Design and Implementation of Benes/Clos On-Chip Interconnection Networks

Yikun Jiang
University of Nevada, Las Vegas

DESIGN AND IMPLEMENTATION OF BENES/CLOS ON-CHIP INTERCONNECTION NETWORKS

By

Yikun Jiang

Bachelor of Computer Science
Qingdao Univerty
2005

Master of Computer Science
Harbin Institute of Technology
2007

A dissertation submitted in partial fulfillment
of the requirements for the

Doctor of Philosophy – Electrical Engineering

Department of Electrical and Computer Engineering
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
August 2016

UNLV | GRADUATE COLLEGE

Dissertation Approval

The Graduate College
The University of Nevada, Las Vegas

July 15, 2016

This dissertation prepared by

Yikun Jiang

entitled

Design and Implementation of Benes/Clos On-Chip Interconnection Networks

is approved in partial fulfillment of the requirements for the degree of

Doctor of Philosophy – Electrical Engineering
Department of Electrical and Computer Engineering

Mel Yang, Ph.D.
Examination Committee Chair

Robert A. Schill Jr, Ph.D.
Examination Committee Member

Brendan Morris, Ph.D.
Examination Committee Member

Yingtao Jiang, Ph.D.
Examination Committee Member

Laxmi Gewali, Ph.D.
Graduate College Faculty Representative

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

ABSTRACT

Networks-on-Chip (NoCs) have emerged as the key on-chip communication architecture for multiprocessor systems-on-chip and chip multiprocessors. Single-hop non-blocking networks have the advantage of providing uniform latency and throughput, which is important for cache-coherent NoC systems. Existing work shows that Benes networks have much lower transistor count and smaller circuit area but longer delay than crossbars. To reduce the delay, we propose to design the Clos network built with larger size switches. Using less than half number of stages than the Benes network, the Clos network with 4x4 switches can significantly reduce the delay. This dissertation focuses on designing high performance Benes/Clos on-chip interconnection networks and implementing the switch setting circuits for these networks. The major contributions are summarized below:

- The circuit designs of both Benes and Clos networks in different sizes are conducted considering two types of implementation of the configurable switch: with NMOS transistors only and full transmission gates (TGs). The layout and simulation results under 45nm technology show that TG-based Benes networks have much better delay and power performance than their NMOS-based counterparts, though more transistor resources are needed in TG-based designs. Clos networks achieve average 60% lower delay than Benes networks with even smaller area and power consumption.
- The Lee's switch setting algorithm is fully implemented in RTL and synthesized. We have refined the algorithm in data structure and initialization/updating of relation values to make it suitable for hardware implementation. The simulation and synthesis results of the switching setting circuits for 4x4 to 64x64 Benes networks under 65nm technology confirm that the trend of delay and area results of the circuit is consistent with that of the

Lee's algorithm. To the best of our knowledge, this is the first complete hardware implementation of the parallel switch setting algorithm which can handle all types of permutations including partial ones.

The results in this dissertation confirm that the Benes/Clos networks are promising solution to implement on-chip interconnection network.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 Background

Networks-on-Chip (NoCs) have emerged as the key on-chip communication architecture for multiprocessor systems-on-chip and chip multiprocessors [1]. Achieving scaling performance for future many-core systems will require high-performance, yet energy-efficient on-chip interconnection networks [2]. Existing NoC topologies can be classified into two categories: (1) Multi-hop interconnection networks, like mesh [3], torus, concentrated mesh [4], etc., and (2) Single-hop non-blocking indirect networks, like crossbar , Benes, Clos, etc.

NoC systems, such as the Tilera Tile64 0, utilize a distributed mesh-based network to avoid the scaling issues of long wires. However, this improved scalability comes at the expense of nonuniform cache access (NUCA) latencies [15] and high variability in memory access latencies [6], as well as increased design complexity to guarantee correctness and fairness. The study in [14] shows that mesh network's accepted throughput at any given node is highly dependent on the location of the destination node, as shown in Figure 1. A number of solutions have been developed to solve the problem but at the cost of more complexity routing algorithms and adoption of additional buffers at each router. These buffers consume significant power and area. According to Intel's projections, the interconnection network itself consumes more than 30% of total chip power [16][30].

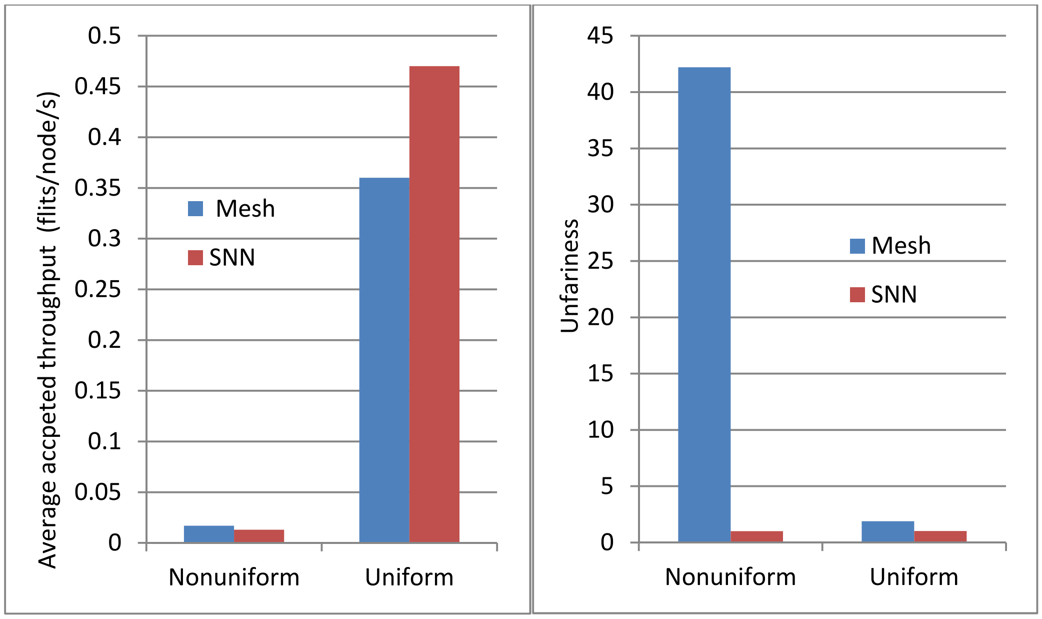


Figure 1 Traffic analysis of mesh- and crossbar-based NoC [14]

Contrastingly, single-hop non-blocking networks eliminate the need of intermediate buffers, and thus can provide uniform latency and throughput, which are very important for cache-coherent many-core systems [14]. Additionally, due to their high bisection bandwidth, non-blocking networks can potentially provide lower complexity solutions with quality-of-service guarantees than multi-hop networks [16][26]. The crossbar-based swizzle-switch network (SSN) achieves significant performance improvement in throughput (21%), cache miss latency (3.0x), and energy savings (25%) than the mesh-based network [14].

Our study is focused on high performance circuit designs of on-chip non-blocking networks, including single-stage networks (i.e., crossbar) and multi-stage networks (Benes and Clos networks). The scalability of crossbar designs is limited by the quadratically increased circuit complexity. Both Benes and Clos networks are rearrangeably non-blocking multi-stage interconnection networks. Benes network is a special case of Clos network which has $N = 2^n$ inputs and outputs. The Benes network is constructed with 2×2 switching nodes recursively.

Due to their non-blocking property and relative smaller number of crosspoints, Benes/Clos networks have received much attention in both academia and industry. Benes/Clos networks have been used in many areas, such as interconnection network in parallel computers, multiprocessors system [31], and networks-on-chip [32][33][34][45][46].

1.2 Circuit Design of Benes/Clos Networks

Compared with direct networks [35][36], Benes/Clos networks can provide uniform latency and throughput, which are very important for cache-coherent many-core systems [37].

As an alternative to crossbars, Benes networks have much lower transistor count and smaller circuit area but longer delay than crossbars [17]. In [20], 3D folded designs of Benes networks and Clos networks built with 2x2 switches are presented. But there is no exploration of Clos networks with larger size switches. In addition, the aforementioned designs are conducted under 130nm or older technology. There is a need to evaluate these network designs under current newer technology.

To reduce the delay, we propose to design the Clos network built with larger size switches. Using less than half number of stages than the Benes network, the Clos network with 4x4 switches can significantly reduce the delay. The circuit designs of both Benes and Clos networks in different sizes are conducted considering two types of implementation of the configurable switch: with NMOS transistors only and full transmission gates (TGs). The layout and simulation results under 45nm technology show that Clos networks achieve average significant lower delay than Benes networks with even smaller area and power consumption.

1.3 Switch Setting Algorithms

In packet switching systems, the switch fabric must be able to provide internally conflict-free paths for the requesting packets in each time slot [38]. This is implemented by setting the states of all switches in the network. It is clear that the routing assignment (i.e., switch setting) scheme in Benes/Clos networks has a strong impact to the efficiency of the Bene/Clos networks.

A number of switch setting algorithms have been developed in the past few decades, including sequential algorithms and parallel algorithms. Sequential algorithms such as looping algorithms [39] are designed for circuit switching systems where the switching configuration can be rearranged at relatively low speed. In [39], a switch setting algorithm with a time complexity of $O(NlogN)$ is proposed based on Waksman's proof. As a matter of fact, using sequential algorithm, the $N \times N$ Benes network cannot be set up in less than $O(NlogN)$ time complexity, because there are $O(NlogN)$ switches. The set-up time is much longer than the latency in Benes networks, which is $O(logN)$ for $N \times N$ network. In order to obtain a switch setting algorithm that has complexity comparable to the network latency, parallel algorithms are needed.

1.4 Contributions

This dissertation is focused on designing high performance Benes/Clos networks and implementing the switch setting circuits for these networks. Specifically, the circuit designs of Benes/Clos networks are completed with two types of implementation of the configurable switch: NMOS transistors only and full transmission gates (TGs). NMOS transistors only and full transmission gates (TGs). The best parallel switch setting algorithm, Lee's algorithm, is implemented in hardware. The switch setting circuit can be integrated with Benes network circuit to be used in high-performance network-on-chip systems.

The following contributions are made in this dissertation:

1. Transmission gates are used to design Benes/Clos networks. The layout and simulation results under 45nm technology show that the TG-based Benes networks have much better

timing and power performance than their NMOS-based counterparts, though more transistor resources are needed in TG-based designs.

2. Simulation results confirm that Benes/Clos networks are promising alternatives to crossbars. Clos networks achieve average 60% lower delay than Benes networks with even smaller area and power consumption.
3. The Lee’s switch setting algorithm is fully implemented in RTL and synthesized. We have refined the algorithm in data structure and initialization/updating of relation values to make it suitable for hardware implementation.
4. The simulation and synthesis results of the switching setting circuits for 4x4 to 64x64 Benes networks under 65nm technology confirm that the trend of timing and area results of the circuit is consistent with that of the Lee’s algorithm. To the best of our knowledge, this is the first complete hardware implementation of the parallel switch setting algorithm which can handle all types of permutations including partial ones.

1.5 Organization

This dissertation is organized as follows. Chapter 2 summarizes the related work done for single-hop networks including Crossbar, Bene/Clos networks. Also, existing parallel switch setting algorithms are reviewed.

Chapter 3 is focused on the circuit design of Benes/Clos networks with two types of implementation of the configurable switch: NMOS transistors only and full transmission gates (TGs). The layout and simulation results of the network circuits under 45nm technology are presented and analyzed.

In Chapter 4, the RTL implementation of Lee’s parallel switch setting algorithm is described. The simulation and synthesis results of the switching setting circuits under 65nm technology are presented.

Chapter 5 concludes the dissertation and suggests the future work.

CHAPTER 2 RELATED WORK

In this chapter, the existing research on NoC will be reviewed, including the following topics: circuit design of single-hop networks and parallel switch setting algorithms.

2.1 Circuit Design of Single-Hop Networks

2.1.1 Crossbar

To provide better bandwidth, Crossbars are used to replace bus-based interconnect fabrics in early multi-core systems, like in the Niagrara2 [7] and IBM BlueGene [13]. Crossbar-based architectures not only can provide the uniform memory access latency that is unachievable in multi-stage NoC systems, but also can potentially provide higher bisection bandwidth and lower complexity solutions for quality-of-service guarantees than NoC designs. Despite these advantages, large crossbars are generally considered infeasible because the area and power of traditional matrix-style crossbars grow quadratically with crossbar radix [8]. Therefore, it is commonly believed that they become overly expensive for radixes above 32 or 64 [9]. In some work, the crosspoint queueing (CQ) was adopted to replace the traditional input queueing (IQ) to simplify complexity and improve the performance of crossbar scheduling [10]. However, crosspoint queueing was found expensive due to the high partitioning of the switch memory; since there is one memory per crosspoint, the total number of memories grows as $O(N^2)$, which is costly for flow and congestion control algorithms [11][12]. Furthermore, the work in [20] proposed the hierarchically-queued crossbar (HQ) as an organization that lowers memory partitioning. In this organization, an $N \times N$ crossbar is partitioned in $(N/k)^2$ $k \times k$ sub-crossbars and memories are placed only at the inputs and outputs of the sub-crossbars. Hence, the total number of memories is reduced from $O(N^2)$ to $O(N^2/k)$. Unfortunately, this organization has a major disadvantage: although partitioning is lowered, it remains unacceptably high, especially when N is large. The reason is that each sub-crossbar has to be relatively small in order to be efficiently scheduled, which, in turn, implies a small k and a quick growth rate of total number of memories.

Existing crossbar circuits mainly adopt MUX-based designs and matrix-based designs [17]. For MUX-based crossbar designs, the latency experienced by a signal depends on crossbar size. For data paths constructed using 2-to-1 multiplexers, doubling the number of inputs results in an additional multiplexer on each line [17]. Pipelined crossbars are proposed to speed up MUX-based designs. Both IBM C64 [18] crossbar and 128x128 crossbar [22] are pipelined MUX-based designs. The IBM Cyclops64 is a 96x96 96-bit-wide crossbar implemented in a 90nm technology, running at 533MHz, and occupying 27mm^2, including the circuits for queueing, arbitration, and flow control [18]. In [22], a 128x128 32-bit-wide crossbar switch is implemented in 90nm CMOS standard-cell ASIC technology. The crossbar operates at 750MHz and provides a port capacity above 20Gb/s, while fitting in a silicon area as small as 6.6mm^2 by filling it at 90% level (control not included). Though the throughput is dramatically improved with pipelined crossbar designs, the port-to-port latency is kept undesirable.

The complex wire interleaving in traditional MUX-based crossbars causes the layout challenge at high bus widths [14]. Most recent crossbars use matrix-style structures. The results in [17] show that for the same size crossbar, compared with the MUX-based design, the matrix-based design reduces the transistor count by 90% and latency up to 50%. Conventional matrix-based crossbars consist of the switching fabric and a separate arbiter that configures the crossbar. This decoupled approach imposes the routing challenge and complexity in arbiter design when the radix of the crossbar increases. Passa's work proposed a novel microarchitecture that inverts the locality of wires by orthogonally interleaving the input with the output arbiters, thus reducing the routing area from $O(N^4)$ to $O(N^2\log^2N)$. However, the prohibitive overhead of the arbiter (consuming 60% of total crossbar area) still limits the design to scale when implementing high-radix crossbar.

2.1.2 Benes/Clos Networks

Though matrix-based crossbars overbeat MUX-based designs in both area and delay [17], their scalability is still limited by the quadratically increased circuit complexity. As an alternative to crossbars, Benes networks have much lower transistor count and smaller circuit area but longer delay than crossbars [17]. The circuit design of 2D Benes network shows that the Benes network significantly reduces the transistor count and power consumption compared with the same size matrix-based crossbar design. While the latency result of the Benes network is worse than that of the matrix-based crossbar counterpart [17]. In [20], 3D folded designs of Benes networks and Clos networks built with 2x2 switches are presented. The numerical analysis shows that 3D folded design can help improving the latency result. But this work only provides the 3D folded design in theoretical aspect, there is no actual layout. As a matter of fact, the TSVs cannot satisfy the density requirement of high-radix crossbar connection because each TSV needs a large pad area to guarantee the quality. Besides, in this work there is no exploration of Clos networks with larger size switches.

Clos networks have been adopted to interconnect multi/many cores [16][19] in 2D and 3D structures. However, to the best of our knowledge, there is no work on using the Clos network as a replacement of crossbars.

In the literature, the studies on circuit design of crossbar and Benes/Clos networks are limited. Due to the lack of appropriate wire models, it’s very inaccurate to conduct circuit level simulation for designs, which completely neglects any effect physical routing might have on circuit performance. The transistor level simulation might yield important information about the behavior of the actual physical circuits. However, there seems to be few studies based on transistor level simulation of these network circuits although delay boundaries in terms of number of switches and path length have been established.

In [17], the transistor level circuit designs of matrix/MUX-based crossbars and Benes network are accomplished. But from the layout view presented by the author, the following problems are observed: 1) the transistor level layout design is not optimized enough; 2) the maximum size of the design is limited to 16x16, which cannot provide enough results to justify the trend for larger size crossbars. In addition, their aforementioned designs are conducted under 130nm or older technology. There is a need to conduct transistor level circuit designs of these networks under current newer technology.

2.2 Parallel Switch Setting Algorithms

In [42], Nassimi and Sahni developed a parallel switch setting algorithm which runs significantly faster than the sequential algorithm based on Waksman's proof [41]. The complexity of this algorithm depends on the parallel computer model and the number of processing elements available. Four SIMD models with different topologies are studied as follows:

1. Completely Interconnected Computer (CIC): In a CIC model, every pair of processing elements is connected directly. The time complexity is $O(log^2 N)$.
2. Mesh-Connected Computer (MCC): In this model, the processing elements are logically arranged as in a k-dimensional array. The time complexity is $O(\sqrt{N}\, log^2 N)$.
3. Cube Connected Computers (CCC): In this model, all processing elements are connected like a cube. The time complexity is $O(log^4 N)$.

4. Perfect Shuffle Computer (PSC): This model employs the shuffle connection of Stone's work [47]. The time complexity is $O(log^4 N)$.

We can see that the time complexity of topologies other than CIC is fairly high. However, CIC is simply too complex to be realized. In addition, this parallel algorithm [42] cannot handle the partial permutations. Implementing the algorithm in SIMD systems is not efficient enough comparing to its complexity. The authors also proposed a self-routing algorithm for Benes network [42] to route through the network using destination tags. However, this algorithm cannot route all permutations [50]. In [48], a fast parallel algorithm is proposed with pipelining which achieves $\mathrm{U(logN)}$ speedup than Nassimi and Sahni's algorithm for unicast assignments on both CIC and extended shuffle-exchange network. Lu and Zheng propose a fast parallel algorithm which can route K connections in $\mathrm{O(logNlogK)}$ for rearrangeable non-blocking networks based on edge-colorings of bipartite graphs [49]. A list of parallel routing algorithms is surveyed in [50].

In [44][49], Lee and Liew present a parallel routing algorithm for Benes Networks. It has time complexity $\mathrm{O(log^2N)}$ which is same as CIC but using only $\mathrm{N/2}$ processing elements [42]. This algorithm was developed based on the previous work in [41] and [42], but can handle the partial permutation problem. In addition, the algorithm can be extended and applied to Clos networks with two's power number of central modules. In the literature, there is nearly no hardware implementation of this parallel algorithm. In [39], a simple hardware design based on Lee's algorithm for 16×16 Benes network in FPGA is presented. However, no detailed design and simulation results are shown in that paper. Another problem about [39] is that, the work is only limited to the switch setting unit for the first stage of 16×16 Benes network. Without the design of the switch setting circuit for different size networks, there is no way to tell the trend of how the hardware cost would increase correspondingly when the network size grows.

In this dissertation, we will focus on designing high performance Benes/Clos on-chip interconnection networks and implementing the switch setting circuits for these networks.

CHAPTER 3 CIRCUIT DESIGN OF BENES AND CLOS NETWORKS

3.1 Non-Blocking Networks

The major performance metrics of the circuit designs of non-blocking networks include delay, area, and power consumption. The number of stages of a network is the key factor determining the delay of the network. Generally, for networks built with the same type of logic units, more stages means longer delay. The determining factor of area and power consumption is the transistor count. In this section, we describe the properties of three types of non-blocking networks that are the determining factors of their performance.

3.1.1 Crossbar

The crossbar is a strictly non-blocking network, i.e., any permutation of inputs and outputs can be realized without confliction. As shown in Figure 2, each input port is connected to each output port through a dedicated logic unit, which is composed of one configurable switch, the basic component used in our circuit design. The number of logic units needed for an NxN crossbar is N^2.

The number of stages traversed from one input output to one output port is only one. However, the circuit complexity of crossbars increases quadratically with the crossbar's size. The resulted high power consumption and die area limits the use of crossbars for large-scale NoCs.

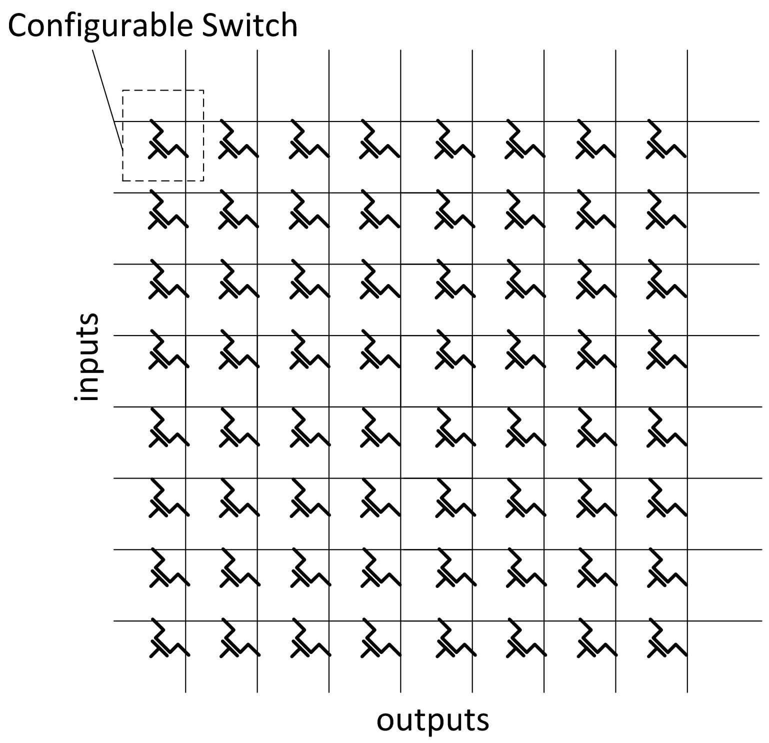


Figure 2 8x8 Crossbar

3.1.2 Benes Network

An NxN Benes network basically is built with two symmetrical NxN butterfly networks. Larger size Benes networks can be built with smaller Benes Networks recursively. The basic logic unit is a 2x2 crossbar switch. As shown in Figure 3.

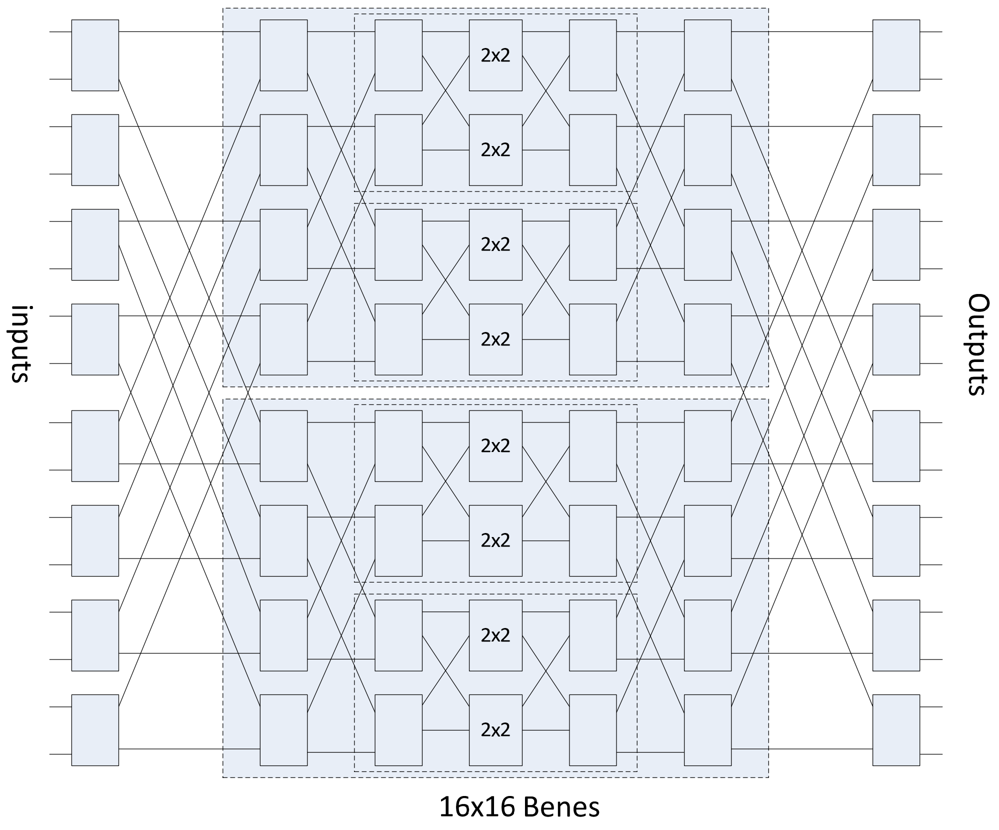


Figure 3 Benes network

Base on this recursive nature, the number of logic units used in NxN Benes network can be derived as:

$$\begin{cases} f(N) = N + 2f(N/2); \\ f(2) \ = 1; \end{cases} \quad =>$$

$$f(N) = Nlog_2^N - \frac{N}{2} \tag{1}$$

Equation(1) shows the amount of logic units used in a Benes network is significantly reduced compared to the amount of logic units used in the same size crossbar.

In Benes networks, the number of stages traversed from an input port to an output port increases as the network size increases. The relation between the number of stages of a Benes network and the network size is derived below.

$$\mathrm{s(N)} = 2\log_2^{\mathrm{N}} - 1 \tag{2}$$

Though $\mathrm{O}(\log_2^{\mathrm{N}})$ is a slow increasing function of N, it is desirable to reduce the number of stages.

3.1.3 Clos Network

To reduce the number of stages for the same size Benes network, we consider Clos networks [22]. A Clos network is composed of three stages of crossbar switches: the input stage, middle stage and the output stage. Each stage is made of a number of same size crossbar switches. A Clos network is defined with a triplet (m, n, r), where m represents the number of switches at the middle stage, n represents the number of input (resp. output) ports of each switch at the input (resp. output) stage, and r is the number of switches at input/output stages. Each input stage crossbar switch has m outputs, each connecting to one of the middle stage switches.

Based on the definition of the Clos network, Figure 4 shows the semi-recursive Clos networks built with 4x4 and 2x2 crossbar switches. Figure 5 (a) and (b) show the structures of the two size switches made by crossbar. The number of logic units (i.e., 2x2/4x4 switches) used in such Clos network is derived as:

$$\begin{cases} f(N) = \dfrac{N}{2} + 4f\left(\dfrac{N}{4}\right); \\ f(2) \ = 1, f(4) = 1; \end{cases} \Rightarrow$$

$$f(N) = \begin{cases} \dfrac{N}{2}(log_4^N) - \dfrac{N}{4}; & N = 4^k, k \in I, k \geq 1 \\ \dfrac{N}{2}(log_4^{2N}); & Otherwise \end{cases} \tag{3}$$

Equation (3) shows the number of logic units needed for Clos networks is much smaller than the number of logic units needed by Benes networks. Notice that in (3), when $\log_4^{\mathrm{N}}$ is not integer, like $\mathrm{N} = 8, 32\ 128, \dots$, the most middle stage is composed of 2x2 switches. Fig. 4(b) and (d) show this type of Clos network.

The relation between the number of stages of a Clos network and the network size is shown in Equation (4).

$$S(N) = \begin{cases} 2(log_4^N) - 1; & N = 4^k, k \epsilon I, k \geq 1 \\ 2(log_4^{2N}) - 1\ ; & Otherwise \end{cases} \tag{4}$$

As we can see from Equation (4), for the $\mathrm{m} = 4$, the number of stages increase 2 every time when the radix of Clos network break through the line of 4^{i}, and i is an integer. For the example shown in Figure 4, for (a) and (b) with radix 8x8 and 16x16 respectively, then they have 3 stages for going through the whole Clos network. Once the radix break s through 16 which is 4^2, as shown in (c), then the Clos has 5 stages.

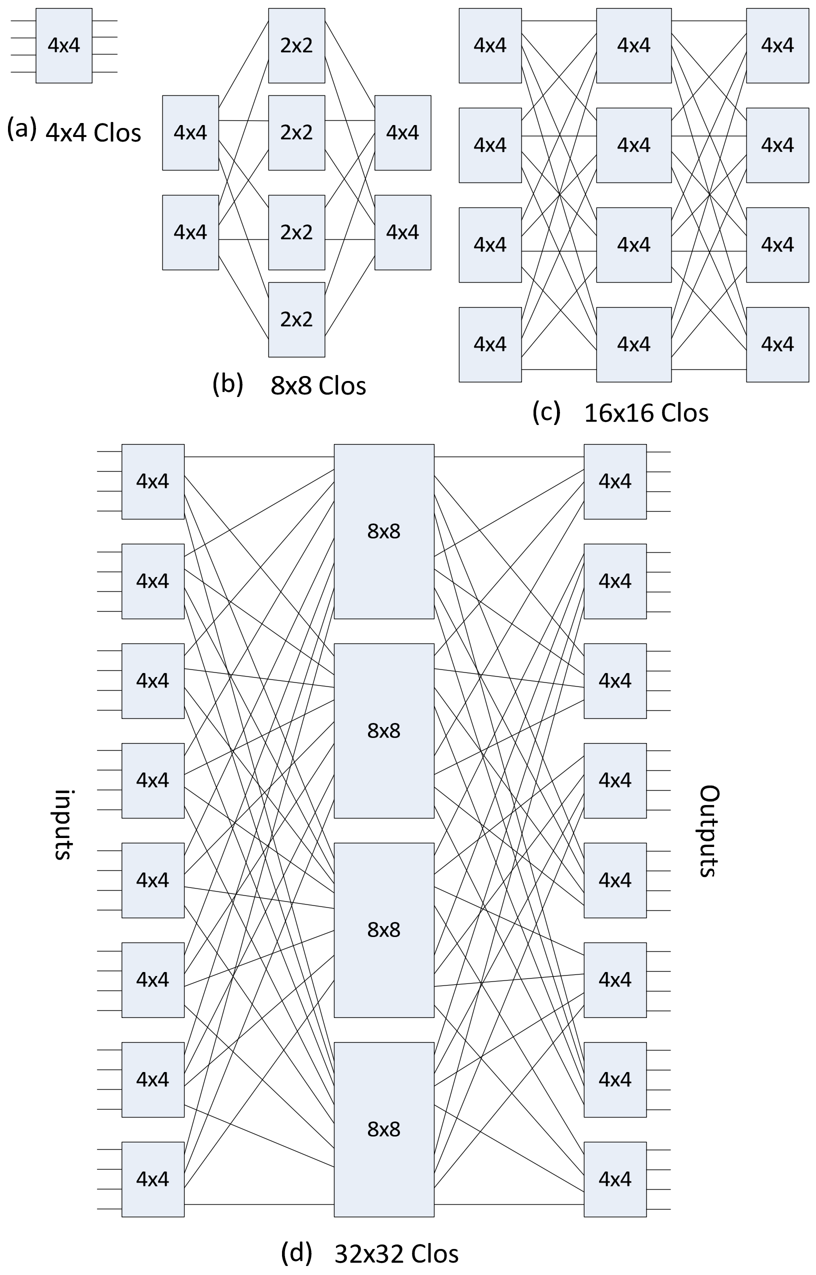


Figure 4 Clos Network

3.1.4 Comparison of Clos and Networks

Comparing Eqns. (1) and (3), the number of logic units of Clos networks is about half of the number of logic units used in Benes networks. Notice that the logic unit represents different size crossbar switches in these two networks. As shown in Figure 5 (a) and (b), the 2x2 crossbar is made of 4 configurable switches, while the 4x4 crossbar is made of 16 configurable switches. Table 1 lists the number of logic units and configurable switches for different sized Benes and Clos networks.

Table 1 Networks

Size	**Clos**		**Benes**	
	Num of logic units	***Num of configurable switches***	***Num of logic units***	***Num of configurable switches***
4×4	1	16	6	24
8×8	8	80	20	80
16×16	12	192	56	224
32×32	48	576	144	576
64×64	80	1280	352	1408
128×128	256	3328	832	3328
256×256	448	7168	1920	7680

Table 2 Number of Stages for Benes and Clos networks

Size	**Clos**	**Benes**
4×4	1	3
8×8	3	7
16×16	3	9
32×32	5	11
64×64	5	13
128×128	7	15
256×256	7	17

Eqns. (2) and (4) show the number of stages needed is reduced from $O(\log_2^N)$ in Benes networks to $O(\log_4^N)$ in Clos networks, when N gets larger, this difference is more significant as shown in Table 2.

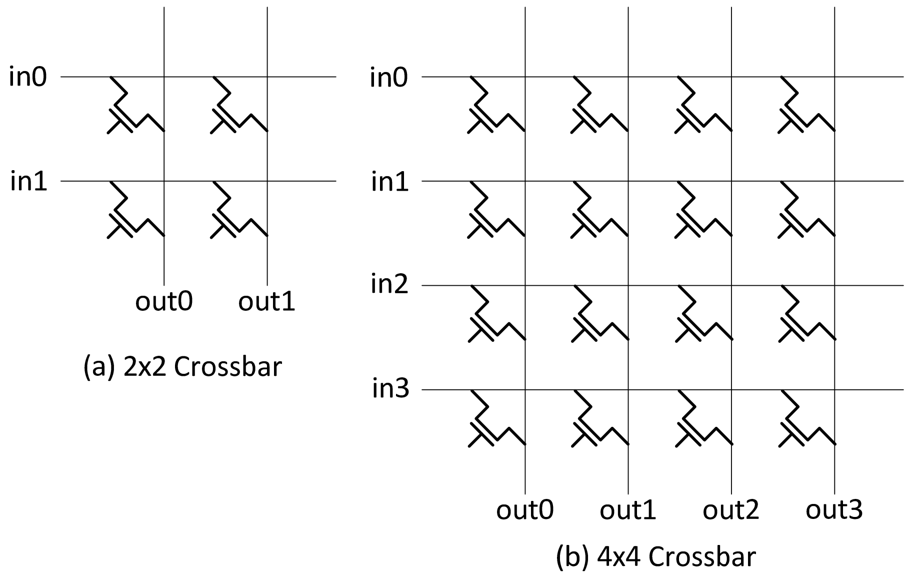


Figure 5 2x2 and 4x4 crossbar switches

As we know, the number of stages traversed by a signal is the determining factor of the delay. For both 2x2 and 4x4 crossbar switches, as shown in Figure 5, only one logic unit will be passed through from an input to an output. The delay for the two crossbar switches should be similar. As such, the total delay experienced from an input port to an output port in Clos shall be much smaller than that in the same size Benes network.

3.2 Design Flow

The Benes and Clos networks of different sizes are designed and simulated through the Cadence design flow, provided by their IC design tools. Circuit level layout and simulations are conducted using Cadence Virtuoso under TSMC 45nm technology. The performance metrics to be compared include critical path delay, area, and power consumption.

Figure 6 illustrates the design flow, which consists of three major steps: schematic design, circuit layout, and simulation. For each specific size Benes/Clos network, the schematic circuit is designed first. After the functional verification for schematic circuit is passed, the layout for each network circuit is drawn based on the schematic circuit. During the layout design period, the DRC checking need be done repeatedly in order to avoid any DRC rule violation.

After the layout is completed without any DRC violation, the netlist with parasitic parameters is extracted from layout. The LVS checking is to ensure the exact matching of the netlist generated match the schematic circuit and the layout circuit.

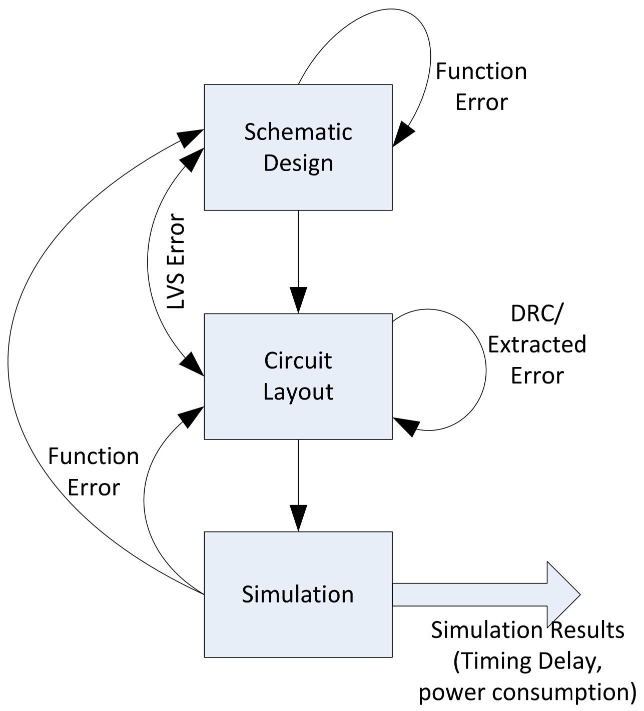
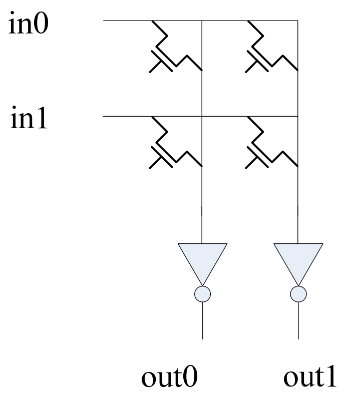


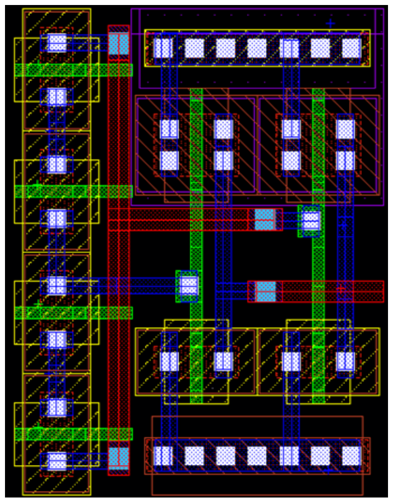
Figure 6 Design Flow

The final step is to simulate the circuits. First the simulation platform need be built using “config” view in Virtuoso. Then the tools embedded in Virtuoso are used to simulate the circuit and generate the delay and power consumption.

3.3 Design of Logic Unit

As described in *Section 2*, the logic unit of a Benes/Clos network is made of 4 or 16 configurable switches. We consider two alternative designs for the basic configurable switch: 1) single NMOS transistor and 2) full transmission gate which uses two transistors (one NMOS and one PMOS).



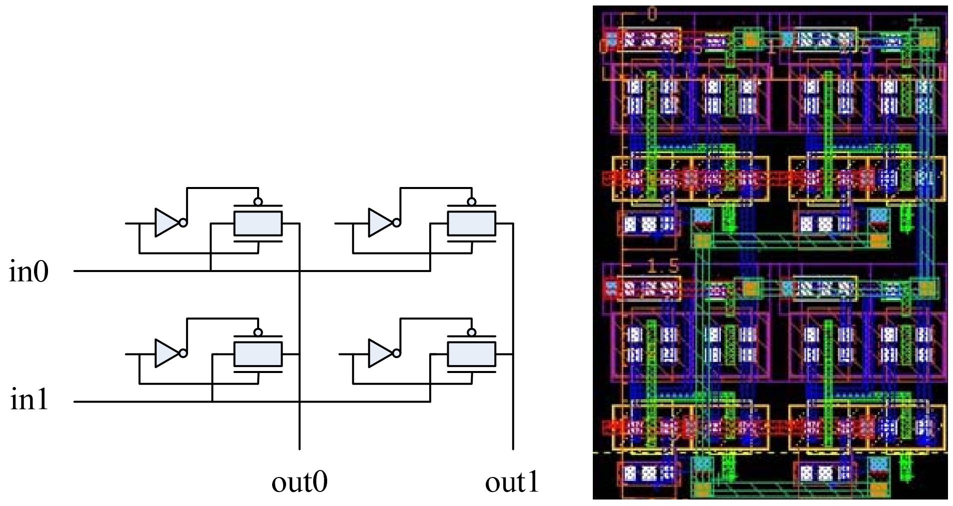


(a) Schematic View (b) Layout View

Figure 7 NMOS-based 2x2 crossbar

Figure 7 (a) and Figure 8 (a) show the schematic diagrams of the two designs of the logic unit of 2x2 crossbar. The number of transistors needed by the first design and the second design is 6 and 16 respectively. But the NMOS transistor has its intrinsic defect known as the “weak 1” problem. The signal passes through the NMOS transistor cannot reach the VDD voltage without the help of a buffer. The transmission gate design uses the complementary manner to control the two transistors turn on or off as shown in Figure 8.

Both transistors are either on or off at the same time to pass or block the signal. When the input signal is '1' ('0'), the PMOS transistor will compensate the weak '1' ('0') from the NMOS transistor.



(a) Schematic View (b) Layout View

Figure 8 TG-Based 2x2 Crossbar

Our experiment shows that under the TSMC 45nm technology, after the signal passes through the first NMOS transistor stage, the strength of the signal can only reach 75% of the original voltage VDD. In order to solve this problem, buffers are added between every two adjacent stages so that the signal strength of all inputs for next stage is kept at VDD, as shown in Figure 7 (a) and (b). The buffers combined with slow rising of signal coming out from the NMOS transistor introduce significant delay for the signal path.

3.4 Schematic and layout Designs

Based on the logic unit design, the schematic circuits of Benes networks are built recursively from 4x4 to 64x64 following Figure 2. Both schematic and layout circuits are laid out manually. In this work, we have completed the layout of NMOS-based Benes networks to 32x32 and TG-based Benes networks to 64x64. The simulation results on delay and power consumption are generated.

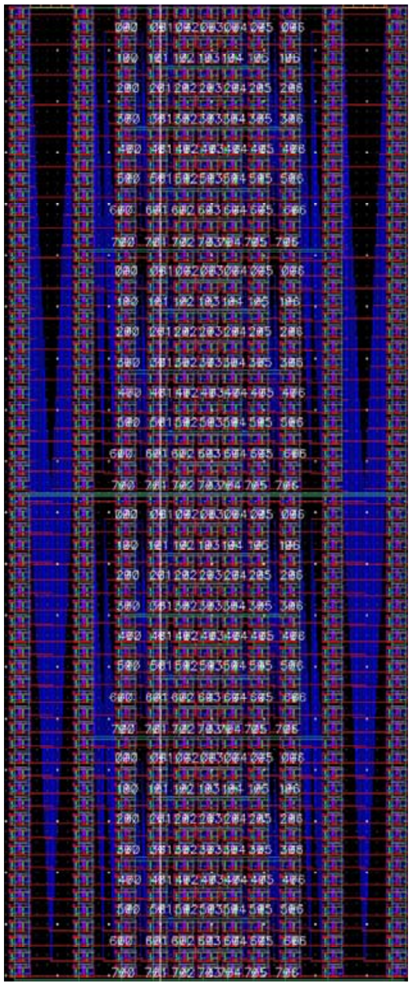


Figure 9 Circuit of 64x64 Benes network

Figure 9 Circuit of 64x64 Benes networkFigure 9 shows layout view of TG-based 64x64 Benes network as an example. When the network size is doubled, the Benes network is duplicated and added with two more stages of crossbar switches. As shown in Figure 9, the 64x64 Benes network includes two 32x32 Benes networks and two input/output stages composed of thirty-two 2x2 crossbar switches.

In the similar way, layout circuits of 4x4 to 64x64 TG-based Clos networks are laid out manually as shown in Figure 10.

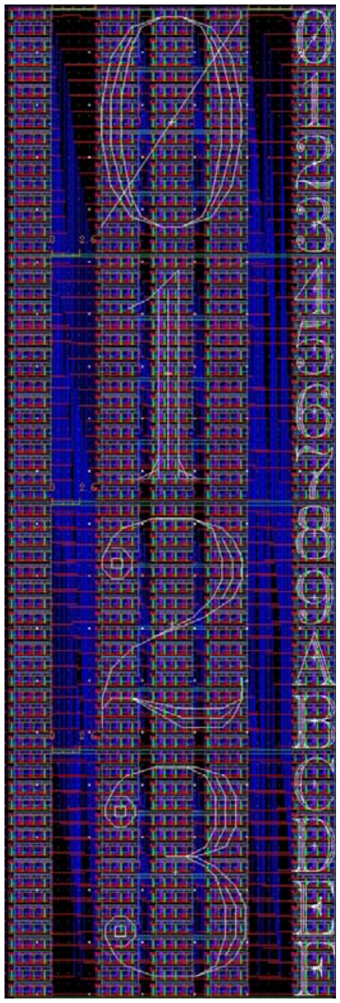


Figure 10 Circuit of 64x64 Clos network

3.5 Experimental Results

In this section, we present the simulation results of all Benes and Clos networks obtained from Cadence Virtuoso simulation tools. The performance metrics including delay, area and power consumption are reported and analyzed. For delay and power consumption metrics, the results are obtained with wire delay model (i.e., RC model).

3.5.1 Delay

Table 3 and Figure 11 show the delay results of NMOS-based and TG-based Benes networks as well as TG-based Clos networks. The delay result shown is the average of the rising-transition and falling-transition delays.

Table 3 Delay *(ns)* of Benes and Clos networks

Size (N×N)	**Benes (NMOS)**	**Benes (TG)**	**Clos (TG)**
4x4	1.622	0.101	0.013
8x8	3.131	0.331	0.192
16x16	4.578	0.625	0.241
32x32	7.034	1.232	0.832
64x64	N/A	1.584	0.920

The wire delay has a significant impact to the delay. And the impact is bigger for larger size designs. This trend is attributed to factors of longer wires for interconnecting the inner and outer stages and increased wire load. As shown in the first two columns, for the same size Benes network, the delay of NMOS-based Benes network is about 10 times of the delay with its TG-based counterpart. The basic reason has been mentioned in Section 4. The rising delay of NMOS transistors along the signal path contributes the most to total path delay. In the design with TGs, the rising delay problem is eliminated, thus the path delay plummets.

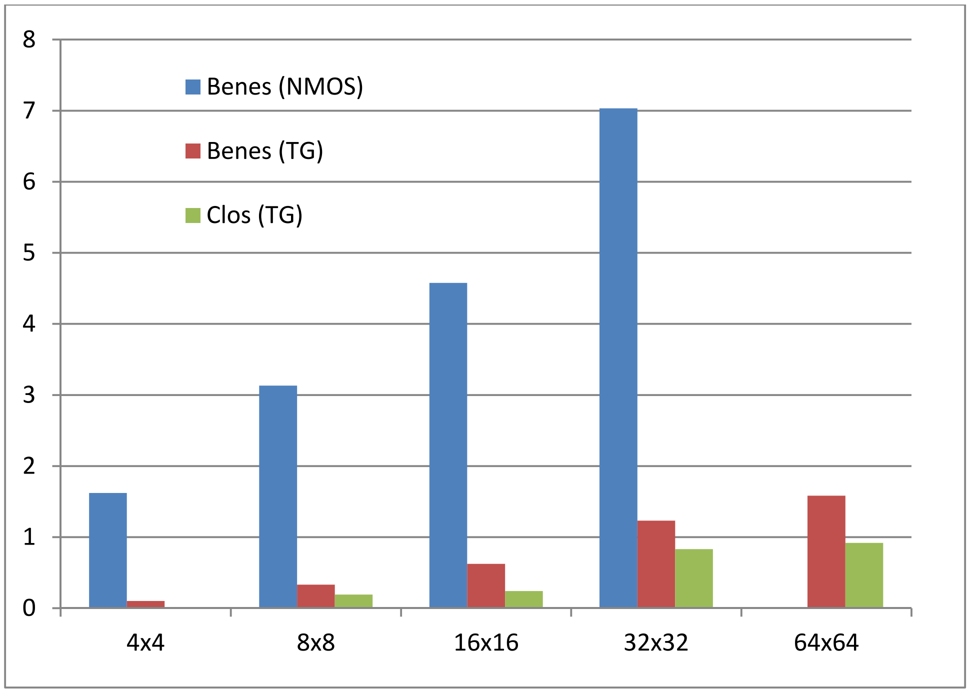


Figure 11 Delay of Benes and Clos networks

The difference of delays between Benes and Clos networks is mostly attributed to the difference between the numbers of stages of these two networks. Table 2 shows the number of stages used for specific size networks, 8x8 and 16x16 Clos networks have as the same number of stages as 4x4 Benes networks. As shown in Figure 11, the delay of 16x16 Clos is larger than that of 8x8 Clos, and both delays are higher than the delay of 4x4 Benes network. The reason is that though the number of stages is the major factor determining delay, the output load also plays an important role in it. The output load of 16x16 Clos is larger than 8x8 Clos, and both are much larger than 4x4 Benes network. This explains the trend of delay results.

3.5.2 Area

Table 4 and Table 5 show the transistor number and area of Benes and Clos networks. The first two columns show that the area of NMOS-based Benes network is much smaller than that of the corresponding TG-based Benes network. Each TG contains four transistors, which is 4 times of NMOS, as explained in Section 4.

Table 4 Transistor Count used in Benes and Clos networks

Size (N×N)	**Benes (NMOS)**	**Benes (TG)**	**Clos (TG)**
4x4	40	96	64
8x8	176	320	320
16x16	540	896	768
32x32	1464	2304	2304
64x64	3696	5632	5120

Table 5 Area of Benes and Clos networks ($\boldsymbol{um^2}$)

Size (N×N)	**Benes (NMOS)**	**Benes (TG)**	**Clos (TG)**
4x4	40.97	75.81	42.56
8x8	154.10	263.05	227.25
16x16	474.42	764.71	624.03
32x32	684.13	1239.2	1125.7
64x64	N/A	3662.8	2954.1

While NMOS-based Benes networks also need adding the inverters (as buffers) between two neighboring stages to reshape the defect signals caused by weak ‘1’ problem. As such, the transistor account of a TG-based Benes network is slightly more than 2 times of its NMOS-based counterpart. The actual area ration between TG- and NMOS-based Benes networks is less than 2:1.

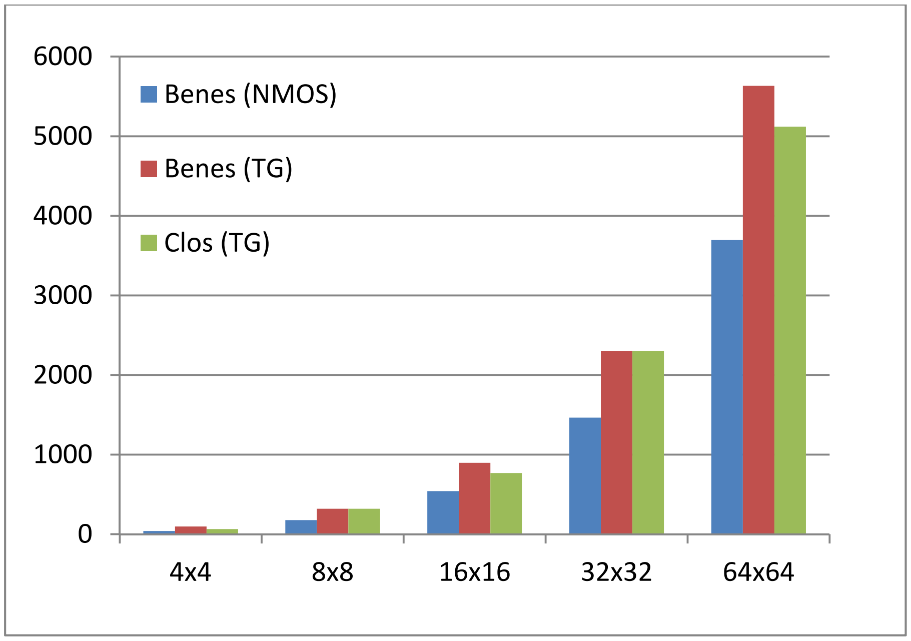


Figure 12 Transistor Count

Figure 12 shows the trend described in Table 4, as we can see, the TG-based Benes and Clos consume similar amount of transistors, which is higher than NMOS-based network. As shown in Figure 13, consistent with the trend of transistor count, the area of NMOS-based Benes network is the smallest among the three designs. The TG-based Clos network has smaller area than TG-based Benes network for all network sizes and the difference is more significant for larger size N. The smaller layout area of Clos networks is attributed to the fact that by using 4x4 crossbars as major building blocks, less interconnects are used and the circuit is more compacted compared with that of Benes networks.

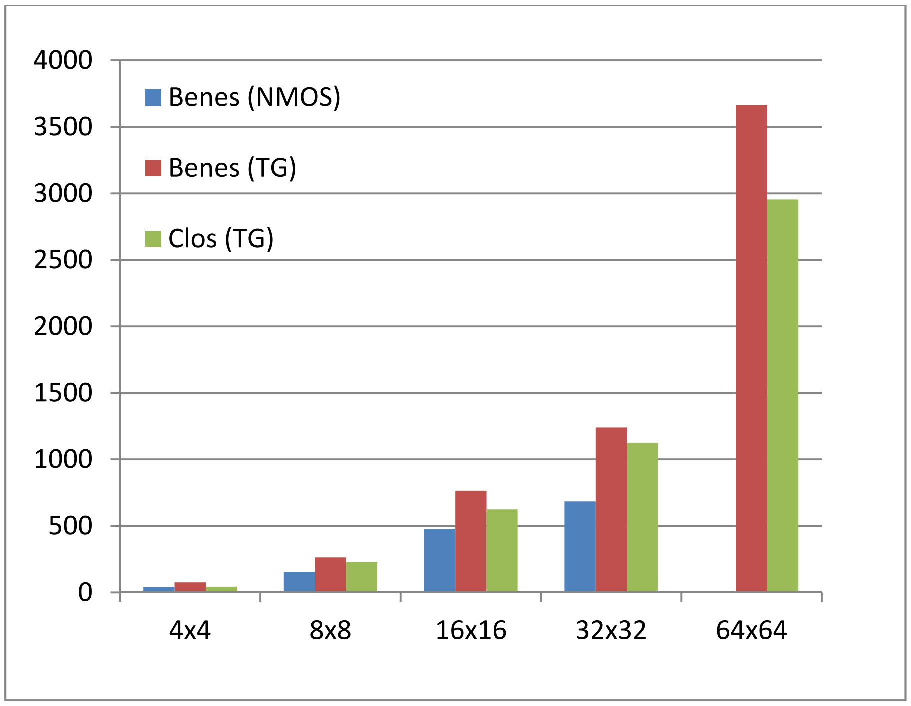


Figure 13 Area of Benes and Clos Networks (um^2)

3.5.3 Power Consumption

Table 6 shows the power consumption for these networks. The results with RC model are much higher than without RC model.

Table 6 Power consumption (uW) of Benes and Clos networks

Size (N×N)	**Benes (NMOS)**	**Benes (TG)**	**Clos (TG)**
4x4	1.149	0.928	0.275
8x8	4.162	2.491	2.084
16x16	8.064	7.954	3.282
32x32	16.23	10.45	8.415
64x64	N/A	32.85	25.7

Similar to delay results, the impact of wire delay is getting bigger with the network size increasing. The power consumption of NMOS-based Benes network is higher than the corresponding TG-based Benes network, because the inverters between two neighboring stages consume significant power to compensate the defect signals caused by NMOS transistor's weak '1' problem.

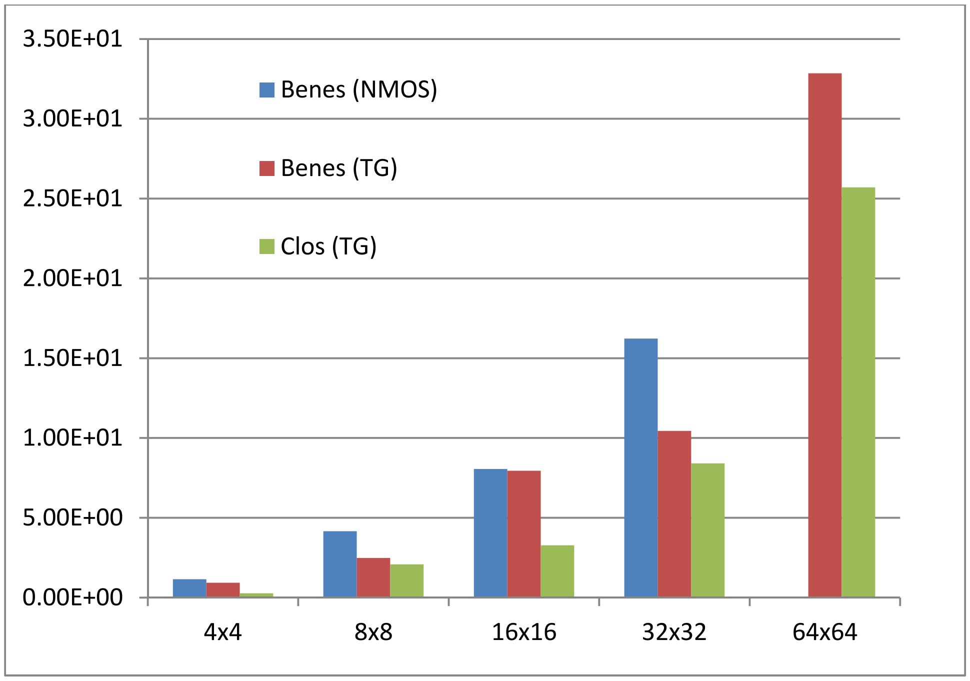


Figure 14 Power consumption ($\boldsymbol{uW}$) of Benes and Clos Networks

Figure 14 shows that comparing TG-based Benes and Clos networks, the Benes networks consume more power. On one hand, the data signals go through more stages in Benes network than in Clos network, on the other hand, the signals in Clos network have larger output load considering the larger size logic unit. The combined effect is that Clos networks have lower power consumption than Benes networks. The improvement (over 20%) is more significant for larger size N as shown in Table 6.

3.6 Summary

This chapter was focused on the circuit designs of different sized Benes and Clos networks considering two types of implementations: NMOS transistor only and full transmission gates. All designs are laid out manually and simulated with using Cadence tools. The experimental results showed that the TG-based Benes networks have much better delay and power performance than their NMOS-based counterparts, though more transistor resources are needed in TG-based designs. Clos networks have 60% delay delay reduction than Benes networks with even smaller area and power consumption. This result confirms that Clos network is a better alternative to Benes networks to replace crossbars in large scale networks.

CHAPTER 4 HARDWARE DESIGN OF PARALLEL SWITCH SETTING ALGORITHM FOR BENES NETWORKS

4.1 Benes Network and Routing Constraints

The Benes network is a special instance of Clos network. An $\mathrm{N} \times \mathrm{N}$ Benes network basically is built with two symmetrical butterfly networks. A Benes network can be considered as a cascaded combination of Omega network and a reverse Omega network overlapped with the middle stage. As such, the Benes network is a symmetric topological structure among the link patterns in the network from center stage. Besides, Benes network is inherently recursive. An $\mathrm{N} \times \mathrm{N}$ Benes network can be built from two $\frac{\mathrm{N}}{2} \times \frac{\mathrm{N}}{2}$ Benes networks recursively, S_{up} and $\mathrm{S}_{\mathrm{down}}$, which represent the up and down Benes subnetwork, respectively. As shown in Figure 15, the 8×8 Benes network can be divided into two 4×4 Benes networks and two extra stages each composed of four 2×2 switching nodes at input side and output side, respectively.

A complete path of Benes network can be decomposed into the forward sub-path and backward sub-path routed in the Omega network and the reverse Omega network, respectively. The two subpaths must meet at one of the switches in the middle stage. Therefore, between any pair of input and output ports of an $\mathrm{N} \times \mathrm{N}$ Benes network, there exist N routing paths.

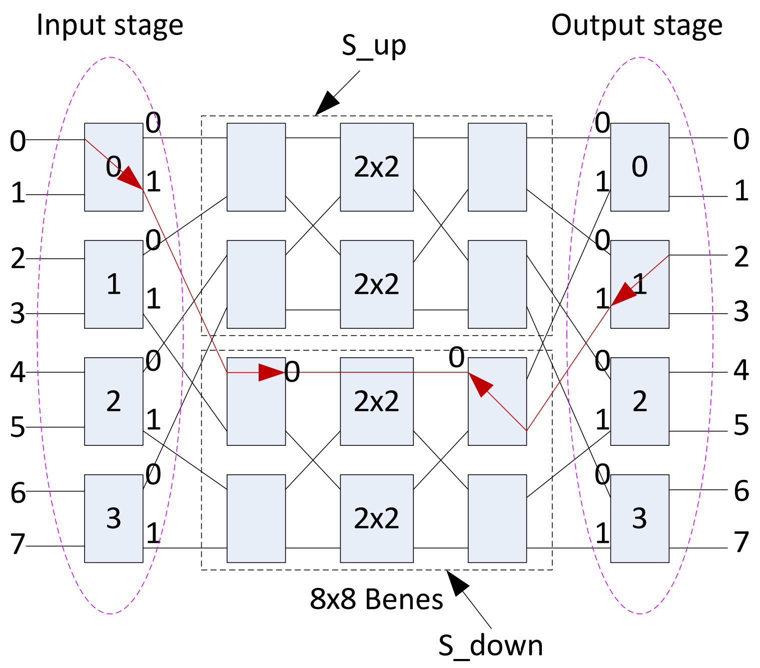


Figure 15 8 × 8 Benes Network

The non-blocking routing in Benes networks is achieved if the following constraints are satisfied:

Symmetric Routing Constraint: To route from input s to output d, either S_{up} or S_{down} subnetwork must be assigned to the subpaths on the Omega network and reserve Omega network simultaneously. This constraint must be held for each inner stage, recursively. As such, when the output state of the switching node at the forward stage in the Omega network is determined, then the input state of the switching node at the symmetric backward stage in the reverse Omega network is also determined.

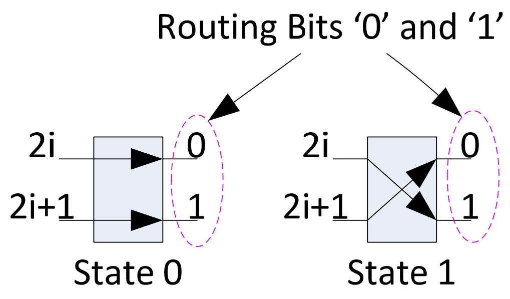


Figure 16 Switching Node State

Internally Conflict-Free Constraint: To avoid confliction between connection requests, the two input ports (resp. output ports) of each input switching node (resp. output switching node) cannot be assigned to the same output port (resp. input port).

Each switching node has two states: '0' (i.e., straight) and '1' (i.e., cross), as shown in Figure 16. Combined with Figure 15, we can see that, any input port of a switching node must connect to the $'0'$ output port to reach S_{up}, or connect to the $'1'$ output port to reach S_{down}. The output states at each stage can be represented as a binary bit (namely, routing bit). The routing bits ($'0'$ or $'1'$), as shown in Figure 16, at all stages compose the path in the Benes network.

Table 7 Routing Bit Values vs. State Values

State of switching node	**0**	**1**
Routing bit of port 2i	0	1
Routing bit of port 2i+1	1	0

Table 7 shows the relation between the switch state and the routing bit corresponding to its input ports. The state of a switching node determines the routing bit value of a port, and vice versa. Following the internal conflict-free constraint, the routing bits of the two input ports of a switching node have to be distinct.

4.2 Lee's Parallel Routing Algorithm

Lee's parallel algorithm can be decomposed into four major steps: initialization, searching, merging and calculating the permutation for subnetworks. Denote the set of input and output ports as I and O, respectively, i.e., $\mathrm{I} = \mathrm{O} = \{0, 1, \dots, \mathrm{N} - 1\}$, and $\pi\colon \mathrm{I} {-}{>}\ \mathrm{O}$ be an input-output permutation indicating connection requests. We use $(\mathrm{i},\ \mathrm{j})$ to indicate the ith input port is going to connect to the jth output port in the permutation. In this part, we will use an example permutation to elaborate the main concept of this algorithm. In the below permutation, $'\ \mathrm{X}'$ means this input port has no output request.

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 3 & 2 & 6 & 4 & 7 & 5 & \mathrm{X} \end{pmatrix}$$

Because of the symmetric routing constraint, the algorithm only need to find out the routing bits of the stages in one Omega subnetwork, then the routing bits of the counterpart stages in the other Omega network will be determined. In Lee's algorithm, the output side switch setting is determined first, and then the input side switch setting is derived.

4.2.1 Initialization

The first step of Lee's algorithm is to build the connections between output switching nodes using relation values. The connection between output switching nodes are built on the internally conflict-free constraint, to avoid this internal conflict, the algorithm need to group switching nodes with the same relation together, and assign the switch state values to them consistently.

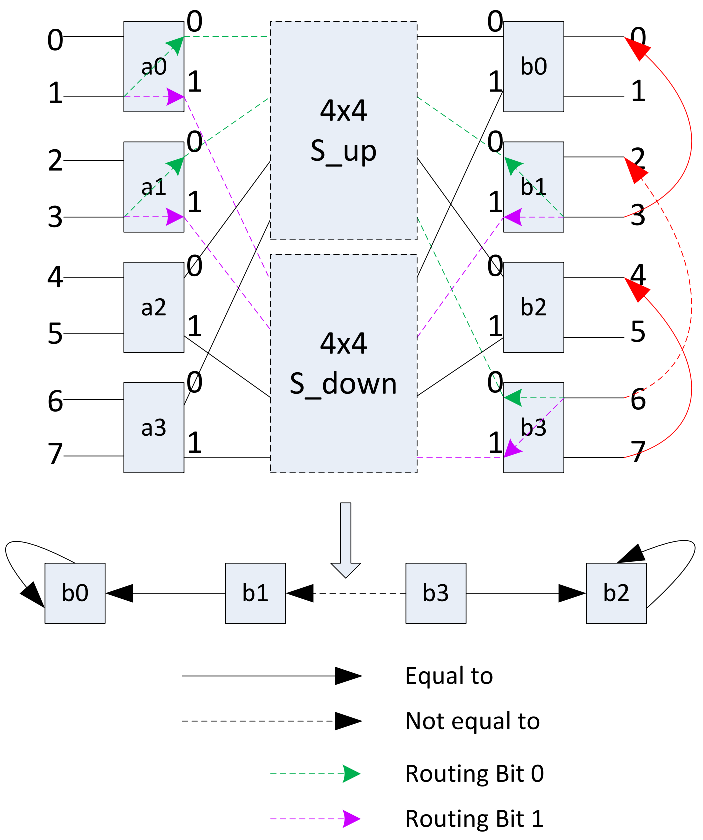


Figure 17 Initialization

Here, we adopt the same notation of [38]. We use $\mathbf{a_i}$ and $\mathbf{b_i}$ to denote the switch state value of input/output switching node $\mathbf{a_i}$ and $\mathbf{b_i}$, respectively. Let $\boldsymbol{\alpha}: \boldsymbol{I} \to \{\mathbf{0}, \mathbf{1}\}$ $\boldsymbol{and}$ $\boldsymbol{\beta}: \boldsymbol{O} \to \{\mathbf{0}, \mathbf{1}\}$, where $\boldsymbol{\alpha}(\boldsymbol{k})$ is the routing bit of **k**th input, and $\boldsymbol{\beta}(\boldsymbol{k})$ is the routing bit from ***k***th output.

From [38], the symmetric self-routing constraint requires that

$$\alpha(k) = \beta(\pi(k)) \qquad k=0, 1, \ldots, N\text{-}1 \qquad (5)$$

The internal conflict-free constraint requires that

$$\alpha(k) = \overline{\alpha}(k+1), \beta(k) = \overline{\beta}(k+1) \qquad k=0, 1, \ldots, N\text{-}2 \qquad (6)$$

The combination of (1) and (2) gives

$$\beta\big(\pi(k)\big) = \alpha(k) = \overline{\alpha}(k+1) = \overline{\beta}(\pi(k+1)) \qquad k=0, 1, \ldots, N\text{-}2 \qquad (7)$$

Then we have

$$\alpha_i = \begin{cases} a_{\frac{k}{2}}, & k \text{ is even} \quad k = 2i \\ \overline{a_{\frac{k-1}{2}}}, & k \text{ is odd} \quad k = 2i+1 \end{cases} \tag{8}$$

$$\beta_i = \begin{cases} b_{\frac{k}{2}}, & k \text{ is even} \quad k = 2i \\ \overline{b_{\frac{k-1}{2}}}, & k \text{ is odd} \quad k = 2i+1 \end{cases} \tag{9}$$

For the given permutation, we have:

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 3 & 2 & 6 & 4 & 7 & 5 & X \end{pmatrix} => $$

$$\begin{pmatrix} a_0 & \overline{a_0} & a_1 & \overline{a_1} & a_2 & \overline{a_2} & a_3 & \overline{a_3} \\ b_0 & \overline{b_1} & b_1 & b_3 & b_2 & \overline{b_3} & \overline{b_2} & X \end{pmatrix}$$

For the ith input switching node, we refer to the output port pair (k, l) corresponding to the input port pair $(2i, 2i+1)$ as a connection pair. Then we obtain:

$$\begin{pmatrix} 2i & 2i+1 \\ k & l \end{pmatrix} => \begin{pmatrix} a_i & \overline{a_i} \\ \beta(k) & \beta(l) \end{pmatrix} \tag{10}$$

Based on Eqn. (11), we have:

$$\beta(k) = \overline{\beta(l)} \tag{11}$$

Consider the given permutation, taking $(\mathbf{1}, \mathbf{3})$ as example. As shown in Figure 17, in order to have the same routing bits ($'\ \mathbf{0}'$ or $'\ \mathbf{1}'$) for input port $\mathbf{1}$ and output port $\mathbf{3}$, the corresponding input switching node must set the state value base on the corresponding output switching node, i.e., for input/output permutation $\binom{\mathbf{1}}{\mathbf{3}}$, we have $\boldsymbol{\alpha}(\mathbf{1}) = \overline{\boldsymbol{a_0}}$, $\boldsymbol{\beta}(\mathbf{3}) = \overline{\boldsymbol{b_1}}$, since $\boldsymbol{\alpha}(\mathbf{1}) = \boldsymbol{\beta}(\mathbf{3})$, then we can get $\mathbf{a_0} = \mathbf{b_1}$. Similarly, for $\binom{\mathbf{3}}{\mathbf{6}}$, we derive, $\overline{\boldsymbol{a_1}} = \boldsymbol{b_3}$. Together, we obtain

$$a_0 = b_0 \quad , \quad \overline{a_0} = \overline{b_1}$$

$$a_1 = b_1 \quad , \quad \overline{a_1} = b_3$$

$$a_2 = b_2 \quad , \quad \overline{a_2} = \overline{b_3}$$

$$a_3 = \overline{b_2} \quad , \quad \overline{a_3} = X$$

After eliminating all **a** from above equations, we can obtain a set of $\boldsymbol{N/2}$ initializing equations as follows:

$$\boldsymbol{b_1 = b_0, \quad b_3 = \overline{b_1}, \quad b_3 = b_2, \quad b_2 = X}$$

These equations about **b** can help us to build the relation connections between output switching nodes as shown in Figure 17. All output switching nodes are connected like a linked list, where the index of the state variable is taken as the node address. Each initializing equation is used to establish a pointer, in which the state variable with larger index points to the other with smaller index.

After initialization step, all output switching nodes can be grouped into equivalent classes. For switching nodes in the same class, the state value of any switching node is relevant to the state value of others. The representative node of each class is the switching node with the smallest index number. For the above example, as shown in Figure 17, all output switching nodes are in the same class. The representative node of the group is $\boldsymbol{b_0}$. Regardless of the Benes network radix, the initialization step is processed at all PEs at the same time with time complexity $\boldsymbol{O(1)}$.

4.2.2 Searching

As shown in Figure 17, there are two pointer types, Type 0 Pointer indicating the two state variables are equal, and Type 1 Pointer indicating the two state variables are not equal. All switching nodes except the representative node in the group will go through the searching step to point to the representative node. The time complexity of searching step is $\boldsymbol{O(logN)}$.

Figure 18 shows the searching result for Figure 17.

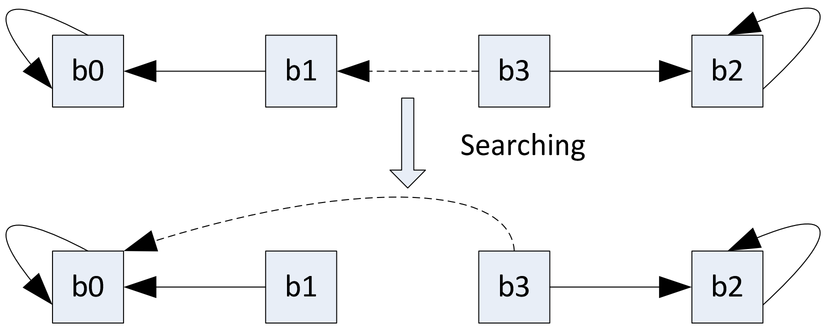


Figure 18 Searching

4.2.3 Merging

Usually, among the nodes belonging to the same class, there should be only one endpoint which is the representative node of the class. If there are two endpoints in one class, then the merging step is needed to eliminate one of them. The time complexity of this merging step is $\mathbf{O(1)}$. Figure 18 shows that the two endpoints $\mathbf{b_0}$ and $\mathbf{b_2}$ are pointed by $\mathbf{b_3}$, which means the value of $\mathbf{b_3}$ will be determined by the values of $\mathbf{b_0}$ and $\mathbf{b_2}$, causing confliction. As shown in Figure 19, after the merging step, the direct connection between two endpoints $\mathbf{b_0}$ and $\mathbf{b_2}$ is found.

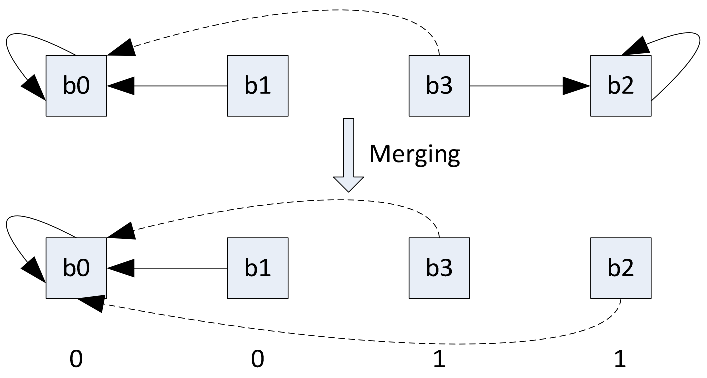


Figure 19 Merging

After all switching nodes point to the representative of the class, the state values of all switching nodes can be determined by assigning the state value of the representative as 0 or 1. One of the assignments of the above example is derived as by letting $\mathbf{b_0 = 0}$:

$$\mathbf{State}\ (\mathbf{b_0}, \mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}) = (\mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{1})$$

By applying the symmetric routing constraint, the state values of input switching nodes should be setup as:

$$\mathbf{State}\ (\mathbf{a_0}, \mathbf{a_1}, \mathbf{a_2}, \mathbf{a_3}) = (\mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0})$$

Figure 20 shows the settings of input/output switching nodes for permutation

$$\begin{pmatrix} \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} \\ \mathbf{0} & \mathbf{3} & \mathbf{2} & \mathbf{6} & \mathbf{4} & \mathbf{7} & \mathbf{5} & \mathbf{X} \end{pmatrix}$$

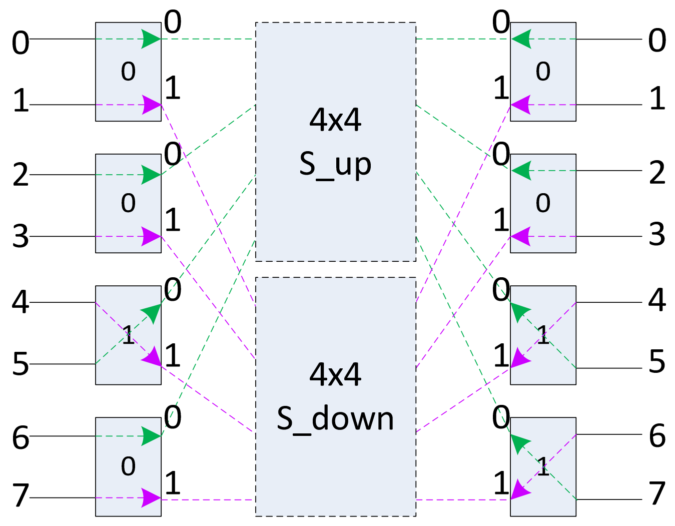


Figure 20 Settings of input/output switching nodes

4.2.4 Permutation for subnetwork

After the state values of input/output switching nodes are determined, the switch settings of two inner $\frac{N}{2} \times \frac{N}{2}$ subnetworks can be determined recursively. The permutations of the two inner subnetworks can be derived by tracing the routing paths from both input and output sides.

Then Lee's algorithm is applied to derive the state values of the input/output switching nodes of the two subnetworks. The time complexity to calculate those permutations for subnetworks is $\boldsymbol{O}(\mathbf{1})$. In a recursive manner, the state values of all stages will be computed by the Lee's parallel routing algorithm.

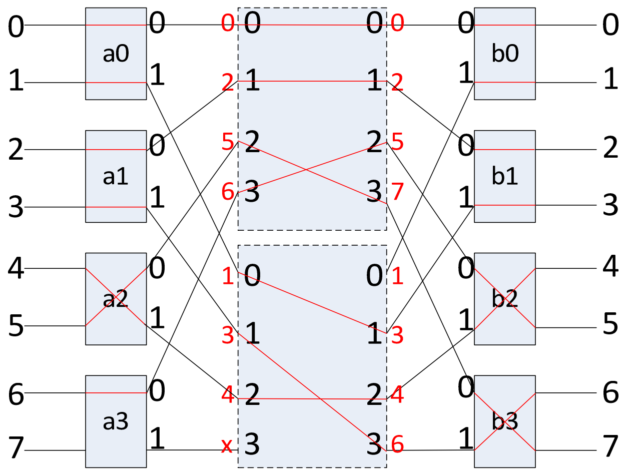


Figure 21 Permutation for Subnetwork

Figure 21 shows the connections of the two inner subnetworks and the derived two permutations π_0 for S_{up} and π_1 for S_{down} for two inner subnetworks, respectively.

$$\pi_0 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 3 & 2 \end{pmatrix}$$

$$\pi_1 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 3 & 2 & x \end{pmatrix}$$

Continue this process until the state values of switching nodes in the middle stage are determined.

As we can see from the description in above section, the searching step is the only procedure which is relevant to the radix of Benes network. All the other procedures could be finished in $O(1)$. The time complexity for each round is determined by the searching procedure which is $O(logN)$.

4.3 Hardware Design of lee's algorithm

4.3.1 Design Flow

As shown in Figure 22, the hardware design of Lee's algorithm follows the common RTL design flow which consists of four steps: 1) Specification, 2) RTL design, 3) simulation of the RTL code, 4) synthesis of the RTL design. In the second step, we use Verilog HDL to implement the RTL design of Lee's parallel algorithm.

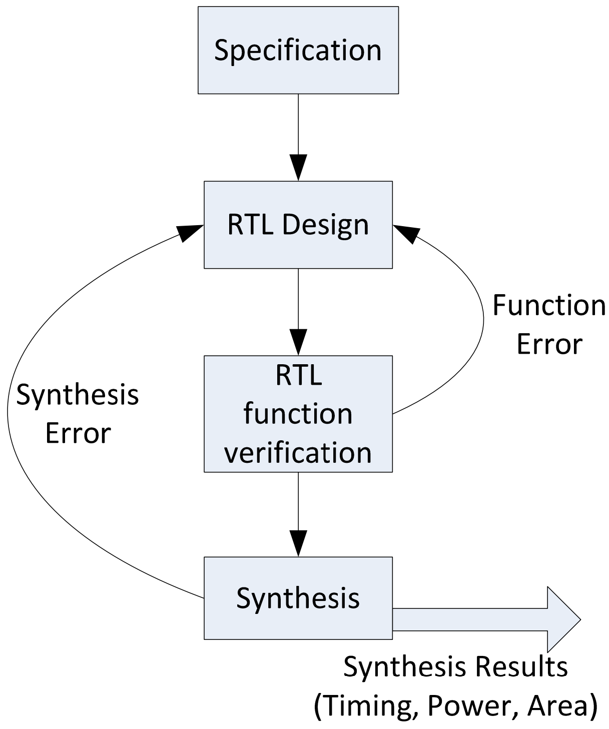

Specification
RTL Design
Function Error
Synthesis Error
RTL function verification
Synthesis
Synthesis Results (Timing, Power, Area)

Figure 22 Design Flow

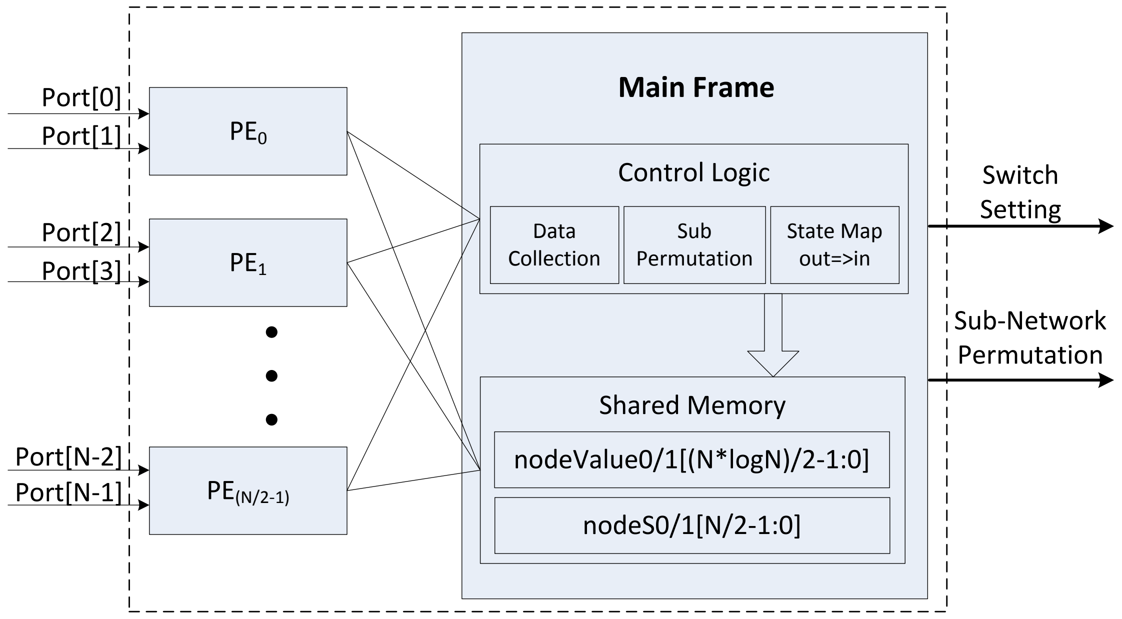


Figure 23 Circuit Architecture

As shown in Figure 23, for the switch setting circuit of $N \times N$ Benes network, there are $N/2$ processing elements (PE), each representing an output switching node, are connected by the main frame. Each PE$_i$ holds several variables. In the main frame, two major parts are the control logic and shared memory. Table 8 lists the variables used in our design. For $N \times N$ Benes network, each variable storing port index has $n = log_2^N$ bits. The global variables are shared among all processing elements.

As each output switching node (represented by one processing element) has two ports, 0 and 1, we adopt a two-register structure for each output switching node to store the pointers associated with port 0/1. In the searching step of Lee's algorithm, each PE may need search in two directions. The two-register structure allows each PE keeps searching in two directions until they reach the representative nodes. Here four variables are used for storing the index of the node ($\mathrm{nodeValue0/1}$) pointed by the port 0/1 pointer and corresponding relation value ($\mathrm{nodeS0/1}$), respectively. The size of these shared registers is determined by the radix of Benes network. For

$N \times N$ Benes network, the size of $\mathrm{nodeValue}0/1$ is $(N/2) * logN$ bits as there are $N/2$ output switching nodes and logN bits are needed to represent the index of each port. The size of $\mathrm{nodeS}0/1$ is $N/2$ as one bit is needed to represent the relation value between two connected switching nodes, ‘0’ represents not equal, ‘1’ represents equal

Table 8 Definition of Variables

Global Variable	**Meaning**	**Size (bits)**
port[N]	Store the output port index of the permutation.	NlogN
nodeValue0/1[N/2]	Store the index of the port which is pointed by the port 0/1 pointer of each output switching node. For example, $\mathrm{nodeValue0/1[i] = j, 0 <= j < i <= N/2}$, means node i points to node j, i.e., there is a relation connection between node i and node j.	$\frac{\mathrm{NlogN}}{2}$
nodeS0/1	Store the relation value for the connection from the port 0/1 pointer of each output switching node.	$\mathrm{N/2}$
inNodeStateValue[N/2]	Store the state value of input switching nodes.	$\mathrm{N/2}$
outNodeStateValue[N/2]	Store the state value of output switching nodes.	$\mathrm{N/2}$
sub0/1_port[N/2]	Store the permutations for two inner subnetworks.	$\mathrm{logN/2}$
Local Variable	**Meaning**	**Size (bit)**
port0/1	Store the output port index of the connection pair corresponding to input port pair $(2i, 2i + 1)$.	logN
preNodeValue0/1	Store the $\mathrm{nodeValueL0/1}$ before each searching procedure.	logN
nodeType	Two-bit value, '00' means the node doesn't point to any other node; '01' if the node points to only one other node, '11' if it points to two other nodes.	2

The control logic is responsible for the following functions:

1. Maintaining and updating the registers' data and status respectively, according to the newest information received from processing elements.
2. Calculating the setting value for switching nodes on the inputs/outputs stage.
3. Calculating the input/output permutation for the subnetworks.

Every clock cycle, the control logic gathers the updated values of $\mathrm{nodeValue0/1}$and $\mathrm{NodeS0/1}$ from all processing elements. All processing elements have the full access (write and read) to all bits of both $\mathrm{nodeValue0/1}$ and $\mathrm{NodeS0/1}$ so that each processing element can modify any bit of these registers at any time. As such, the design must guarantee there are no more than one processing elements writing the same bit of these registers in the same clock cycle. The Lee's algorithm ensures that when there is no confliction in permutation, each element of $\mathrm{nodeValue0/1}$ and $\mathrm{nodeS0/1}$ will only be updated by one processing element in each step. The instinct exclusive property can guarantee that, for each bit of $\mathrm{nodeValue0/1}$ and $\mathrm{NodeS0/1}$, in each clock cycle, there will be only one processing element modifying it and no conflict would happen.

The second task of the control logic is to calculate the state values for the input/output switching nodes. The state values of output switching nodes can be obtained from $\mathrm{NodeS0/1}$. The state values for the input switching nodes are based on the symmetric routing constraint.

The last task of the control logic is calculating the input/output permutation for the subnetworks. The Lee's algorithm calculates the switch setting values recursively from the outmost stages to the most inner stages. Take 16×16 Benes network as an example, according to the state values of the input/output switching nodes, the control logic will derive the permutation for two inner 8×8 Benes networks. This part will be discussed in details in the following subsections

4.3.2 Finite State Machine

In this part, the RTL design of Lee's parallel algorithm is presented. Following the process of Lee's parallel routing algorithm, we derive the finite state machine of each processing element as shown in Figure 24 which encloses five steps.

1. IDLE
2. INIT
3. SEARCH

4. MERGE

5. DONE

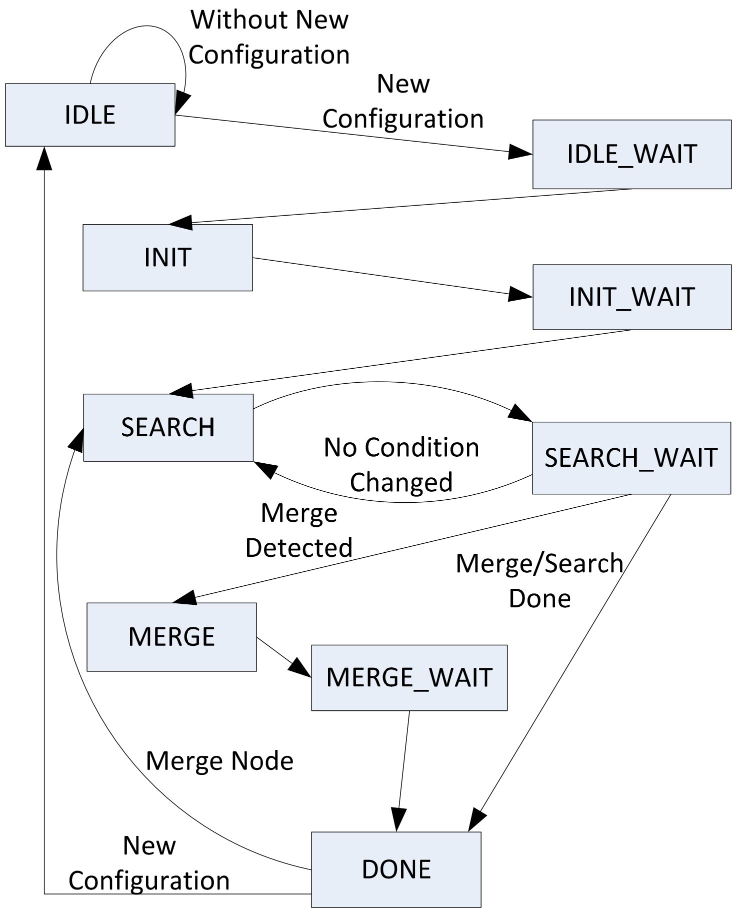


Figure 24 State Diagram

Each step could be divided into several states to complete the function that this step is supposed to do. Those states named with 'WAIT' as appendix are used to synchronize processing elements. All the processing elements need to wait one clock cycle so that the register values updated by other processing elements become valid in all processing elements. In the following part of this section, we will describe these five main steps.

IDLE

At the starting point, all processing elements are in the IDLE state to wait for the new permutation between input and output ports. When the new permutation arrives by setting input ports of all input switching nodes, all processing elements will enter the INIT state to conduct initialization functions. Before the processing element enters the INIT state, the control unit needs one clock cycle to synchronize with all other processing elements.

In the IDLE state, all register values are reset to default values, where $nodeValue0/1$ and $preNodeValue0/1$ are set to the current node index and $nodeS0/1$ are all reset to 0.

INIT

In Lee's parallel routing algorithm, the first step is to initialize the pointers and relation values between output switching nodes. This initialization process is determined by the permutation between inputs and outputs of Benes network. Consider the following permutation for a 16×16 Benes network:

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 10 & 14 & 9 & 2 & 8 & 13 & 12 & 15 \end{pmatrix.$$

$$\begin{matrix} 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 1 & \times & 7 & 11 & 5 & 0 & 4 & 6 \end{matrix}\Big)$$

As discussed in Section 3, there are two types of relation between two output switching nodes that have connection, equal or not equal, represented as $'0'$ or $'1'$ respectively. In Lee's parallel routing algorithm, in order to find out the relation between these two output switching nodes, the equations between routing bits of input/output switching nodes need be derived first. In our design, the relation between two output switching nodes can be derived directly from the parity of two output port indexes corresponding to the two input ports of each PE.

Given the connection pair (k, l) for an input port pair $(2i, 2i + 1)$ (i.e., $port0$ and $port1$ in our design), according to Eqns. (8), (9) and (11), we derive the four possibilities of the above equation:

Case 1: k is even and l is even, we have

$b_{\frac{k}{2}} = \overline{b_{\frac{l}{2}}}$;

Case 2: k is even and l is odd, we have

$b_{\frac{k}{2}} = \overline{\overline{b_{\frac{l}{2}}}} \Rightarrow b_{\frac{k}{2}} = b_{\frac{l}{2}}$;

Case 3: k is odd and l is even, we have

$\overline{b_{\frac{k}{2}}} = \overline{b_{\frac{l}{2}}} \Rightarrow b_{\frac{k}{2}} = b_{\frac{l}{2}}$;

Case 4: k is odd and l is odd, we have

$\overline{b_{\frac{k}{2}}} = \overline{\overline{b_{\frac{l}{2}}}} \Rightarrow b_{\frac{k}{2}} = \overline{b_{\frac{l}{2}}}$

As we can see from above options, when k and l have the opposite odd-even property, then their corresponding output switching nodes will have the same state value, or, they have the opposite state value. The relation between two output switching nodes can be set according to odd-even property of k and l by checking $port0[0]$ and $port1[0]$ as shown in Eqn. (12).

$$NodeS0/1 = \sim(port0[0]\ XOR\ port1[0])$$
$$= \begin{cases} 0 & Equal \\ 1 & Not\ Equal \end{cases} \tag{12}$$

At each processing element P_i, the following code is used to set $nodeValue0/1$ and $NodeS0/1$, where $n = log_2^N$.

// pNode is the temporal variable to hold the larger node index

if $(port0[n - 1{:}1] < port1[n - 1{:}1])$ {

```
        $pNode = port1[n-1:1];$
        $\boldsymbol{if}\ (port1[0])\ \{$
          $nodeValue1[pNode] = port0[n-1:0];$
          $nodeS1[pNode] = (port0[0] == port1[0])?$
                                        $1'b1 : 1'b0;$
        $\}$
        $\boldsymbol{else}\ \{$
          $nodeValue0[pNode]] = port0[n-1:0];$
          $nodeS0[pNode] = (port0[0] == port1[0])?$
                                        $1'b1 : 1'b0;$
        $\}$
$\}$
$\boldsymbol{else\ if}\ (port1[n-1:1] < port0[n-1:1)\ \{$
        $pNode = port0[n-1:1];$
        $\boldsymbol{if}\ (port0[0])\ \{$
          $nodeValue1[pNode] = port1[n-1:0];$
          $nodeS1[pNode] = (port0[0] == port1[0])?$
                                        $1'b1 : 1'b0;$
        $\}$
        $\boldsymbol{else}\ \{$
          $nodeValue0[pNode]] = port1[n-1:0];$
          $nodeS0[pNode] = (port0[0] == port1[0])?$
                                        $1'b1 : 1'b0;$
        }
}
else
        $null$;
```

Note that each port register has width of log_2^N bits with the top $(log_2^N - 1)$ bits representing the output switching node number and the least significant bit representing the port number (0 or 1) of the output switching node as well as the parity of the output port index.

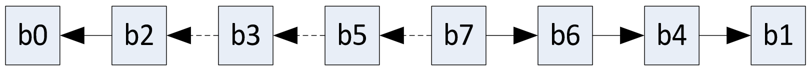


Figure 25 Initialization

After the initialization step, all output switching nodes will be divided into one or more classes depending on the permutation of inputs/outputs as shown in Figure 25. All output switching nodes in the same class are bounded together such that once the state value of any switching node is determined, then the state values of all the other switching nodes will be determined. For the example shown above, if the switch setting value of $b0$ is 0, then the state values of the whole class are shown in Figure 26.

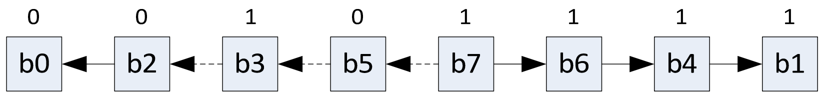


Figure 26 State Value Setting

SEARCH

As discussed in last section, in the searching step, all processing elements parallelly search and update the node pointer till reaching the representative node of the class, i.e., the switching node with the smallest index number in the class. The number of searching steps is bounded by $\frac{N}{2}$. As shown in Figure 24, right after the state machine runs into the SEARCH state, each processing element P_i updates $\mathrm{nodeValue0/1[i]}$ and relation values $\mathrm{nodeS0/1[i]}$ stored locally till the pointer's values do not change in the current searching iteration. To detect the ending condition of searching step, before searching in SEARCH state, the node pointer's current value $\mathrm{nodeValue0/1}$ will be stored in $\mathrm{preNodeValue0/1}$.

Figure 27 shows that after searching all processing elements point to one endpoint except the one representing $b7$, which reaches two endpoints $b1$ ($node[1]$) and $b0$ ($node[0]$). In each class, there is only one representative node. In order to solve this problem, we must merge these two end nodes pointed by the same processing element, as shown in Figure 19, this process will be done in the MERGE state.

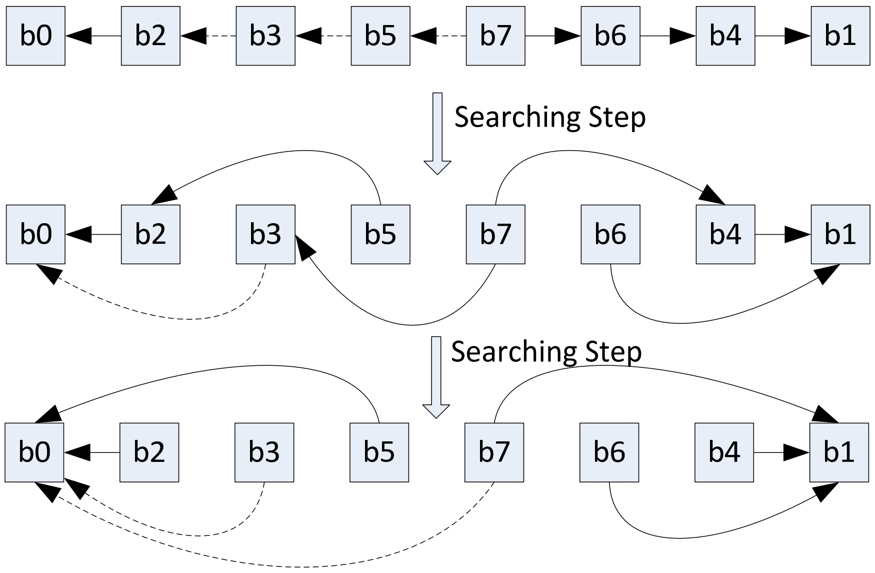


Figure 27 SEARCH

The following two conditions need be satisfied before transferring to the MERGE state.

- After one searching step, the value contained in register preNodeValue doesn't change.
- The switching node has type value "$nodeType == 2'\ b11$", which means the switching node points to two endpoints.

At each processing element P_i, the following code is used to determine if transiting to the MERGE state.

```
$\boldsymbol{if}$ (($preNodeValue0 = nodeValue0$) and
                    ($preNodeValue1 == nodeValue1$))
    $\boldsymbol{if}$ ($nodeType[1:0] == 2'b11$)
      $\boldsymbol{if}$ ($nodeValue0 == nodeValue1$)   // Both registers   // in the current node point to the same endpoint
        $state$ = MERGE_SN ;
      $\boldsymbol{else}$ $state$ = MERGE;
    $\boldsymbol{else}$ $state$ = DONE;
$\boldsymbol{else}$ $state = SEARCH$;
```

If the two pointers of the switching node point to the same endpoint, then FSM transits to MERGE_SN state, in which one of two pointers of the switching node will be reset to its initial value; otherwise, the FSM transits to the MERGE state.

MERGE

When the processing element reaches the endpoints in both direction and the two endpoints are different, the merging step will be conducted. As in the initialization step, the node pointer with larger node index is updated with smaller node index number. As shown in Figure

28, the processing element merges the endpoints of b7 overwriting the nodeValue register storing b1 to b0. We can also see that, after the merging process, the switching nodes previously pointing to node b1 need be updated to pointing to b0. For the example in Figure 28, after the merging step, nodes b6 and b4 need go through searching step again to update their pointers to the representative node b0.

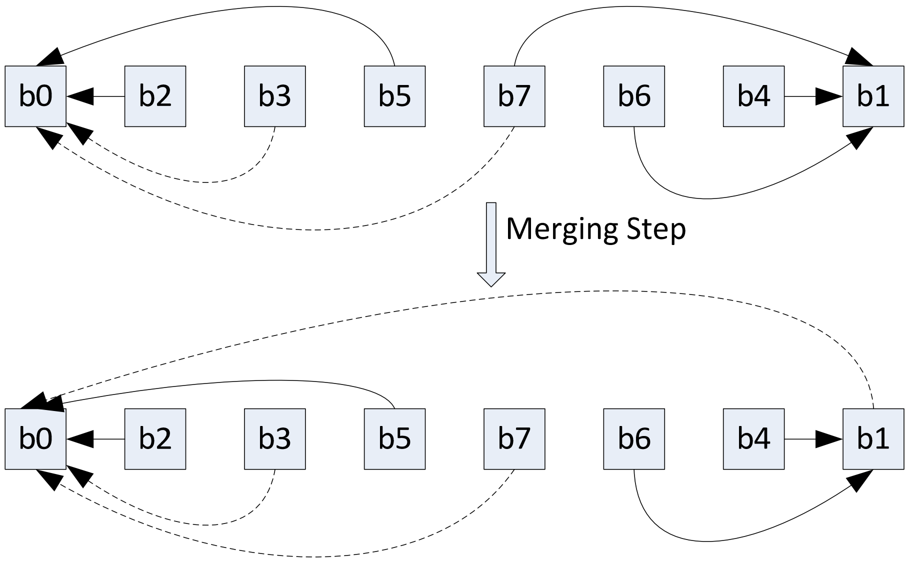


Figure 28 MERGE

For each processing element P_i, the following code is used to update pointers.

```
$\boldsymbol{if}\ (nodeValue0[i] < nodeValue1[i])$
        $\boldsymbol{if}\ (nodeValue1[i][0])$ {
          $nodeValue1[nodeValue1[i]/2] = nodeValue0[i];$
          $nodeS1[nodeValue1[i]/2] =$
                              $nodeS0[i]\ XOR\ nodeS1[i];$
        }
        $\boldsymbol{else}$ {
```

```
        $nodeValue0[nodeValue1[i]/2] = nodeValue0[i];$
        $nodeS0[nodeValue1[i]/2] =$
                        $nodeS0[i]\ XOR\ nodeS1[i];$
    }
$\boldsymbol{else\ if}(nodeValue0[i] > nodeValue1[i])$ {
    $\boldsymbol{if}\ (nodeValue0[i][0])$ {
        $nodeValue1[nodeValue0[i]/2] = nodeValue1[i];$
        $nodeS1[nodeValue0[i]/2] =$
                        $nodeS0[i]\ XOR\ nodeS1[i];$
    }
    $\boldsymbol{else}$ {
        $nodeValue0[nodeValue0[i]/2] = nodeValue1[i];$
        $nodeS0[nodeValue0[i]/2] =$
                        $nodeS0[i]\ XOR\ nodeS1[i];$
    }
}
$\boldsymbol{else}$
    $null;$
```

After the merging step, the processing element will notify the other processing elements so that all the other processing elements will transit to the SEARCH state. As shown in Figure 29, after the searching step, all the switching nodes point to the representative node of this class. The initial state value for the representative node ‘b0’ of this class is ‘0’, then the state value of all other switching nodes can be determined by the relation value $\mathbf{nodeS}$ in parallel. And the switch state values shown in Figure 29 is exactly the same as those values shown in Figure 26.

In our design, after all processing elements are in DONE state, the mainframe will set the state values of output and input switching nodes.

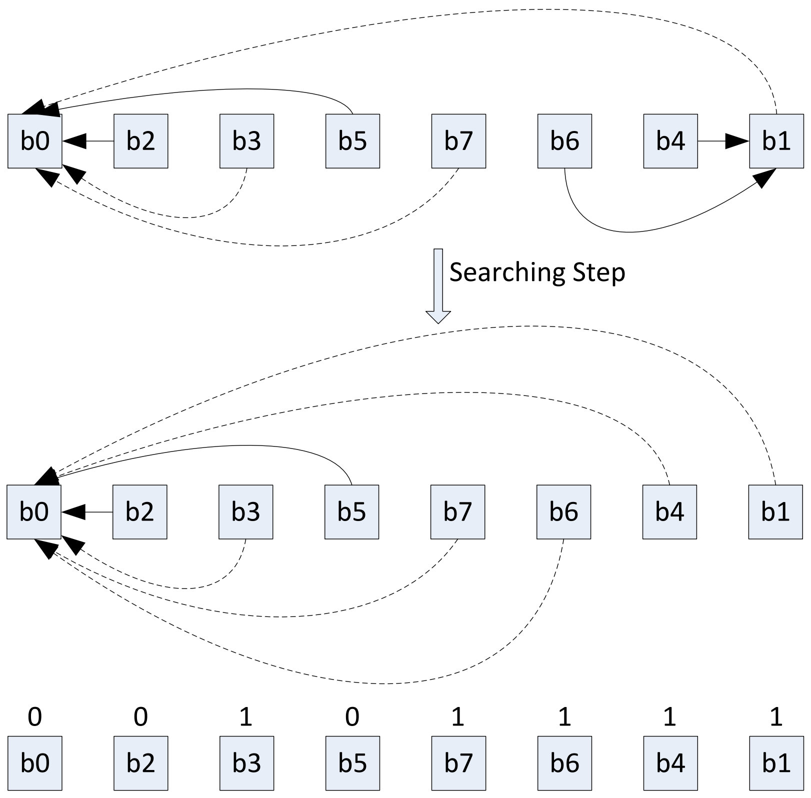


Figure 29 Searching after Merging

4.3.3 Setting State Values of Input/Output Switching Nodes

The state values for output switching nodes outNodeStateValue $\left[\frac{\mathrm{N}}{2}-1:0\right]$ can be obtained directly from the relation value nodeS0 $\left[\frac{\mathrm{N}}{2}-1:0\right]$ or nodeS1 $\left[\frac{\mathrm{N}}{2}-1:0\right]$ as follows.

$$outNodeStateValue[j] = NodeS0[j] \mid NodeS1[j]; \qquad (13)$$

After the state values of output switching nodes are determined, the state values of input switching nodes are determined too. According to the symmetric routing constraint, the state value of an input switching node is equal to or opposite to the state value of its corresponding output switching node which depends on the relation of the input/output port index number.

Given permutation pair (k, l), where k is the input port number, and l is the output port number, due to the symmetric self-routing constraint, i.e., Eqn. (13) and (14) in Section 3.1, we have:

$$\mathrm{inNodeStateValue}\ \left[\frac{\mathrm{k}}{2}\right] = \begin{cases} \ \mathrm{outNodeStateValue}\left[\frac{\mathrm{l}}{2}\right] & \text{if k, l are same parity} \\ \sim\mathrm{outNodeStateValue}\left[\frac{\mathrm{l}}{2}\right] & \text{if k, l are opposite parity} \end{cases} \tag{14}$$

where $\mathrm{k}/2$ and $\mathrm{l}/2$ give the corresponding input/output switching node index. As we can see, either $\mathrm{port}[2\mathrm{i}]$ or $\mathrm{port}[2\mathrm{i}+1]$ can be used to determine the relation between state values of input switching node and its corresponding output switching node. Here we use $\mathrm{port}[2\mathrm{i}]$ to do the calculation. And $\mathrm{port}[2\mathrm{i}][0]$ gives the parity of the output port.

```
// For i=0, 1, …, N/2
$if\ (port[2i][0])$
        $inNodeStateValue[i] = \sim outNodeStateValue\left[\frac{port[2i]}{2}\right];$
$else$
        $inNodeStateValue[i] = outNodeStateValue\left[\frac{port[2i]}{2}\right];$
```

4.3.4 Permutation configuration for sub Benes network

After the state values of input/output switching nodes are set, the permutation for subnetworks will be determined. Given state values in $\mathrm{outNodeStateValue}\left[\frac{\mathrm{N}}{2}-1:0\right]$, the permutation for two subnetworks sub0 and sub1 can be calculated by the control unit as below:

```
$\boldsymbol{for}\ \left(i = 0; i < \frac{N}{2}; i++\right)$ {
        $\boldsymbol{if}\ (inNodeStateValue[i])$ {
            $sub0\_port[i][n-2:0] = port[2i+1][n-1:1];$
            $sub1\_port[i][n-2:0] = port[2i][n-1:1];$
        }
        $\boldsymbol{else}$ {
            $sub0\_port[i][n-2:0] = port[2i][n-1:1];$
            $sub1\_port[i][n-2:0] = port[2i+1][n-1:1];$
        }
}
```

Figure 30 shows the timing diagram for the whole process of the example permutation. In this example, there are three searching steps, two consecutive ones and one after the merging step which is consistent with the Lee's algorithm. Each step needs two clock cycles to finish, because each step needs one more clock cycle to update data in the shared memory. After the state values of input/output switching nodes are determined, one more clock is needed to calculate the permutation for two subnetworks. Totally, 17 clock cycles are used to finish the whole process. Consistent with Lee's algorithm, during the whole process, only the number of searching steps is relevant to the radix of Benes network. And all other steps are in constant.

The pseudocode of the implemented parallel switch setting algorithm for $\mathrm{N} \times \mathrm{N}$ Benes network ($\mathrm{N} > 4$) is listed in Appendix A.

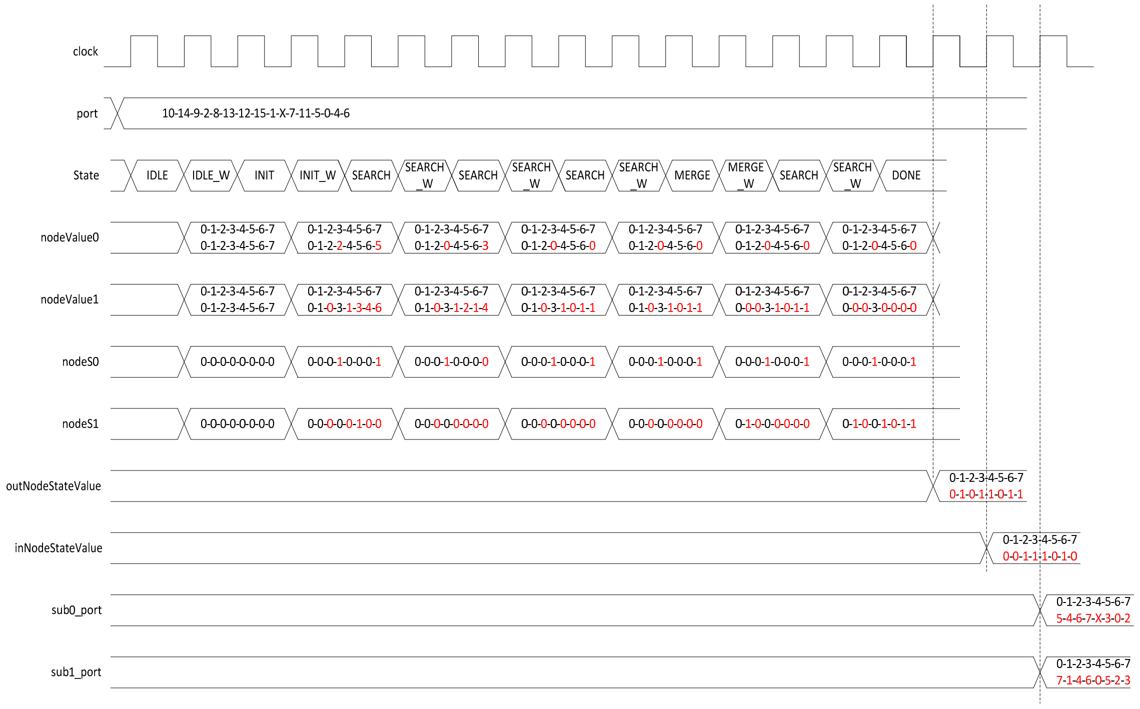


Figure 30 Timing Diagram

4.3.5 Special case

Because of the simplicity of 4×4 Benes network, there is no need to run the whole process. As shown in Figure 31, there are $P_4^4 = 24$ permutations between input and output ports which fall into two cases: 1) either these two output switching nodes are in the same class, 2) they are in the two separated classes. For both cases, there is no need to do the searching and merging procedure thus significantly reducing the logic complexity of each processing element.

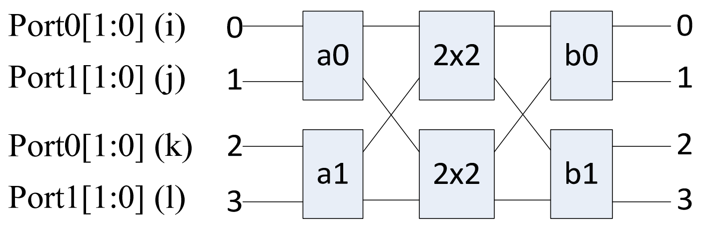


Figure 31 4x4 Benes Network

Consider the permutation of the 4x4 Benes network shown in Figure 17, we can derive below:

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ i & j & k & l \end{pmatrix} => \begin{pmatrix} a_0 & \overline{a_0} & a_1 & \overline{a_1} \\ \beta(i) & \beta(j) & \beta(k) & \beta(l) \end{pmatrix}$$

Consider the connection pair $(i\,,j)$, there are two cases:

Case 1: both i and j belong to the same output switching node, then k and l must belong to the other switching node. There is no connection between two output switching nodes, i.e., they belong to two separated classes. The state value of each output switching node can be assigned independently.

Case 2: i and j belong to two output switching nodes, respectively. Then there exists a connection between two output switching nodes, i.e., they belong to the same class. The state value will be assigned correlately.

Each processing element has two registers $\mathrm{port0}[1:0]$ and $\mathrm{port1}[1:0]$ holding the two output port index number, where $\mathrm{port0}[1]$ and $\mathrm{port1}[1]$ can be used to determine which output switching node the port belongs to and $\mathrm{port0}[0]$ and $\mathrm{port1}[0]$ can be used to determine the relation value between these two output switching nodes if there is connection between them. The following logic code is designed.

```
$\boldsymbol{if}\ (port0[1]\ XOR\ port1[1])\ then$
    $\boldsymbol{if}\ (port0\ XOR\ port1[0])\ then$
```

```
                $nodeS0[1] = 1'b0;$
            $\boldsymbol{else}$ $then$
                $nodeS0[1] = 1'b1;$
            $\boldsymbol{end}$
$\boldsymbol{end}$
```

4.4 Experiment Results

We have implemented the Lee's algorithm for finding the switch settings for input/output stages of 4x4 to 64x64 Benes networks in Verilog, simulated and synthesized the designs using Cadence tools. The RTL code is written in parameterized way so that it is easy to expand to larger sizes. In the simulation process, ModelSim is adopted as the simulation tool. For each design, five categories of permutations are used for validation including bit reversal, perfect shuffle, butterfly, matrix transpose, and random permutations. Under each category, one or more different permutations have been tested. The worst case permutation would cause all output switch nodes in the same group and connected in order with all pointers in same directions. Under the worst case, the algorithm needs the most steps to search representative node which has complexity $O(\log N)$, there is only one worst case in each size network. In the synthesis process, Cadence Encounter RTL-Compiler is used with TSMC 65nm technology library. All size designs are synthesized under the same settings. The synthesized results of delay, area in number of cells, and power consumption are presented below.

Table 9 Delay result

Benes Size	**4x4**	**8x8**	**16x16**	**32x32**	**64x64**
Delay (ns)	0.1	0.8	2.3	3.7	5.6
Time Complexity $O(\log^2 N)$	4	9	16	25	36

The delay is mainly decided by the time complexity of the algorithm. While the size of the processing element will not affect the delay as much as that does to area and power consumption as shown in Table 9 and Figure 32.

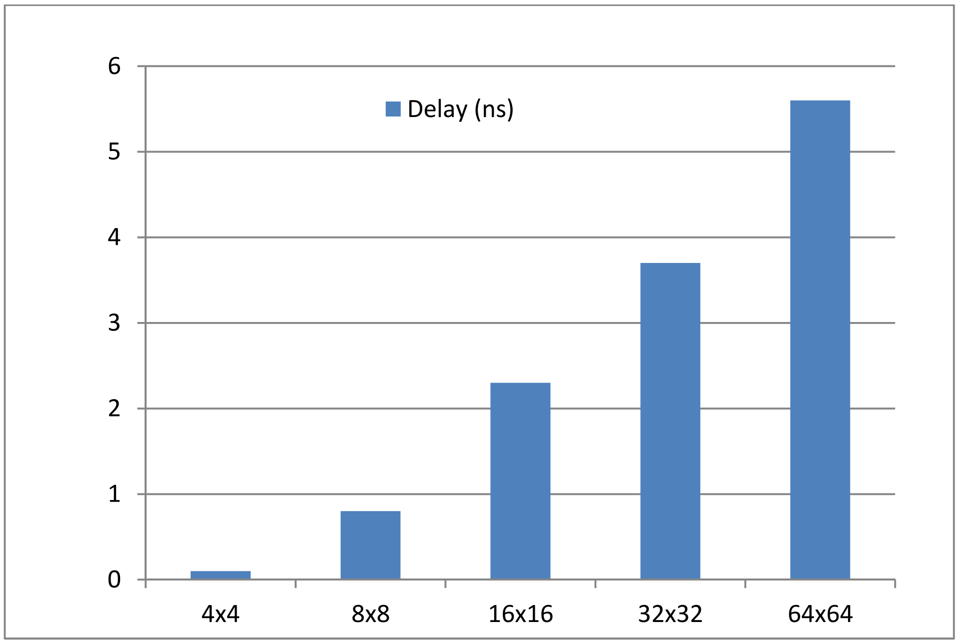


Figure 32 Delay Result

As discussed in last section 4.2, the time complexity of algorithm is determined by the number of searching steps. The simulation results show that the number of searching steps follows $O(\log N)$. Except the 4x4 network, the synthesized delay result has about the same trend as that of the time complexity of Lee's algorithm. For 4x4 Benes, there is no searching step in the switch setting algorithm. That's why the delay is much lower than that of 8x8 Benes.

Table 10 Cell number and Area

Benes Size	**4x4**	**8x8**	**16x16**	**32x32**	**64x64**
Number of Cells	1.0E+01	1.81E+03	8.11E+03	3.62E+04	1.32E+05

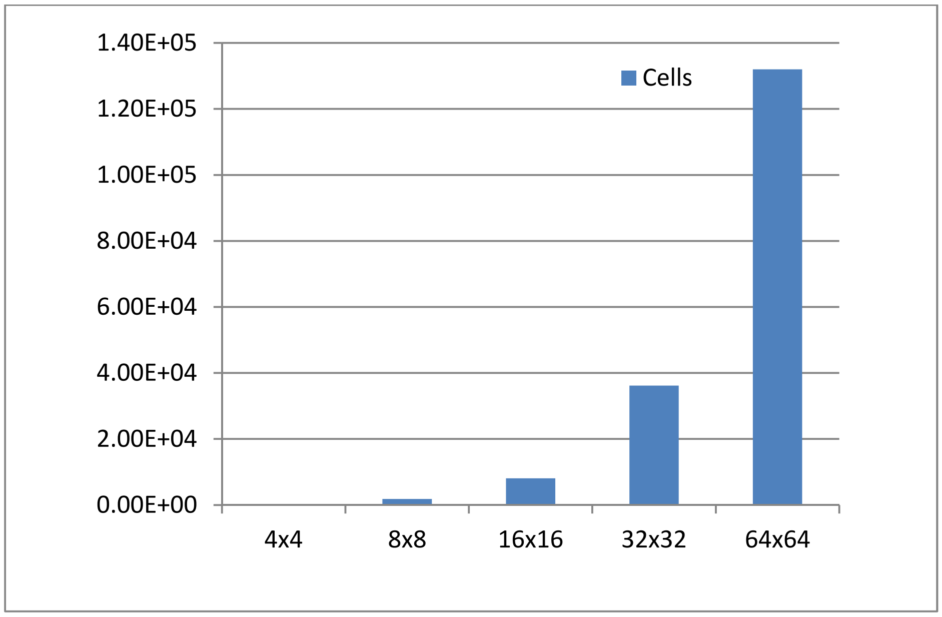


Figure 33 Number of Cells

Table 10 and Figure 33 show the area result in terms of number of cells, the basic design unit used to measure the logic complexity. When the network size is doubled, the number of cells increases by about 4 times except for the 4x4 network. It is clear that in Lee’s algorithm, when the network size is doubled, the number of processing elements needed in each stage is doubled. For example, the 8×8 Benes has 4 processing elements and the 16×16 Benes network has 8 processing elements. Besides, the logic complexity of the processing element nearly doubles when the network size is doubled. Overall, the logic complexity of the processing element should be increased by four times when the network size is doubled. This explains the trend of number of cells in Table 10.

Table 11 Power Consumption (uW)

Size / Power Type	4x4	8x8	16x16	32x32	64x64
Leakage	0.75	92.4	386	1,760	7,070
Internal	1.23	84.6	385	1,690	7,280
Net	0.18	29.8	142	604	2,280
Switching	1.41	114	528	2,290	9,560

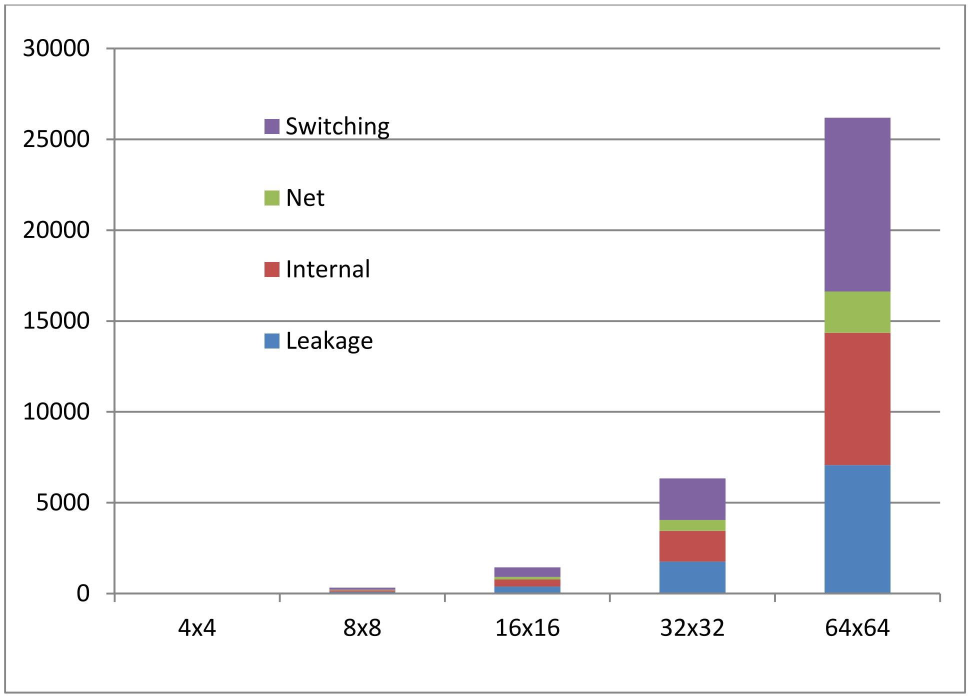


Figure 34 Power Consumption

Table 11 shows the power consumption of the design in terms of static (internal) power, dynamic (mainly switching), net and leakage power. Each portion of power increases significantly as the radix of Benes network increases. The power consumption increasing trend is consistent with the increasing trend of number of cells. As shown in Figure 34, the switching power is the most significant portion, followed by internal (Static) and leakage power which occupies 36%, 28% and 27% of total power, respectively. Together the three portions of power dominate the power consumption at more than 90%.

4.5 Summary

This chapter presents the RTL design of a parallel switch setting algorithm in Benes Networks. We have refined the algorithm in data structure and initialization/updating of relation values to make it suitable for hardware implementation. The RTL code is written in parameterized way so that it is easy to expand to larger sizes. The RTL design of the switch setting circuit for 4x4 to 64x64 Benes networks are simulated and synthesized using Cadence tools. The simulation and synthesis results confirm that the trend of delay and area results of the circuit is consistent with that of the Lee’s algorithm.

CHAPTER 5 CONCLUSION AND FUTRURE WORK

In Chapter 3, the result confirms that Clos network is a better alternative than Benes networks to replace crossbars in large scale networks-on-chip systems. Though the Clos network is the extension of Benes network, the Clos network provides way more flexibility than the Benes network. As shown in the definition of Clos network, and demonstrated by our CMOS circuit design, the larger size of switching logic unit can reduce the number of stages to traverse. Furthermore, by using transmission gates in our circuit design, both the delay and power consumption results are improved significantly.

The parallel switch setting algorithm is the key to satisfy the requirements of high performance switching networks. We implemented the fastest parallel switch setting algorithm, Lee’s parallel routing algorithm, in hardware. The RTL design of Lee’s algorithm has been fully implemented by Verilog. During the RTL design, we have refined Lee’s routing algorithm in a few ways to make it suitable for hardware implementation. The RTL design of the switch setting circuit for 4x4 to 64x64 Benes networks are simulated and synthesized using Cadence tools. The simulation and synthesis results of the switching setting circuits for 4x4 to 64x64 Benes networks confirm that the trend of delay and area results of the circuit is consistent with that of the Lee’s algorithm.

The future work includes: 1) integration of the switch setting circuit with the Benes network circuit; 2) evaluation of the network performance of Benes/Clos-based NoCs under both synthetic and real life benchmarks.

APPENDIX A

The pseudocode of the implemented parallel switch setting algorithm for N × N Benes network (N > 4) is listed below. All variables are defined in Table 8.

Parallel Switch Setting Algorithm

Inputs: port[N] - Permutation for input ports ➔ output ports

Outputs: inNodeStateValue[N/2] - Switch setting values for input switching nodes

outNodeStateValue[N/2] - Switch setting values for output switching nodes

sub0/1_port[N/2]: Permutations for subnetworks

Function: Calculate the switch settings for input and output switching nodes

```
$\boldsymbol{Main}$ () {
  $//initialize\ global\ variables$
  $\boldsymbol{for}\ (i = 0;\ i\ <\ N/2;\ i = i + 1)$ {
    $nodeValue0[i]\ =\ i;$
    $nodeValue1[i]\ =\ i;$
    $nodeS0/1[i]\ \ =\ 0;$
  }
  $Call\ initialization\ procedure;$
  $Call\ searching\ procedure;$
  $Call\ merging\ procedure;$
          $//calculate\ input, output\ switching\ node\ state\ values$
$outNodeStateValue[i]\ \ =\ nodeS0[i]\ OR\ nodeS1[i];$
  $\boldsymbol{for}\ (i = 0;\ i\ <\ N/2;\ i = i + 1)$ {
    $\boldsymbol{if}\ (port[2i]\ is\ even)$ {
        $inNodeStateValue[i] = outNodeStateValue[port[2i]/2]$ ;
      }
      $\boldsymbol{else}$ {
        $inNodeStateValue[i] = \sim outNodeStateValue[port[2i]/2]$;
      }
    }
    $// Calculating permutation for subnetworks$
    $\boldsymbol{for}$ $(i = 0;\ i < N/2;\ i = i + 1)$ {
      $\boldsymbol{if}$ $(inNodeStateValue[i])$ {
        $sub0\_port[i] = port[2i + 1]/2$;
        $sub1\_port[i] = port[2i]/2$;
      }
      $\boldsymbol{else}$ {
        $sub0\_port[i] = port[2i]/2$;
        $sub1\_port[i] = port[2i + 1]/2$;
      }
    }
} $// End of main$


$// Initialization procedure$
$\boldsymbol{for}$ each processing element $P_i$ in parallel do{
    $port0 = port[2 * i]$;
```

```
  $port1 = port[2 * i + 1];$
   $// find\ the\ node\ with\ larger\ index$
  **if** $(port0[n-1:1] > port1[n-1:1])$ {
        $//set\ the\ port\ 0\ pointer$
        $nodeValue0[port0[n-1:1]] = port1[n-1:1];$
        $//set\ the\ relation\ value\ of\ port\ 0\ pointer$
        $nodeS0[port0[n-1:1]] = \sim(port0[0]\ XOR\ port1[0]);$
  }
  **else if** $(port1[n-1:1] > port0[n-1:1])$ {
        $nodeValue1[port1[n-1:1]] = port0[n-1:1];$
        $nodeS1[port1[n-1:1]] = \sim(port0[0]\ XOR\ port1[0]);$
  }
  **else**
        $null;$
} $// End\ of\ Initialization$

$// Searching\ procedure$
**for** each processing element $P_i$ in parallel do{
$// if\ the\ nodeValue0/1\ keeps\ the\ same\ value\ after$
 $//\ searching, then\ stop$
  $preNodeValue0 = nodeValue0[i]/2;$
  $preNodeValue1 = nodeValue1[i]/2;$
```

```
    $\boldsymbol{if}(nodeValue0[nodeValue0[i]/2] < nodeValue1[nodeValue0[i]/2])$ {
        $nodeValue0[i] = nodeValue0[nodeValue0[i]/2];$
        $nodeS0[i] = nodeS0[i]\ XOR\ nodeS0[nodeValue0[i]/2];$
    }
    $\boldsymbol{else}$ {
        $nodeValue0[i] = nodeValue1[nodeValue0[i]/2];$
        $nodeS0[i] = nodeS0[i]\ XOR\ nodeS1[nodeValue0[i]/2];$
    }
    $\boldsymbol{if}(nodeValue0[nodeValue1[i]/2] < nodeValue1[nodeValue1[i]/2])$ {
        $nodeValue1[i] = nodeValue0[nodeValue1[i]/2];$
        $nodeS1[i] = nodeS1[i]\ XOR\ nodeS0[nodeValue1[i]/2];$
    }
    $\boldsymbol{else}$ {
        $nodeValue1[i] = nodeValue1[nodeValue1[i]/2];$
        $nodeS1[i] = nodeS1[i]\ XOR\ nodeS1[nodeValue1[i]/2];$
    }
}$\boldsymbol{while}((preNodeValue0 \neq nodeValue0[i])$
                          $and\ (preNodeValue1 \neq nodeValue1[i]));$
$//\ Merging\ procedure$
$\boldsymbol{for}$ each processing element $P_i$ in parallel do{
        $\boldsymbol{if}\ (nodeValue0[i] < nodeValue1[i])$
                $if\ (nodeValue1[i][0])$ {
```

```
                $nodeValue1[nodeValue1[i]/2] = nodeValue0[i]$
            $nodeS1[nodeValue1[i]/2] = nodeS0[i]\ XOR\ nodeS1[i];$
            }
            $\boldsymbol{else}$ {
                $nodeValue0[nodeValue1[i]/2] = nodeValue0[i];$
                $nodeS0[nodeValue1[i]/2] = nodeS0[i]\ XOR\ nodeS1[i];$
            }
        }
        $\boldsymbol{else\ if}(nodeValue0[i] > nodeValue1[i])$ {
            $if\ (nodeValue0[i][0])$ {
                $nodeValue1[nodeValue0[i]/2] = nodeValue1[i];$
                $nodeS1[nodeValue0[i]/2] = nodeS0[i]\ XOR\ nodeS1[i];$
            }
            $\boldsymbol{else}$ {
                $nodeValue0[nodeValue0[i]/2] = nodeValue1[i];$
                $nodeS0[nodeValue0[i]/2] = nodeS0[i]\ XOR\ nodeS1[i];$
            }
        }
        $\boldsymbol{else}$
            $null;$
    }
```

REFERENCE

[1] P. Kundu, "On-die interconnects for next generation CMPs," in *Proc. Workshop On- and Off-Chip Interconnection Networks for Multicore Systems (OCIN)*, 2006.

[2] C. O. Chen, S. Park, T. Krishna, L. Peh, "A low-swing crossbar and link generator for low-power network-on-chi," in *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD)*, 2011, pp. 779-786.

[3] R. Das, S. Eachempati, A. K. Mishra, V. Narayanan, and C. R. Das, "Design and evaluation of a hierachical on-chip interconnect for next-generation CMPs," in *Proc. IEEE 15th Symp. HPCA*, 2009, pp. 175-186.

[4] X. Wang, M. Yang, Y. Jiang, and P. Liu, "A power-aware mapping approach to map IP cores onto NoCs under bandwidth and latency constraints," *ACM Trans. Architecture and Code Optimization*, vol. 7, no. 1, Apr. 2010.

[5] D.Wentzlaff and P. Griffin, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15-31, 2007.

[6] C. Kim, D. Burger, S. W. Keckler, "Nonuniform cache architectures for wire delay dominated on-chip caches," *IEEE Micro*, vol. 23, no. 6, pp. 99-107, Nov.-Dec. 2003.

[7] T. Johnson and U. Nawathe, "An 8-core, 64-thread, 64-bit power efficient SPARC SOC (niagara2)," in *Proc. IEEE Int'l Solid-State Circuits Conf.*, 2007, pp. 108–590.

[8] L. Mhamdi, K. Goossens, I.V. Senin., "Buffered crossbar fabrics based on networks on chip," in *Proc. Communication Networks and Services Research Conf. (CNSR),* 2010, pp.74-79.

[9] G. Passas, M. Katevenis, D. Pnevmatikatos, "A 128 x 128 x 24Gb/s crossbar interconnecting 128 tiles in a single hop and occupying 6% of their area," in *Proc.* 4th *ACM/IEEE Int'l Symp., Networks-on-Chip (NOCS),* 2010, pp. 87-95.

[10] J. Kim, J. Dally, B. Towles, A. K. Gupta, "Microarchitecture of a high-radix router," in *Proc. 32nd Int't Sym. Computer Architecture,* 2005, pp. 420-431

[11] J. Duato, I. Johnson, J. Flich, F. Naven, P. Garcia, and T. Nachiondo, "A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks," in *Proc. Int'l Symp. HPCA,* 2005, pp. 108-119

[12] G. Mora, J. Flich, J. Duato, P. L´opez, E. Baydal, and O. Lysne, "Towards an efficient switch architecture for high radix switches," in *Proc. ACM/IEEE ANCS,* 2006, pp. 11-20

[13] IBM Blue Gene team, "The IBM blue gene/q computer chip," *IBM Journal of Research and Development,* vol. 57, no. 1/2, pp. 1:1-1:13, Jan.-March 2013.

[14] K. Sewell, R.G. Dreslinski, T. Manville, S. Satpathy, N. Pinckney, G. Blake, M. Cieslak., "Swizzle-switch networks for many-core systems," *IEEE J. Emerging and Selected Topics in Circuits and Systems,* vol. 2, no. 2, pp. 278-294, Jun. 2012,

[15] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-

delay dominated on-chip caches," in *Proc. 10th Int'l Conf. Architectural Support Programm. Languages Operat. Syst.,* 2002, pp. 211-222.

[16] A. Zia, S. Kannan, G. Rose and H. J. Chao, "Highly-scalable 3D Clos NoC for many-core CMPs," in *Proc. IEEE 8th Int'l NEWCAS Conf. (NEWCAS),* 2010, pp. 229-232.

[17] E. Coen-Alfaro, "Crossbar architectures for VLSI systems: a comparative study," *Master Thesis, University of Idaho,* Apr. 2004.

[18] Y. Zhang, J. Taikyeong, F. Chen, H. Wu, and N. R. Gao, "A study of the on-chip interconnection network for the IBM Cyclops64 multi-core architecture," in *Proc. Parallel and Distributed Processing Symp. (IPDPS),* 2006, pp. 25-29.

[19] Y. Kao, M. Yang, N.S. Artan, and H. J. Chao, "CNoC: High-radix Clos network-on-chip," in *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 30, no. 1, pp. 1897-1910, Nov. 2011.

[20] D. L. Lewis, S. Yalamanchili, and H. S. lee, "High performance non-blocking switch design in 3D die-stacking technology," in *Proc. IEEE Computer Society Annual Symp. VLSI,* 2009, pp. 25-30

[21] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal,* vol. 32, no. 2, pp. 406-424, Mar. 1952.

[22] G. Passas, M. Katevenis, D. Pnevmatikatos, "VLSI micro-architectures for high-radix crossbar schedulers," in *Proc. 5th IEEE/ACM Int. Symp. Networks Chip,* 2011, pp. 217-224.

[23] C. Kim, D. Burger, S. Keckler, "An Adaptive, Non-uniform cache structure for wire-delay dominated on-chip caches," in *Proc. 10th Int'l Conf. ASPLOS*, 2002, pp. 211-222.

[24] S. Satpathy, Z. Foo, B. Giridhar, R. Dreslinski, D. Sylvester, T. Mudge, D. Blaauw, "A 1.07Tbit/s 128×128 swizzle network for SIMD processors," in *Proc. IEEE Symp. VLSI Circuits,* 2010, pp. 16-18.

[25] S. Satpathy, K. Sewell, T. Manville, Y. Chen, R. Dreslinski, D. Sylvester, T. Mudge, D. Blaauw, "A 4.5Tb/s 3.4Tb/s/W 64×64 switch fabric with self-updating least-recently-granted priority and quality-of-service arbitration in 45nm CMOS," in *Proc. IEEE ISSCC,* 2012, pp. 478-480.

[26] H. Moussa, O. Muller, A. Baghdadi, M. Jezequel, "Butterfly and Benes-based on-chip communication networks for multiprocessor turbo decoding," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE),* 2007, pp. 1-6.

[27] A. Abbas, M. Ali, A. Fayyaz, A. Ghosh, A. Kalra, S. U. Khan, "A survey on envergy-efficient methodologies and architectures of network-on-chip," in *Computers & Electrical Engineering,* 2014, pp. 333-347.

[28] H. Liu, L. Xie, J. Liu and L. Ding, "Application of butterfly Clos-network in network-on-chip," *Scientific World Journal,* vol. 2014, pp. 1-11, Jan. 2014

[29] H. Richer, "Real-time interconnection network for single-chip many-core computers," in *Proc. Design & Test Symposium,* 2013, pp. 1-4.

[30] Y. Kao and H. J. Chao, "Design of a bufferless photonic Clos network-on-chip architecture," *IEEE Transactions on Computers,* vol. 63, no. 3, pp. 764-776, Mar. 2014.

[31] K. N. Levitt, M. W. Green, and J. Goldberg, "A study of the data commutation problems in a self-repairable mutiprocessor," *in Proc. Spring Joint Computer Conf.,* 1968, pp. 515-527

[32] Y. Kao, M. Yang, N. S. Artan, and H. J. Chao, "CNoC: high-radix Clos network-on-chip," *IEEE Trans. Computer-Aided Design of Intergrated Circuits and Systems*, vol. 30, no. 12, pp.1897-1910, Dec. 2011.

[33] H. Liu, L. Xie, J. Liu and L. Ding, "Application of butterfly Clos-network in network-on-chip," *The Scientific World Journal,* vol. 2014, pp.1-11, Jan. 2014.

[34] A. Joshi, C. Batten, Y. Kwon, S. Beamer, I. Shamim, K. Asanovic and V. Stojanovic, "Silicon-photonic Clos networks for global on-chip communication," in *Proc. 3rd ACM/IEEE Int'l Symp. Networks-on-Chip (NoCS),* 2009, pp.124-133.

[35] J. H. Bahn, S. E. Lee and N. Bagherzadeh, "Design of a router for network-on-chip," *Int'l J. High Performance Systems Architecture,* vol. 1, no.2, pp.98-105, Oct. 2007.

[36] P. Liu, Y. Liu, B. Xia, C. Xiang, and X. Wang, "A networks-on-chip emulation/verification framework," *Int'l J. High Performance systems Architecture,* vol. 3, no. 1, pp.2-11, 2012.

[37] K. Sewell, R. G. Dreslinski, S. Satpathy, G. Blake and M. Cieslak, "Swizzle-switch networks for many-core systems," *IEEE J. Emerging and Selected Topics in Circuits and Systems*, vol. 2, no. 2, pp. 278-294, Jun. 2012.

[38] T. T. Lee and S. Y. Liew, "Parallel routing algorithms in Benes-Clos networks," *in Proc. 15th Annual Joint Conf. IEEE Computer Societies. Networking the Next Generation (INFOCOM)*, 1996, vol.1, pp. 279-286.

[39] Y. Hamada, K. Kai, Y. Miao and H. Obara, "Design of partially-asynchronous parallel processing elements for setting up Benes networks in O(log2N) time," in *Proc. Int'l Conf. Photonics in Switching*, 2009, pp.1-2.

[40] Y. Yeh and T. Feng, "On a class of rearrangeable networks," *IEEE Trans. Computers,* vol. 41, no. 11, pp.1361-1397, Nov. 1992.

[41] A. Waksman, "A permutation network," *J. Ass. Comput. Mach.*, vol. 15, no. 1, pp.159-163, Jan. 1968.

[42] D. Nassimi and S. Sahni, "Parallel algorithms to set up the Benes permuation network," *IEEE Trans. Computer,* vol. c-31, no. 2, pp.148-154, Feb. 1982.

[43] K. N. Levitt, M. W. Green and J. Goldberg, "A study of the data commutation problems in a self-repairable multiprocessor," in *Proc. Spring Joint Computer Conf.*, 1968, pp. 515-527, 1968.

[44] T. T. Lee and S. Y. Liew, "Parallel routing algorithms in Benes-Clos networks," *IEEE Trans. Communications,* vol. 50, no. 11, pp.1841-1847, Nov. 2002.

[45] H. Richter, "Real-time interconnection network for single-chip many-core computers," *in Proc. Design & Test Symp.*, 2013, pp. 27-30.

[46] H. Moussa, O. Muller, A. Baghdadi and M. Jezequel "Butterfly and Benes-based on-chip communication networks for multiprocessor turbo decoding," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2007, pp.1-6.

[47] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.,* vol. C-20, no. 2, pp.153-161, Feb. 1971.

[48] C. Y. Lee and A. Y. Qruc "Fast parallel algorithms for routing one-to-one assignments in Benes networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 3, pp.329-334, Mar. 1995.

[49] E. Lu and S. Q. Zheng, "Parallel routing algorithms for non-blocking electronic and photonic switching networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 8, pp. 702-713, Aug. 2005.

[50] A. B. Rathod, S. M. Gulhane and A. R. M. Khan, "Parallel routing algorithms in Benes and Clos networks: a survey," *Int'l J. Advance Foundation and Research in Computer,* vol. 2, no. 1, pp. 21-31, Jan. 2015.

CURRICULUM VITAE

Yikun Jiang, Ph.D.

4500 S. Maryland Pkw.

Department of Electrical and Computer Engineering

University of Nevada, Las Vegas

Las Vegas, NV, 89119

Cell Phone: (702)748-4515

E-mail address: jiangy3@unlv.nevada.edu

Education

Qingdao University, Qingdao, China

Bachelor, Computer Science, 2005

Harbin Institute of Technology, Harbin, China

Master, Computer Science, 2007