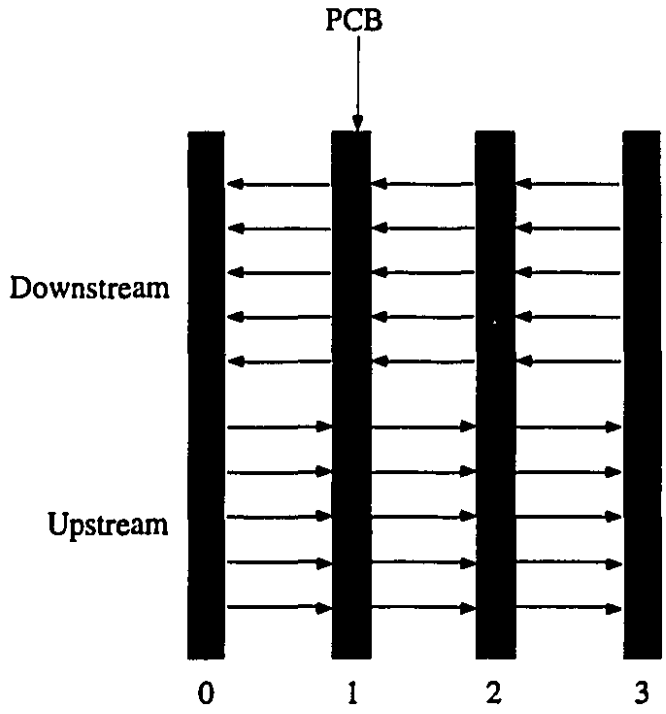

Figure 3.5 Typical structure of a Dual Stream Linear Hyperplane.

The upstream transmission means that PCB i will transmit to another PCB j where $i < j$. The downstream transmission means that PCB i will transmit to another PCB j where $i > j$. The fact that there are two uni-directional data streams implies that each PCB will need two packaged SPAs. One SPA will be used for upstream transmission while the other will be used for downstream transmission [9].

3.3.2 Circular Hyperplane

An alternative structure to the dual stream linear Hyperplane is the Circular Hyperplane. Figure 3.6 shows the Circular Hyperplane for four boards. We see that there is only one stream necessary in this implementation. Using bulk optics, PCB 3 in Figure 3.6 is connected to PCB 0.

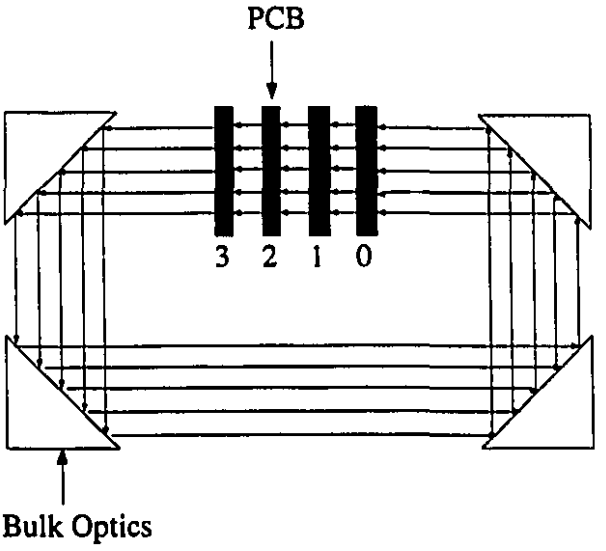

Figure 3.6 Typical structure of a Circular Hyperplane.

3.3.3 Reconfigurable and Intelligent Operating Modes of the Hyperplane

The Hyperplane can operate in two general modes. The first is called the reconfigurable mode while the second is called the intelligent mode [13]. The mode that the Hyperplane operates in is dependent on the application it is being used in (eg. telecommunications, massively parallel processing). In the reconfigurable mode, the Hyperplane can be reconfigured to embed the network topology of any graph. This is only limited by the number of electrical and optical channels. When operating in the intelligent mode, the SPAs will process packets of data as they propagate through the backplane. The processing electronics on each SPA will decide whether to extract a packet based on predefined extraction criteria [13].

3.4 Summary

Chapter 3: Terabit Photonic Backplane Project

In this chapter, a detailed explanation of the Terabit Photonic Backplane was given. The concepts of multiple quantum well technology were also outlined. The topics covered in this chapter is enough to give the reader an adequate background to understand the topics covered in the subsequent chapters. In the next chapter, we demonstrate that the network topology of a Cray T3D Supercomputer can be embedded on to the Hyperplane. The purpose is to demonstrate that modern high performance systems can be implemented using the Hyperplane

Chapter 4
Graph Contraction and Embeddings

4.1 Introduction

In previous chapters, we have motivated the need for optical technology and described the Terabit Photonic Backplane (otherwise known as the *Hyperplane*). One of the primary advantages of the Hyperplane is its ability to embed a variety of networks allowing it to satisfy the spectrum of requirements demanded by different applications including parallel processing applications. Each printed circuit board could accommodate one or more processors that would be interconnected optically to other processors on other boards. One of the main bottlenecks inhibiting the performance of parallel processing systems is the communication time between processors. As the number of processors in the system increases, the limitations imposed by the communication between processors become increasingly significant. This limitation is due to the two-dimensional nature of the system that often implements a multidimensional network configuration leading to long interconnection lines [6]. The network configuration itself also differs for differing applications. How well the network configuration suits the demands for the particular application will directly impact the overall performance of the system [25]. Since the Hyperplane uses optical channels to interconnect its boards and is designed to be reconfigurable, a greater degree of connectivity is thus achievable alleviating the communication bottleneck to a great extent. In this chapter, we will demonstrate the usefulness of the Hyperplane in a massively parallel processing architecture. We will examine the Cray T3D Supercomputer and embed its network topology, through *graph contractions,* on to the dual-stream linear Hyperplane (the embeddings of the Cray T3D Supercomputer on to the *Circular Hyperplane* using similar schemes has been done in [13][31]). The chapter commences by giving a formal description of graph contraction. Citing specific

examples, we will carry the reader through the embedding process and demonstrate that the Cray T3D can be implemented on to the dual-stream linear Hyperplane. We will, in addition, discuss the number of printed circuit boards and the number of smart pixel arrays per printed circuit board necessary to achieve this.

4.2 Graph contraction

4.2.1 Introduction to Graph Contraction

Graph contraction [25][28] is often used in parallel processing when a set of tasks for a software application must be assigned to the parallel processing system whose network topology is fixed. The scheduling of such tasks, in practice, is not always straightforward because the execution times of each task may vary. Since the tasks are often dependent on one another, a schedule for processors needs to be developed so that each processor will execute at the appropriate time [26]. To help simplify the problem, graph contraction is normally used for assigning the tasks to the processors [25][28]. However, graph contraction is also a useful tool, as we will shortly see, when embedding networks onto the Hyperplane. Although the concepts in graph contraction are straightforward, a formal description of it will be given, in the next subsection, for the sake of completeness.

In parallel processing, the software application consists of various tasks which are executed by processors of the network. Ideally, the number of processors in the network should be the same as the number of tasks in the application. Unfortunately, the overall cost and complexity of the system rises proportionally to the number of processors it contains. Fortunately, however, applications consisting of a large number of tasks can be processed by smaller networks by assigning multiple tasks to each processor. The tasks will share the resources of the processor [28]. This approach not only reduces the number of processors required but also alleviates the interprocessor communication latency to some degree [28]. The challenge therefore lies in the assigning of tasks to a given topology that yields the best performance. The process by which this is achieved is termed *graph contraction* [25].

Parallel software applications are often represented using standard graph notation in which a vertex represents a process or task to be executed and an edge between two vertices represents the communication between the two processes. Figure 4.1, shows an example of such a graph, $G(V,E)$. The set of vertices is represented by $V(G)$ or simply V where each vertex $v \in V$ and $|V|$ represents the total number of vertices in $G(V,E)$. Similarly, the set of edges is represented by $E(G)$ or simply E where each edge $e \in E$ and $|E|$ represents the total number of edges in $G(V,E)$. Two vertices, u and v, of $G(V,E)$ are termed neighbours if the edge $(u,v) \in E$ where (u,v) represents an edge connecting vertex u with vertex v. Additionally, $G'(V',E')$ spans $G(V,E)$ if $E' \subseteq E$ [25][27][28].

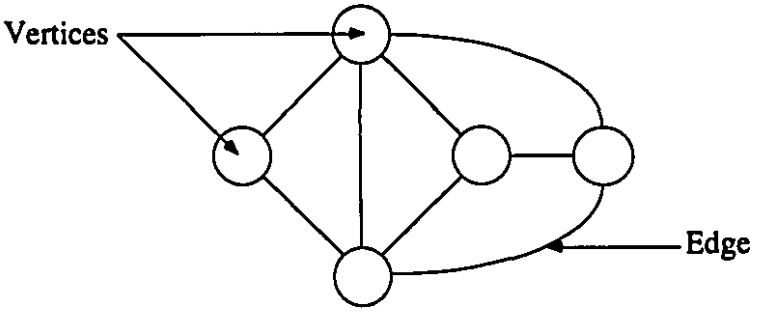

Figure 4.1 Typical graph topology consisting of five vertices and eight edges.

4.2.2 Formal description of Graph Contraction

The task of assigning a set of processes to a given network topology can be represented by a mapping function φ which will transform or map a graph G (representing a set of tasks) to a reduced graph G' that maintains certain properties of $G(V,E)$. We call the mapping function, a contraction of $G(V,E)$ [28].

Chapter 4: Graph Contraction and Embeddings

Let *G(V,E)* be any arbitrary graph with the same characteristics described in the previous subsection. Partition the set of vertices, V, into a set of mutually exclusive subsets, V_1, V_2, ... such that the number of vertices represented in each subset does not exceed a positive integer, l (i.e. $|V_i| \leq l$ for $i = 1, 2, \ldots$). Such a partitioning of the graph G is called an *l-partition* of G. The contraction φ will transform the graph *G(V,E)* into a contracted graph $G(V', E')$ by performing an *l-partition* of *G(V,E)* and replacing each resulting subset of vertices by a single vertex v_i' where $v_i' \in V'$. The edges are then added according to the following rule. A pair of vertices (v_i', v_j') with $i \neq j$ are joined by an edge in $G'(V', E')$ if and only if there is an edge connecting a vertex in V_i with a vertex in V_j. This process is termed graph contraction [28]. Notice that when the subgraphs, formed by the *l*-partition of G, are replaced by the set of vertices in V', all the edges within each subgraph are internally absorbed by the vertices in V'. We define the width of a contracted edge connecting v_i' with v_j' to be equal to $|e_{v_i v_j}|$ where $e_{v_i v_j} \in E$ and $e_{v_i v_j}$ represents the set of edges that connects a vertex in V_i with a vertex in V_j. Additionally, the number of vertices that a vertex v_i' contracts under the mapping φ is called the *cardinality* of v_i'. A *bounded contraction of degree l* is a contraction from $G(V,E)$ to $G'(V',E')$ such that the degree of any vertex $v_i' \in V'$ is less than or equal to its cardinality. Lastly, a *bounded contraction of minimal degree*, is simply a bounded contraction of G whose degree l has been minimized [25].

Repeated applications of graph contraction to a problem results in an exponential reduction in the size of the problem which can then be assigned to networks with fewer and fewer processors. With this method, almost any parallel application could be executed using a processing system of any size. However, as the number of processors decreases, the number of tasks assigned to each processor will increase [28][29].

Thus far, the description of graph contraction has focused on its software application. That is, the task is to assign a number of software tasks to a finite number of processors with a fixed topology. It is important to note that for our purposes, graph contraction will be used for the embedding of network topologies onto the Hyperplane and not for parallel software applications. In this context, the vertices of a graph represent processing elements of the Cray. After contraction, each of the contracted vertices represents a set of processing elements placed on a single board. Given the network

topology, we wish to embed these processors on to the dual-stream linear Hyperplane. However, the Hyperplane has a fixed number of boards that it interconnects (the number of boards varies from 16 to 64). If the number of vertices (processors) we wish to embed is less than or equal to the number of boards accommodated by the Hyperplane, then we merely assign one processor per board. If, on the other hand, the number of processors is greater than the number of boards then we have to assign multiple processors to each board. We use graph contraction to assign these processors to the .boards.

4.3 Graph Embeddings

The connection intensive nature of the Hyperplane allows a number of different network configurations to be embedded on to it [13][12]. The embedding process consists of mapping a graph representing a network configuration on to the Hyperplane itself. The Hyperplane represents a fixed architecture whose interconnection scheme varies to suit different networks. The embedding describes the manner in which the boards need to be connected to achieve the desired network topology. Figure 4.2 shows an example of a 4 board embedding on the Hyperplane. The nodes at the bottom of the template (numbered 0 to 3) represent the printed circuit boards connected to the Hyperplane. The vertical lines connected to the nodes are the electrical connections between the board and its associated SPA or SPAs (recall each board can have more than one SPA associated with it). The bold, horizontal lines represent the *optical connections* between two boards (through the SPAs). The bold squares symbolize an optical transmitter (or modulator) while the circles symbolize an optical receiver. The template is divided into two halves. The top half represents the downstream transmission while the bottom half represents the upstream transmission. For each board a separate SPA is needed for the upstream and the downstream transmission. Downstream transmission simply means that node i will transmit to another node j where $i > j$. Similarly, upstream transmission means that node i will transmit to another node j where $i < j$.

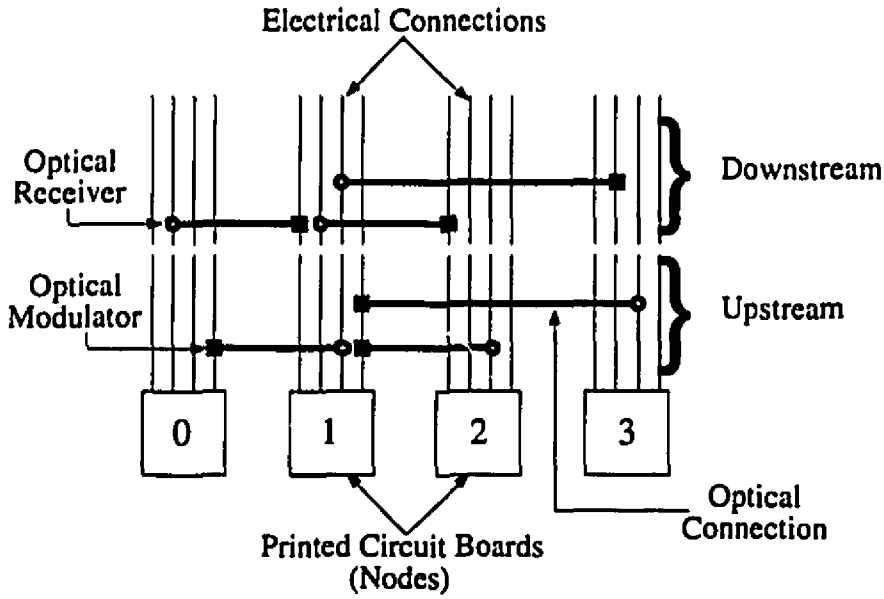

Figure 4.2 Example of a 4 board embedding on the Hyperplane.

Figure 4.3 shows the convention we have chosen for the embedding process. The left half of the electrical connections are used for downstream transmission and reception of data. The right half are used for upstream transmission and reception of data.

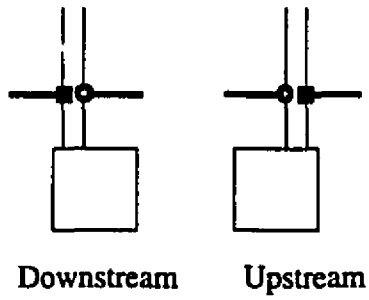

Figure 4.3 Convention for placing upstream and downstream optical connections.

4.4 Architecture of the Cray T3D Supercomputer

In this section a brief examination of the Cray T3D Supercomputer is given. The Cray T3D represents the first of a three phase project undertaken by Cray Research. The goal of the project is to attain a massively parallel processing system (or MPP) that can reach a sustained performance of one trillion floating point operations per second (1 Teraflop) on real customer codes [30]. The system is designed to support various styles of MPP programming such as work-sharing, data parallel and message passing. The Cray T3D comprises four types of components. These are the processing elements (PE), the interconnection network that connects the nodes, the Input/Output gateways and the system clock which operates at a period of 6.67 ns (or a frequency of 150 MHz). For our purposes, we will concentrate only on the PEs and the interconnection network [30].

4.4.1 Processing Element

The Cray T3D, as with any MPP computer system, contains hundreds or thousands of microprocessors each accompanied by some local memory. The combination of two microprocessors, local memory and some support circuitry make up a processing element. Depending on the system configuration, the Cray T3D contains between 32 and 2048 PEs [30].

The microprocessor in each PE is a 64 bit reduced instruction set computer (RISC) microprocessor developed by Digital Equipment Corporation capable of performing arithmetic and logical operations on 64-bit integer and 64-bit floating point registers. In addition, the microprocessor is furnished with data and instruction cache memory. The local memory is a dynamic RAM for system data storage and is connected to the microprocessor using a low latency and high bandwidth data path. The size of the DRAM varies from 2 Megawords to 8 Megawords (where each word is defined to be 8 bytes). The support circuitry aids the microprocessor by extending the control and addressing functions of the microprocessor and includes an interface to the rest of the network as well as a block transfer engine. System data is redistributed through the block transfer engine which simply consists of an asynchronous direct memory access controller [30].

4.4.2 Interconnection Network Topology

The interconnection network provides the connectivity among the PEs. Each connection between 2 PEs is a 24-bit bidirectional link. We will refer to a connection between two PEs as a *Cray channel*. The interconnection topology of the Cray is a 3-dimensional mesh. The mesh interconnection scheme interconnects PEs in each dimension in a bidirectional ring or torus as shown in Figure 4.4. In this manner, the lowest numbered PE in a dimension is directly connected to the PE with the largest number in the same dimension. Figure 4.4 shows a one dimensional torus network.

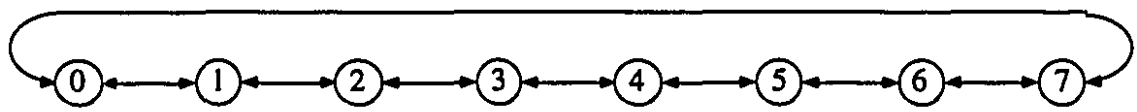

Figure 4.4 One dimensional bidirectional torus network

There are several advantages to this interconnection scheme. One such advantage is throughput of information. In Figure 4.4, PE 1 can communicate with PE 7 by passing through only one PE (PE 0). In the absence of the wrap around (the connection between PE 0 and PE 7), PE 1 would have to transmit through five PEs to reach PE 7. Additionally, the torus network offers some fault tolerance. Supposing, for example, PE 4 could not directly transfer data to PE 5 because of a bad channel between the two PEs. PE 4 could still transfer the data by sending it the long way around through all the other PEs in the torus [30].

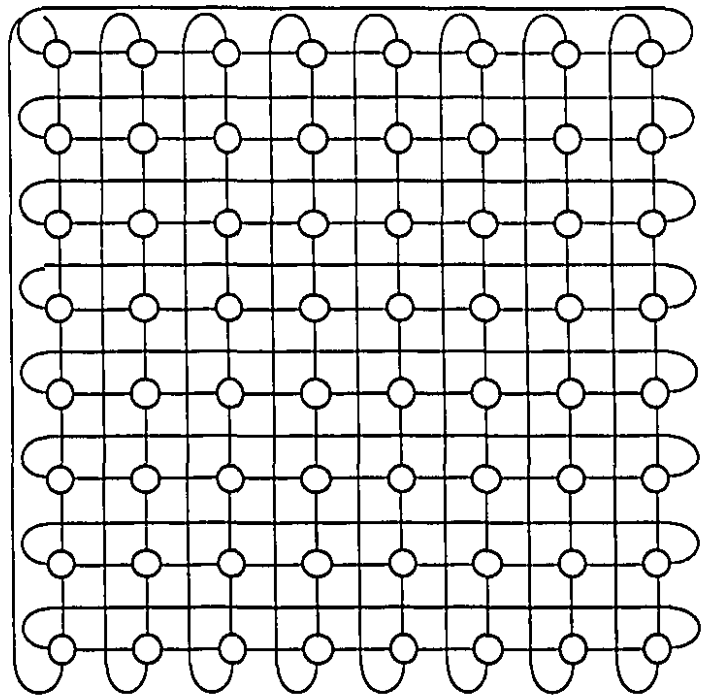

Figure 4.5 Two dimensional bidirectional mesh network

Figure 4.5 shows an 8x8 mesh. Although not shown, each Cray channel is bidirectional and is 24 bits wide in each direction. Figure 4.6 shows an 8x8x8 mesh. For clarity most of the wrap around links have been suppressed but it is important to keep in mind that these links still exist. The 3-dimensional mesh is a the network topology used in a 512 PE Cray T3D. It is this network that we will contract into various contracted networks and embed onto the Hyperplane.

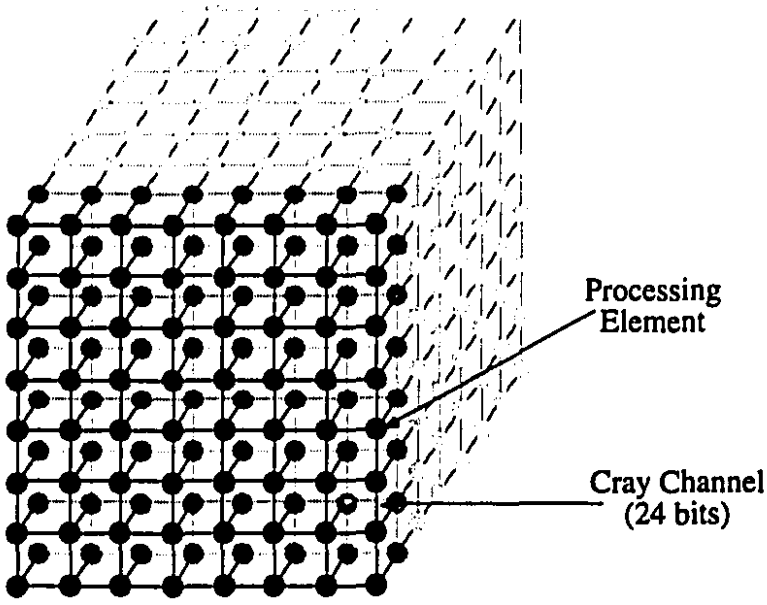

Figure 4.6 Network topology of Cray T3D Supercomputer (wrap around connections are suppressed for clarity).

4.5 Contraction and Embedding of the Cray T3D Interconnection Network

In this section we will present several possible schemes to contract the network topology of the Cray T3D Supercomputer (similar contraction schemes were described in [31]). We will contract the network topology using graph contractions described earlier in this chapter. Let the network shown in Figure 4.6 be represented by a graph $G(V,E)$ where V represents the set of PEs and E represents the set of Cray channels that connect the PEs. Recall that in the contraction process, our goal is to transform G into a simpler graph G' which will retain certain properties of G. To accomplish this we must first divide the set of vertices V into a set of mutually exclusive subsets such that the number of vertices in each set does not exceed a positive integer l. Recall that such a partitioning is called an *l-partition*. The manner in which V is partitioned is completely arbitrary and therefore leads to various contracted graphs. In the following, different contractions of the Cray T3D interconnection network will be presented.

4.5.1 Case 1: Contraction by Two Columns

The first contraction we will perform is contraction by two columns [31]. In this method, the set of vertices of G is partitioned into mutually exclusive subsets two columns at a time. That is, each subset represents two columns of vertices of G. Figure 4.7 shows G with one partitioned subset (two columns) highlighted in black [31]. Since there are a total of 512 vertices in G and each partition contains 16 vertices, there are a total of 32 mutually exclusive subsets (i.e. V_0, $V_1 \ldots V_{31}$). Each subset is then replaced by a separate contracted vertex v_i'. Neighbouring vertices are subsequently connected with edges. The width of each edge is then determined using the procedure described in Section 4.2.2.

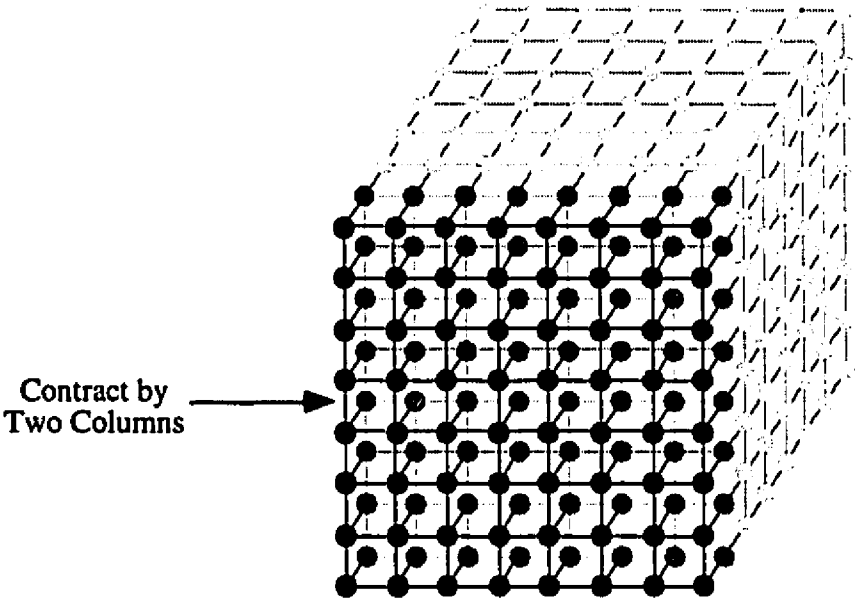

Figure 4.7 Contraction of G through two column partitioning.

The process leads to a contracted graph G' shown in Figure 4.8. Under this contraction scheme, we are left with 32 contracted vertices. The vertices have been numbered from 0 to 31 to aid us when we described the embedding procedure. In general, any number can be assigned to any vertex. However, as we will see, the best embedding

results generally arise when the numbering is ordered in the dimension of the fewest number of vertices. In this case, the vertices in dimension *d0* are numbered first. When there are no more vertices to number in *d0*, then we move up one step in direction *d1* and continue counting again in direction *d0* [31].

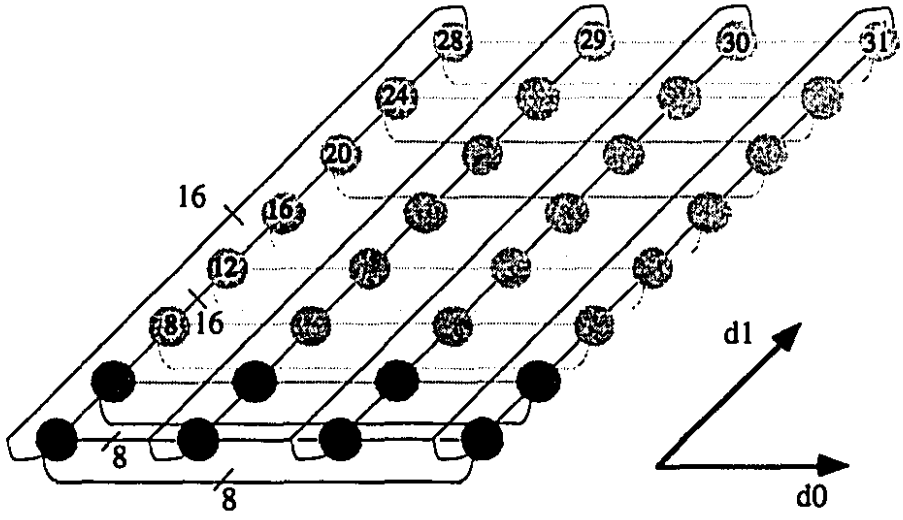

Figure 4.8 Result from contracting *G* by partitioning by two columns.

Figure 4.9 shows the embedding of the resulting contracted graph. The contracted graph has a total of 32 vertices, so the embedding shown in Figure 4.9 will have 32 nodes. Recall that each node in the embedding represents a PCB. Since there are a total of 512 PEs in *G*, each PCB will host 16 PEs.

Each connection between the contracted vertices of Figure 4.8 has a width of 8 Cray channels in the *d0* dimension and a width of 16 Cray channels in the *d1* dimension. In this context a Cray channel is a single *unidirectional* connection between two PEs in *G*. In addition, each Cray channel has a capacity of 24 bits. The optical connections of the embedding shown in Figure 4.9 represent the connections between the contracted vertices of Figure 4.8. To achieve this, we let each optical connection in the embedding represent 8 Cray channels. The choice to let each optical connection represent 8 Cray channels is mainly a matter of convenience. Therefore, each connection shown in Figure 4.8 that has a

width of 8 Cray channels will be represented by one optical connection in Figure 4.9. Moreover, each connection shown in Figure 4.8 that has a width of 16 Cray channels will be represented by two optical connections in Figure 4.9.

The upstream connections have the exact same connection scheme as the downstream. However the direction in which the data is transmitted in the upstream is opposite to that of the downstream.

In Figure 4.9 there is bisecting line that divides the embedding into two halves of equal size. We are interested in knowing the *bisection width* [5] of this embedding. The bisection width is defined, in this context, as the number of optical connections that pass through the bisecting line. The bisection width is a good indicator of the maximum communication bandwidth in the embedding. The communication bandwidth along any other cross section should be bounded by the bisection width [5].

In Figure 4.9, the bisection width is 18 optical connections in the upstream direction and another 18 optical connections in the downstream direction. The bisection width is useful in determining the total number of SPAs needed to implement the embedding. We will calculate the total number of SPAs needed later in this chapter.

4.5.2 Case 2: Contraction by 4x4 Clusters

An alternative way of contracting G is to partition V by selecting a 4x4 cluster of vertices as shown in Figure 4.10 [31]. The resulting contraction is a three dimensional graph again with a total of 32 vertices as shown in Figure 4.11. Most of the wrap around connections have been suppressed for clarity.

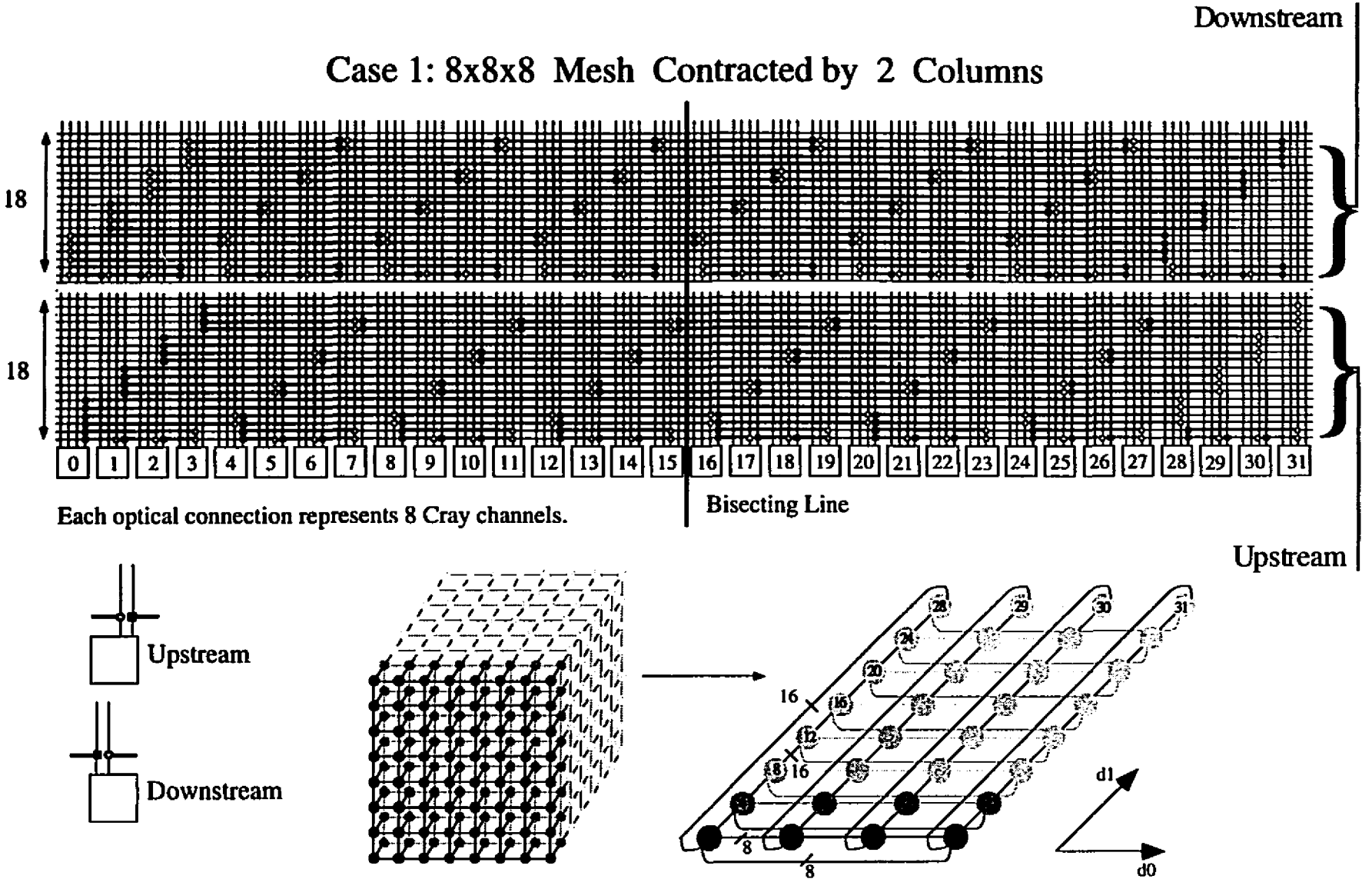

Figure 4.9: Embedding of G when contracted by 2 columns.

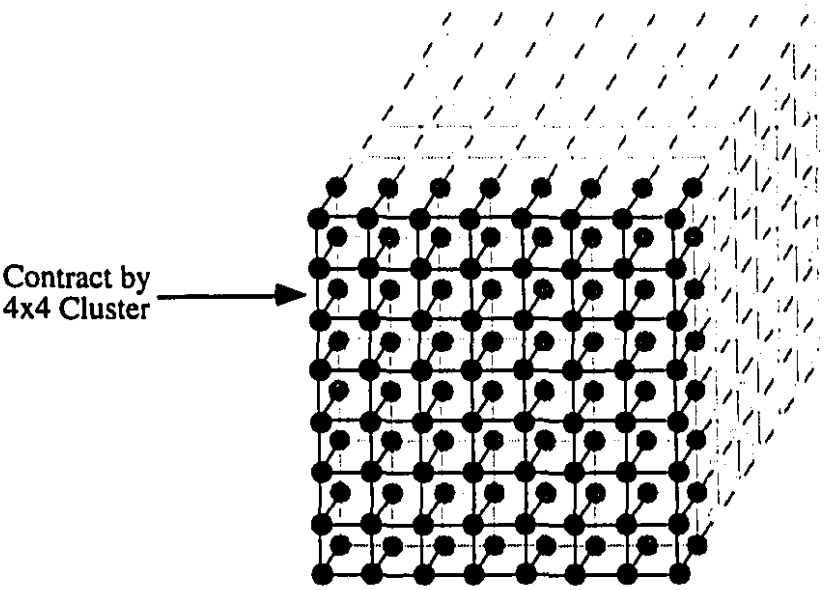

Figure 4.10 Contraction of *G* through 4x4 cluster partitioning.

Each connection between the contracted vertices of Figure 4.11 has a width of 4 Cray channels in the d0 and d1 dimensions and 16 Cray channels in the d2 dimension. The embedding of the contracted graph is shown in Figure 4.12. Each optical connection in the embedding represents 8 Cray channels. Notice, however that the connections in the d0 and d1 dimensions of Figure 4.11 have widths of 4 Cray channels. The question remains how are these connections (which are 4 Cray channels wide) to be represented in the embedding when each optical connection is 8 Cray channels wide. The solution to this is straightforward. For example, the connection between Vertex 1 and Vertex 3 of Figure 4.11 has a width of 4 Cray channels. However, recall that there is a wrap around edge between Vertex 1 and Vertex 3 that has been suppressed in the figure. Therefore, there are two connections (each having a width of 4 Cray channels) between Vertex 1 and Vertex 3 . As a result, a single optical channel in the embedding can be used to represent both connections. This is true for all the connections in dimensions d0 and d1 in Figure 4.11.

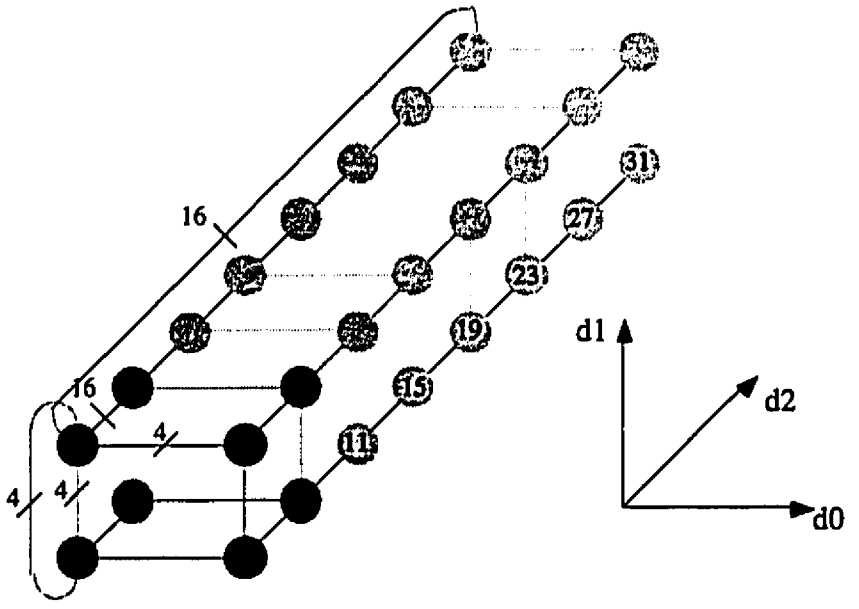

Figure 4.11 Contracted result from 4x4 cluster partitioning

Referring to Figure 4.12, the bisection width is 19 optical connections in the upstream direction and 19 optical connections for the downstream. Once again, the connection topology of the upstream is the same as that of the downstream. The only difference is that they transmit data in opposite directions thereby establishing the bidirectionality of the embedding.

Many of the optical connections can be shuffled between rows. By rearranging the optical connections between rows, we can compact the embedding which will reduce the bisection width. In Figure 4.13, the bisection width 18 is optical channels in the upstream and 18 in the downstream directions. This is slightly less than the uncompacted embedding shown in Figure 4.12.

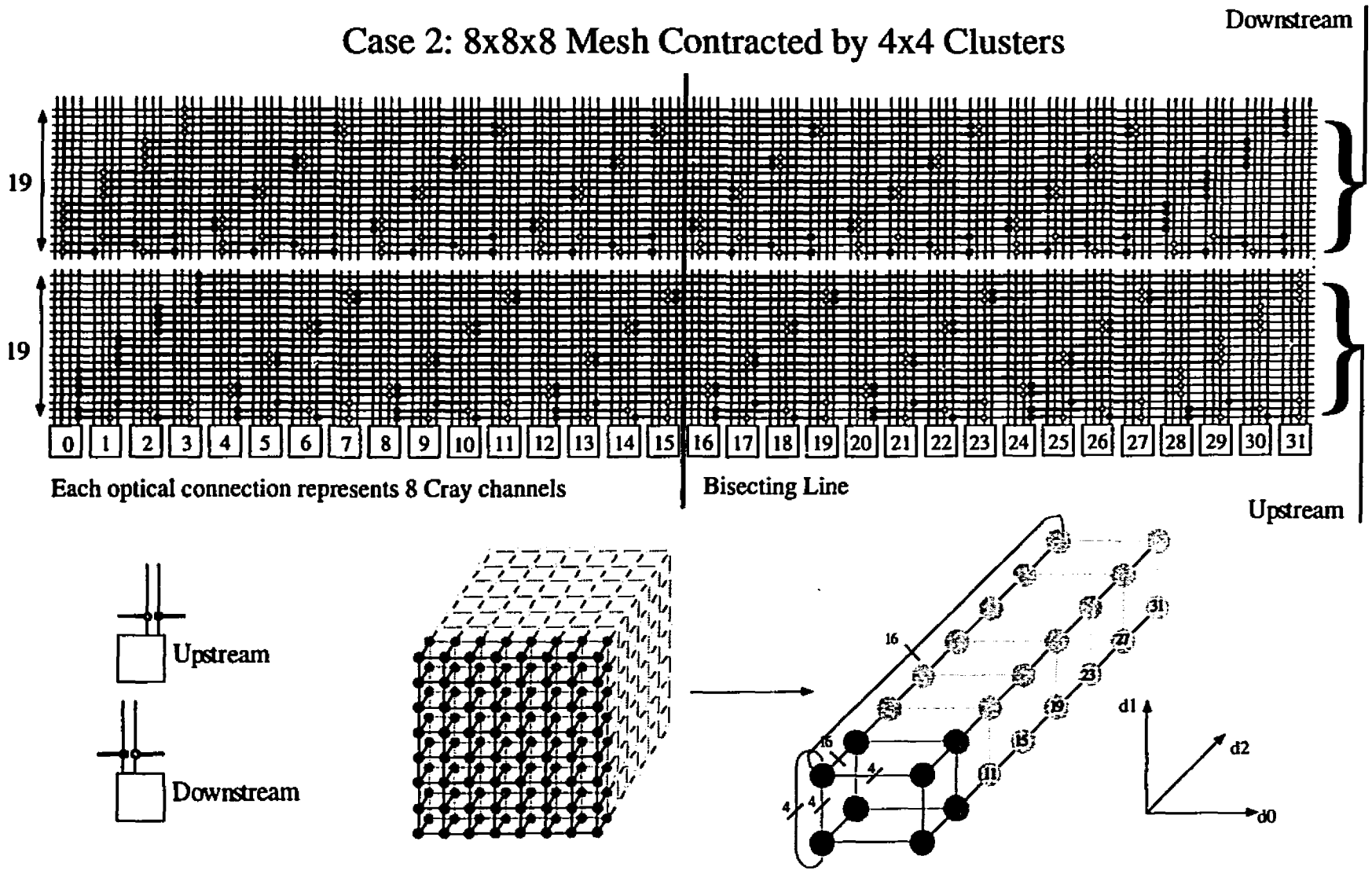

Figure 4.12: Embedding of G when contracted by 4x4 clusters

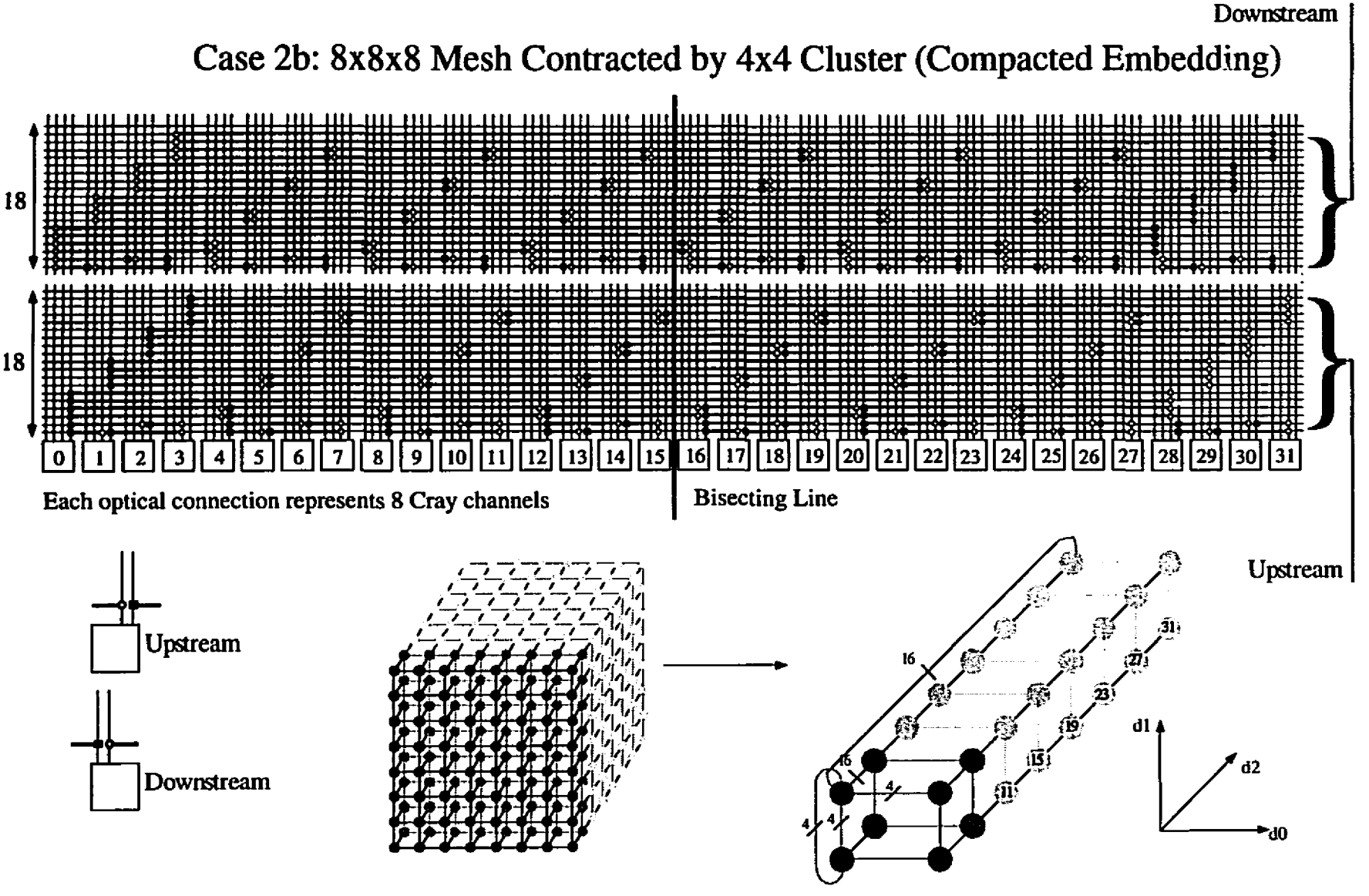

Figure 4.13: Compacted embedding of G when contracted by 4x4 clusters.

4.5.3 Case 3: Contract by 2x4x2 Cluster

The final example of contraction that we will perform is to contract by partitioning using a 2x4x2 cluster shown in Figure 4.14 [31]. The result is another 3 dimensional contracted graph with 32 contracted nodes as shown in Figure 4.15. Once again keep in mind that only two of the wrap around edges are shown. The rest of the wrap around edges have, in Figure 4.15, been suppressed for clarity. Each connection between the contracted vertices of Figure 4.15 has a width of 8 Cray channels in dimensions *d1* and *d2* and 4 Cray channel in the *d0* dimension. Figure 4.16 shows the embedding of the contracted graph. Each optical connection has a width of 8 channels. We are again confronted with the issue (first described in Section 4.5.2) that each optical connection in the embedding is 8 Cray channels wide while the connections in the *d0* dimension of Figure 4.15 are only 4 Cray channels wide. We resolve this issue in the same manner described in Section 4.5.2. The embedding has a bisection width of 21 optical connections in each direction. Of all the embeddings presented so far, this embedding has the highest bisection width.

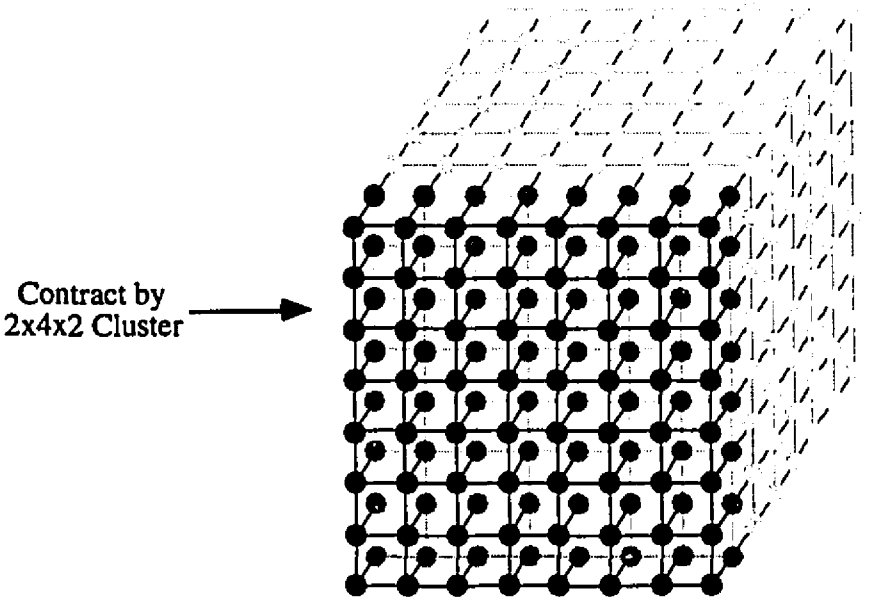

Figure 4.14 Contraction of *G* through 2x4x2 cluster partitioning.

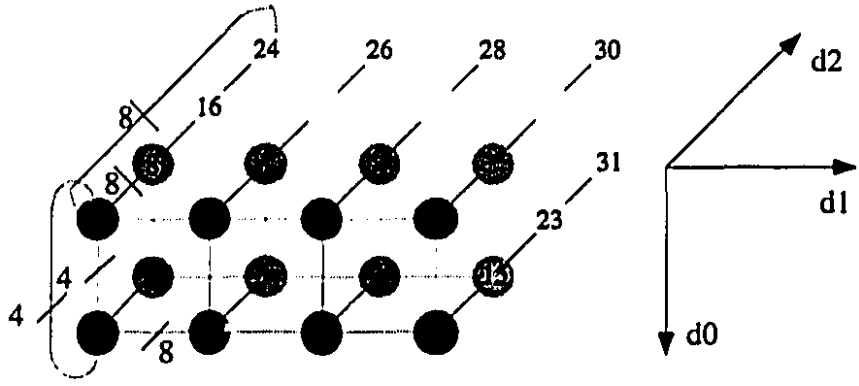

Figure 4.15 Contracted result from 2x4x2 cluster partitioning.

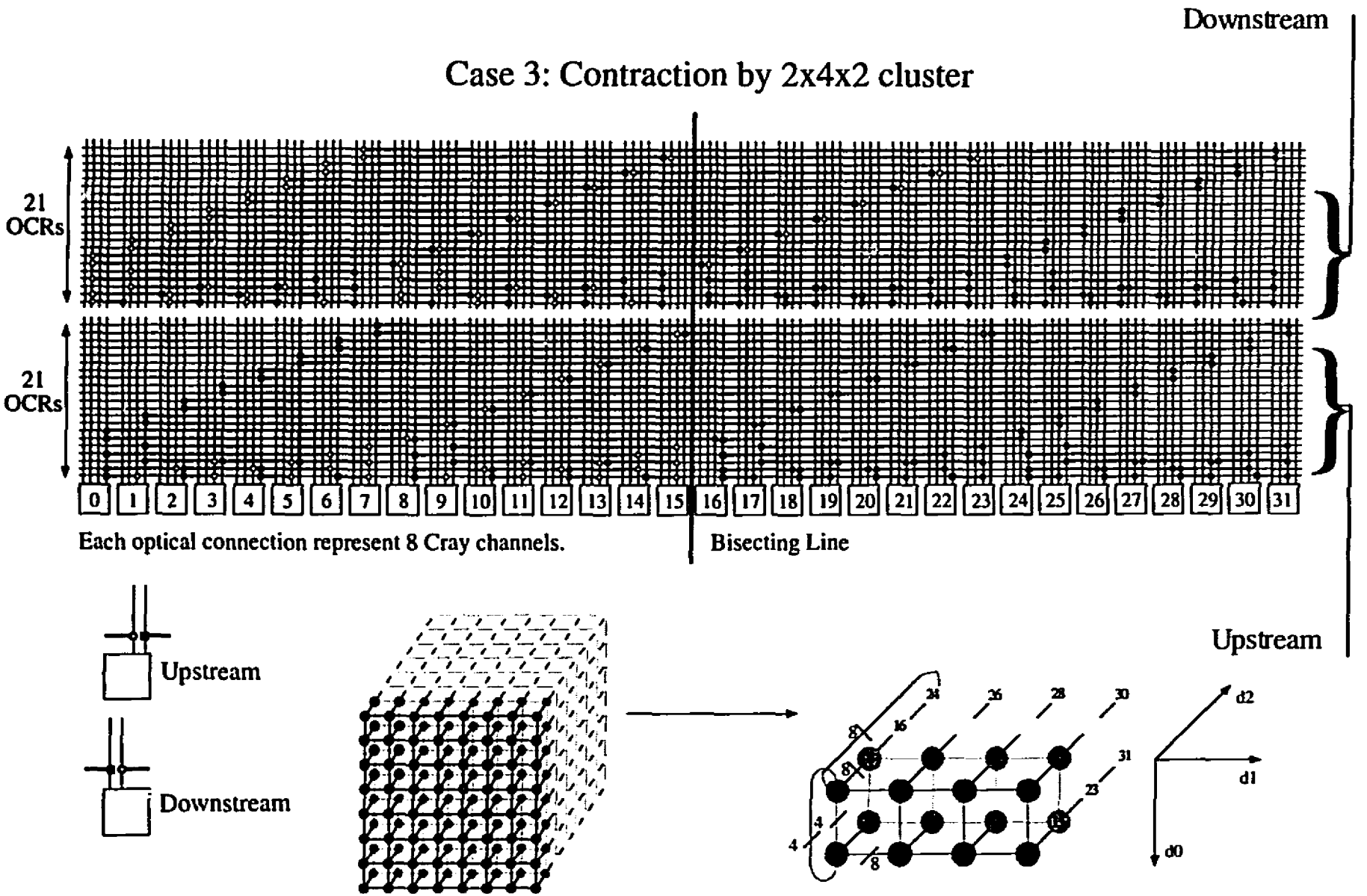

Figure 4.16 Embedding of G when contracted by 2x4x2 cluster.

4.6 Hardware Requirements for Implementation

In this section, the hardware requirements to implement the various embeddings described in the previous section will be discussed. In essence, we want to determine the total number of SPAs needed to implement each of the embeddings described. The purpose of this exercise is to demonstrate that the Cray T3D Supercomputer can be implemented using the Hyperplane. Recall that in each embedding, we calculated its bisection width. As we will see in this section, the bisection width will help us determine the total number of SPAs needed. We measured the bisection width in terms of the number of optical connections that passed through the bisecting line. In each embedding, an optical connection represented 8 Cray channels. Recall that in this context, a Cray channel is a single unidirectional connection between two PEs in the Cray T3D. Each Cray channel is 24 bits wide. Since each Cray channel is 24 bits wide, each optical connection is 8×24 or 192 bits (24 bytes) wide. Given this, our next task is to calculate, for each embedding scheme, the total number of smart pixel arrays (SPAs) necessary to implement that embedding.

Each SPA (as described in Chapter 3) is an array of pixels which can be divided into logical optical channels each 8 bits wide [13]. Because these logical optical channels are 8 bits wide, we refer to them as *byte channels* [13]. The number of byte channels that comprise a SPA is somewhat arbitrary. However, with existing CMOS technology, the number of byte channels would typically be between 32 and 512. The SPA design described in Chapter 3 supports 128 byte channels.

By determining the number of byte channels needed for a given embedding, we can calculate the number of SPAs needed. We assume that the SPAs are completely imaged from one to another. This means that the SPAs on each PCB must be able to support the bisection width (which represents the maximum communication bandwidth along any cross section of the embedding). In addition, each PCB must be interchangeable. Therefore each PCB will have the same number of SPAs connected to it. The number of byte channels needed is determined by the bisection width. For example, the embedding for

Case 1 (shown if Figure 4.31) has a bisection width of 18 optical connections in each direction (or a total of 36 in both directions). Since each optical connection is 24 bytes wide, each optical connection will require 24 byte channels. Since the embedding for Case 1 has a bisection width of 18 optical channels in each direction, a total of 18×24 (or 432) byte channels per PCB will be needed in each direction. Table 4.1 summarizes the bisection width, the number of Cray channels and the number of byte channels needed in each direction for all four embeddings.

Embedding	Bisection Width (optical connections)	# of Cray Channels	# of Byte Channels
Case 1	18	144	432
Case 2	19	152	456
Case 2b	18	144	432
Case 3	21	168	504

Table 4.1: Resource requirements for various contraction schemes.

Now that the total number of byte channels needed (in each direction) has been calculated, our next objective is to determine the number of SPAs per PCB needed. The number of SPAs per PCB is dependant on the number of byte channels comprising each SPA. Let's look at Case 1 as an example. Suppose each SPA comprises 32 byte channels. Since we require that each PCB accommodate 432 byte channels, then 13.5 SPAs in each direction are needed to implement this embedding. Since we cannot have half of a SPA, we actually need 14 SPAs in each direction. However, half of the fourteenth SPA will be unused. These unused channels could be used to provide fault tolerance. Table 4.2 summarizes the number of SPAs needed per PCB as the number of byte channels comprising each SPA varies from 32 to 256.

Embedding	# of SPAs/PCB 32 byte channels	# of SPAs/PCB 64 byte channels	#of SPAs/PCB 128 byte channels	# of SPAs/PCB 256 byte channels
Case 1	14	7	4	2
Case 2	15	8	4	2
Case 2b	14	7	4	2
Case 3	16	8	4	2

Table 4.2: Number of SPAs (in each direction) needed per PCB as the number of byte channels per SPA varies.

Now that the number of SPAs per PCB has been calculated it is useful to see which embedding and which SPA make most efficient use of the available resources. Recall that when we calculated the number of SPAs needed for the Case 1 embedding (using 32 byte channels per SPA), we needed 13.5 SPAs per PCB. Since we can't have half of a SPA, we needed to round that value up to 14. As a result, 16 byte channels (or 3.7 % of the total byte channels required) are unused. Table 4.3 tabulates the number of unused byte channels for different SPAs and different embeddings. The percent values in the table represent the total percentage of unused byte channels.

Embedding	Byte channels unused (32 byte channel SPA)	Byte channels unused (64 byte channel SPA)	Byte channels unused (128 byte channel SPA)	Byte channels unused (256 byte channel SPA)
Case 1	16 (3.7%)	16 (3.7%)	80 (18.5%)	80 (18.5%)
Case 2	24 (5.3%)	56 (12.2%)	56 (12.2%)	56 (12.2%)
Case 2b	16 (3.7%)	16 (3.7%)	80 (18.5%)	80 (18.5%)
Case 3	8 (1.6%)	8 (1.6%)	8 (1.6%)	8 (1.6%)

Table 4.3: Tabulation of the number of byte channels unused. The percentages in parentheses represent the total percentage of byte channels not used.

The values tabulated by Tables 4.1 to 4.3 give us an idea which of the embeddings and which of the SPAs are better to use. When interpreting the data presented in these tables, there are two criteria that should be kept in mind. In general, it is preferable to use the embedding that uses the fewest number of SPAs. However, it is also preferable to use

those SPAs which yield the highest percentage of unused byte channels. The more unused byte channels we have, the more fault tolerance we can give the system.

Table 4.2 shows that Case 3 is inefficient because it uses the most number of SPAs. In addition, Table 4.3 shows that Case 3 has the lowest percentage of unused byte channels. Therefore, Case 3 does not seem to match our criteria. Case 1 and 2b seem to be the better embeddings because they require the fewest number of SPAs (Table 4.2). Furthermore, if we implement these embeddings using 128 or 256 byte channel SPAs, we will get the highest percentage of unused byte channels leading to the maximum amount of fault tolerance.

4.7 Chapter Summary

In this chapter, we presented various embeddings of a Cray T3D Supercomputer on to the Hyperplane. We commenced the chapter by introducing a method of graph contraction and used this method to reduce the network topology of the Cray T3D so that it could be embedded onto the Hyperplane. We contracted the Cray T3D topology in 3 different ways and then presented their embeddings onto the Hyperplane. Using these embeddings, we calculated the total number of SPAs needed to implement these embeddings as the number of byte channels comprising each SPA varied. We also calculated the percentage of unused byte channels for the various embeddings and SPAs. We noted that the in general, we would prefer those embeddings that used the fewest number of SPAs. Furthermore, we preferred those SPAs that yielded the highest percentage of unused byte channels. The more unused byte channels available, the more fault tolerance we can give to the system. Using the values tabulated in Tables 4.1 to 4.3, we determined that Case 1 and Case 2b used the fewest number of SPAs. We also noted that these embeddings are best implemented using 128 or 256 byte channel SPAs since they yield the highest percentage of unused byte channels (and therefore the maximum fault tolerance).

Chapter 5
Functional Specifications of the June 1995 Backplane

5.1 Introduction

In the previous chapter, we demonstrated that the dual-stream linear Hyperplane could be used to interconnect the processors of a Cray T3D Supercomputer. In order to accomplish this, the functional specifications of the Hyperplane have been evolving over the past several years. One of the long term goals of the Major Project is to develop monolithic Smart Pixel Arrays using Indium Phosphide based VCSEL/MSM technology [1]. However, as we mentioned in the Introduction, demonstrators are constructed periodically to highlight milestones achieved. Because Indium Phosphide technology is still evolving, the SPAs fabricated thus far are based on CMOS SEED technology. In this chapter, we will review the design specifications for the June 1995 Backplane (also known as the June 1995 Hyperplane) and the June 1995 SPA which are based on existing CMOS technology. We use a time stamp in the design names to emphasize the fact that these designs are constantly evolving. The complete structural specifications for the June 1995 Hyperplane can be found in [31]. The functional specifications specify functions or behaviours without necessarily specifying structural implementations.

5.2 June 1995 Hyperplane

Figure 5.1 shows a typical representation of the June 1995 Hyperplane[2]. The number of PCBs connected could vary as the design evolves. For our purposes here, we will assume that there are four PCBs connected. The Hyperplane primarily operates in one of two modes. These modes are called the *Reconfigurable Mode* and the *Intelligent Mode* [12][13]. These modes were described earlier in Chapter 3. Reconfigurable mode could be used for massively parallel processing systems for example and intelligent mode could be used for telecommunications applications.

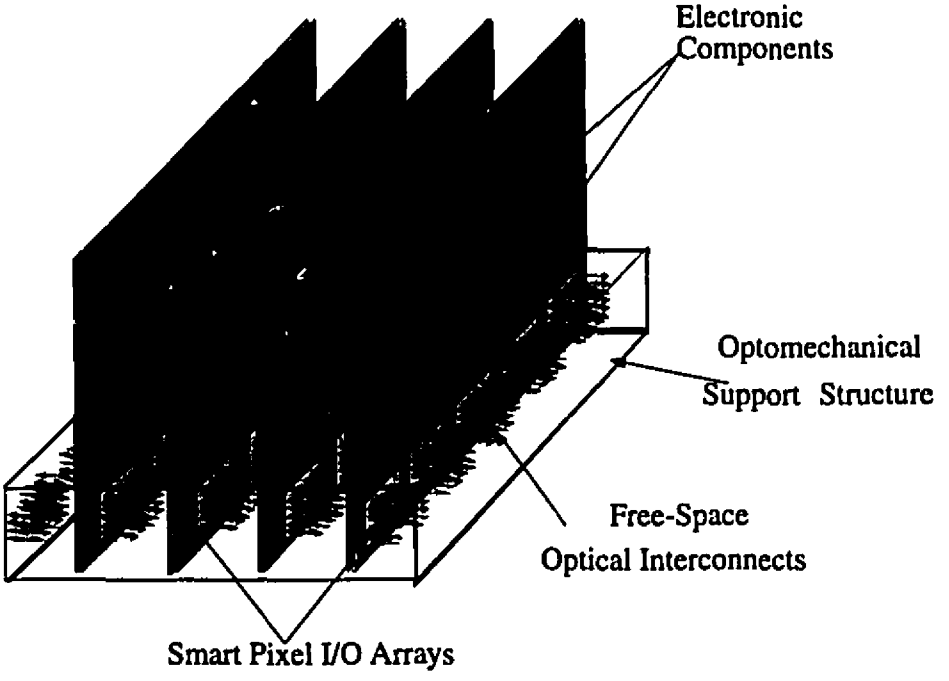

Figure 5.1 Typical architecture of the June 1995 Hyperplane.

The architecture of the Hyperplane can be implemented using either the Circular structure or the Dual Stream Linear Structure. Most of the designs reviewed in this chapter

---

[2] From here on, the term *Hyperplane* will refer to the June 1995 Hyperplane and the term *Smart Pixel Array (SPA)* will refer to the June 1995 SPA.

can support either structure. For brevity, we will focus only on the circular structure. Recall that PCBs wishing to send data will connect to one of the optical communication channels [12] through its associated SPAs. Data is grouped together in the form of packets. A PCB wishing to send a packet will divide the packet into bytes and send the packet through the backplane one byte at a time. At the rising edge of each clock, the PCB will typically insert a new byte into the backplane through the SPAs. At the same time, the bytes circulating in the backplane will typically move from one PCB to the next at every rising edge of a clock . That is during each clock cycle, the byte will move from *PCB j* to *PCB (j+1) mod 4* until it reaches it point of origin where it will be overwritten or discarded. When operating in the intelligent mode, each packet will have a header that will identify its destination. The logic on each SPA will process the packet headers to decide whether to extract the packet as they travel through the backplane. When the Hyperplane is operating in the reconfigurable mode, the SPAs do not perform any packet processing or filtering [13].

5.3 Smart Pixel Array Interface Signals

The Hyperplane is composed of a large number of optical communication channels (OCCs) which are created by optically interconnecting smart pixel arrays (SPAs). Figure 5.2 shows the interface signals of the SPA [31].

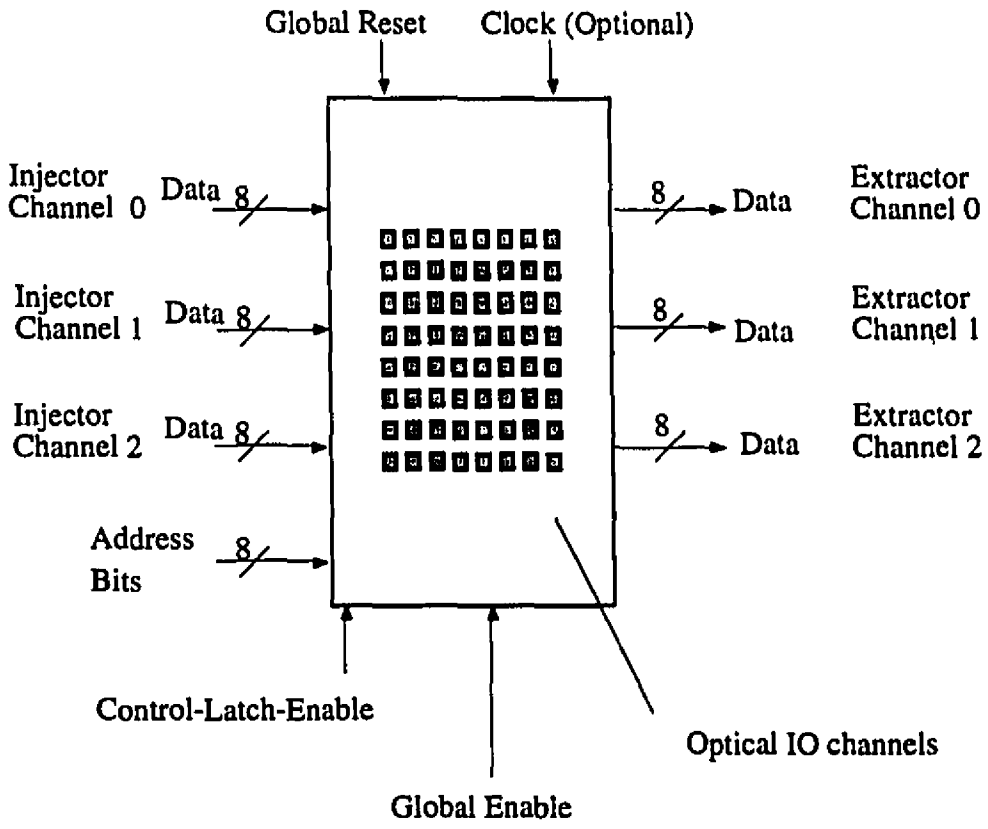

Figure 5.2 Typical electrical interface signals for the June 1995 SPA (3 injector channels, 3 extractor channels and 8 optical byte channels shown).

For our purposes, interface signals refer to the electrical connections of the SPA that would typically be connected to its associated printed circuit board. The optical windows are also shown. Table 5.1 tabulates a brief description of the interface signals [13].

Electronic I/O Signal	Signal Description
Clock	Global synchronizing clock signal (typically 80 MHz.).
Global Reset	Resets entire system to "idle" state.
Injector Channel x (x=0,1,2)	Eight bit wide input channel for data transmission.
Ctl Latch Enbl.	Configures control latch.
Global Enable	Initiates data transmission and reception.
Address Bits	Eight bits used to differentiate between SPA boards.
Optical I/O	8x8 array of optical windows optically connecting boards.
Extractor Channel (x=0,1,2)	Eight bit wide output channel for data reception.

Table 5.1: Electrical interface signals of the June 1995 SPA.

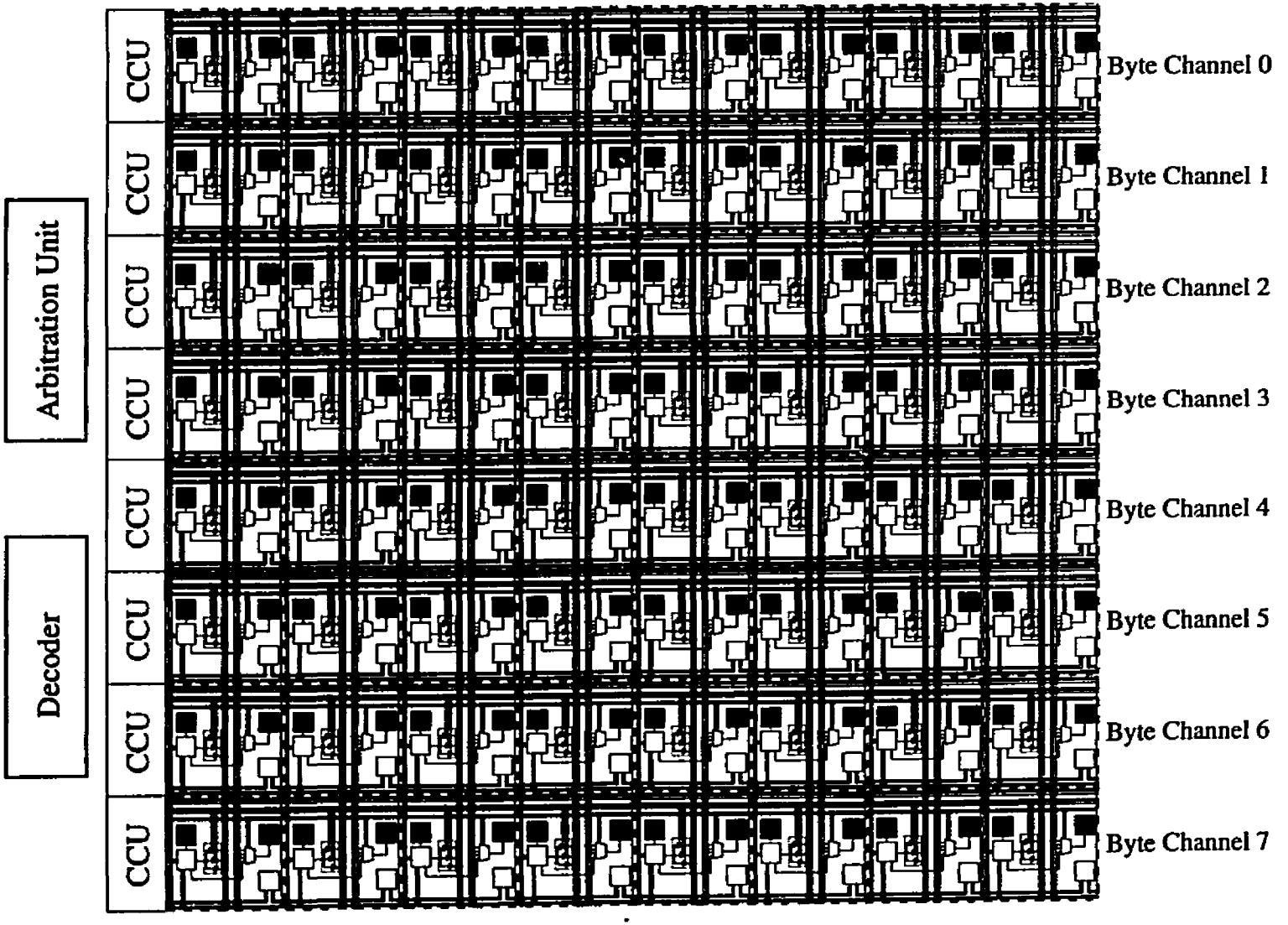

Figure 5.3: Functional diagram of an 8x8 Smart Pixel Array [13].

Figure 5.3 shows a functional schematic of the June 1995 SPA [31]. It consists of an 8x8 array of smart pixels which is partitioned into 8 logical optical channels. Each logical optical channel is 8 pixels (bits) wide and is therefore referred to as a byte channel [13]. Therefore, the connection set by the byte channels establishes an 8 bit data path. However, as we will see in a later section, if the packets are being submitted asynchronously, then the SPA will need to be a 9x8 array of pixels. The difference between synchronous and asynchronous packet transmission was discussed in Chapter 3.

Each byte channel has an associated channel control unit (CCU) and eight smart pixels. Figure 5.4 shows the top left quadrant of Figure 5.3 [31]. In this figure only four pixels per byte channel and four byte channels are shown for clarity. The CCU is used to regulate the flow data through the channel. It consists primarily of an 8 bit control latch configured before the Hyperplane begins transmitting and receiving data. The control latch sets certain parameters that determine the way the channel will operate. In later sections we will describe how the control latch accomplishes this. The number of pixels and the number of I/O electronic pins that each SPA can accommodate is limited by the fabrication technology of SPAs. It is expected however, that 32x32 SPA chips will be available in the near future. Each smart pixel consists of an optical input port and an optical output port. Therefore, each pixel is capable of handling one bit at any given time. Data is entered through one of the three *Injector Channels* and is transmitted over the backplane over a single byte channel. In addition to the CCU and smart pixels, the SPA consists of a decoder and an arbitration circuit. The decoder is used to configure the control latch in each CCU. In this way, the user can specify how the backplane will behave.

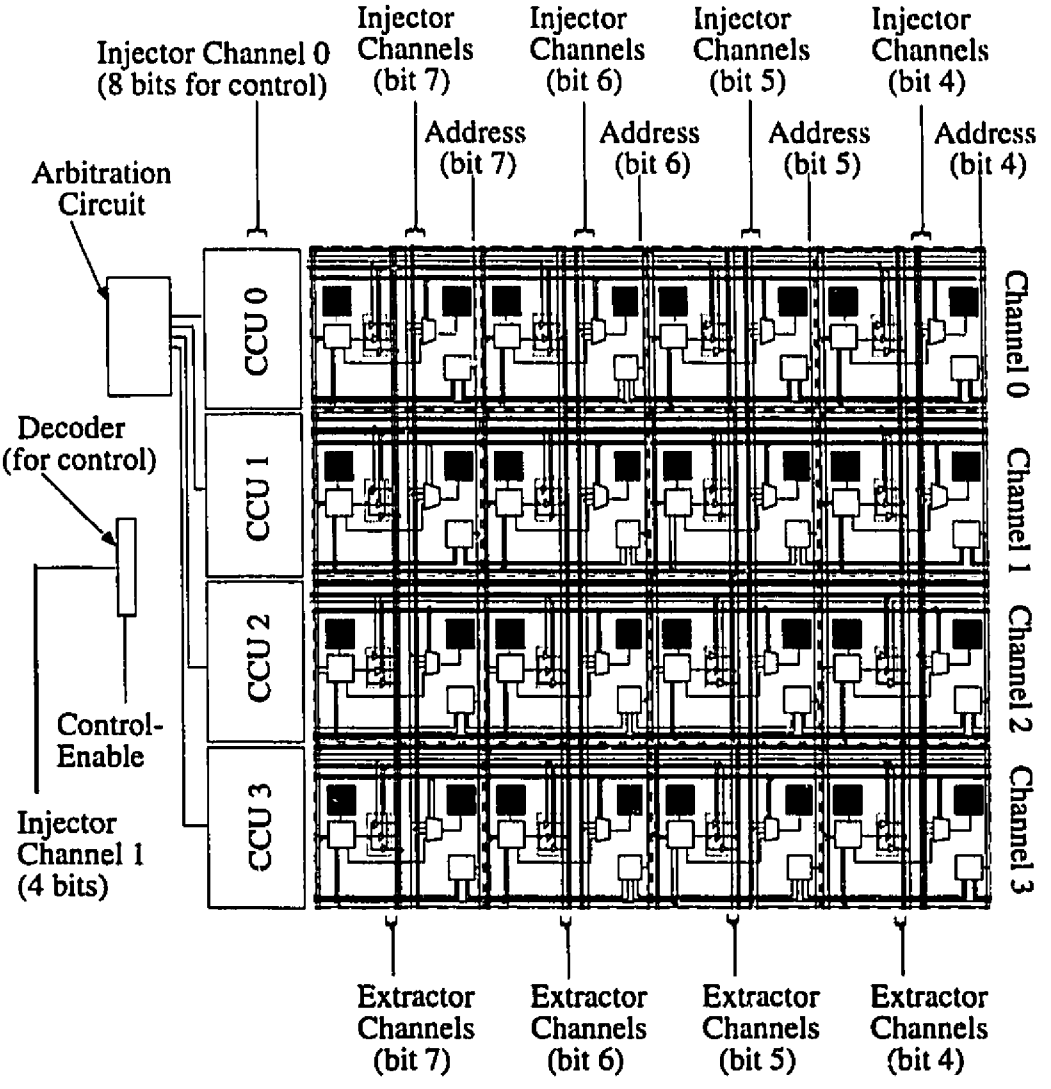

Figure 5.4 Functional logic description of the top left quadrant of Figure 5.3

5.4 Smart Pixel Cell and Channel Specifications

Figure 5.5 shows the functional block diagram of a single smart pixel [13]. The Injector Selector Circuit is a mux that will select which bit to transmit optically to the next board. The injector selector control is controlled by the injector selector control bits that come from the control latch in the CCU. The injector selector control will select a bit from

one of the three Injector Channel or the Optical Input Port. The Extractor Selector Circuit consists of three tristate buffers used to select which of the three Extractor Lines to extract the incoming data. The extractor selector control which consists of three tri-state buffers is controlled by the extractor selector control bits. If the backplane is operating in reconfigurable mode, then the extractor selector control bits come from the control latch in the CCU. If the backplane is operating in the intelligent mode, then the extractor selector control bits will come from the arbitration unit (the CCU and the arbitration circuit will be discussed in later sections).

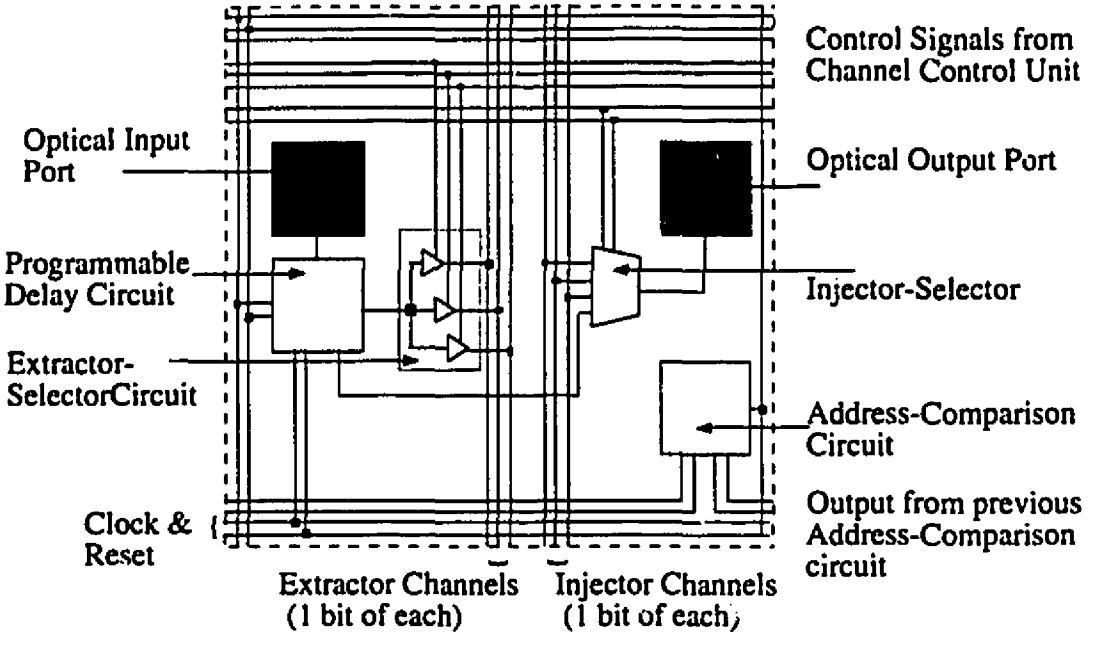

Figure 5.5 Logic diagram of a single smart pixel.

5.4.1 Address Comparison Circuit

When operating in the intelligent mode, each packet sent through the Hyperplane has a header that encodes the address of the packet's destination. When an byte channel receives a header, it must read the header and decide whether it should extract this packet and send it to its associated PCB. The address comparison circuit is responsible for

making this decision. In this section, we will describe the operation of the address comparison circuit. Figure 5.6 shows the header of a packet when the packets are synchronously transmitted. Each board has its own unique 5 bit address that separates it from the other boards. Bits 0 through 4 of the header are used to encode the address of the boards. Since the board address is five bits long, the maximum number of boards that the Hyperplane can accommodate is 32. However, for the purposes of the June 1995 Hyperplane, only four boards will be used. Bits 5, 6 and 7 designate groups of boards. This will allow transmission to boards that belong to a certain group. After the header has been sent, all subsequent bytes transmitted are the packet payload (data).

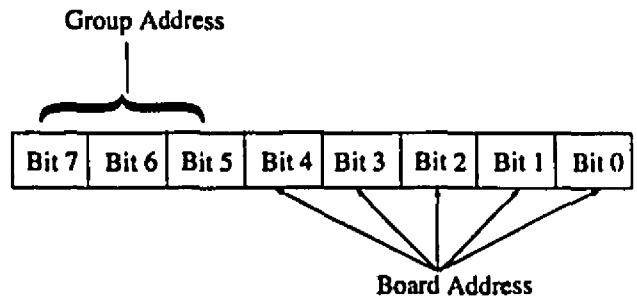

Figure 5.6 Typical packet header used for synchronous packet transmission.

In asynchronous packet transmission, a PCB can begin transmitting a packet at the rising edge of any clock cycle. This is in contrast to the synchronous packet transmission in which every PCB must begin transmitting its packets at the same time. However, in asynchronous packet transmission, the address comparison circuit of each byte channel must be able to differentiate between a header and data. Since packet can be transmitted at any rising edge of the clock, the address comparison circuit will not know whether the byte it is receiving is a header or data. One way to resolve this is to increase the width of the byte channel so that it is 9 bits wide. Figure 5.7 shows a typical header of a packet when the packets are asynchronously transmitted. The most significant bit of this header is a Valid Header bit. The sole purpose of this bit is to give the address comparison circuit some means to differentiate between the header and the data. By increasing the width of the byte channels to 9 bits, we maintain the 8 bit data path.

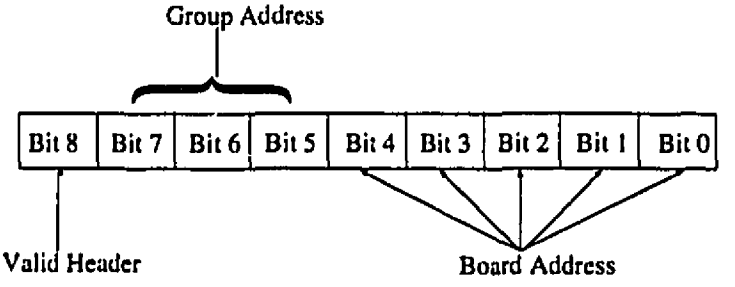

Figure 5.7 Typical packet header used for asynchronous packet transmission.

Figure 5.8 shows a schematic of a typical address comparison circuit found in each smart pixel while Figure 5.9 shows a block diagram that describes a typical interconnection of address comparison circuits within an byte channel when the packets are synchronously transmitted. The address comparison circuit for bits 0 through 4 (board address comparison) will check to see if the board address matches while the address comparison circuit for bits 5, 6 and 7 will check if there is a group address matches (group address comparison).

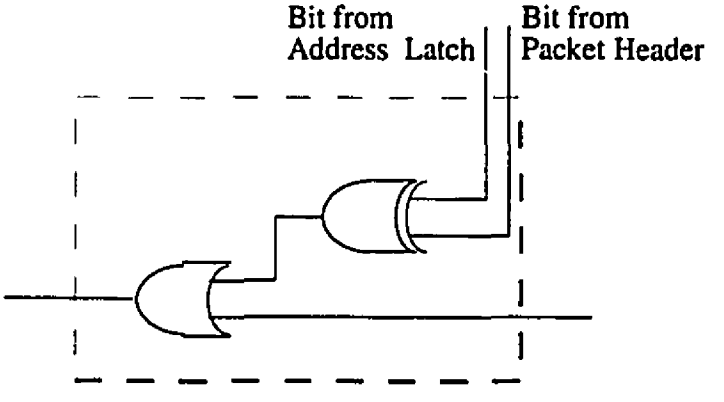

Figure 5.8 Address comparison circuit.

The results of the comparison are sent to the CCU. If either (or both) of the comparisons match, then the byte channel will extract this byte and all subsequent bytes until the entire packet has been received.

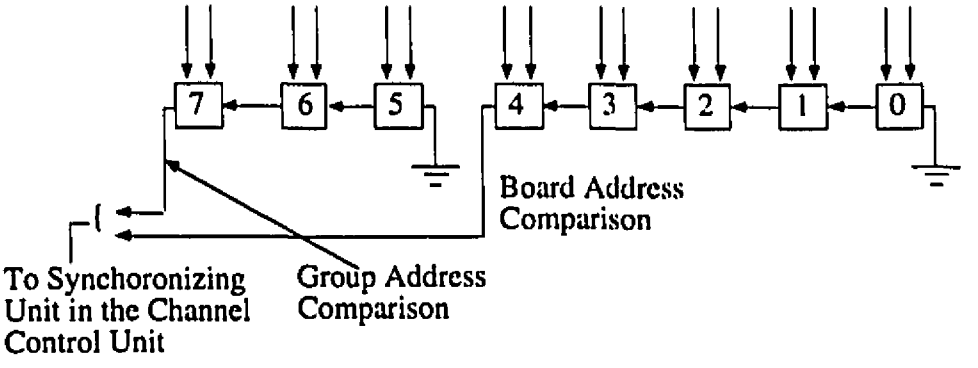

Figure 5.9 Interconnection of the address comparator circuits within a byte channel (packets are transmitted synchronously).

Figure 5.10 shows the interconnection of the address comparator circuit when the packets are sent asynchronously. This works in the exact same manner as the circuit shown in Figure 5.9 except for the last comparator which compares the ninth bit of the header to check whether the byte received is a header or just data. It must be pointed out that the ninth bit in the header does not constitute any part of the data path established by the byte channels.

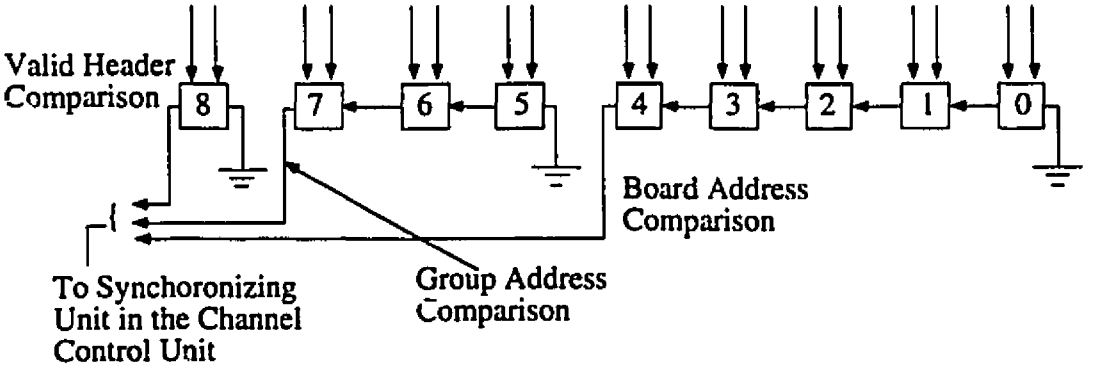

Figure 5.10 Interconnection of the address comparator circuits within a byte channel (packets are transmitted asynchronously).

5.4.2 Programmable Delay Circuit

In addition to the address comparison circuit, each smart pixel is equipped with a programmable delay circuit. The schematic of the programmable delay circuit is shown in Figure 5.11.

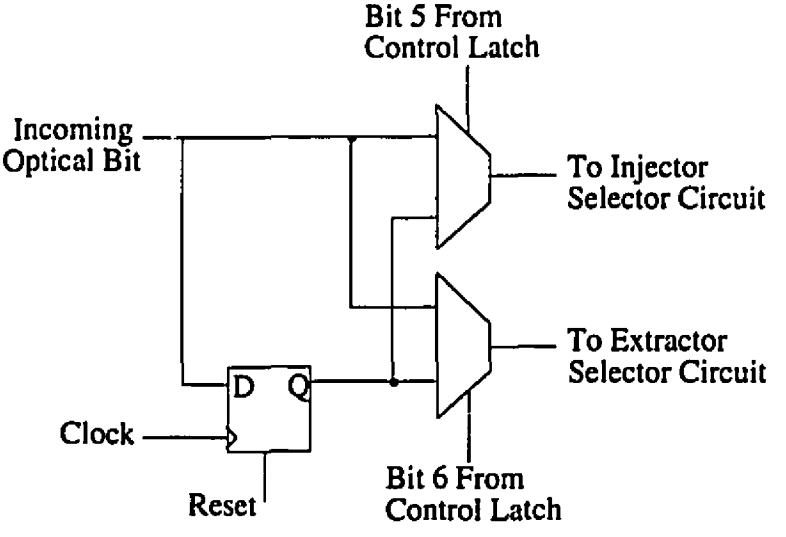

Figure 5.11 Programmable delay circuit.

Depending on the configuration settings of the control latch (in the CCU), the programmable delay circuit will either route data entering it directly to the output or delay it by one clock cycle (using the D-Flip Flop). The multiplexers are used to select whether the data should be delayed or not. The multiplexers are controlled by the control latch in the CCU. The programmable delay circuit enables the backplane to have buffering capabilities.

5.5 Channel Control Unit

Associated with each channel is a *Channel Control Unit* (CCU). Its primary function is to regulate the flow of data through a channel. Typically, the CCU consists of a control latch and a synchronizing unit as shown in Figure 5.12. The functionality of the CCU depends on whether the backplane is operating in the intelligent mode or reconfigurable mode. If the backplane is operating in the intelligent, then the CCU will provide the interface between byte channels and the arbitration circuit. A full description of the arbitration circuit will be given later in this chapter.

The control latch in the CCU is an 8 bit latch that is used to configure the manner in which the Hyperplane will operate. The most significant bit of the control latch determines whether the backplane will be operating in the intelligent mode of the reconfigurable mode. The next two bits configure the programmable delay circuit (this circuit will be described in a later section). The next two bits form the Injector Selector Control bits discussed in the last section. The last three bits are the Extractor Selector Control bits. These three bits are only used if the backplane is operating in the reconfigurable mode (notice that the mux in the channel control unit will select whether the extractor selector control bits come from the control latch or the arbitration circuit).

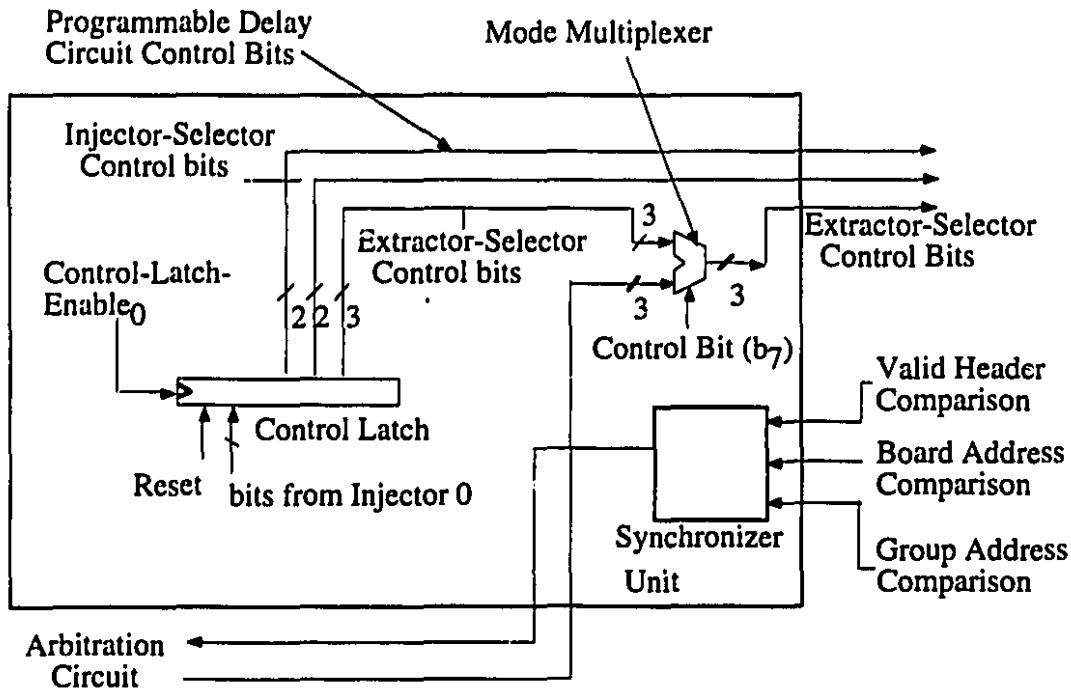

Figure 5.12 Functional description of a typical Channel Control Unit.

In this particular design of the CCU, the synchronizing unit is only used when the backplane is running in the intelligent mode of operation. Notice that the signals from the address comparator circuits of Figures 5.9 or 5.10 are fed directly to the synchronizing circuit. The design of the CCU shown in Figure 5.12 is the CCU used when the backplane transmits packets asynchronously (notice the valid header comparison signal). With minor adjustments (eg. removing the valid header comparison input signal), the CCU can be modified to operate in a backplane where the packets are transmitted synchronously.

Figure 5.13 shows the schematic of the synchronizing circuit. This circuit is only used when the backplane is operating in the intelligent mode. The inputs to the synchronizing circuit are signals from the address comparator circuit. The Valid Header Comparison signal is only used when the backplane is transmitting packets asynchronously. If a matching header arrives at a byte channel, the output signal (called *Req*) will be asserted. The request signal will be sent to the arbitration circuit.

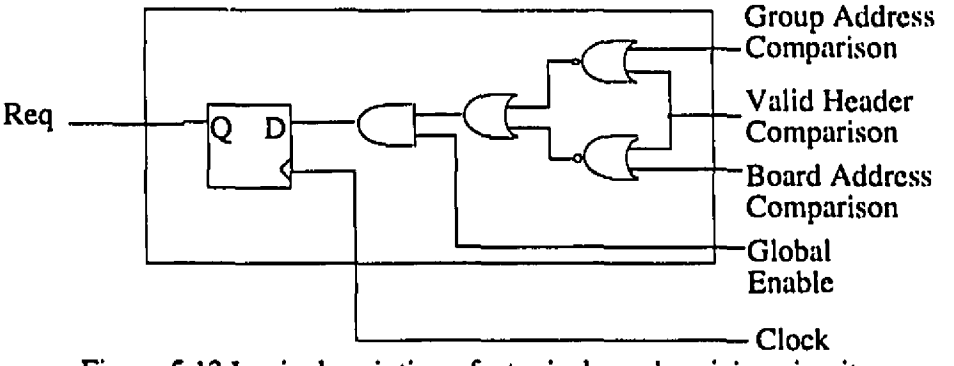

Figure 5.13 Logic description of a typical synchronizing circuit.

5.6 Arbitration Circuit

One of the more complex components of the SPA is the arbitration circuit[3]. The arbitration circuit is used only when the SPA is operating in intelligent mode. There are several issues associated with the reception of data. Recall that there are typically eight byte channels per SPA but only three extractor channels. A SPA, on occasion, may attempt to extract more packets than the number of extractor channels available (recall, there are 3 extractor channels per SPA in the June 1995 design) [13]. In this case only it is important that only three of these packets are extracted at any given time. In order to ensure this, an arbitration circuit is used to map the byte channels onto the extractor channels [13]. All other requests for extraction will be ignored. The mapping scheme used by the arbitration circuit will be described in the following sections.

5.6.1 Overview of the Arbitration Circuit

Figure 5.14 shows a typical functional block diagram of the arbitration circuit. The arbitration circuit consists of two components. The first is an arbiter and the second is a set of counters. The arbiter has inputs which are connected to the synchronizing circuit of each byte channel (the output signal *Req* in Figure 5.13 connects to one of the input signals of

[3] The arbitration circuit described in this chapter is one structural implementation (developed by this author) of the generic arbitrator described in [31].

the arbitration circuit). The output of the arbiter consists of 8 signals each 3 bits in width. Each output signal connects to the extractor selector control bits of a byte channel.

When the arbiter receives the *Req* from one of the synchronizing circuits, it checks to see whether there are any free extractor channels. If none of them is free then the packet will not be received. If there is a free extractor channel, then the arbitration circuit will assign the extractor channel to the byte channel. The arbitration circuit does this by appropriately setting the byte channel's extractor selector control bits. At the same time, the arbiter will send a start signal to one of the counters. The counter will increment at the rising edge of every clock pulse. The counters are used to track the number of words received by the channel. The size of each packet is somewhat arbitrary. If the Hyperplane is used in an ATM application then each packet would be 53 bytes in length. In this implementation of the Hyperplane, we have assigned each packet to be 16 bytes (including the header) in length. Therefore, each counter in the arbitration circuit is a four bit counter.

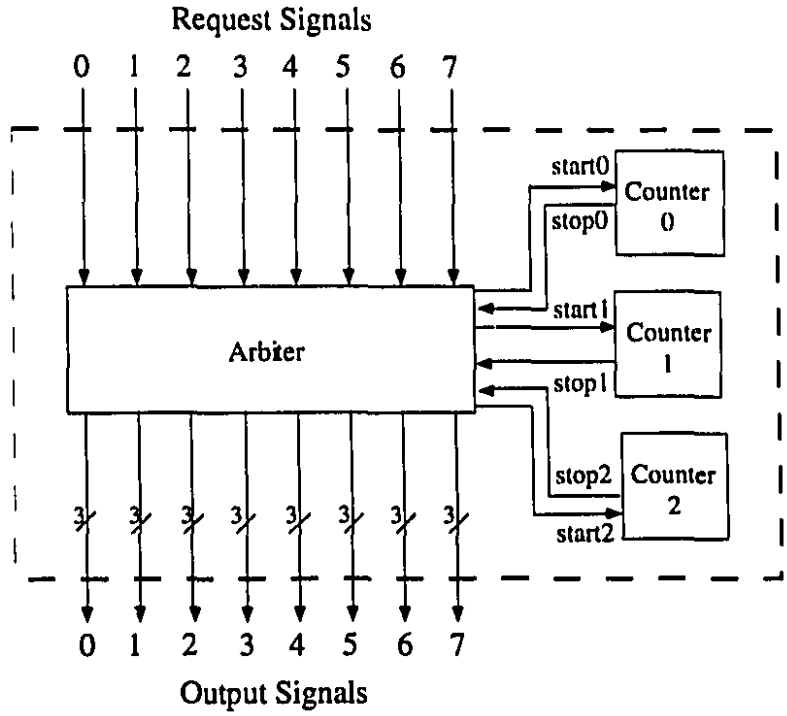

Figure 5.14 Typical functional diagram for the arbitration circuit.

When the entire packet has been received, the extractor selector control bits need to be reset so that the extractor channel is free to receive further incoming data. When the counter reaches a predetermined value (which represents the number of bytes in a packet), its sends a *stop* signal to the arbiter. At this time the extractor selector control bits for that channel are reset and the counter is reset and remains idle until another packet is to be received. Since there are three extractor channels, three separate counters are needed.

5.6.2 Functional Specifications of the Arbitration Circuit

Each transition is assumed to occur at the rising edge of the clock. The states are encoded using three bits. Each bit represents an extractor channel. If the most significant bit is high, then extractor channel 2 is busy. Similarly, if the least significant bit is high, then extractor channel 0 is busy. Each byte channel has a request signal. When there is an address match, the synchronizing circuit of the CCU will assert a request signal to the

arbitration circuit on the next rising edge. The arbitration circuit will receive this signal and cause the arbiter to change state. In addition there are two sets of internal signals that the arbitration circuit will use. These are the *startx* and the *stopx* signals (where *x* ranges from 0 to 2). The *startx* signals are issued from the arbiter which instructs one of the counters to begin counting. When the counter has reached a predetermined value, it will issue a *stopx* signal which will instruct the arbiter that the channel has received the entire packet and its extractor selector control bits can be reset. The output signals from the arbiter are the extractor selector control bits fed to the channels. These signals are more difficult to demonstrate on the state diagram. They will, however, be discussed shortly.

Figure 5.15 shows a typical state transition diagram for the arbiter. Each state is encoded using three bits. When none of the extractor channels is busy, the arbiter is in state 000. If one of the request signals is asserted high (denoted by *req* in the state diagram), then the arbiter will change state. In addition, whenever a request signal is asserted high, a *startx* signal is also asserted high. This will cause one of the counters to begin counting. The *stopx* signals are received from one of three possible counters. The stop signal will also cause the arbiter to change state as shown in Figure 5.15.

The only signals that are not shown on the state diagram are the 8 output signals. Each output signal is 3 bits wide and connects to the extractor selector control of a byte channel. Depending on which of the eight request lines was asserted and depending on the state of the arbiter, the appropriate output signal would be asserted. For example, suppose the arbiter is in state 011 (extractor channels 0 and 1 are busy) and signal *request 5* is asserted high. This means that byte channel 5 has a packet to extract. The arbiter will move to *state 111* and set start2 to a high state. This will cause *Counter2* to begin counting.

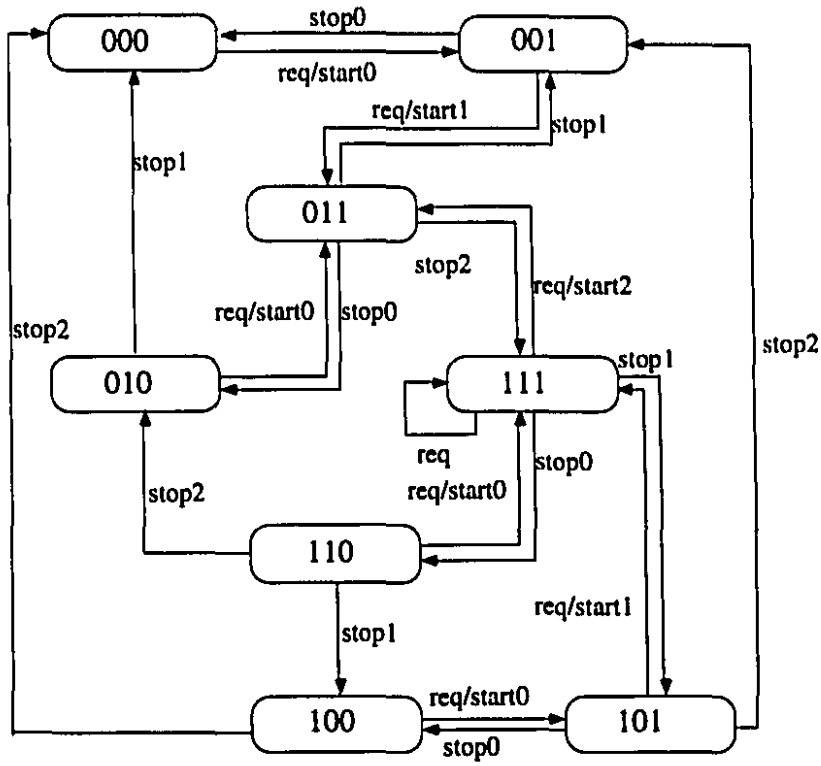

Figure 5.15 State transition diagram of the arbitration circuit.

At the same time, *output 5* will be set so that byte channel 5 will extract over the extractor channel 2.

Notice that in this implementation of the arbitration circuit, when there is a header match, there is a delay of two clock cycles until the extractor selector control bits are set. When there is a match, the request signal will go high on the next clock cycle. On the following rising edge of the clock, the arbitration circuit will change state and the counter will start counting. Since there is a two clock cycle delay, the header of each packet will have to be held in place for at least that many clock cycles. Therefore, in this implementation, when a packet is sent through the backplane, the header is sent at least two times before any data is sent.

5.7 Configuring the Smart Pixel Array

Before the Hyperplane can be used for transmission and reception of data, it first needs to be configured. As mentioned in a previous section, the Hyperplane has two modes of operation. The configuration process will instruct the Hyperplane which mode to operate in.

The configuration process involves loading the *Control Latch* in the Channel Control Unit. The loading of the Control Latch is a straightforward process. Each SPA has a 1 to 8 decoder as shown in Figure 5.16. The decoder is used to select one of the eight Control Latches. Thus the loading of the Control Latches is done one latch at a time. The inputs of the decoder are connected to the last three bits of *Injector Channel 1* while the eight input bits of the Control Latch are connected to *Injector Channel 0*. The control data to be loaded by the Control Latch are presented on to *Injector Channel 0*. The control bits for the decoder (which will select the control latch on channel 0) are then presented on the first three bits of *Injector Channel 1*. Finally, the Control Latch Enable is strobed high at which time the data on Control Latch (of channel 0) are loaded. The process is repeated until all eight control latches on all eight channels are configured.

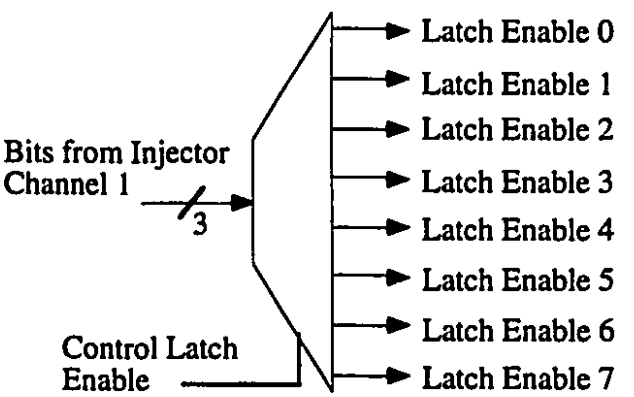

Figure 5.16 Decoder for configuring the control latch.

5.8 Chapter Summary

In this chapter, we have outlined the functional specifications of the June 1995 Smart Pixel Array. We began be describing the two primary modes of operation. These modes are the reconfigurable mode and the intelligent mode. We then proceeded to give a top-down description of the designs for the June 1995 Hyperplane and SPA. We described the various components that comprise the SPA including the arbitration circuit which was shown to be one of the more complex components in the design. In the next chapter, we shall describe the June 1995 Hyperplane using a hardware description language called VHDL. Using this language, we will simulate the design to demonstrate its functional correctness. We will then synthesize the design so that it can be downloaded on to an FPGA. Once the design has been configured, the FPGA will behave as a hardware simulator of the Hyperplane.

Chapter 6
Simulation of the June 1995 Hyperplane using VHDL and FPGAs

6.1 Introduction

In the previous chapter, we took a detailed look at the functional specifications of the June 1995 SPA and then proceeded to describe how these devices are interconnected to form the June 1995 Hyperplane[4]. Our next objective is to test the design in some manner to verify its functional correctness. One way of accomplishing this is to develop a simulator for the Hyperplane which can operate in both hardware and software.

In this chapter, we describe the development of a simulator for the Hyperplane. The software simulation is achieved using a hardware description language called VHDL that describes the design and is simulated by a CAD tool. The CAD tool allows us to feed our design with test vectors and give us the resulting output based on our design specifications. Once the functionality of the design has been verified in software, the CAD tool will then "transform" our design into a format which can be downloaded onto an programmable device. The programmable chip will then behave as an hardware emulator of the Hyperplane. In addition to verifying the its functional correctness, there is one additional

[4] Recall, that Hyperplane and SPA refer to the June 1995 designs of the Hyperplane and SPA

benefit to developing a hardware emulator for the Hyperplane. Recall that the Hyperplane will interconnect a number of printed circuit boards. These printed circuit boards each have electronic ICs for processing. By developing a hardware simulator for the Hyperplane, we can simulate how the ICs will behave when connected to Hyperplane. The VHDL model for the *June 1994* Hyperplane was developed and designed in [3].

The rest of this chapter is organized as follows. We begin by describing the hardware description language VHDL and briefly discuss the CAD tool which will simulate the design described by the VHDL code. Next, we will describe the Xilinx 4000 series field programmable gate array (FPGA). These are the programmable devices that will form the hardware implementation of our simulator. We then describe the structure of our VHDL code and then simulate our design. The results of the simulation are subsequently discussed. We then compile our design to generate a file that can be used to program the FPGA. This process is called synthesis for FPGA technology. In the final section of this chapter, we will discuss the timing and area statistics generated during the synthesis process.

6.2 Design Entry Language

When designing a digital system, the designer can describe his design using one of several methods. Historically, most of the design process was performed using schematic capture tools. In this methodology, the designer would draw logic gates and connect them using a CAD tool. Recently, a more popular method has emerged in which the designers could describe their design using a high level language similar in many ways to the C programming language. It is a language which can readily describe complex digital systems. VHDL stands for VHSIC Hardware Description Language where VHSIC stands for Very High Speed Integrated Circuits [32].

There are several advantageous features to VHDL. Among the most important is its ability to support hierarchy. That is, a digital system can be modelled as a set of interconnected components. These components can each, in turn, be modelled as a set of

interconnected subcomponents. In this way, complex designs are easily described using VHDL.

Each component or subcomponent in VHDL has an entity description associated with it. The entity description describes the input and output ports to that component. In addition to the entity, each component has an architecture description. The architecture description describes the behaviour of the entity [32][33].

Once the entire design has been specified using VHDL, the designer is ready to simulate it. The simulation requires two things. The first is a VHDL test bench, the second is a CAD tool which can simulate VHDL designs. The CAD tool which we will be using is the Synopsys 3.0b Design Compiler and Vhdldbx software. The test bench is another VHDL file that assigns values to the inputs of our design at specified times. The CAD tool will read the test bench and generate the output based on the design specifications and the input vectors specified in the test bench.

6.3 Introduction to FPGAs

State of the art VLSI technology has provided the capability to implement powerful digital devices such as microprocessors at an ever decreasing cost. It is now possible to fabricate integrated circuits housing millions of transistors. Such devices are often designed using the full custom layout approach. In this approach, each device on the integrated circuit has been tailored to meet a set of specifications and, to date, utilizes the least amount of silicon real estate. A less cumbersome approach to VLSI design would be to use standard cells when laying out the design. This semi-custom approach reduces the layout time but often requires larger areas of silicon. Both approaches, however, are expensive by nature. They require several months of design and manufacturing effort resulting in an increased cost to the consumer unless they are mass produced. In our highly competitive society, the need to minimize the overall cost and the time to market is paramount [34]. Given this need, field programmable gate arrays (FPGAs) have emerged as the ideal solution to these time to market problems. In essence, an FPGA is a device whose logic structure is configured by the end user without the need for fabrication

facilities allowing for rapid prototyping and testing of designs. The use of FPGAs dramatically reduce the time to market because the design needs only to be described at a high level such as a schematic. CAD tools are used to "transform" the design into a bit stream which is then downloaded to the FPGA [34][35]. The FPGA will behave according to the design specifications. FPGAs are similar to programmable read only memories or programmable logic devices except their capabilities are far more extensive. In the following, we describe in some detail the architecture of the field programmable gate array and how it will relate to the modelling of the Phase III Smart Pixel Array.

An FPGA is a device with an array of uncommitted logic elements that can that are generically interconnected. Figure 6.1 shows a conceptual view of an FPGA. As shown in Figure 6.1 the FPGA consists of a two dimensional array of logic blocks and interconnection resources that are used to interconnect the logic blocks. The logic elements are blocks that can implement a small logic design. They typically have some form of programmable devices (such as a look up table) to implement combinational circuitry and one or more flip-flops to implement sequential circuitry. The interconnection resources comprises segments of wires whose lengths typically vary [34][36][37]. In addition to the logic elements and interconnection resources, FPGAs also house programmable switches that are used to connect the logic elements to the segments of wire and to connect or one segment of wire to another. In order for FPGAs to be a viable technology, it must be able to implement a very large spectrum of designs. It is therefore important that the logic blocks and the interconnection resources be as versatile as possible. There are many manufacturers of FPGAs each with their own advantages and tradeoffs. In this chapter, we will concentrate on the design of the XC4013 FPGA manufactured by Xilinx Corporation. However, before a detailed look at the architecture of the FPGA is undertaken, it is useful to understand the implementation process when a design is downloaded on to an FPGA [34][36].

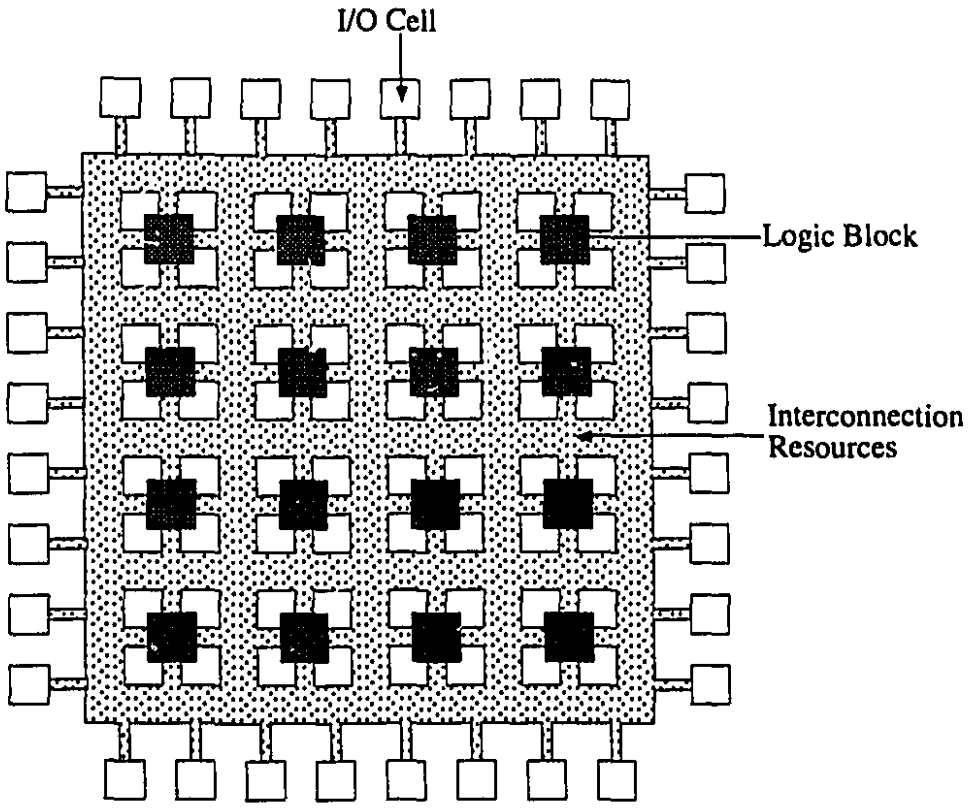

Figure 6.1 Conceptual view of an FPGA.

The implementation of a logic design on an FPGA is for the most part carried out by a CAD tool. Figure 6.2 shows a flow chart which depicts the steps that a CAD tool will typically take during the implementation process of a design on to an FPGA. Irrespective of how the design is described, the design entry is then transformed by the CAD tool into some standard format such as Boolean expressions. These expressions are then sent through the logic optimizer whose primary responsibility is to manipulate the Boolean expressions in order to minimize the overall area or maximize the speed of the final circuit [34].

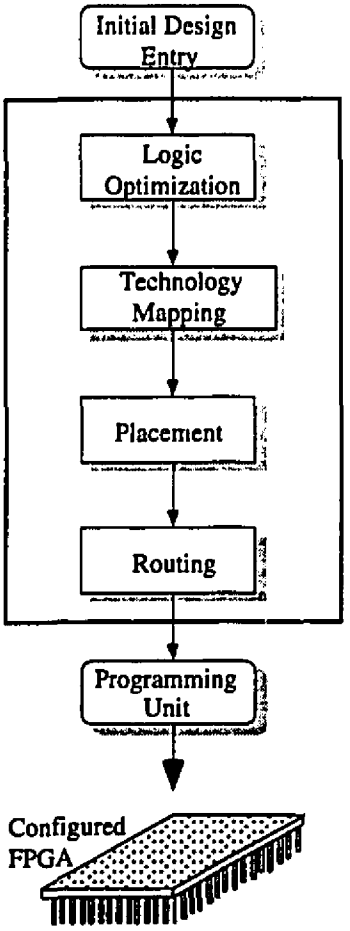

Figure 6.2 Synthesis process from design entry to configured FPGA.

The next step in the process in the technology mapping. Once the Boolean expressions (and hence the original design) have been optimized, the technology mapper will transform them into an equivalent circuit consisting of only logic elements. In essence, the mapper will have partitioned the design into subcircuits so that each subcircuit can be implemented by a logic element. The technology mapper will also perform some optimization algorithms. Depending on the user specifications, the mapper will attempt to

reduce the number of logic elements consumed (for area optimization) or reduce the number of stages in the critical path (for timing optimization) [34].

The next step is "place" these logic elements on the FPGA. A placement algorithm is used to do this. The placement algorithm is used to determine the logic elements that will implement the subcircuits. The main objective of the placement algorithm is to place the subcircuits in a manner which minimizes the total amount of interconnection resources used [34].

The final routine used by CAD tool is the routing software. The routing software assigns wire segments of the FPGA and configures the programmable switches in order to establish the specified connections among the logic blocks. In essence, the subcircuits are interconnected so that the original design is achieved. If the routing software is unable to route the design (a phenomenon which is not uncommon when implementing large designs), then it is possible that more than one FPGA will be required to implement the design. Since the interconnection resources of the FPGA are fixed in place, the task of routing is a difficult by nature.

Once the preceding steps have been successfully completed, the CAD tool will feed its output to a programming unit which will then configure the FPGA. Once the FPGA is configured, it will implement the initial design entry and the entire implementation process has been completed.

6.3.1 Xilinx 4013 FPGA

As mentioned in the previous section, each manufacturer of FPGAs will design their FPGA differently from other manufacturers. There are various types of FPGA implementations ranging from one time programmable FPGAs to reprogrammable FPGAs. In this section, the architecture of the Xilinx 4013 FPGA is described. The Xilinx 4013 FPGA has a general architecture similar to that of Figure 6.3. In the Xilinx architecture, the logic elements are termed Configurable Logic Blocks (CLBs) and the I/O Cells are termed I/O Blocks (IOBs). The programmable connections are achieved using n-channel pass transistors which are controlled by static-RAM (SRAM) cells. The programmable

connections are called switching matrices and are shown in Figure 6.4. The interconnection resources consists of horizontal and vertical routing channels that interconnect the CLBs. There are three types of interconnection lines differentiated by their relative segment lengths. These are the single-length line, the double-length line and the long lines. The single length lines are used primarily for short, local interconnections as shown in Figure 6.4. The single length lines are arrays of horizontal and vertical lines each intersecting a switch matrix between each block. The double length lines are double the length of the single length lines and runs past two CLBs before intersecting a switching matrix [34][36].

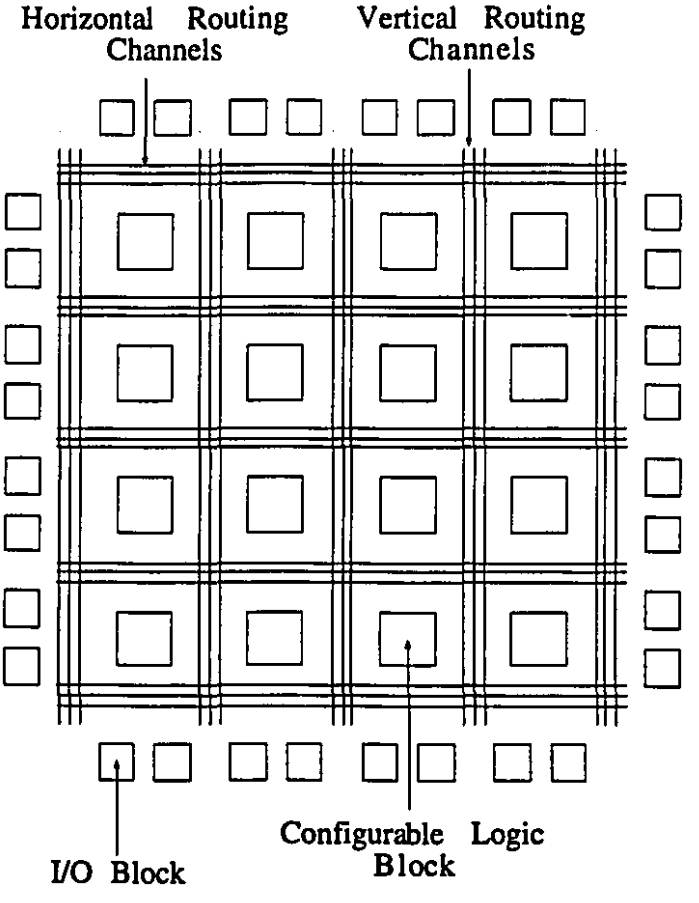

Figure 6.3 Architecture of an Xilinx 4000 series FPGA

The long lines are an array of metal segments that traverse the entire length of the or width of the FPGA. Long lines are used for high fan-out, time-critical signals which can also be used to distribute clock signal with a minimum amount of skew.

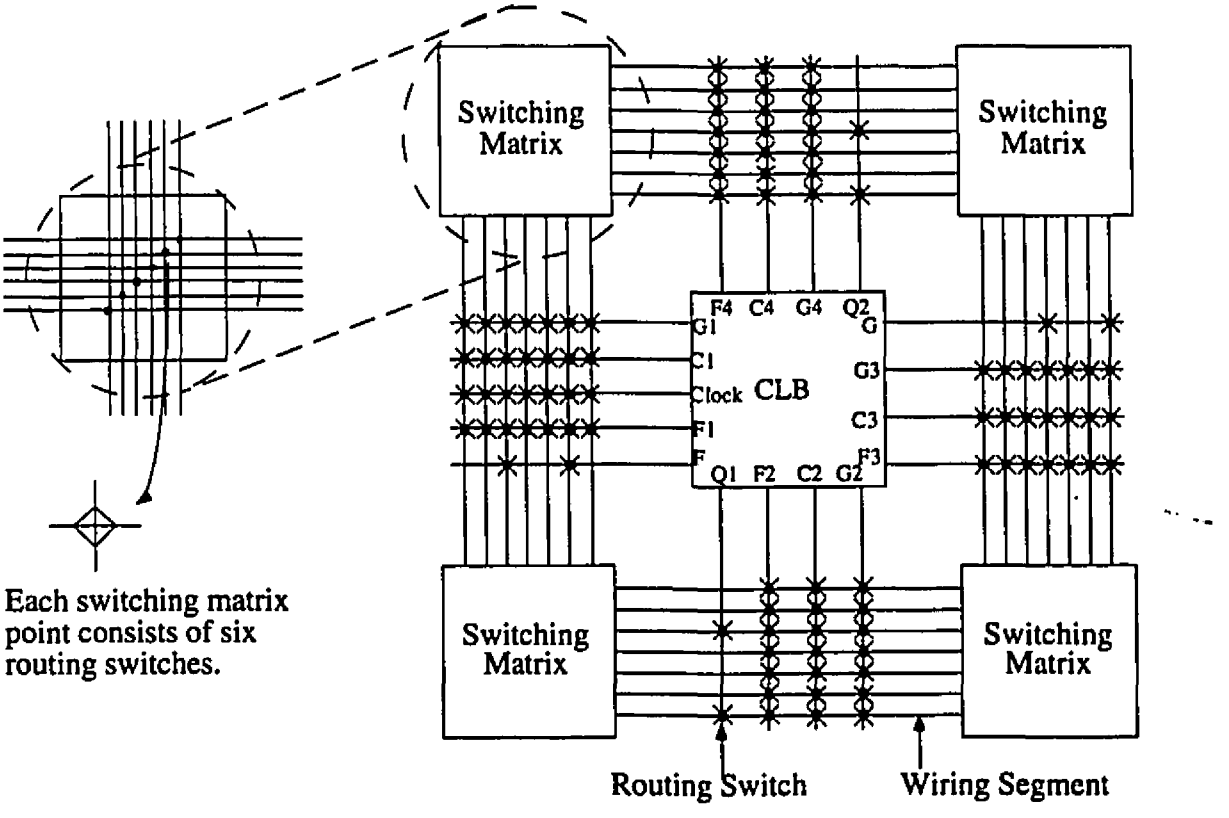

Figure 6.4 Routing resources to interconnect CLBs.

The configurable logic blocks (CLBs) are generic devices used to implement a small subset of the entire design. The functional logic description of the CLB is shown in Figure 6.5. It consists of two 4-input logic function generators labelled F and G and a third function generator with three inputs labelled H. These function generators provide the combinational circuitry of each CLB. They can perform two independent functions of four

variables, one function of five variables, any function of four variables in conjunction with some functions of five variables and some functions of nine variables [34][36].

In addition to the combinational circuitry, each CLB consists of the D-Flip Flops which provide the sequential circuitry of each CLB. The D-Flip Flops are equipped with asynchronous set and reset which are also controlled by the selector module. The set of multiplexors on each CLB determine, in part, the functionality of the each CLB. The outputs of the combinational function generators can be fed into the inputs of the D-Flip Flop or directly to the CLB output. If the combinational function generators are fed directly to the outputs, the D-Flip Flops can receive inputs from the selector in which case, the CLB can be actually performing two different functions (one combinational and one sequential).

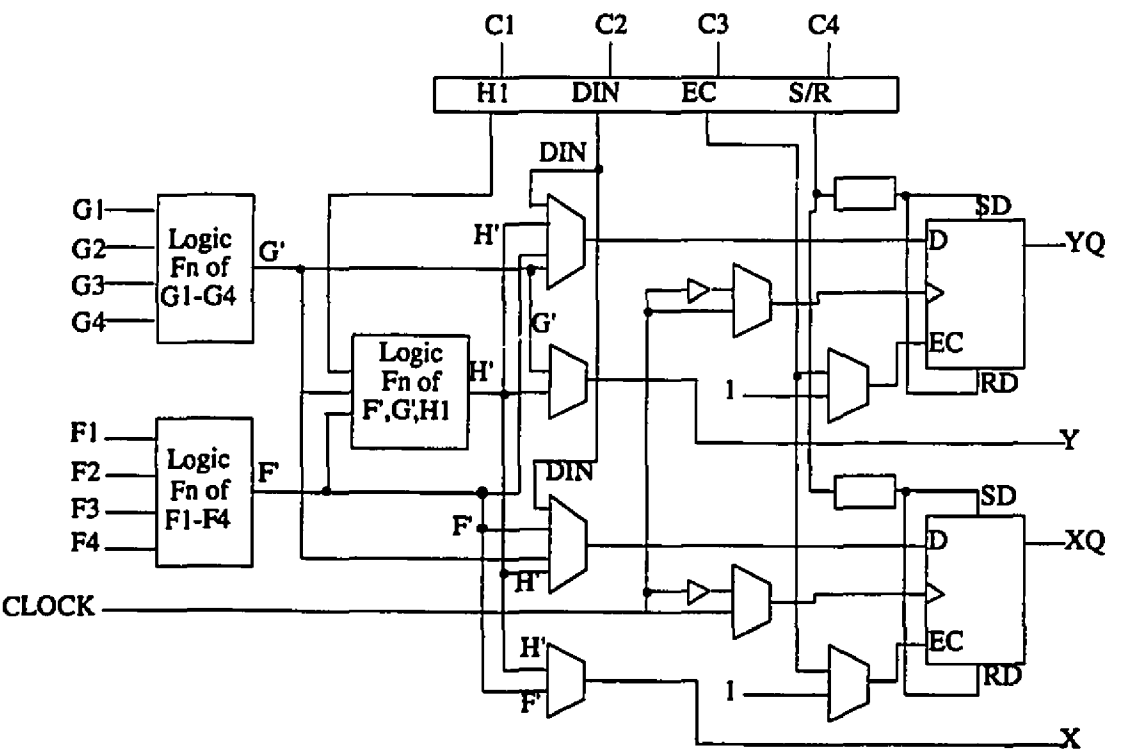

Figure 6.5 Logic specification for the Configurable Logic Block of the Xilinx 4000 series FPGA.

The IOBs provide the interface between the internal logic of the FPGA and the packaged pins [34][36].

6.4 VHDL Hierarchy of the Hyperplane

As mentioned in Section 6.2, one of the main features of VHDL is its ability to support hierarchy. Before we describe the our design using VHDL, it is first prudent to develop a hierarchical description. Figures 6.6 through 6.10 show the various levels of the hierarchy for our design. Each bounding box represents a separate entity. Figure 6.6 shows the highest level in the hierarchy. It consists of two entities. The first is the test bench which we will use to simulate our design The second is the Backplane entity. The Backplane entity will instantiate the entity of four SPAs and interconnect them in a unidirectional ring. Notice that within the entity Backplane is another entity Smart_Pixel_Array. This indicates that the entity Backplane will use the entity Smart_Pixel_Array as its components. Since Backplane interconnects four smart pixel arrays in a unidirectional ring, we say that the entity Backplane instantiates the Smart_Pixel_Array four times.

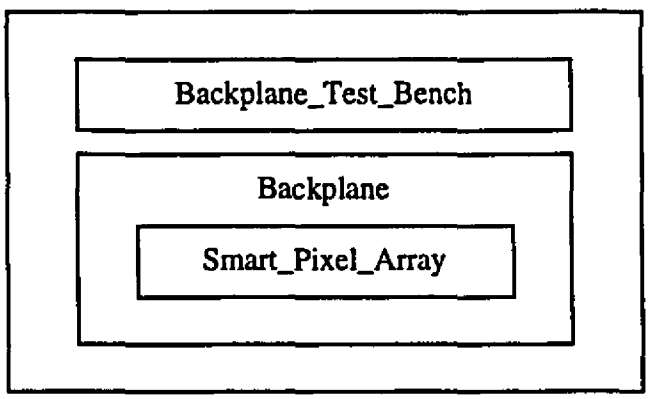

Figure 6.6 Top level of the hierarchy

The next level in the hierarchy (shown in Figure 6.7) describes the components that make up Smart_Pixel_Array. As we described in Chapter 5, each SPA consists of eight SPC's

(smart pixel channels), an arbitration circuit and a decoder. Therefore the entity Smart_Pixel_Array will instantiate entity Smart_Pixel_Channel eight times (byte channels are occasionally referred to as Smart Pixel Channels). Notice that Smart_Pixel_Channel will instantiate entities Smart_Pixel and Channel_Control_Unit. In addition, Smart_Pixel_Array will instantiate entities Arbitration_Unit and Decoder once each.

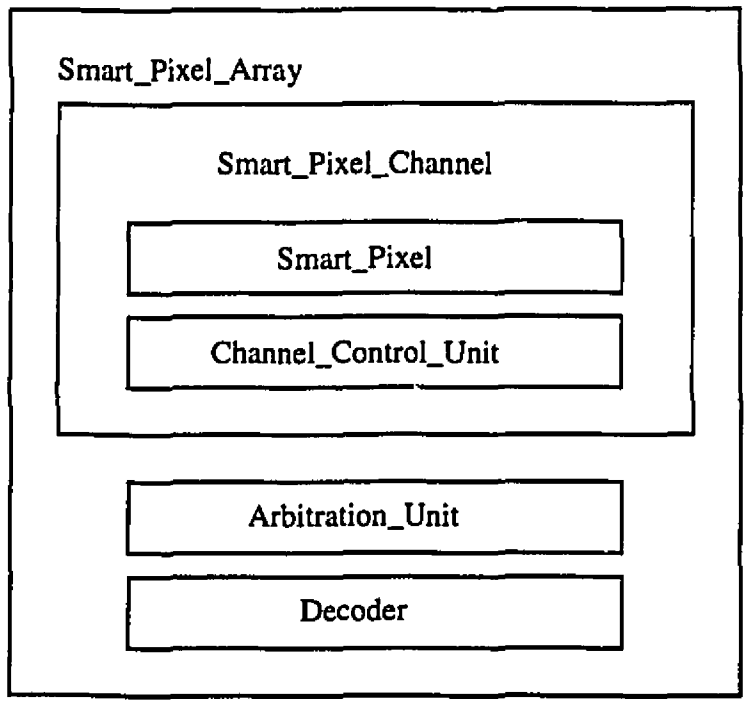

Figure 6.7 Second level in the hierarchy, illustrating a typical SPA.

Figures 6.8 to 6.10 show the third and fourth levels in the hierarchy for the rest of the design. Once the hierarchy of our design was completed, the VHDL code for each level of the hierarchy was created and then simulated using the Synopsys CAD tool. A complete copy of the VHDL code can be found in [40].

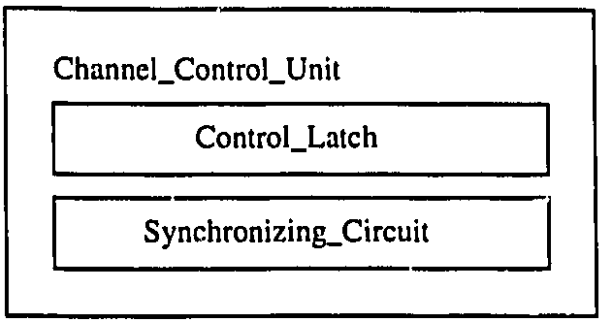

Figure 6.8 Third level in hierarchy illustrating a typical channel control unit.

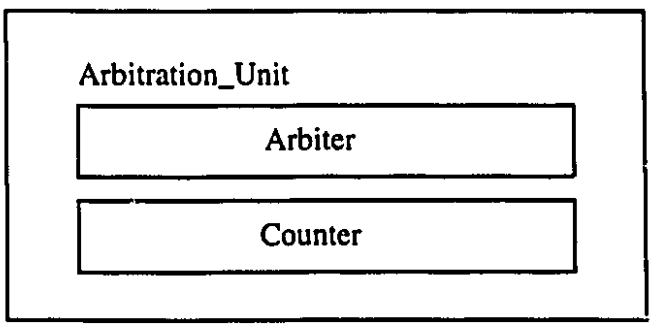

Figure 6.9 Third level in hierarchy illustrating the arbitration unit.

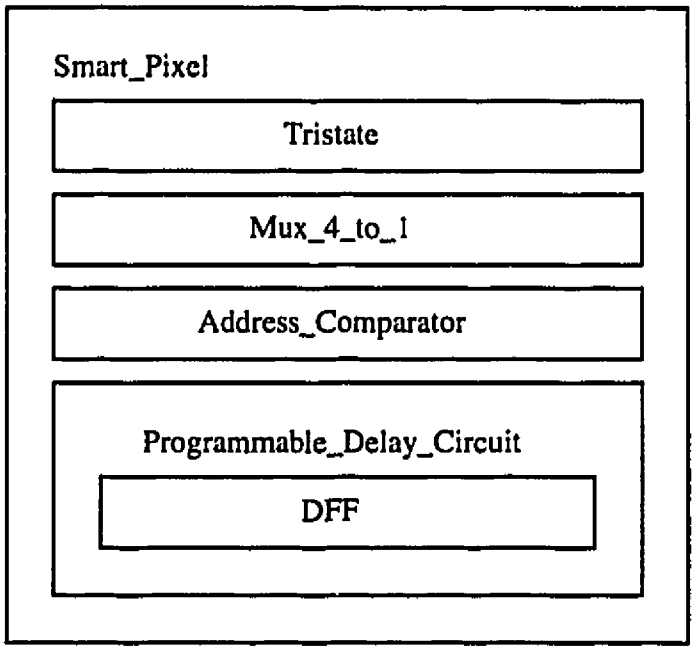

Figure 6.10 Third and fourth levels in the hierarchy, illustrating a typical smart pixel and the programmable delay circuit.

6.5 Design Verification Through Simulation

6.5.1 Propagation Delays Through the Logic Devices

Once the design has been described using VHDL, the next step is to verify its functional correctness through simulation. The task of simulation is indeed difficult. It would be unrealistic to demonstrate how the Hyperplane would behave under all contingencies since this would constitute an exhaustive test. The number of test vectors in an exhaustive test rise exponentially to the number of inputs to the system [38]. This section will present some simulation results. It is the author's belief that the simulations presented in this chapter will demonstrate and verify the functional correctness of our design.

In order to establish proper timing analyses, the propagation delays through the various components need to be taken into account. To determine these propagation delay,

it is necessary to know what technology will be used. At the current time, it is expected that upcoming demonstrators will be implemented using either .5 or .8 μ CMOS technology. In order to establish a worst case analysis, the delay specifications for 1.2 micron CMOS technology was included in the VHDL code [39]. Table 6.1 summarizes the delay specifications used. Adding the delays to the various components allows us to also approximate the maximum speed the final design can accommodate. As can be seen, the main bottleneck is the delay through the bond pads. The bond pads are used to inject and extract data to and from the backplane. This delay will vary depending on the capacitive load it needs to drive. However, a conservative estimate assumes the delay through the bond pads to be about 25.0 ns.

Logic Device	Propagation Delay
2 Input And	3.86 ns
4 Input And	4.67 ns
2 Input Or	2.73 ns
2 Input Xor	4.27 ns
2 Input Nor	2.73 ns
3 State Buffer	3.96 ns
D Flip Flop	4.39 ns
4 to 1 Mux	7.39 ns
4 Bit Counter	5.67 ns
Arbiter (approx.)	12.0 ns
Bond Pad (approx.)	25.0 ns

Table 6.1: Summary of worst case propagation delays through various logic devices using 1.2 micron CMOS technology [39].

6.5.2 Simulation Setup and Results

Using the delay values specified in the previous section, the test bench was created. Recall that there are two modes that the Hyperplane can operate in. For the sake of brevity, the simulation of only the intelligent mode will be described. In addition, the simulation is

set so that the packets are *asynchronously* transmitted. In order to operate in the intelligent mode, each board (and hence each SPA) must be assigned a unique address. Table 6.2 summarizes what address we have assigned to each SPA

SPA	Address (Binary)
0	10000010
1	10000011
2	10000100
3	10000101

Table 6.2: Address for the SPAs connected to the Hyperplane.

In Chapter 5, we noted that in order to preserve an 8 bit data path, the byte channel had to be increased from 8 pixels to 9 pixels. However, for the sake of convenience, we have elected to keep the byte channel at 8 pixels (8 bits) and still send packets asynchronously. However, with minor adjustments to the VHDL code, the byte channel can be increased to 9 pixels.

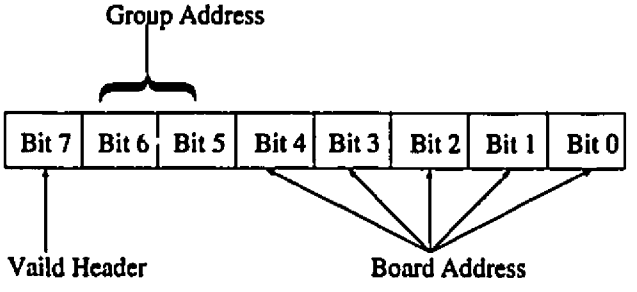

Figure 6.11 Description of address bits.

For our VHDL simulations, the clock has a period of 30 ns. Before the Hyperplane can begin transmitting and receiving data, it must first be configured. The configuration process loads data into the control latches of all the channel control units. This process was described in some detail in Chapter 5. Figure 6.12 shows the simulation of the configuration process. The figure is labelled with lines as an aid in our discussion. At *Line A*, we see that the Reset signal is set high (see the bold dot on *Line A*). At this time all latches and

outputs are initialized to a low state. Once all components have been reset, the configuration process begins. Figure 6.12 shows the status of all 12 injector channels and all 12 extractor channels. We will follow the configuration of SPA 0. Recall that two injector channels are needed to configure a SPA. Injector channel 0 carries the data to be latched by the control latches. Injector channel 1, carries the control bits of the decoder (Figure 5.16) that selects the control latch to configure (the configuration process was described in Chapter 5, Section 6).

The labelling of the signals in Figure 6.12 and 6.13 follow a unchangeable convention created by the Synopsys 3.0 CAD tool. Injector0(1), for example, means injector channel 1 of SPA 0. At *Line B* in Figure 6.12, we see that Injector0(1) has a value of 00 and Injector0(0) has a value of F8 (see the two bold dots on *Line B*). Injector0(1) controls the decoder in each SPA while Injector0(0) holds the data that the control latches will be loaded with. Therefore at *Line B*, the first control latch of SPA 0 will be loaded with the data F8 (at the rising edge of the clock). The same holds true for all the other SPAs. At *Line C* we see that Injector0(1) has changed from 00 to 01. That means that the next control latch has been selected for loading. Injector0(0) remains the same. All this means is that the control latch will be loaded with the same data as the previous control latch. The configuration process continues until *Line D* where the last control latch is loaded with a value EC (Injector0(1) has a value of 07 and Injector0(0) holds a value of EC)

Once the configuration process has been completed, the Hyperplane is ready to transmit and receive data. The test bench will begin feeding data through the Injector Channels. Since it is difficult to visually trace the transmission of packets through the backplane, Table 6.3 summarizes where the packets from a given SPA are destined.

Notice that four packets are destined for SPA 2 but only three of those packets can be extracted from the backplane (there are only three extractor channels per SPA). We set the headers of four packets for SPA 2 so that we can ensure that the arbitration circuit will ignore all incoming packets if all three extractors are busy.

Source	Header	Destination
SPA 0	11000101	SPA 3
SPA 0	10100100	SPA 2
SPA 1	11100010	SPA 0
SPA 1	10100100	SPA 2
SPA 2	10100011	SPA 1
SPA 2	11100010	SPA 0
SPA 3	10100100	SPA 2
SPA 3	10100100	SPA 2

Table 6.3: Summary of packet origin and destination.

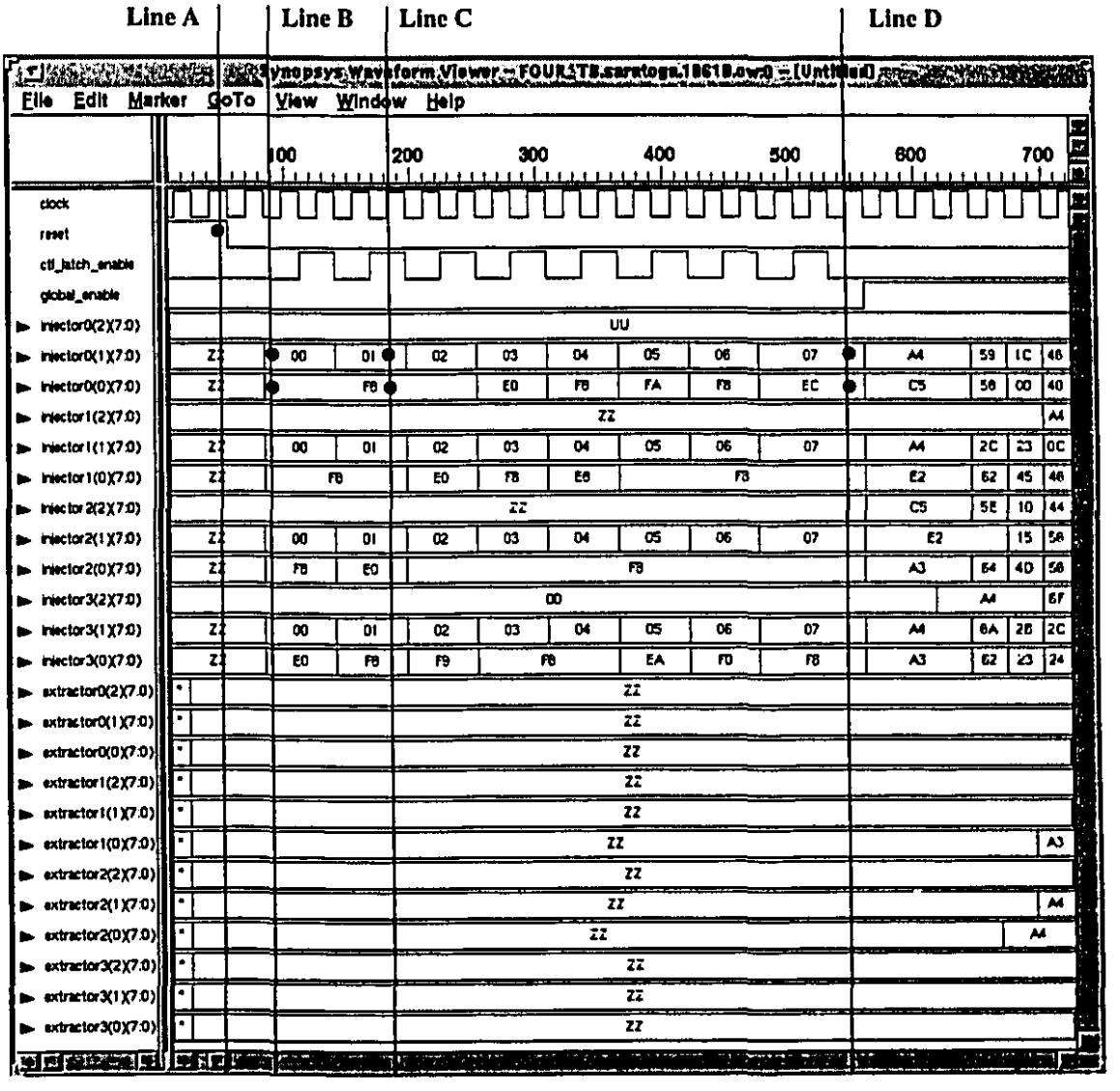

Figure 6.12: Simulation of the configuration process.

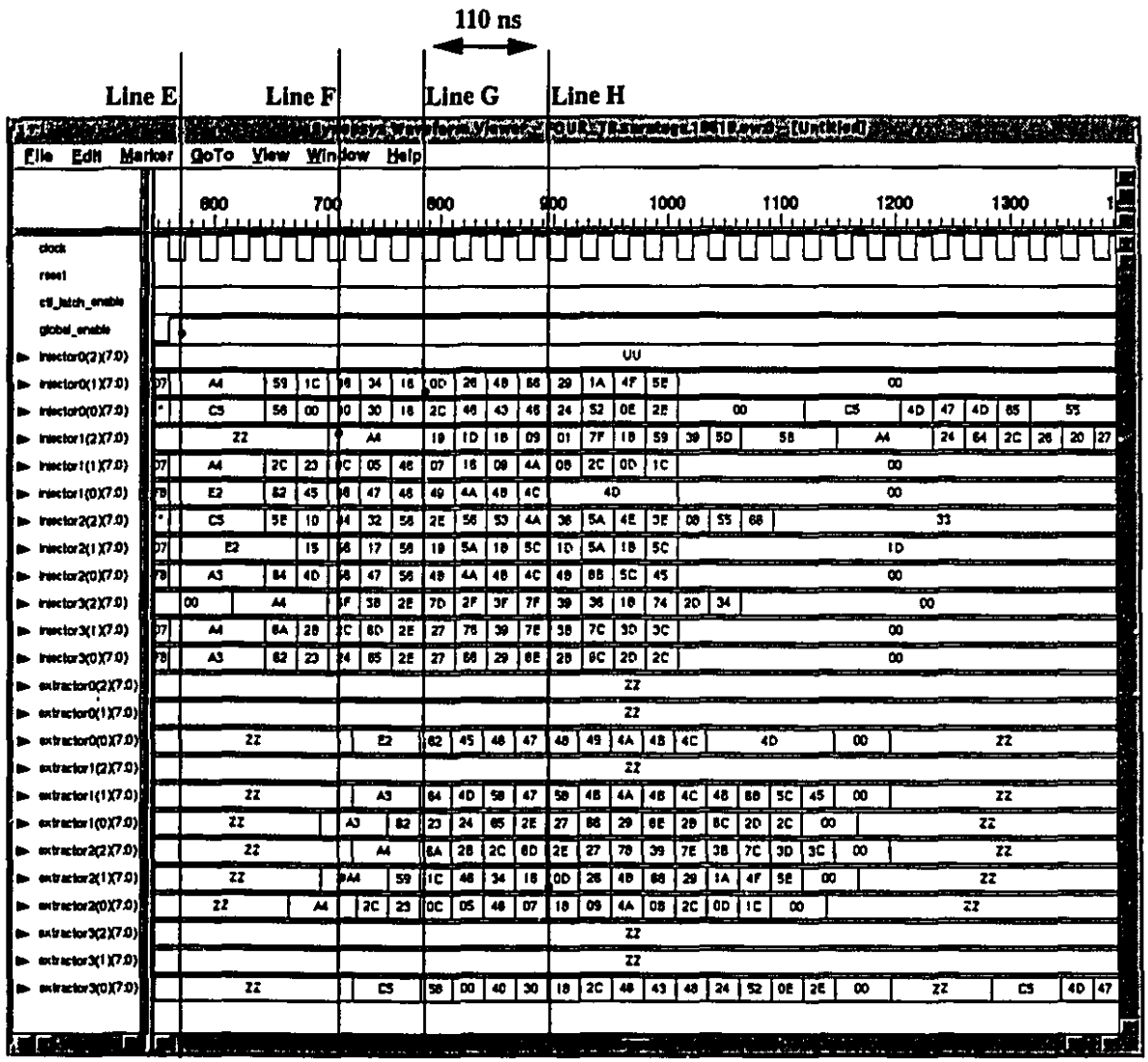

Figure 6.13: Simulation of packet transmission and reception.

The transmission of data begins when the *Global Enable* signal goes high (see the bold dot on *Line E* in Figure 6.13). At this time, most of injector channels are fed with the header of various packets. The header needs to be held for at least two clock cycles to allow time for the arbitration circuit to complete its task (to be conservative, all packet headers were held for 3 clock cycles). To show that the backplane does transmit packets asynchronously, the transmission of a packet from Injector1(2) has been delayed until *Line F*. Once the header has been transmitted, a new byte is introduced at the falling edge of every clock cycle. At the same time, all bytes in the backplane move from *SPA j* to *SPA (j+1) mod 4*. Thus if SPA 3 is transmitting to SPA 2, then each word would take 3 clock cycles to travel from SPA 3 to SPA 2.

If we follow Table 6.3, we see that the packets arrive at the appropriate SPA and are handled correctly. For example, the packet transmitted by Injector0(1) has a header whose hexadecimal address is A4. This indicates that the packet is destined for SPA 2. If we look at Extractor2(1), at *Line F* (see the bold dot near the bottom of *Line F*) we see that the data appearing on this extractor is the same data that was injected through Injector0(1). We can measure the delay from when the data was injected by Injector 0(1) to the time it was extracted by Extractor2(1). *Line G* shows the word 0D entering the backplane over Injector0(1). *Line H* shows the same word being extracted by Extractor2 (1). The time between *Line G* and *Line H* is 110 ns (please see bold dots on *Line G* and *Line H*). This delay is to be expected. There is a 25 ns delay to pass through the bond pads of the injector channel. From there the data will hop from SPA to SPA at every rising edge of the clock. Since the data is travelling from SPA 0 to SPA 2, it must make two hops. Therefore, the data will take 2 clock cycles (or 60 ns) to move from SPA 0 to SPA 2. Finally, there is another 25 ns delay to pass through the bond pads of the extractor. The sum of all these delays add to 110 ns.

The only packet that did not arrive at its destination is the packet sent by Injector3 (2). However this is to be expected. Table 6.3 shows that there are four packets destined for the SPA 2. The last injector to transmit a packet destined for SPA 2 is Injector3(2). If

the arbitration circuit is working correctly, then this packet should just be ignored. This is indeed the case. This shows that the arbitration circuit simply ignored the packet from Injector3(2).

Given the delays shown in Table 6.1, the simulation was executed for various clock speeds. To ensure proper operation of the VLSI design, the clock must have a minium period of 28 ns or 35.7 MHz. Referring to Table 6.1, we observe that the minimum clock period for the VLSI design is 25 ns which represents the delay through the bond pads. The additional 3 ns delay that we notice through simulations occurs because logic gates through the datapath. It should be noted however, that the delays tabulated are worst case figures based on 1.2 μ CMOS technology. It is expected that the actual implementation of the Hyperplane (which is expected to be fabricated using either .8 μ or .5 μ CMOS technology) will be able to run at higher clock speeds.

6.6 Design Compilation and Synthesis for FPGA Technology

The final step in the development of our simulator is to synthesize the VHDL description for FPGA technology. The major steps in the synthesis procedure were shown in Figure 6.2. The logic optimization and the technology mapping are performed by the Synopsys CAD tool. The last placement and routing of the design are performed by the Xact software developed by the Xilinx corporation. When completed, the Xact software will report the number of CLBs the design will require.

One of the obstacles that we faced was the shortage of pins available. Referring to Figure 5.2 (on Page 62), we see that each SPA requires 60 *electrical* pins (*i.e.* not including the optical I/O). If four SPAs were to be interconnected, then a total of 240 pins would be needed. The target technology for synthesis was the Xilinx 4013 FPGA which has a maximum total of 192 I/O pins. Therefore, our FPGA simulator will only simulate two SPAs connected together on one FPGA. This is not a serious problem since the main motivation for developing a hardware simulator is to test the electronic components. We do not need to simulate all four SPAs connected in a unidirectional ring to meet this end. Therefore, our hardware simulator will only simulate two SPAs connected in a unidirectional ring.

Table 6.4 summarizes the resource utilization of our synthesized design. Notice that there is a relatively large utilization of packed CLBs. This is largely due to the Synopsys 3.0 compiler. During the technology mapping phase, the compiler does not capitalize of the built in latch enable on the CLB flip-flops. The result is a significant increase in the overall logic area required to implement latch enable system. The target libraries for the newer version of Synopsys were not readily available during the time of compilation and synthesis. We expect to resynthesize the design when the target libraries become available.

Resource	Used	Available	%-Utilization
Occupied CLBs	530	576	92
Packed CLB's	421	576	73
Bonded IO Pins	105	192	54
FG Fn. Generators	843	1152	73
H Fn. Generators	214	576	37
CLB Flip Flops	281	1152	24
Tri-State Buffers	64	1248	5
Tri-State Half Long Lines	64	96	66

Table 6.4: Resource utilization after partition, placement and routing of the design on a Xilinx 4013 FPGA.

Path	Timing (Speed Grade=6)	Timing (Speed Grade=4)
Pad to Pad	102.3 ns	89.1 ns
Pad to Setup	210.5 ns	156.4 ns
Clock to Pad	92.0 ns	70.2 ns
Clock to Setup	212.0 ns	144.1 ns

Table 6.5: Timing statistics for two speed grades of the Xilinx 4013 FPGA.

The timing statistics for the synthesized design are tabulated in Table 6.5. These statistics are obtained using the Xdelay software also produced by Xilinx. The Xilinx 4000 series offers various speed grades. Table 6.5 shows the timing statistics for two such speed grades.

Timing Statistic	Speed Grade -6	Speed Grade -4
Minimum Clock Period	212.0 ns	156.4 ns
Maximum Clock Speed	4.7 MHz	6.4 MHz

Table 6.6: Maximum clock speeds for the Hyperplane emulator using different speed grades of the Xilinx 4013 FPGA.

Table 6.6 tabulates the maximum clock speed of the emulator based on the critical path calculated by the Xdelay package. Even with the faster FPGA, the maximum clock speed of the emulator is restricted to 6.4 MHz. Again this is partly due to the Synopsys 3.0b compiler. We mentioned earlier that the compiler does not capitalize on the built in latch enable system of each flip flop. The result is an increase in the overall logic area that the final design will consume. This extra logic will also lead to propagation delays affecting the overall speed of the design. We expect better performance values when the newer libraries become available and our design is recompiled using the new versions of Synopsys. Nevertheless, a hardware emulator that runs at 4 MHz would allow other hardware modules (processing ICs mounted on each PCB) to be thoroughly tested. As little as 5 years ago, a wire-wrapped prototype would be the only feasible method to construct an emulator (since FPGA technology was not readily available). Such a prototype would likely run at speeds less than 1 MHz. Hence the FPGA emulator is very fast and convenient compared to the alternatives (i.e. wire wrapped prototypes).

6.7 Chapter Summary

This chapter discussed the development of a simulator for the Hyperplane. The simulator was developed in software and in hardware. The software simulator was created using VHDL (a hardware description language). The VHDL code allowed us to describe our design in a manner which could be read by a CAD tool Synopsys. The Synopsys tool would then simulate our design based on the specifications of the VHDL description. We presented the hierarchy of the VHDL description and then demonstrated the functional correctness of our design by displaying some of the simulation results. We then proceeded to synthesize our design for FPGA technology. We saw that the major consumer of configurable logic blocks was the arbitration circuit. We also saw that the Synopsys compiler that we used did not capitalize on the latch enable system causing an increase in the overall area.

One of the novel aspects of this chapter is the usage of FPGA technology to simulate an optoelectronic device and an optical backplane in real time. Based on previous experiences with the modelling of an optical backplane using FPGAs our hardware simulator will undoubtedly become an invaluable tool. The simulator allows us to develop electronic hardware and analyze how it will behave when interfaced with the actual SPAs at reasonable clock rates (in the MHz range).

Chapter 7
Conclusion

As our information hungry society continues place greater and greater demands on high bandwidth systems, the need for new paradigms in which data is transmitted and processed begins to arise. The speed at which electronics can switch is limited by the capacitive and inductive effects inherent to all systems. These effects are especially more pronounced as the frequency at which data is transmitted increases. Moreover, as the density of the electrical interconnect increases so do the deleterious effects of inductive coupling.

In recent years, a considerable amount of research has focused on integrating optics with electronics. The main advantage of these systems arises from the fact that optical signals are naturally non-interactive. Therefore they do not suffer from the same high frequency and cross coupling effects that plague electronic systems. Initially, much of this research revolved around replacing entire electronic modules with equivalent optical devices. Several years later it was discovered that greater benefits could be achieved if optical technology was used to complement electronic technology and with this realization, research in optoelectronic technology became increasingly widespread.

The Canadian Institute for Telecommunications Research is funding a Major Project in Photonic Devices and Systems. One of the main goals of this five phase five year project is to develop an optical backplane capable of interconnecting several printed circuit boards with an aggregate bandwidth on the order of 1 Terabit per second. The backplane is

appropriately referred to as the Terabit Photonic Backplane and much of this thesis revolves around this project. We began by motivating the need for optics in technology and then proceeded to discuss some of the work previously done in this area. Chapter 3 discussed in some detail the Terabit Photonic Backplane (which we later called the Hyperplane) and reviewed a rather new device which we termed Smart Pixel Arrays (SPAs). These SPAs are based on Multiple Quantum Well (MQW) technology and perform the optoelectronic conversions necessary to implement the Hyperplane.

In Chapter 4, we described a procedure called graph contractions. We used this procedure to present several possible embeddings of a Cray T3D Supercomputer on to the Hyperplane. For each embedding we calculated the bisection width. We then calculated the number of SPAs (as the number of byte channels varied) needed to implement each embedding. The bisection width was used to help calculate these numbers.

As part of the mandate for the Major Project, demonstrators are periodically constructed to highlight the milestones achieved. Chapter 5 reviewed the functional specifications of the June 1995 Hyperplane and the June 1995 SPA which can be used to realize the embeddings of Chapter 4. The June 1995 Hyperplane is a representative subset of the Terabit Photonic Backplane.

Finally, in Chapter 6 we developed a hardware and software simulator for the June 1995 backplane. The design was described using a hardware description language called VHDL and simulated using the CAD tool Synopsys. We then compiled and synthesized the design for FPGA technology. The synthesized design could be downloaded on to an FPGA which would then behave, in hardware, as the backplane. The hardware simulator will allow us to rigorously test any electronic components that will eventually be interfaced with the backplane at realistic clock rates (in the MHz range).

References

1. Canadian Institute for Telecommunications Research, *Annual Report 1995*. Available from McGill University by calling (514) 398 8104.

2. Jurgen Jahns and Sing H. Lee ed., "Optical Computing Hardware," Academic Press Inc., San Diego, Ch 1, written by Thomas Cloonan, "Architectural Considerations for Optical Computing and Photonic Switching" pp 1-41, 1994.

3. Palash Desai and Ted H. Szymanski, "Simulation and Modelling of a Smart Pixel Array Optical Backplane Using FPGA's" *Proceedings of the Third Canadian Workshop on Field Programmable Devices*, Montréal, Canada, pp 38-43, May 29-June 1, 1995.

4. Manoj Verghese, *A Software Based Design Space Exploration of a Free-Space Photonic Backplane*, Master's Thesis, McGill University, June 1995.

5. Kai Hwang, *Advanced Computer Architecture. Parallelism, Scalability, Programmability*, New York, NY, McGraw Hill, 1993.

6. Julia J. Brown, J.T. Gardner and Stephen R. Forrest, "An Integrated, Optically Powered, Optoelectronic "Smart" Logic Pixel for Interconnection and Computing Applications," *IEEE Journal of Quantum Electronics*, Vol. 29, NO. 2, pp. 715-726, February 1993.

7. Anthony L. Lentine and David A. B. Miller, "Evolution of the SEED Technology: Bistable Logic Gates to Optoelectronic Smart Pixels", *IEEE Journal of Quantum Electronics*, Vol. 29, No. 2, pp. 655-669, February 1993.

8. H.S. Hinton, *An Introduction to Photonic Switching Fabrics*, Plenum Press, New York, New York, 1993.

9. T.H. Szymanski and H.S. Hinton, "Architecture of a free-space circular photonic Hyperplane", Submitted.

10. Ted H. Szymanski, "Intelligent optical backplanes," International Conference on Optical Computing - 95, Salt Lake City, Utah, March 12-17, 1995.

11. Stephen R. Forrest and H. Scott Hinton, "Introduction to the Special Issue on Smart Pixels", *IEEE Journal of Quantum Electronics*, vol. 29, no. 2, 598 -599, February 1993.

12. H. Scott Hinton and Ted H. Szymanski, "Intelligent Optical Backplane", To Appear *Conference on Massively Parallel Processing with Optical Interconnections*, San Antonio, Texas, October 23-24, 1995.

13. Ted H. Szymanski and H. Scott Hinton, "Design of a Terabit Free-Space Photonic Backplane for Parallel Computing", To Appear, *Second International Conference on Massively Parallel Processing using Optical Interconnects 1995,* San Antonio, Texas, October 23-24, 1995.

14. T.H. Szymanski and H.S. Hinton, "Smart pixel designs for a terabit free-space photonic backplane", IEEE/LEOS Summer Topical Meeting '94, Lake Tahoe, Nevada, July 11-13, 1994.

15. R.G.A Craig and Frank Tooley, "All-optical programmable logic gate," *Applied Optics*, Vol 29, pp. 2148-52, May 1990.

16. T. Main, R. J. Feuerstein, H. F. Jordan, V.P. Heuring, J. Feehrer and C. E. Love, "Implementation of a general-purpose stored-program digital optical computer", *Applied Optics* Vol. 33, No. 8, 1619-28, March 10, 1994.

17. T.J. Cloonan, "Free-space optical implementation of a feed forward crossbar network," *Applied Optics,* 31, pp. 3213-3224, June 1992.

18. V.P. Heuring et al., "Bit-serial architecture for optical computing," *Applied Optics,* 29, 3213-3224, June 1992.

19. A. Guha et al., "Designing massively parallel optical computers: a case study," *Applied Optics,* 29, pp. 2187-2200, May 1990.

20. B. Jenkins, et al., "Optical computing: Introduction by the feature editors," *Applied Optics,* 31, pp. 5423-5425, Sept. 10, 1992.

21. F.B. McCormick, T. J. Cloonan, A. L. Lentine, J. M. Sasian, R. L. Morrison, M G. Beckman, S. L. Walker, M.J. Wojcik, S. J. Hinterlong, R. J. Crisci, R. A. Novotny, and H. S. Hinton, "Five-stage free-space optical switching network with field-effect-transistor self-electro-optic-effect-device smart-pixel arrays", Applied Optics, Vol. 33. No 8. 1601-1618, March 10, 1994.

22. Motorola, *OPTOBUS Technical Information*, October 1994.

23. S.Matsuo et al. "Photonic switch monolithically integrating an MSM PD, MESFET's, and a vertical-cavity surface-emitting laster", *LEOS'94 Postdeadline Paper,* Boston, MA, October 31-November 3, 1994, pp. PD2.1.

24. David R. Rolston, "Multiple Quantum Well Devices and their Impact on Optical Architectures", Proceedings of the Topical Meeting on Photonic Devices and Systems, pp 77-82, November 2-3, 1994.

25. Y. Lyuu and E. Schenfeld, " Parallel Graph Contraction with Applications to a Reconfigurable Parallel Architecture", *International Conference on Parallel Processing*, St. Charles, Illinois, pp. III-258 - III-265, 1994.

26. Ramaswamy Govindarajan, Guang Gao and Palash Desai, "Minimizing Buffer Requirements under Rate-Optimal Schedule in Regular Dataflow Networks" submitted to the *IEEE Transactions on VLSI Systems*, 1995.

27. Michael R. Gary and David S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness,* W.H. Freeman and Company, New York, New York, 1979.

28. A. Barak and R. Ben-Natan, "Bounded Contractions of Full Trees", *Journal of Parallel and Distributed Computing*, 17, 363-369, 1993.

29. A. Rosenfeld, "Arc Coloring, Partial Path Groups, and Parallel Graph Contractions", *Journal of Parallel and Distributed Computing*, Vol. 7, 335-354, 1989.

30. Cray Research, Inc., *Cray T3D System Architecture Overview,* Revision 1.C, September 23, 1993.

31. T.H. Szymanski, "Notes on the Optical Backplane", Internal Document, June 1995 available by calling (514) 398 5934.

32. Jayaram Bhasker, *A VHDL Primer,* Englewoods Cliffs, NJ, PTR Prentice Hall, 1995.

33. Douglass L. Perry, *VHDL 2nd ed.,* McGraw-Hill, New York, NY, 1994.

34. S.D. Brown, R.J. Francis, J. Rose, Z. G. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, USA, 1992.

35. Betina Hold, Pramod C. Bhatt and Vinod K. Agarwal, "Rapid Prototyping and Synthesis of a Self-testing ABS Controller using CAD Tools", *Proceedings of the Second Canadian Workshop on Field Programmable Devices*, Kingston, Ontario, pp. 2.1.1-2.1.8, June 13-16, 1994.

36. Xilinx, *The Programmable Logic Data Book*, 1994.

37. S. M. Trimberger ed., *Field-Programmable Gate Array Technology*, Kluwer Academic Publishers, USA, 1994.

38. Miron Abramovici, Melvin A. Breuer, Arthur D. Friedman, *Digital Systems Testing and Testable Design*, W.H. Freeman and Company, New York, 1990.

39. Mosis, *CMOSN Cell Library Notebook Vol. 1 1.2 Micron*, Revision 3.0A, 1992.