Article

Novel Benes Network Routing Algorithm and Hardware Implementation

Dimitris Nikolaidis [1,*], Panos Groumas [2], Christos Kouloumentas [2] and Hercules Avramopoulos [1]

[1] School of Electrical and Computer Engineering, National Technical University of Athens, Iroon Polytechniou 9, 15773 Athens, Greece; hav@mail.ntua.gr
[2] Optagon Photonics, Eleftheriou Venizelou 47, 15351 Pallini, Greece; panos.groumas@optagon-photonics.eu (P.G.); christos.kouloumentas@optagon-photonics.eu (C.K.)
* Correspondence: dimnikolaidis@mail.ntua.gr

Abstract: Benes/Clos networks constitute a particularly important part of interconnection networks and have been used in numerous areas, such as multi-processor systems, data centers and on-chip networks. They have also attracted great interest in the field of optical communications due to the increasing popularity of optical switches based on these architectures. There are numerous algorithms aimed at routing these types of networks, with varying degrees of utility. Linear algorithms, such as Sun Tsu and Opferman, were historically the first attempt to standardize the routing procedure of this types of networks. They require matrix-based calculations, which are very demanding in terms of resources and in some cases involve backtracking, which impairs their efficiency. Parallel solutions, such as Lee's algorithm, were introduced later and provide a different answer that satisfy the requirements of high-performance networks. They are, however, extremely complex and demand even more resources. In both cases, hardware implementations reflect their algorithmic characteristics. In this paper, we attempt to design an algorithm that is simple enough to be implemented on a small field programmable gate array board while simultaneously efficient enough to be used in practical scenarios. The design itself is of a generic nature; therefore, its behavior across different sizes (8×8, 16×16, 32×32, 64×64) is examined. The platform of implementation is a medium range FPGA specifically selected to represent the average hardware prototyping device. In the end, an overview of the algorithm's imprint on the device is presented alongside other approaches, which include both hard and soft computing techniques.

Keywords: Benes network; routing algorithm; hardware implementation; FPGA; optical switching; data center

Citation: Nikolaidis, D.; Groumas, P.; Kouloumentas, C.; Avramopoulos, H. Novel Benes Network Routing Algorithm and Hardware Implementation. *Technologies* **2022**, *10*, 16. https://doi.org/10.3390/technologies10010016

Academic Editor: Joshua M. Pearce

Received: 23 December 2021
Accepted: 18 January 2022
Published: 25 January 2022

1. Introduction

Communication between different integrated systems has been one of the cornerstones of technological advancement in the last decades. As time passes, more devices and autonomous circuits require connection to some kind of network in order to function properly. In the contemporary digital world, a sophisticated and complex structure ranging from data center (DC) servers to multicore processors are in need of fast and reliably functioning interconnection networks in order to attain maximum efficiency.

One such type of interconnection network that emerged from the Clos family [1] with 2×2 switching elements and N/2 switches per stage is the Benes network [2]. Benes networks are a well-known type of non-blocking multistage interconnection networks [3,4]. They are rearrangeable and non-blocking [5], which practically means that all possible routing schemes ($n!$) can be satisfied if the entire network is reconfigured from the beginning every time there is a change in routing. For this reason, it is also a permutation network [2]. Benes networks have 2^r inputs and $2^r - 1$ stages of 2×2 switches.

In the DC network application domain, optical switches based on the Benes architecture are attracting a lot of research interest due to their non-blocking nature, which is highly desirable for optical interconnects within DC. Optical switches are an attractive alternative to traditional electrical packet switching (EPS), as they hold the promise to obviate the need for optical-to-electrical and electrical-to-optical conversion, which requires an ever-increasing number of serdes lanes on the switch silicon. For example, electrical packet switches with 25.6 Tb/s capacity use 256 serdes lanes at 50 Gbaud pulse amplitude modulation 4-level (PAM-4) with 100 Gb/s per lane capacity, and the next step to 51.2 Tb/s would require to either double the serdes lanes to 512 at 100 Gb/s per lane or double the per lane capacity to 200 Gb/s while keeping the serdes lanes at 256 [6]. Both approaches face scalability problems related to power consumption, size, and cost. Increasing the number of serdes lanes translates to increased power dissipation and a larger application-specific integrated circuit (ASIC) footprint to accommodate the extra pins. Increasing the speed for the serdes lanes will also increase power dissipation on the ASIC and on the pluggable transceiver modules located at the switch's front panel. What is more, further shrinking of the ASIC's transistors that naturally happens approximately every 2 years in order to evolve to the next generation, apart from being extremely costly, will soon reach its physical limits, as moving to <5 nm complementary metal–oxide semiconductor (CMOS) node technology will be very challenging.

On the other hand, optical switches are transparent to the incoming signal data rates and can have similar radix (e.g., 16, 32) with their EPS counterparts. In general, they have been proposed in various implementations, such as, for example, arrayed waveguide grating routers (AWGRs) in combination with microcombs [7], 4 × 4 AWGRs [8] or 8 × 8 cyclic AWGR (CAWGR)s [9], or cascades of (nested) Mach–Zehnder switches (MZS) [10]. The AWGR implementations require that the optical transmitters operate at different wavelengths. This is typically achieved by using arrays of lasers or tunable wavelength transmitters. In [11], a switching concept based on the combination of a high-capacity tunable optical transceiver and several wavelength selective switches (WSSs) were presented; however, these were commercial bulk devices, they were limited in port count and had rather slow (tens of milliseconds) switching speed. WSSs rely on complex assemblies of micro-optic elements necessitating accurate alignments, making their fabrication costly and limiting any integration potential. The Benes architecture is based on simple cross bar switches, is typically realized as Mach–Zehnder structures with thermo-optical or electro-optical actuation, can be fully integrated in photonic integrated circuit (PIC), and it requires the fewest switch elements to realize the end-to-end connections; this in turn translates to lower insertion loss, fewer components to actuate, and less power consumed, eventually resulting in compact switches. Benes optical switches can be realized in various technologies, such as microelectromechanical systems (MEMS) [12] or PICs. More specifically, PIC-based Benes optical switches have been reported in lithium niobate ($LiNbO_3$) [13], silicon [14,15], and indium phosphide (InP) [16]. InP has the advantage of the very fast (ns) electro-optic actuation; however, the achievable switch size is not large, due to yield issues. Silicon photonic switches can also achieve fast (few ns) electro-optic actuation, but when scaling to higher than 8 × 8 sizes, their performance will deteriorate in terms of insertion loss and crosstalk. A 32 × 32 Benes optical switch fabricated on optical polymer is being pursued for the first time in the EU project POETICS [17]. Optical polymer switches can provide polarization independence, low insertion loss and transparency over a wide wavelength range [18]. Actuation is based on the thermo-optic effect which provides millisecond switching speeds, and despite the fact that they are not suited for packet level switching, they are expected to find their place within hybrid optical–electrical DC networks [19], where the EPS switches would handle only the dynamic, rapidly changing traffic ("mice flows") and off-load the slowly varying traffic ("elephant flows") to the optical switch. They can handle multi-gigabit traffic flows of parallel [20] or multiplexed optical lanes without the need for demultiplexing as in the case of EPS switches. In this manuscript, we focus only on the implementation of the algorithm, which is shown for the first time as previous work focused on optical transmission experiments using high-capacity arrayed optical transceivers for use in intra-DC optical interconnects.

A simple and efficient routing algorithm taking up only minimal hardware resources on the control electronics is what is required to unlock the full potential of PIC-based Benes switches and increase their throughput. In the electronic (digital) domain, the throughput increase is achieved by increasing the number of ports while keeping the algorithm's procedure as undemanding and cost effective as possible [21].

The Benes network is a well-established solution to the problem of connecting ports in distinct types of networks. It has been used in shared memory multiprocessor systems [22]. Due to its utility, a great number of algorithmic solutions have been proposed, parallel and sequential. One of the most famous linear algorithms is Opferman's [23] looping algorithm, which starts from the switches in the outer stages and ends in the center stage. It dissects the network into smaller networks and recursively routes all of them, creating a complete path in the process. There is also an expansion of this method by Andresen that enables the routing of base 2^k networks [24]. Waksman [25] proposed a recursive procedure in service of a uni-processor system. Both examples have the complexity of n*log$n$, which is the lowest possible in a 1-proccessing unit system [23].

In the parallel domain, Nassimi and Sahni [26,27] presented a parallel algorithm which is significantly faster than the linear methods, but its complexity varies from $(\log n)^2$ to $(\log n)^4$ depending on the type of system that it is applied on. Lee's parallel algorithm [28] has been implemented in hardware since it offers the lowest complexity $(\log n)^2$ while consuming only n/2 processing units. There are many other algorithms that route Benes networks [21,29,30]; however, essentially, they are all variations of the aforementioned ones. The solution presented in [31] is the closest, in terms of function, to the solution covered in this paper. Even though it is one of the solutions that use two-dimensional (2D) matrixes, in some instances, it produces the same switch setting. It is, however, purely a theoretical approach and does not include any hardware implementation.

There have been some attempts to translate certain algorithms to hardware implementations [32]. However, to our knowledge, all of these attempts focus primarily on parallel routing algorithms for Benes networks. Moreover, all the already established methods, linear and parallel, involve complex calculations. Linear methods generally focus on the use of 2D matrix permutations [29], with some of them requiring backtracking to previously set switches and changing their setting [23], thus heavily impeding their throughput. Parallel implementations are centered around Lee's algorithm [28], which although faster, it consumes a lot of resources when translated into hardware. The intent of this paper is to introduce an original, simple algorithmic solution while presenting the corresponding implementation method. The implementation is of a generic nature and was experimentally tested on a various number of inputs. By taking advantage of the recursive nature of the network, the routing problem is reduced to an elementary sorting of the input–output pairs in each layer. The position each pair occupies on the sorted list determines the switches' setting. All switches are routed linearly, and there is no back warding. The design was implemented as part of a larger project and was therefore thoroughly tested experimentally. The major obstacle in this endeavor was the condensation of the 2D matrix permutations used in other algorithms for each layer of the network, as well as the substitution of the aforementioned 2D-matrices' convolutions with the simple rearranging of elements. The latter part specifically was instrumental to giving our approach an advantage over other proposed algorithms.

The organization of the paper is as follows. Section 2 provides the propositions which serve as the foundations for the method. Section 3 includes the theoretical explanation as well as the proof of the design's functionality. Section 4 covers the hardware implementation of all ideas discussed in Section 3. Section 5 is where the implementation results are presented. Finally, we conclude in Section 6.

2. Preliminaries

The major goal of our work is to find an efficient, scalable, and functional way to route a Benes network. The act of "routing" is to produce a valid setting scheme for all switches in the network in order to satisfy an input–output routing table given as an input. Specifically for the case of Benes optical switches for application inside DCs, sizes of 8 × 8, 16 × 16, 32 × 32 and 64 × 64 are particularly interesting, as they offer similar port numbers with their electric switch counterparts, while at the same time, it is feasible to fabricate them in PICs. The algorithm itself was made for a generic number of inputs, but for the sake of brevity, both in the theoretical explanation and in the implementation section, in many cases, we consider only an 8 × 8 Benes network. It can be inferred, however, that the algorithmic steps presented herein can route any generic network, and in the following sections, the validity of our approach will become apparent. This section provides a list of propositions which are going to set the basis of this paper's approach. In all propositions listed below, all letter variables signify switches and ports (input and output) and thus should be considered integers.

Proposition 1. The general recursive form of a Benes network is shown in Figure 1. It has $n = 2^r$, $r > 1$, inputs, and $n = 2^r$ outputs. All n ports (0, 1, 2, 3) are the input ports and all n′ ports (0′, 1′, 2′, 3′) are the output ports. There are $m = 2^r - 1$ input switches and $m' = 2^r - 1'$ output switches. There are 2 subnetworks in the center, upper (U) and lower (L). These 2 subnetworks are Benes networks with 2^{r-1} inputs and 2^{r-1} outputs (half of the original network).

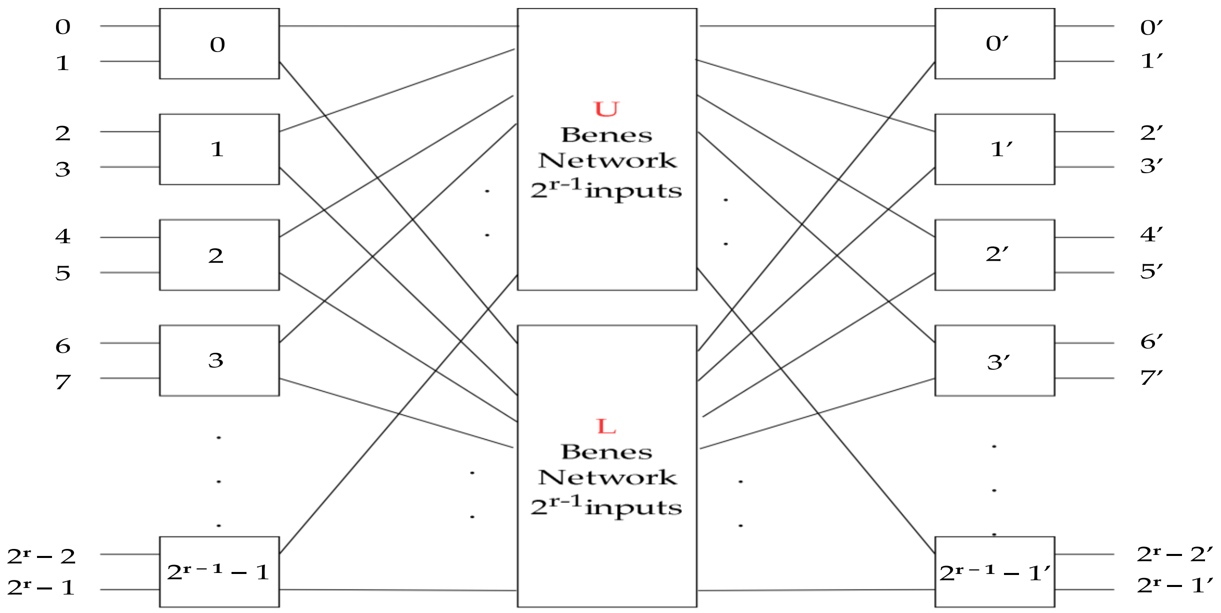

Figure 1. Recursive form of a Benes network.

Proposition 2. In a Benes network with 2^r inputs and outputs, there are r layers. $r - 1$ layers with 2^r switches (input and output switches) and 1 layer (the center layer) with 2^{r-1} switches. Layer $k \in [0, r-1]$, k has 2^k subnetworks with $2^{(r\text{-}k)}$ inputs, outputs each. The number of subnetworks doubles in each layer, while the number of inputs per subnetwork is halved. Figure 2 presents an 8 input-output Benes network. As we can see, it is a Benes network of $2^3 = 8$ $(r = 3)$ input–output ports. Red is layer 0 $(k = 0, 2^3 - 0 = 8$ input–output ports per subnetwork, $2^0 = 1$ subnetwork which is the original), blue is layer 1 $(k = 1, 2^{(3-1)} = 4$ input–output ports per subnetwork, $2^1 = 2$ subnetworks), and green is layer 2 $(k = 2, 2^{(3-2)} = 2$ input–output ports per subnetwork, 4 switches/subnetworks). One layer has multiple subnetworks. With every layer, the number of subnetworks is doubled and number of inputs per subnetwork is halved. For this reason, the number of input and output switches of each layer remains the same, except in the last layer. Layer $k = r - 1$ is the center of the network and consists only of switches and not subnetworks (switches and subnetworks coincide).

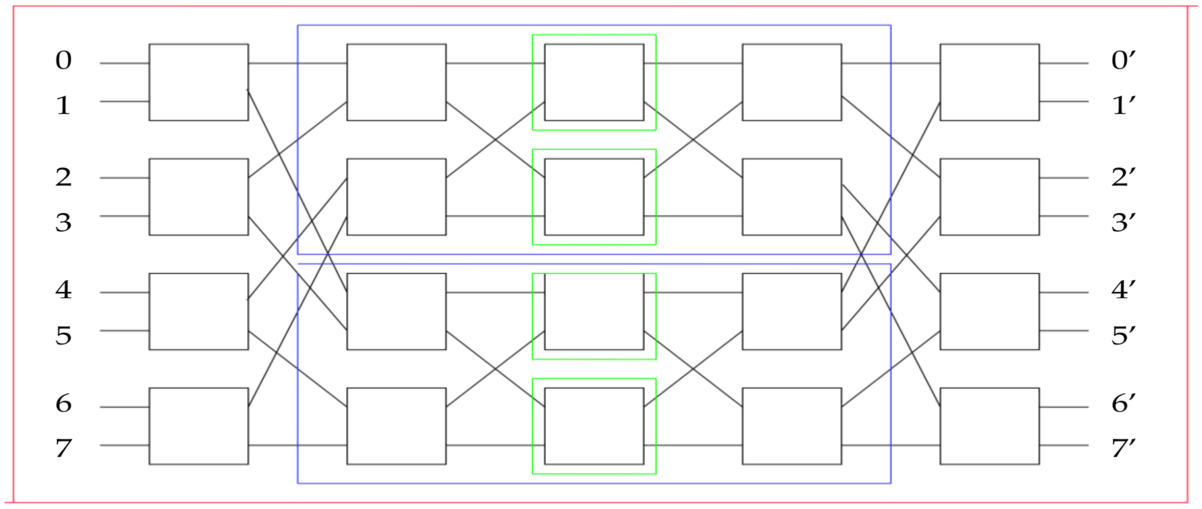

Figure 2. Internal overview of an 8 input–output Benes network. Layer 0 is marked red, layer 1 is marked blue and layer 2 is marked green.

Proposition 3. A switch (input or output) z ∈ [0, m − 1] (Proposition 1) has 2 input ports, 2 × z, and 2 × z + 1 and 2 output ports, 2 × z′ and 2 × z + 1′. For every switch, 2 settings are possible (Figure 3). When 2 × z is connected to 2 × z + 1′ and 2 × z + 1 is connected to 2 × z′ the setting is cross. When 2 × z is connected to 2 × z′ and 2 × z + 1 is connected to 2 × z + 1′ the setting is bar. Satisfying the requested routing paths by setting all switches in the correct setting is the purpose of the routing algorithm.

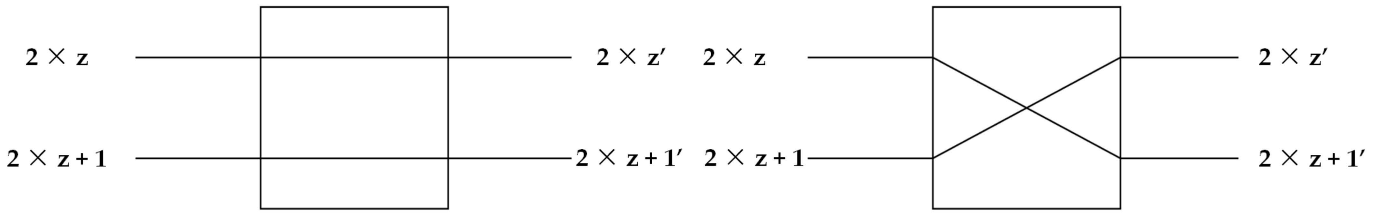

Figure 3. The 2 types of switch settings, bar (left) and cross (right).

Proposition 4. Because of the general form of the network (Proposition 1) and since 2 × z and 2 × z + 1 are ports of the same switch, they are connected to different subnetworks (up and down) regardless of the setting the switch has (cross or bar).

Proposition 5. If the switch is bar, input 2 × z is connected to the upper subnetwork and 2 × z + 1 is connected to the lower subnetwork; if the switch is cross, the opposite is true. The same stands for the outputs.

Proposition 6. If the subnetwork, with which every input and output is connected, is known, then by merit of Proposition 5, all switches can be configured correctly. Additionally, it is clear that the knowledge of the input–output table of the 2 subnetworks of the subsequent layer can be used to configure them. Therefore, the sequence that successfully routes layer 0 (the outmost layer) can be recursively used to route the inner layers and eventually the entire Benes network. This is related to the general form of the Benes network. The middle networks (the ones presented as "black boxes") follow the exact same form as layer 0 (the outermost layer). The only thing required to route the outer layer is the input–output routing table. The process of routing the outer layer itself produces the input–output routing tables of the inner networks. This is shown in Figure 4j clearly, where the lines connecting inputs–outputs of the inner networks can be seen. Each of these networks plays the same role for its own subnetworks and propagates the new routing tables. This process continues until all layers are routed fully.

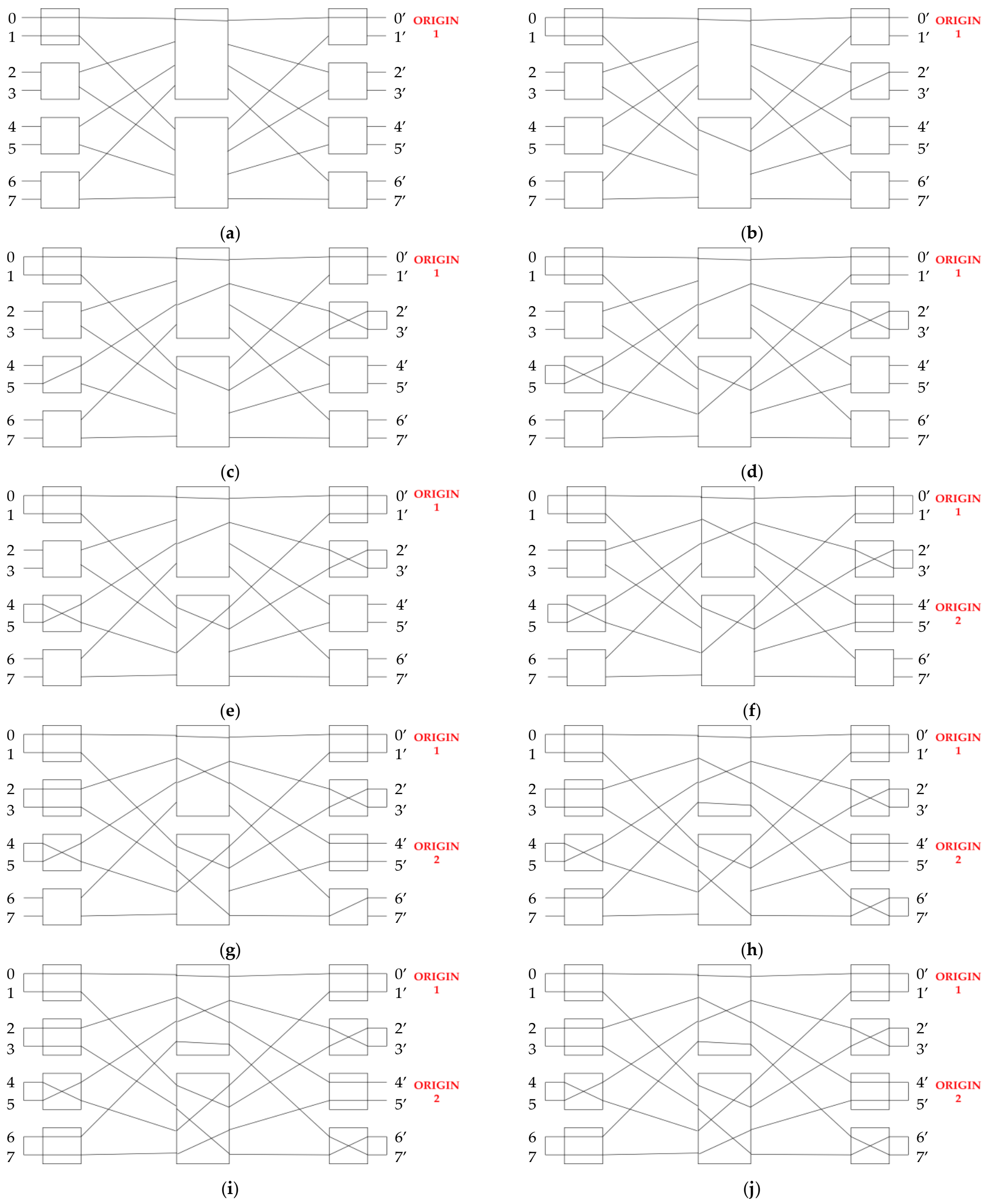

Figure 4. As we can see in Figure 4, the patch goes back and forth from inputs to outputs and up and down from the upper to lower subnetwork, constantly. The order of routing as seen in the subfigures is (**a**) path 0-0′, (**b**) path 1-2′, (**c**) path 5-3′, (**d**) path 4-1′, (**e**) end of loop/no routing takes place, (**f**) path 2-4′, (**g**) path 3-6′, (**h**) path 6-7′, (**i**) path 7-5′, (j) entire network is routed.

3. Proposed Routing Algorithm

3.1. Algorithm

According to Proposition 6 and because of the recursive nature of the Benes network, successfully routing layer 0 of a Benes network equates to routing the entire network. The algorithm described in this section aims to do just that. Set all 2^r (Proposition 2) switches of layer 0 to a setting that satisfies the given input–output table. With that completed, routing the entire network is simply a matter of producing the routing tables of the rest of the layers and successfully routing them. A typical run of the algorithm is presented in Figure 4.

Step 1. Select an output port of an unset switch and a subnetwork (upper or lower). If there are no unset switches, end the process. Generally, when the algorithm starts routing for the first time, we pick output $0'$. If we come back to this step during the routing, we can pick any unvisited port. No matter what the starting output port or subnetwork is, the algorithm is going to produce a correct setting.

Step 2. Route the output port through the selected subnetwork all the way to its corresponding output (Figure 4a).

Step 3. Pick the input of the same switch and route it back to its corresponding output by using the opposite subnetwork of what was used in Step 1 (Figure 4b). This is particularly important. Whether the upper or lower subnetwork is selected in Step 1, it will not change the outcome. It is, however, necessary to change subnetworks. If Step 1 used the upper subnetwork, Step 3 must use the lower subnetwork. If Step 1 used the lower subnetwork, Step 3 must use the upper network. This ensures the balance of occupied paths between the two possible routes (upper and lower) and essentially guarantees that for every routing toward a switch, there is always a routing out of it.

Step 4. If the output switch that the path leads to is already visited, go to Step 1 (Figure 4d). If the output switch is unoccupied, select the output of the same switch and the original selected subnetwork and proceed to Step 2 (Figure 4c). In the first case, a path loop is reached. Switches that belong to the same path loop can be treated as belonging to the same cluster. If a cluster occupies paths equally in both subnetworks (Step 3), its routes do not affect the rest of the network. The origin point of every cluster is the starting output of Step 1. In the second case, we select the output of the same switch to maintain the path course. The subnetwork selected in Step 1 for every cluster is always the opposite of the subnetwork, routes, which originate from an input, pass through. In other words, paths exchange subnetworks constantly.

Table 1 is the input-output routing table of the network routed in Figure 4.

Table 1. Example of an 8-input port Benes routing Table.

Input Ports	Output Ports
0	$0'$
1	$2'$
2	$4'$
3	$6'$
4	$1'$
5	$3'$
6	$7'$
7	$5'$

A complete algorithmic run of the proposed algorithm for input–output routing table mentioned on Table 1. In Figure 4a, as mentioned, when starting the routing of a layer, we select output port $0'$ and upper subnetwork (Step 1). We then proceed to route the first path (Step 2). In Figure 4b, the input of the same switch is routed back to the outputs (Step 3). In Figure 4c, the output switch is not visited, so we proceed with Step 2 (Step 4). The output is routed to the input through the upper subnetwork (Step 2). Figure 4d presents the routing of the input of the same input switch to the output (Step 3). In Figure 4e the output switch (switch $0'$ with ports $0'$, $1'$) is already routed, so we proceed to Step 1 (Step 4) and pick output 4′ as the origin of path 2 and the upper subnetwork as the starting subnetwork (Step 1). Figure 4f shows the routing of output 4′ to the corresponding input (Step 2); in Figure 4g, the path routes back to the outputs (Step 3). In Figure 4h, Output 7′ is selected (Step 4) and routed (Step 2) since it is in the same switch as 6′. Finally, in Figure 4i, we route the input of the same switch back to the outputs (Step 3). Figure 4j presents the completed routing.

When the process is completed, all switches of the layer are set. Moreover, all inputs and outputs of both subnetworks are consumed, providing the input–output tables for the next layer. In each layer k of a network of 2^r inputs, there are 2^k (subnetworks) $\times$ 2^{r-k} (inputs per subnetwork) = 2^r switches. There are r layers. The overall complexity is $2^r \times$ r. If $2^r = n$ inputs, the complexity becomes $n \times \log n$. This is the typical linear complexity of a Benes routing algorithm. There is no linear solution that can complete the routing in fewer steps [23]. Considering the fact that this is a linear algorithm, this is no surprise. One thing that should be mentioned is that in contrast to Opferman's algorithm, there is no backtracking to change the settings of the switches [31].

3.2. Proof of Concept

In this section, we present why the algorithm described in the previous section always leads to a valid solution. As per Proposition 6, successfully routing layer 0 of the network leads to routing the entire network. This is because, starting from layer 0 (outmost layer), all inner layers are recursively constructed by Benes networks with half the inputs of the layer before them. The example in Figure 4 clearly showcases how the inner subnetworks have 4 input–output pairs instead of the 8 that the outer layer has. In other words, if the validity of the algorithm for layer 0 is proved, then the entire network can be routed using the same steps on all subnetworks of all layers.

For the sake of simplicity and shortening the proof, the steps between the routing of one output and the routing of the next output (Steps 1 through 4) are referred to as an iteration. In every iteration, at least two switches are set; therefore, in layer 0 with 2^r inputs–outputs, 2^{r-1} iterations are needed. We prove that iteration a_0 leads to a_1 and iteration a_i leads to a_{i+1}, (in short $a_0 \rightarrow a_1$, ai $\rightarrow a_{i+1}$) without fail.

This means that the first step is always viable and all steps after that are always viable until all switches are set. In the following figures (5,6,7) *** denotes the starting output switch of an iteration.

Case $\mathbf{a_o \rightarrow a_1}$. It is obvious that all paths are available, and the first iteration can always be completed without fail (Figure 5).

Case $\mathbf{a_k \rightarrow a_{k+1}}$. Firstly, it is clear that since both inputs of an input switch are routed in one iteration, the routing to an input switch (Step 2) is immediately followed by the routing of the other input port to the corresponding switch (Step 3). It is impossible that an input switch is partly routed (Figure 6).

This means that in the a_k iteration, the target input switch has both paths open. For the path back to the next output, there are two possibilities.

1. Output switch at the end of the iteration is not set. It has not been visited before and can simply be set in the same way as all others: in this case, the algorithm proceeds as usual by selecting the output of the same switch as the starting point of a_{k+1} iteration.
2. If the output switch is routed, then we have reached the end of the loop path, and a new path needs to be selected. The possibility of conflict does not exist since the path which closes the loop always returns to the output side through a different subnetwork than the one who started the loop. Notice that in Figure 7, path 7-1′, which ends the loop, is routed through the lower subnetwork, and path 4-0′, which starts the loop, is routed through the upper subnetwork. Between paths 4-0′ and 7-1′, there are two paths, 5-3′ and 6-2′. The order of routing is 4-0′, 5-3′, 6-2′, 7-1′. If we were to add some more paths to the loop, these must be in multiples of two. For example, in Figure 7, if we change path 7-1′ to 7-5′, then we have to add 3-4′ to the input side and then again end on the output side with 2-1′. The entire routing now is 4-0′, 5-3′,

6-2′, 7-5′, 3-4′, and 2-1′. The path needs to visit the inputs first and the return and end at the outputs (where it started). This costs at least two paths or multiples of two. This is the reason the first and last paths are on different subnetworks. 4-0′ (Upper), 5-3′ (Lower), 6-2′ (Upper), 7-5′ (Lower), 3-4′ (Upper), 2-1′ (Lower). No matter how many paths come between path x-0′ and y-1′, they are always on different subnetworks.

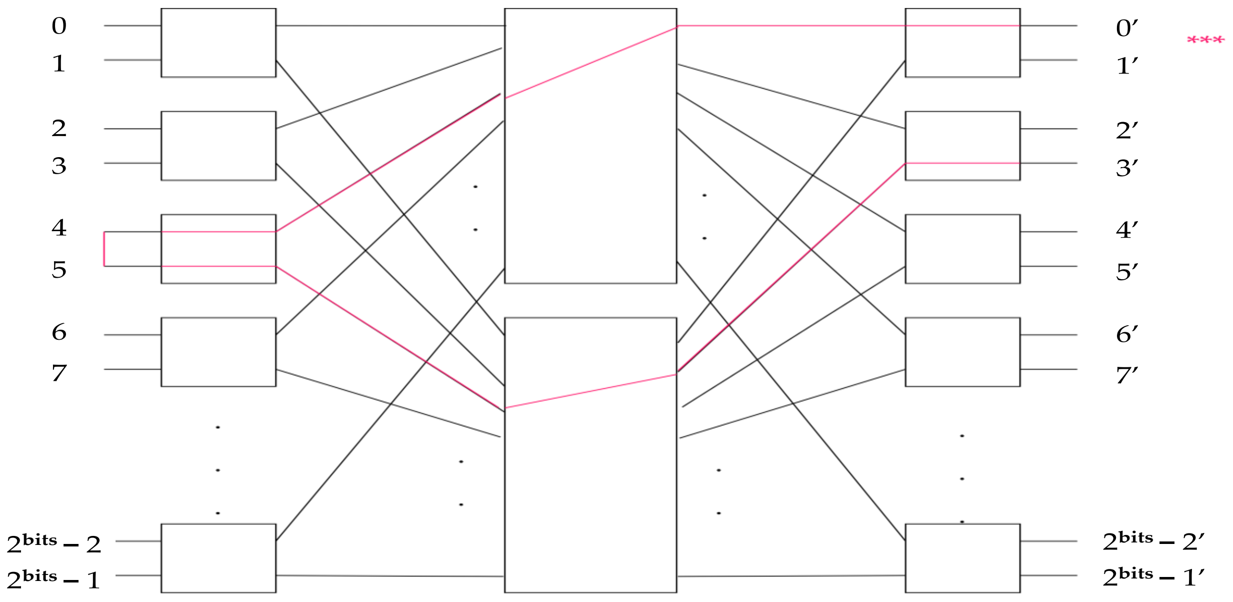

Figure 5. Start of routing sequence. (The dots mean that an arbitrary number of ports up to 2bits-1 (could be 8, 16, 32 etc) are implied but not all sketched for the sake of space. *** denotes the starting output switch of an iteration.)

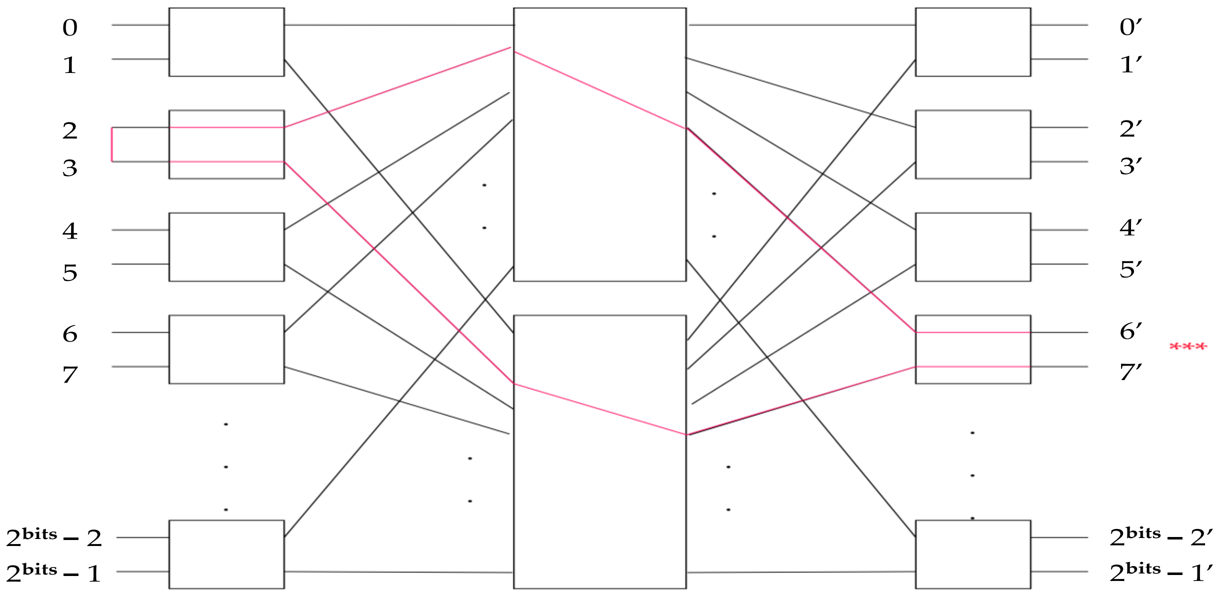

Figure 6. Random iteration during the algorithmic run. Input switch's 2 ports (2 and 3) are routed back-to-back. *** denotes the starting output switch of an iteration.

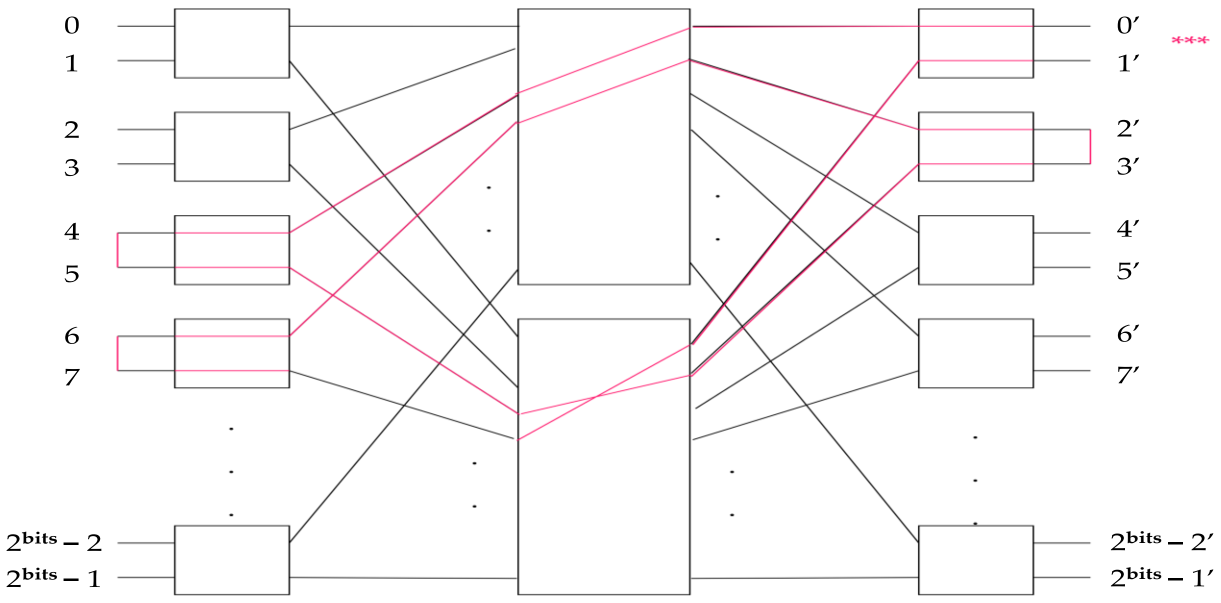

Figure 7. All switches are fully routed, while switch (0′,1′) ends the loop and forms an independent path. The routing of switches inside the path has no effect on the routing of the rest of the network. *** denotes the starting output switch of an iteration.

To summarize, we now know that $a_0 \rightarrow a_1$ and $a_k \rightarrow a_{k+1}$. The process successfully routes the entire outer layer. By applying the same algorithm to subsequent layers, we can route the entire network.

4. Implementation

This section covers the hardware implementation of the algorithm described in the previous sections. The input of the circuit is the input–output routing table, and the output is the switches' setting. After the routing is complete, it is expected that every output can be reached through its corresponding input.

The recursive nature allows the use of the same circuit for every subnetwork albeit with a different number of inputs–outputs. The overall architecture of the design for a generic number of inputs is shown in Figure 8. All subnetworks work identically. In each layer, the number of subnetworks is doubled, and the number of inputs per subnetwork is halved. The main output of each layer is the switches setting. The number of switches is the number of inputs–outputs (half input switches and half output switches) and is the same (2^r switches) for every layer, except the center (2^{r-1} switches). Apart from the switches' setting, each subnetwork also creates the input–output tables that it feeds to the next two subnetworks that replace it. In each instance, the act of "routing" inputs and outputs is translated to a simple sorting of the input–output entries. This sorting ensures that in the new list, pairs in $2 \times z$ positions are routed through the upper subnetwork and pairs in $2 \times z + 1$ positions are routed through the lower subnetwork. Figure 8 presents the subnetwork of the first layer with 2^r inputs. There are four modules. Modules 1 and 4 handle the input and output formatting, respectively. Modules 2 and 3 constitute the main system, and it is where the sorting takes place.

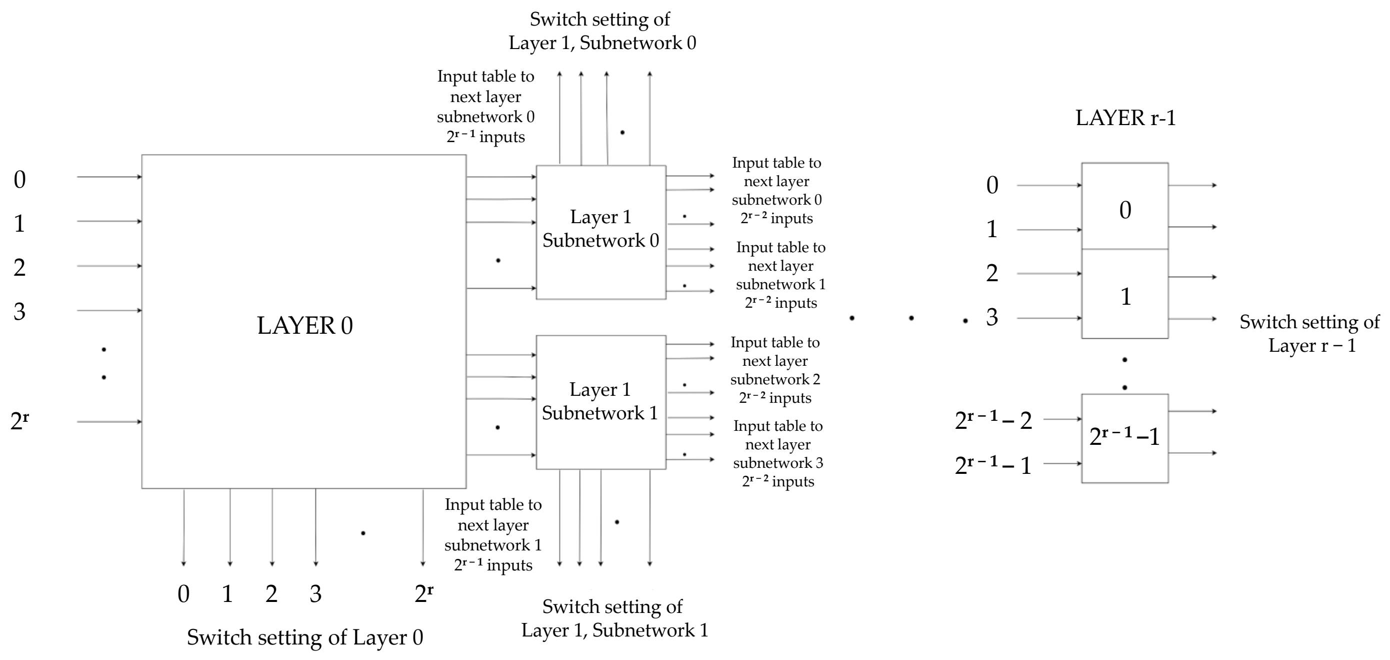
Switch setting of Layer 1, Subnetwork 0
Input table to next layer subnetwork 0 2^{r-1} inputs
LAYER 0
Layer 1 Subnetwork 0
Layer 1 Subnetwork 1
Input table to next layer subnetwork 0 2^{r-2} inputs
Input table to next layer subnetwork 1 2^{r-2} inputs
Input table to next layer subnetwork 2 2^{r-2} inputs
Input table to next layer subnetwork 3 2^{r-2} inputs
Input table to next layer subnetwork 1 2^{r-1} inputs
Switch setting of Layer 0
Switch setting of Layer 1, Subnetwork 1
LAYER r-1
Switch setting of Layer r − 1
$2^{r-1}-2$
$2^{r-1}-1$
2^r

Figure 8. Implemented system for a generic number of inputs.

For the more detailed explanation of the system, we used an example of an 8 input–output network (Figure 9). Out of all the subnetworks (Proposition 2) in the network, we focus on the function of the first layer (Figure 10) (8 input–output ports, 8 switches). Proposition 6 ensures that once the first layer is routed, all others follow in the same way.

The choice to present things in this way was primarily made in order to present the concept of each module in a concise way, which is easy to follow. The relationship between the 8-port network and the generic version is examined in each module section separately.

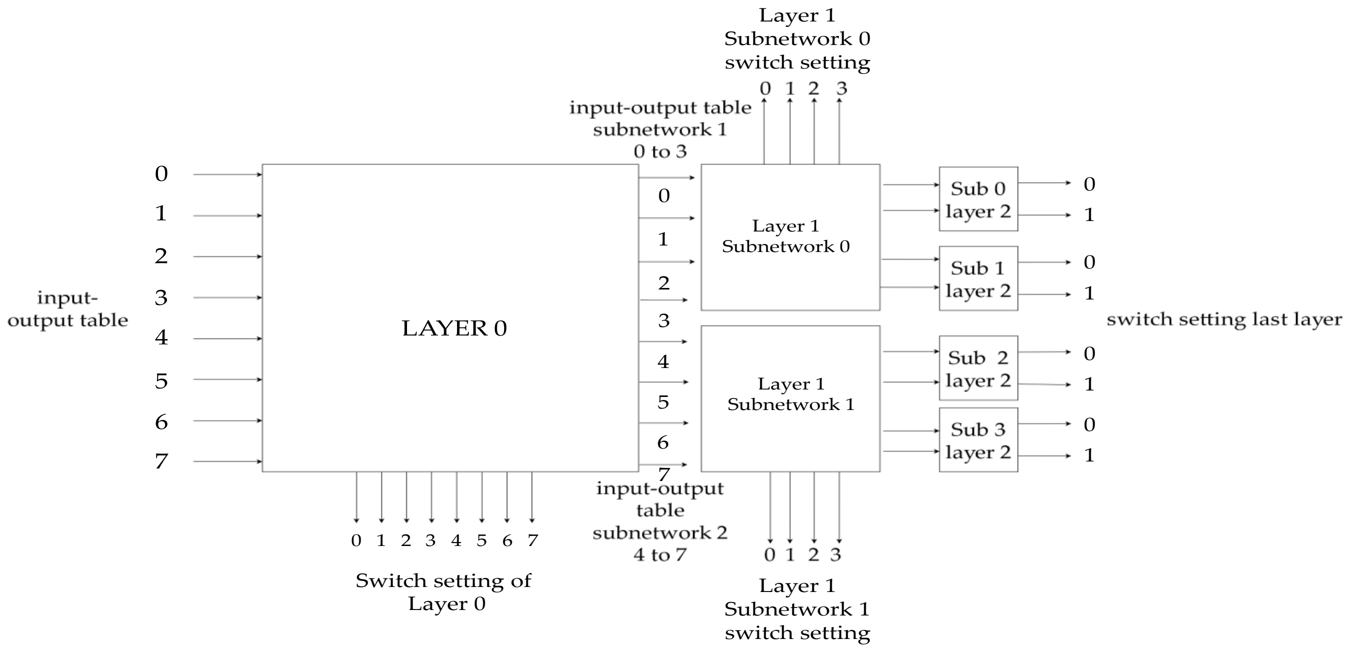
Layer 1 Subnetwork 0 switch setting
input-output table subnetwork 1 0 to 3
input-output table
LAYER 0
Layer 1 Subnetwork 0
Layer 1 Subnetwork 1
Sub 0 layer 2
Sub 1 layer 2
Sub 2 layer 2
Sub 3 layer 2
switch setting last layer
input-output table subnetwork 2 4 to 7
Switch setting of Layer 0
Layer 1 Subnetwork 1 switch setting

Figure 9. The 8 input–output Benes network routing circuit.

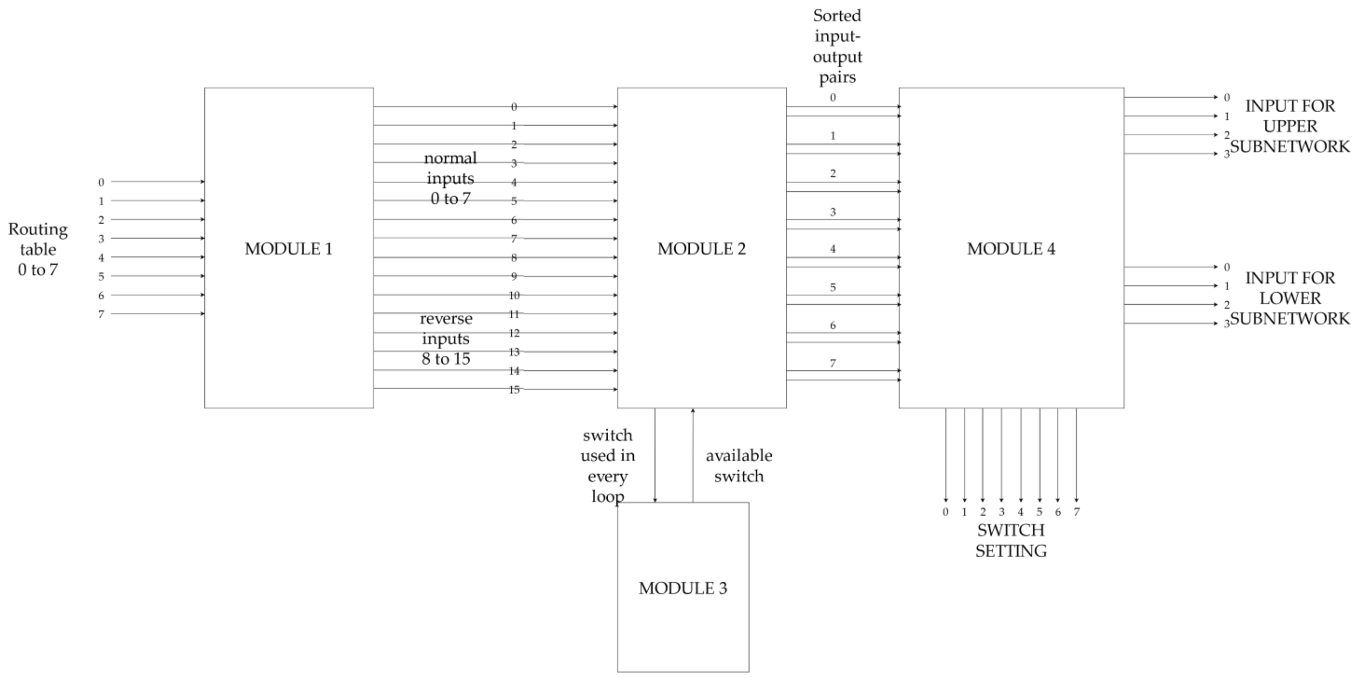

Figure 10. The 8 input–output Benes network routing circuit.

4.1. Module 1: Reverse Input Vector

This module creates the inverse input vector of the input–output routing table. The process uses the number of the output as the index of the inverse vector entries. For example, Table 2 presents the input–output routing table of an 8-port Benes network.

Table 2. Input–output routing table.

Input Ports	Output Ports
0	$0'$
1	$2'$
2	$4'$
3	$6'$
4	$1'$
5	$3'$
6	$7'$
7	$5'$

Input port 0 leads to output port $0'$. Input port 1 leads to output port $2'$. Port 2 to $4'$, etc. The inverse vector contains the output ports on the left side in an ascending order and their respective input port on the right side. In our example, position $2'$ in the new vector is filled with the number 1, position $0'$ with the number 0, and $4'$ with 2. The entire reverse vector in shown in Table 3.

Table 3. Inverse input–output table.

Output Ports	Input Ports
$0'$	0
$1'$	4
$2'$	1
$3'$	5
$4'$	2
$5'$	7
$6'$	3
$7'$	6

The left side of both vectors are always in an ascending order. That is why we can simply keep the right side and use the index of the input vectors as the left side. The right side of these two vectors is the input that is fed to the first layer and starts the entire routing process.

In the case of a generic number of input ports, nothing changes. We use the left side as an index and complete the inverse vector. If there is more than one zero elements, two or more input ports are connected to the same output. With $n = 2^r$ inputs, the creation of the inverse vector needs 1 clock cycle. The creation of the inverse vector is necessary in order to be able to find the corresponding port to an input or an output without searching the entire vector.

4.2. Module 2: Sorting Mechanism

Module 2 implements the algorithm described in the Section 3. In every cycle, it routes two paths: one from the outputs to the inputs and one from the inputs back to the outputs. In the circuit environment, "routing" means sorting input–output pairs in a way in which pairs in $2 \times m$ positions are routed through the upper network and pairs in $2 \times m + 1$ positions are routed through the lower network. Whether it is up and down, or down and up, makes no difference to the solution. Utilizing the two indexed vectors (Tables 2 and 3), a detailed run of the algorithm is shown below. As we mentioned in Section 3, one iteration consists of all steps, essentially routing an output to an input and routing the input of the same switch back to the output.

1st iteration. Any output can be selected to start the algorithm, but in our case, we select output $0'$ as the origin point of the first path (Table 4).

Table 4. 1st iteration.

Input Ports	Output Ports
0	$0'$
1	$2'$

2nd iteration. The output for the next iteration is $3'$ since it in the same switch as $2'$ (Table 5). This is detected simply by dividing the number of the 2 ports by 2.

Table 5. 2nd iteration.

Input Ports	Output Ports
0	$0'$
1	$2'$
5	$3'$
4	$1'$

3rd iteration. $1'$ is in the same switch as $0'$ so a new path must be created. Output port $4'$ is the new origin point (Table 6).

Table 6. 3rd iteration.

Input Ports	Output Ports
0	$0'$
1	$2'$
5	$3'$
4	$1'$
2	$4'$
3	$6'$

4th iteration. $7'$ is in the same switch as $6'$. Last iteration since all switches are routed (Table 7). This is the same example that is presented in Figure 4.

Table 7. 4th iteration.

Input Ports	Output Ports
0	0′
1	2′
5	3′
4	1′
2	4′
3	6′
6	7′
7	5′

In the case of a generic number of inputs, the algorithm is the same. We keep repeating the same steps until all switches are routed. The process is linear, and, in every clock cycle, two switches are routed. For $2^r = n$ input, $2^{r-1} = n/2$ clock cycles are needed.

4.3. Module 3: Unrouted Switch Pointer

Module 3 is a complementary module to the previous part. It is responsible for selecting the next starting point (origin) in the case of a path loop. For example, in iteration 2 (Table 5), the path ends in a loop on output switch 0′ (output ports 0′,1′). To continue the process, a new starting point is necessary. When there are no more available switches in Module 3, all loops are closed, and the algorithm ends. In order to avoid the long critical path of the priority encoder needed for the 1st implementation, a different design is considered. Every switch element, with the use of two auxiliary vectors, becomes a part of a linked list. Vector (left) contains the list entries on the left of each element and vector (right) contains the list entries on the right of each element. The head of the list is the first available pair. When an element is used in Module 2, it is ejected from the list by connecting the two adjacent entries. If the head is selected, then it is ejected, and the head pointer moves to the left entry. The list is always organized from left to right. An equivalent run to the first implementation is as follows.

1st iteration. The two vectors (left) and (right) have four entries, one for every output switch. The initial values for these entries are the adjacent switches. For example, position 1 refers to the 1′ switch (ports 2′,3′). Left of 1 is 2, and therefore, left [1] = 2; right of 2 is 3, and therefore right [1] = 3. Switch 0′ (ports 0′,1′) does not contribute since it is always routed first. The other vector cells are initialized with the same logic (Table 8). The head of the list is switch 1′ (ports 2′,3′), head = 1′.

Table 8. 1st iteration.

Number of Output Switch	Left of Switch	Right of Switch
3′	0′	2′
2′	3′	1′
1′	2′	0′
0′	X	X

2nd iteration. Output switch 1′ is ejected from the list. Switch 2′ (4′,5′) becomes head of the list, head = 2′ (Table 9).

Table 9. 2nd iteration.

Number of Output Switch	Left of Switch	Right of Switch
3′	2′	2′
2′	3′	3′
1′	X	X
0′	X	X

3rd iteration. A path loop is reached. Output switch $2'$ (head of list) is used to proceed with the routing. Consequently, switch $2'$ is ejected from the list (Table 10).

Table 10. 3rd iteration.

Number of Output Switch	Left of Switch	Right of Switch
$3'$	$3'$	$3'$
$2'$	X	X
$1'$	X	X
$0'$	X	X

4th iteration. Output switch $3'$ is ejected from the list. The routing is complete (Table 11).

Table 11. 4th iteration.

Number of Output Switch	Left of Switch	Right of Switch
3	3	3
2	X	X
1	X	X
0	X	X

Such an implementation dramatically improves the speed of the circuit. It allows us to avoid using logical constructs with long critical paths, such as a priority encoder to decide the availability of switches.

4.4. Module 4: Switch Routing and Input-Output Routing Tables of the Next Layer

The last module of the design reads the sorted list of pairs and routes the input and output switches accordingly. It reads every two entries in each side since adjacent entries belong to the same switch. In the case of a path loop, since the path has $2 \times \mathrm{k} + 1$ steps, it never samples the same switch. If a $2 \times \mathrm{m}$ input or output is the first input/output of a switch, then the switch is bar. If it is the second one, then the switch is cross. It also prepares the input–output table for the two subnetworks. The process is completed in $n/2$ cycles.

After sorting the entries with the previous modules, we now have the mapping of the paths (Table 12). All pairs in $2 \times \mathrm{m}$ positions are routed through the upper Benes subnetwork, and all pairs in $2 \times \mathrm{m} + 1$ are routed through the lower subnetwork. We check all $2 \times \mathrm{m}$ positions of the table and determine the routing of both input and output switches. In position 0, there is input port 0 and output port $0'$. Both ports 0 and $0'$ are the first port of their respective switch, so both switches are routed bar according to Proposition 5 (Table 13).

Table 12. U: upper subnetwork, L: lower subnetwork.

Input Ports	Output Ports	Subnetwork
0	$0'$	U
1	$2'$	L
5	$3'$	U
4	$1'$	L
2	$4'$	U
3	$6'$	L
6	$7'$	U
7	$5'$	L

Table 13. 1st iteration.

	Input Switch	Output Switch
0	Bar	bar
1		
2		
3		

The next 2 × m position is position 2 (3rd row). The respective ports are 5 and $3'$. Both are the second port of their respective switch, so both switches are cross (switch 2 for 5 and switch $1'$ for $3'$) (Table 14).

Table 14. 2nd iteration.

	Input Switch	Output Switch
0	Bar	bar
1		cross
2	Cross	
3		

Position 4 (5th row) contains ports 2 and $4'$. They are both the first port of their respective switch, so both switches are set to bar (Table 15).

Table 15. 3rd iteration.

	Input Switch	Output Switch
0	Bar	bar
1	Bar	cross
2	Cross	bar
3		

Position 6 (7th row) contains ports 6 and $7'$. Here, port 6 is the first input of input switch 3 and $7'$ is the second output of switch $3'$. Switch 3 is set to bar, and switch $3'$ is set to cross (Table 16).

Table 16. 4th iteration.

	Input Switch	Output Switch
$0'$	Bar	bar
$1'$	Bar	cross
$2'$	Cross	bar
$3'$	Bar	cross

The entire layer is presented routed in the picture below (Figure 11).

Regarding the input routing tables to the next layers (upper and lower subnetworks), each switch can only be connected to a subnetwork once. The number of a subnetwork port that a switch is connected to is the same as the number of the switch (Proposition 2). For this reason, in order to produce the input–output tables for the next layer, we separate the input–output pairs (Table 12) by their subnetwork (Tables 17 and 18).

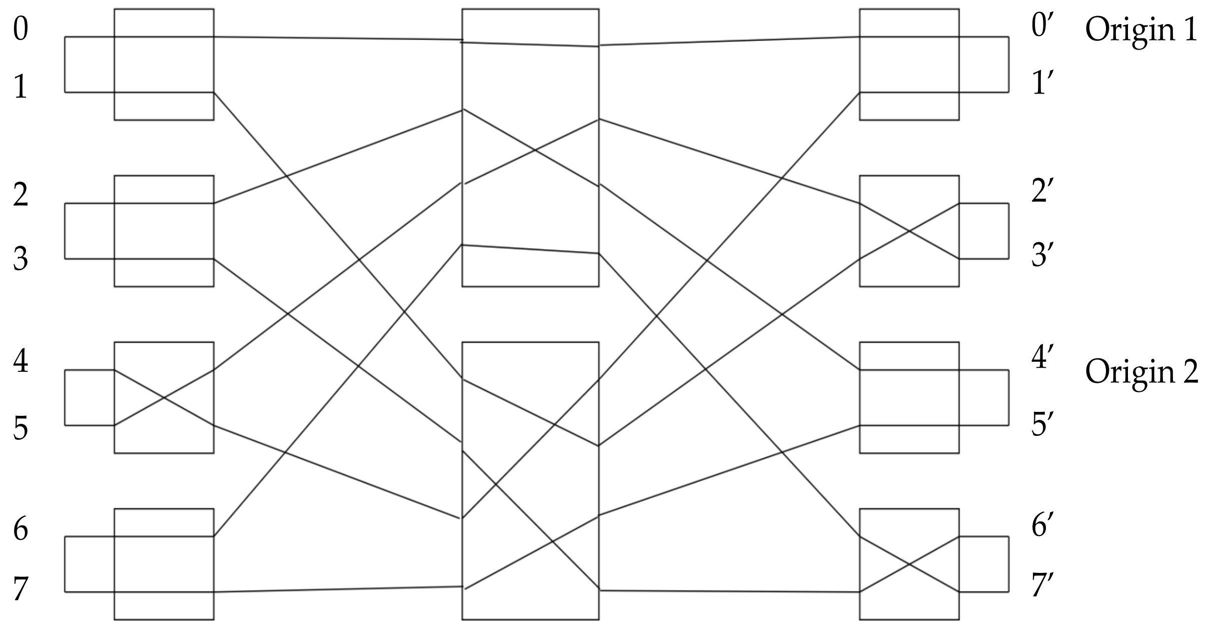

Figure 11. After setting all the switches according to the sorting.

Table 17. Input–output pairs that are routed through the upper subnetwork.

Input Ports	Output Ports	Subnetwork
0	0′	U
5	3′	U
2	4′	U
6	7′	U

Table 18. Input–output pairs that are routed through the lower subnetwork.

Input Ports	Output Ports	Subnetwork
1	2′	L
4	1′	L
3	6′	L
7	5′	L

We then must divide by 2. This division is necessary in order to produce a valid routing table since subnetworks have half the number of input and output ports (Tables 19 and 20).

Table 19. Upper pairs are divided by 2.

Input Ports	Output Ports	Subenetwork
0	0′	U
2	1′	U
1	2′	U
3	3′	U

Table 20. Lower pairs divided by 2.

Input Ports	Output Ports	Subnetwork
0	1′	L
2	0′	L
1	3′	L
3	2′	L

Finally, the pairs are organized in the same manner (ascending input order) as the original input–output routing table (Tables 21 and 22).

Table 21. Routing table of the upper subnetwork.

Input Ports	Output Ports	Subnetwork
0	0′	U
1	2′	U
2	1′	U
3	3′	U

Table 22. Routing table of the lower subnetwork.

Input Ports	Output Ports	Subnetwork
0	1′	L
1	3′	L
2	0′	L
3	2′	L

At this point, the tables are passed as inputs to the subnetworks of the next layer and are used to route it (Proposition 6). The process repeats until all switches on all layers are set. The entire fully routed network can be seen in Figure 12.

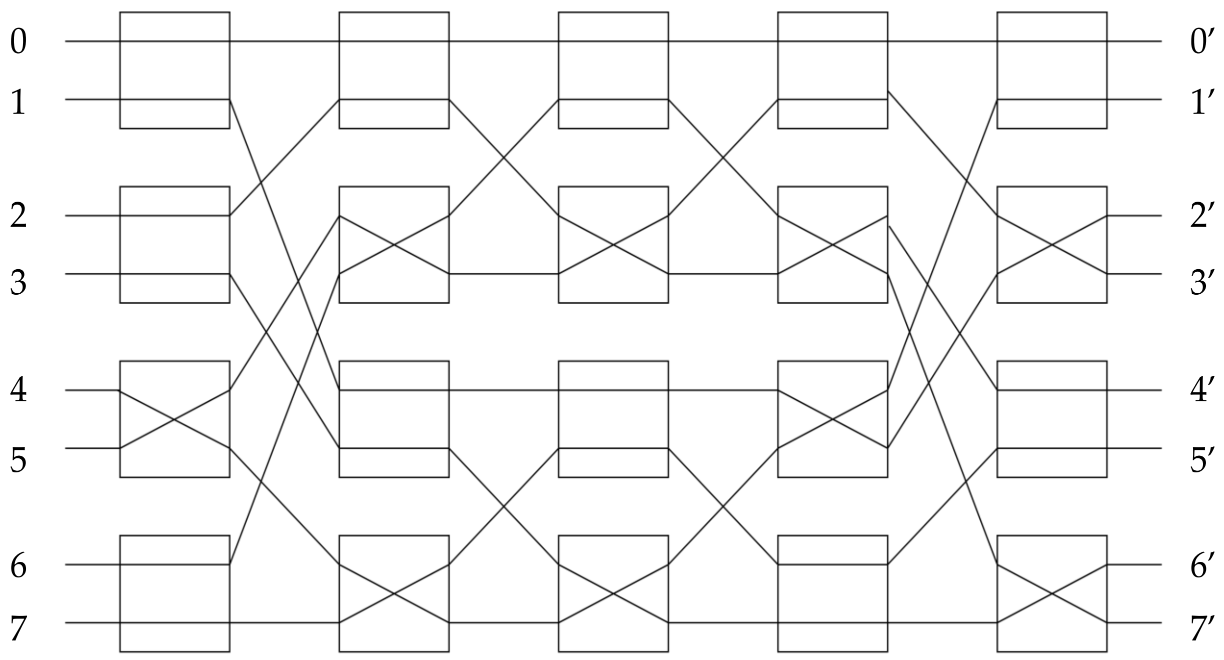

Figure 12. Fully routed 8 × 8 Benes network.

The process is linear in this case also, and it is the same for a generic number of inputs, as nothing changes. We simply keep setting the switches until the end. Since we use only 2 × m positions for 2^r inputs, 2^{r-1} = $n/2$ clock cycles are needed.

The cycles of the individual modules are Module 1: 1, Module 2: $n/2$, Module 3: 1, Module 4: $n/2$, and thus, $1 + n/2 + 1 + n/2 = 2 + n \approx n$ total cycles. Due to the recursiveness of the Benes network (Proposition 1) there are log$n$ = r layers of subnetworks. The complexity of the entire system is log$n \times n$, the minimum complexity to route a Benes network with one processing unit [23].

5. Board Implementation, Results and Discussion

In [32], the author used the Taiwan Semiconductor Manufacturing Company (TSMC) ASIC libraries to implement their work. In our case, we implemented the design on the mid-range FPGA board, ZCU-104 Evaluation board (Figure 13) using Vivado. In order to leave as many resources as possible available for the algorithm to utilize, we used a single pin for the input and a single pin for the output.

Figure 13. Photograph of the ZCU-104 evaluation board used to implement the routing algorithm.

The input–output routing table entries were linearly inserted to a shift register (input vector), whose elements became the corresponding output port (Table 23). In the example, port 0 is paired with port 0′. So, the first 3 bits are 000. Port 1 is paired with port 2′, so the next 3 bits are 010. Using this logic, the corresponding input vector of the example is 000 010 100 110 001 011 111 101. For a generic size of 2^r inputs, the input vector's size is $r \times 2^r$ bits.

Table 23. Shift input register for 8 × 8 Benes network.

Shift Input Register ($3 \times 2^3 = 24$ bits)								
Input ports	0	1	2	3	4	5	6	7
Bits	000	010	100	110	001	011	111	101
Output ports	0′	2′	4′	6′	1′	3′	7′	5′

The same logic applies to the output. For all layers, the setting of all switches is loaded to a shift register connected to the single output pin. The size of the output shift register for a Benes network of 2^r inputs is $(r - 1) \times 2^r + 2^{r-1}$ bits. Each switch is represented by one bit: 1 for cross setting and 0 for bar setting (Proposition 3). There are $(r - 1)$ layers with 2^r switches and one layer with 2^{r-1} switches (Proposition 2). Table 24 shows the output vector of the 8 × 8 example per layer (Proposition 2). The first 4 bits are the input switches of the first layer (red). The next 4 are the output switches of the first layer, etc.

Table 24. Shift output register for 8 × 8 Benes network.

Shift Output Register [$(3 - 1) \times 2^3 + 2^{3-1} = 20$ bits (1 bit for Every Switch)]			
Layers (r = 3)	1st layer	2nd layer	3rd layer
Bits	00100101	01010110	0101

The design was implemented originally for Benes networks of size 8, 16, 32, and 64 (Table 25).

Table 25. Resource utilization.

Size of Benes Network	8	16	32	64
CLB LUTs as Logic	1416 (0.61%)	3656 (1.59%)	13549 (5.88%)	35925 (15.59%)
CLB Registers as Flip Flop	561 (0.12%)	1558 (0.34%)	4002 (0.87%)	9993 (2.17%)
CARRY 8	67 (0.23)	145 (0.5%)	315 (1.09%)	657 (2.28%)
F7 Muxes	2 (<0.01%)	94 (0.08%)	610 (0.53%)	2405 (2.09%)
F8 Muxes	0 (0.00%)	9 (0.02%)	91 (0.16%)	486 (0.84%)

The operational frequency along with the time needed to produce one fully routed setting is listed below (Table 26). Complexity of the algorithm is $n \times \log n$ so in order to find the time need to produce one complete result, we multiply the required clock cycles with the minimum period (Table 26). No DSP, RAM blocks or other specialized blocks were used.

Table 26. Operating frequency.

Size of Benes Network	8	16	32	64
Minimum Operational Period (ns)	4	4	6	8
Maximum operation frequency (MHz)	250	250	166	125
Time needed for 1 complete setting (ns)	96	256	960	3072

Lastly, we have the power consumed as reported by the Vivado tool (Table 27).

Table 27. Power.

Size of Benes Network	8	16	32	64
Power consumed (W)	0.036	0.105	0.241	0.437

In [32], the author provides a utilization report on a synthesized ASIC implementation of Lee's algorithm [28]. For a 64-port network, the parallel design consumes around 1.32×10^5 logic cells and needs $5.6 \times 36 = 201.6$ ns (5.6 ns being the critical path and 36 the needed clock cycles to complete 1 operation) to complete one routing.

A rather rough comparison between these two implementation approaches could be done as follows. According to [33], the fact that architectures built on FPGAs are, on average, 35 times larger and 4 times slower than those build on ASICs, these values can be converted to FPGA terms. However, the difference in the technology nodes (65 nm vs. 16 nm of the used FPGA) should be taken into account since there is a proportional increase in devices speed, according to the scaling factor. More specifically, the above correspond to an area of $132{,}000 \times 35 = 4{,}620{,}000$ cells and a required time of $201.6 \times 4/4 = 201.6$ ns. By merit of this analogy, it seems that the metric of space $\times$ time is lower in our design ($49{,}466 \times 3072 < 4{,}620{,}000 \times 201.6$) by a factor of 6.12.

Apart from parallel and linear routing algorithms, there is another approach to routing interconnection networks and networks in general. Machine learning has risen to prominence in recent years and is being used in many different areas of networking, including network routing. In our case, machine learning-assisted routing of Benes networks primarily concerns photonic switches. Researchers have opted to used machine learning for Benes switch routing since it is less resource demanding compared to the already established algorithms. The author of [34] conducted extensive work on the subject. In [34], a photonic 4×4 Benes switch was routed using machine learning techniques. The author stated that there are $2^6 = 64$ possible combinations. This is drastically more than our model which needs only $4 \times \log 24 = 4 \times 2 = 8$ clock cycles ($n \times \log 2n$, $n = 4$) to complete the routing. In [35], the same trend is continuous. For a classic 8×8 Benes network, the possible combinations are in the range of millions. This would require much more time than our approach of $8 \log 28 = 8 \times 3 = 24$ clock cycles/steps. The author also used non-classic, arbitrary size Benes networks [36] with ports not equal to a power of 2. The modification needed for our proposed algorithm to route all kinds of Benes networks (arbitrary size and classic) will be the subject of our future work. It seems that the proposed algorithm can replace both hard and soft existing computing methods. It is straightforward enough to be implemented easily and does not consume a lot of resources. As it was presented, parallel solutions require too many resources while not offering the counterweight advantages in terms of speed (metric speed $\times$ space) and machine learning has to contend with very big data sets (millions for 8 $\times$ 8 Benes switch).

6. Conclusions

In this paper, we presented a linear algorithm for the purpose of routing Benes networks of all sizes. The purpose of the development of the algorithm is to create an architecture that can perform the routing in a reasonable amount of time while simultaneously being confined enough to fit on a low-cost FPGA. Other linear solutions involve complex matrix calculations or require backtracking. This impairs their effectiveness and causes them to demand additional resources. On the other hand, the parallel hardware solutions, although faster, are very resource demanding. The design presented in this work is straightforward enough to be coded easily in any type of hardware description language and does not use any specialized blocks (DSP, RAM blocks). Its implementation is possible, even on relatively small boards, and the resources it consumes are very few. There is strong evidence to suggest that with a slight modification, it can handle the routing of AS-Benes networks (arbitrary size Benes networks). Despite its efficiency however, at the present time, the proposed algorithm can only be used to route a classic Benes network (2000 inputs). Classic Benes networks are only a subset of the Clos family of interconnection networks and are only used extensively in optical applications. An immediate improvement would be to make slight modifications in order to be able to handle the routing of AS-Benes networks (arbitrary size Benes networks). After that, the same principle of sorting input–output entries can be applied to the routing of other 3-stage Clos interconnection networks in order to establish whether or not it can be used for a wider selection of these types of structures.

Author Contributions: Conceptualization, D.N. and P.G.; methodology, D.N.; software, D.N.; validation, D.N.; formal analysis, D.N.; investigation, D.N.; resources, P.G.; data curation, P.G.; writing—original draft preparation, D.N.; writing—review and editing, P.G.; visualization, P.G.; supervision, P.G. and C.K.; project administration, P.G., C.K. and H.A.; funding acquisition, P.G., C.K. and H.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by the European Commission under the Horizon 2020 project POETICS, grant number 871769. The APC is covered by a waver.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of the data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Clos, C. A Study of Non-Blocking Switching Networks. *Bell Syst. Tech. J.* **1953**, *32*, 406–424. [CrossRef]
2. Benes, E. *Mathematical Theory of Connecting Networks and Telephone Traffic*; Academic Press: New York, NY, USA, 1965.
3. Sandeep, S.; Bansal, P.K.; Kahlon, K.S. On a Class of Multistage Interconnection Network in Parallel Processing. *Int. J. Comput. Sci. Netw.* **2008**, *8*, 287–291.
4. Shao, F.-C.; Our, A.Y. Efficient Nonblocking Switching Networks for Interprocessor Communications in Multiprocessor Systems. *IEEE Trans. Parallel Distrib. Syst.* **1995**, *6*, 132–141. [CrossRef]
5. Jajszczyk, A. Nonblocking, repackable, and rearrangeable Clos networks: Fifty years of the theory evolution. *IEEE Commun. Mag.* **2003**, *41*, 28–33. [CrossRef]

6. Minkenberg, C.; Krishnaswamy, R.; Zilkie, A.; Nelson, D. Co-packaged datacenter optics: Opportunities and challenges. *IET Optoelectron.* **2021**, *15*, 77–91. [CrossRef]
7. Raja, A.S.; Lange, S.; Karpov, M.; Shi, K.; Fu, X.; Behrendt, R.; Cletheroe, D.; Lukashchuk, A.; Haller, I.; Karinou, F.; et al. Ultrafast optical circuit switching for data centers using integrated soliton microcombs. *Nat. Commun.* **2021**, *12*, 5867. [CrossRef]
8. Yin, Y.; Proietti, R.; Ye, X.; Nitta, C.J.; Akella, V.; Yoo, S.J.B. LIONS: An AWGR-Based Low-Latency Optical Switch for High-Performance Computing and Data Centers. *IEEE J. Sel. Top. Quantum Elec.* **2013**, *19*, 3600409. [CrossRef]
9. Xiao, X.; Zhang, Y.; Zhang, K.; Proietti, R.; Yoo, S.J.B. Multi-FSR on-chip optical interconnects using silicon nitride AWGR. In *Conference on Lasers and Electro-Optics (CLEO), OSA Technical Digest*; Optical Society of America: Washington, DC, USA, 2019.
10. Dupuis, N.; Proesel, J.E.; Boyer, N.; Ainspan, H.; Baks, C.W.; Doany, F.; Cyr, E.; Lee, B.G. An 8 × 8 Silicon Photonic Switch Module with Nanosecond-Scale Reconfigurability. In *Optical Fiber Communications Conference and Exhibition (OFC)*; Optical Society of America: Washington, DC, USA, 2020.
11. Groumas, P.; Zhang, Z.; Katopodis, V.; Konczykowska, A.; Dupuy, J.Y.; Beretta, A.; Dede, A.; Choi, J.H.; Harati, P.; Jorge, F.; et al. Tunable 100 Gbaud Transmitter Based on Hybrid Polymer-to-Polymer Integration for Flexible Optical Interconnects. *J. Light. Technol.* **2016**, *34*, 407–418. [CrossRef]
12. Maier, G.; Savastano, L.; Pattavina, A.; Bregni, S.; Martinelli, M. Optical-switch Benes architecture based on 2-D MEMS. In Proceedings of the 2006 Workshop on High Performance Switching and Routing, Poznań, Poland, 7–9 June 2006.
13. Watson, J.E.; Milbrodt, M.A.; Bahadori, K.; Dautartas, M.F.; Kemmerer, C.T.; Moser, D.T.; Schelling, A.W.; Murphy, T.O.; Veselka, J.J.; Herr, D.A. A low-voltage 8 × 8 Ti:LiNbO$_3$ switch with a dilated-Benes architecture. *J. Light. Technol.* **1990**, *8*, 794–801. [CrossRef]
14. Qiao, L.; Tang, W.; Chu, T. 32 × 32 silicon electro-optic switch with built-in monitors and balanced-status units. *Sci. Rep.* **2017**, *7*, 42306. [CrossRef]
15. Liangjun, L.; Shuoyi, Z.; Linjie, Z.; Dong, L.; Zuxiang, L.; Minjuan, W.; Xinwan, L.; Jianping, C. 16 × 16 non-blocking silicon optical switch based on electro-optic Mach-Zehnder interferometers. *Opt. Express* **2016**, *24*, 9295–9307.
16. Ding, M.; Wonfor, A.; Cheng, Q.; Penty, R.V.; White, I.H. Hybrid MZI-SOA InGaAs/InP Photonic Integrated Switches. *IEEE J. Sel. Top. Quantum Electron.* **2018**, *24*, 3600108. [CrossRef]
17. Eurpean Commission. CoPackaging of Terabit Direct-Detection and Coherent Optical Engines and Switching Circuits in mulTI-Chip ModuleS for Datacenter Networks and the 5G Optical Fronthaul. Available online: https://cordis.europa.eu/project/id/871769 (accessed on 23 December 2021).
18. Zhang, Z.; Felipe, D.; Katopodis, V.; Groumas, P.; Kouloumentas, C.; Avramopoulos, H.; Dupuy, J.-Y.; Konczykowska, A.; Dede, A.; Beretta, A.; et al. Hybrid Photonic Integration on a Polymer Platform. *Photonics* **2015**, *2*, 1005–1026. [CrossRef]
19. Farrington, N.; Porter, G.; Radhakrishnan, S.; Bazzaz, H.H.; Subramanya, V.; Fainman, Y.; Papen, G.; Vahdat, A. Helios: A hybrid electrical/optical switch architecture for modular data centers. In Proceedings of the ACM SIGCOMM 2010 Conference, New Delhi, India, 30 August–3 September 2010; Volume 40, pp. 339–350.
20. Groumas, P.; Katopodis, V.; Choi, J.H.; Bach, H.G.; Dupuy, J.Y.; Konczykowska, A.; Zhang, Z.; Harati, P.; Miller, E.; Beretta, A.; et al. Multi-100 GbE and 400 GbE Interfaces for Intra-Data Center Networks Based on Arrayed Transceivers with Serial 100 Gb/s Operation. *J. Light. Technol.* **2015**, *33*, 943–954. [CrossRef]
21. Chakrabarty, A.; Collier, M.; Mukhopadhyay, S. Matrix-Based Nonblocking Routing Algorithm for Benes Networks. In Proceedings of the 2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, Computation World, Athens, Greece, 15–20 November 2009.
22. Leighton, F.T. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, 1st ed.; Kaufmann, M., Ed.; Elsevier: San Mateo, CA, USA, 1992.
23. Opferman, D.C.; Tsao-Wu, N.T. On a Class of Rearrangeable Switching Networks, Part I: Control Algorithm. *Bell Syst. Tech. J.* **1971**, *50*, 1579–1600. [CrossRef]
24. Andresen, S. The looping algorithm extended to base 2 t rearrangeable switching networks. *IEEE Trans. Commun.* **1977**, *25*, 1057–1063. [CrossRef]
25. Waksman, A. A Permutation Network. *J. ACM* **1968**, *15*, 159–163. [CrossRef]
26. Nassimi, D.; Sahni, S. A self-routing Benes network and Parallel Permutation Algorithm. *IEEE Trans. Comput.* **1981**, *30*, 332–340. [CrossRef]
27. Nassimi, D.; Sahni, S. A self-routing Benes network. In Proceedings of the 7th Annual Symposium on Computer Architecture, La Baule, France, 6–8 May 1980; pp. 190–195.
28. Lee, T.; Liew, S. Parallel routing algorithms in Benes-Clos networks. *IEEE Trans. Commun.* **2002**, *50*, 1841–1847. [CrossRef]
29. Aghakhani, K.; Karimi, A. A Novel Routing Algorithm in Benes Networks. *Int. J. Educ. Advan.* **2016**, *7*, 168–177.
30. Lee, K.Y. A new Benes network control algorithm and Parallel Permutation Algorithm. *IEEE Trans. Comput.* **1981**, *30*, 157–161.
31. Karimi, A.; Aghakhani, K.; Manavi, S.; Zarafshan, F.; Al-Haddad, S.A.R. Introduction and Analysis of Optimal Routing Algorithm in Benes Networks. *Procedia Comput. Sci.* **2014**, *42*, 313–319. [CrossRef]
32. Jiang, Y. Hardware Implementation of Parallel Algorithm for Setting Up Benes Networks. *Int. J. High Perform. Syst. Archit.* **2017**, *7*, 26–40. [CrossRef]
33. Boutros, A.; Betz, V. FPGA Architecture: Principles and Progression. *IEEE Circuits Syst. Mag.* **2021**, *21*, 4–29. [CrossRef]

34. Khan, I.; Chalony, M.; Ghillino, E.; Masood, M.U.; Patel, J.; Richards, D.; Mena, P.; Bardella, P.; Carena, A.; Curri, V. Machine learning assisted abstraction of photonic integrated circuits in fully disaggregated transparent optical networks. In Proceedings of the 2020 22nd International Conference on Transparent Optical Networks (ICTON), Bari, Italy, 19–23 July 2020; pp. 1–4.
35. Khan, I.; Tunesi, L.; Masood, M.U.; Ghillino, E.; Bardella, P.; Carena, A.; Curri, V. Automatic Management of N $\times$ N Photonic Switch Powered by Machine Learning in Software-Defined Optical Transport. *IEEE Open J. Commun. Soc.* **2021**, *2*, 1358–1365. [CrossRef]
36. Chang, C.; Melhem, R. Arbitrary Size Benes Networks. *Parallel Process. Lett.* **1997**, *7*, 279–284. [CrossRef]