Design and implementation of fast and hardware-efficient parallel processing elements to set full and partial permutations in Beneš networks

Labson Koloko | Takahiro Matsumoto | Hitoshi Obara

Graduate School of Engineering Science, Akita University, Tegata Gakuen, Akita, Japan

Correspondence
Labson Koloko, Graduate School of Engineering Science, Akita University, Tegata Gakuen, Akita 010-8502, Japan.
Email: labinct@gmail.com

Abstract
A new design for parallel and distributed processing elements (PEs) is proposed to configure Beneš networks based on a novel parallel algorithm that can realise full and partial permutations in a unified manner with very little overhead time and extra hardware. The proposed design reduces the hardware complexity of PEs from $O(N^2)$ to $O(N(\log_2 N)^2)$ due to a distributed architecture. In the proposed design, asynchronous operation was introduced in parts to reduce the time complexity per PE stage down to $O(1)$ within a certain $N$, while it takes $O(\log_2 N)$ time per PE stage in conventional algorithms. A prototype parallel was constructed and PEs were distributed in a field programmable gate array to investigate performance for the switch size of $N = 4$ to 32. The experimental results demonstrate that the proposed design outperforms a recent method by at least several times in terms of hardware and processing time complexities.

1 Introduction

The Beneš network (BNW) is a typical multistage switching network that provides unicast connections between $N$ inputs and $N$ outputs, where $N = 2^n$. An $N \times N$ BNW comprises $2n - 1$ stages, each of which comprises $N/2 \times 2 \times 2$ switch elements (SE) [1]. There is a total of $O(N \log_2 N)$ SEs, which satisfies the theoretical lower bound of nonblocking switch complexity [2]. The BNW is rearrangeably nonblocking, that is, existing connections can be diverted to alternative routes to create space for a new call. Meanwhile, when all connection requests are given simultaneously, it can be configured without such rearrangements using complex routing algorithms. These characteristics make BNWs effective for interconnection networks that provide a limited number of permutations for which sets of switch control bits (SCBs) are precomputed and indexed [3, 4]. In fact, the switch control complexity of the BNW is $O(N \log_2 N)$ under sequential algorithms [5, 6], and this $O(N \log_2 N)$ time simply reflects the number of SEs that must be set in the BNW. This is much longer than the $O(1)$ time of crossbar switches, which, in contrast, have up to $O(N^2)$ SEs [7].

Several parallel algorithms have been investigated to apply BNWs to high-speed time-division multiplexed systems of agile responsibility to arbitrary permutations [8]. The initial work emerged in two different areas concurrently and independently. In 1981, Lev et al. formulated parallel algorithms for Beneš/Clos networks based on a mathematical (graph theory) approach [9], and, in 1982, Nassimi and Sahni developed a parallel algorithm for BNWs based on an engineering (parallel computing) approach [10]. Both of these techniques achieved $O((\log_2 N)^2)$ time complexity with completely interconnected parallel computers for full permutations, where each input corresponds to a unique output.

In many practical applications, some inputs may be idle, resulting in partial permutation. Parallel algorithms for BNWs should handle both full and partial permutations [11]. However, the original parallel algorithms suspend processing when they encounter an idle connection. There are two major approaches to address this issue. First, in 1995, Lee and Oruc introduced quadruple datasets to match idle inputs with idle outputs as dummy destinations [12]. Once each input is assigned to a unique destination output, conventional parallel algorithms
work effectively. However, this approach requires considerable pre-processing and a complicated data structure. Second, in 2002, Lee and Liew proposed an additional merging process that fits in efficiently with the original algorithms [13]. In 2017, Jiang and Yang implemented Lee’s algorithm in a field programmable gate array (FPGA) for \(N = 8–32\) [14]. They found that it worked well but incurred significant overhead time. For example, the first stage of a parallel control unit (PCU) for \(N = 16\), of which the main process can be completed in only four (i.e. \(\log_2 N\)) clocks, required up to 17 clocks. In addition, this approach incurs a steep increase of PCU hardware in \(O(N^3)\) due to a crossbar-like centralized architecture [15].

In 2009, Kai et al. preceded Jiang and Yang in designing a PCU to configure BNWs. Their design only focused the first stage of the PCU for a \(16 \times 16\) BNW [16]. However, they employed a distributed architecture rather than a centralized architecture, which reduced the hardware complexity from \(O(N^2)\) to \(O(\log_2 N)^3\). In addition, they suggested that the time complexity can be reduced from \(O((\log_2 N)^2)\) to \(O(\log_2 N)\) using a pipeline architecture suggested in Ref. [12]. Despite these potential advantages, their algorithm could not handle partial permutations and no detailed experimental results were shown.

In contrast to Ref. [16], we propose a parallel algorithm that realises partial permutations as well as full permutations. We also implement a whole PCU in an FPGA and show experimental results in detail. Our approach is to construct a PCU with distributed PEs that generate SCBs in a pipelined and in part asynchronous manner. In Ref. [12], partial permutations were addressed by matching the idle inputs to idle outputs as dummy destinations; however, our algorithm does not need such overhead processing but can handle partial permutations in a unified manner with full permutations.

The remainder of the paper is organised as follows. In Section 2, beginning with several definitions, we present our routing principle in the form of sequential processing as an introduction to our parallel algorithm. In Section 3, we describe a design of parallel and distributed PEs to be implemented in an FPGA with an emphasis realising full and partial permutations in a simple but unified manner. In addition, we discuss two design options relative to speed and hardware complexity perspectives. In Section 4, we describe the implementation of the parallel algorithm in an FPGA to realise higher speed than clock rate using asynchronous operation in part. We show experiment results to highlight the performance of the design. Finally, the paper is concluded in Section 5.

## 2 Definitions and Proposed Routing Principle

Here, we consider parallel construction of BNWs [17]. Figure 1 shows a primitive model of an \(N \times N\) BNW in a three-stage structure, where a parallel pair of half-sized (i.e. \(N/2 \times N/2\)) matrices are in between two sets of \(N/2 \times 2\) SEs. Each \(N/2 \times N/2\) matrix may be replaced in a three-stage BNW with a pair of \(N/4 \times N/4\) matrices. The reduction proceeds until the minimum size matrices, that is, \(2 \times 2\) SEs, appear in the centre stage. As a result, a complete BNW can be obtained (Figure 2), where each SE has two connection modes, that is, bar (\(=\)) and cross (\(\times\), flipped by a SCB. Note that \(S_{p,q}\) represents the \(q\)-th top SE at the \(p\)-th stage from the left, where \(0 \leq p \leq 2n - 2\) and \(0 \leq q \leq N/2 - 1\). Assume \(s_{p,q} \in \{0, 1\}\) indicates its status, and \(s_{p,q}\) of 0 and 1 makes \(S_{p,q}\) bar and cross.

Let \(i\) and \(j_i\), where \(0 \leq i \leq N - 1\) and \(0 \leq j_i \leq N - 1\), be the input and output port numbers in an \(N \times N\) BNW. For now, we consider a full permutation \(\pi_0\) as follows:

\[
\pi_0 = \begin{pmatrix}
0 & 1 & \cdots & i & \cdots & N-1 \\
\end{pmatrix},
\]

(1)

where \(j_i\) is the designated output at input \(i\). For example, in Figure 1, we assume \(\pi_0\) as follows:

\[
\pi_0 = \begin{pmatrix}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
3 & 2 & 5 & 0 & 4 & 6 & 7 & 1 \\
\end{pmatrix}
\]

(2)
We express a duplet \(i:j\) as an individual corresponding pair of input \(i\) and output \(j\). Let \(b(j)\) be the binary address of output \(j\), defined by

\[
b(j) = (b_j^{(n-1)}, b_j^{(n-2)}, \ldots, b_j^0) \tag{3}\]

where \(b_j^i \in \{0, 1\}, 0 \leq i \leq n - 1, \) with

\[
b(j)_{\text{MSB}} = b_j^{(n-1)},
\]

\[
b(j)_{\text{LSB}} = b_j^0. \tag{4}\]

Each SE in the last stage in Figure 2 has a pair of outputs \(j\) and \(j_c\), of which \(b(j)_{\text{LSB}}\) and \(b(j)_{\text{LSB}}\) are different (or complementary). We define \(m(j)\) as the output SE number with \(j\), where \(0 \leq m(j) \leq N/2 - 1\), and we have

\[
m(j) = m(j_c). \tag{5}\]

In addition, relative to a binary form, let \(\overline{b(j)}\) be the truncated binary address of \(b(j)\), defined by

\[
\overline{b(j)} = (b_j^{(n-1)}, b_j^{(n-2)}, \ldots, b_j^1). \tag{6}\]

Then we have

\[
b(j) = \overline{b(j)}. \tag{7}\]

Similar relationships hold between \(i\) and \(i\), in an input SE, which has a pair of duplets \(i:j\) and \(i:j_c\), where \(j_c\) is the associated destination output in the input SE. Note that we use \(j\) and \(j_c\) (and \(i\) and \(i_c\)) interchangeably, i.e. their positions in an SE are not fixed. Here, \(j_c\) and \(j_c\) may be coincident to each other for a given \(j\). For example, in Figure 1, \(j_c = j = 2\) holds for \(j = 3\) at \(S_{0,0}\).

The original permutation \(\pi_0\) is divided into a pair of sub-permutations as follows:

\[
\pi_{0,\phi^0} = u^0_{\phi^0}, u^1_{\phi^0}, \ldots, u^0_{\phi^0} - 1
\]

\[
\pi_{0,\phi^0} = u^0_{\phi^0}, u^1_{\phi^0}, \ldots, u^0_{\phi^0} - 1 \tag{8}\]

where \(u^0_{\phi^0}\) and \(u^0_{\phi^0}\) (\(0 \leq u^0_{\phi^0} \leq N/2 - 1\) and \(0 \leq u^0_{\phi^0} \leq N/2 - 1\) for \(0 \leq l \leq N/2 - 1\)) are output SE numbers to which a designated output \(j\) belongs, as shown in Figure 1. Previously, a sub-permutation has been referred to as a complete residue system [6] or equivalent class [10]; however, sub-permutations constitute systems of distinct representative (SDR) [18, 19], and we refer to the sub-permutations as SDR.

Assume that the three-stage structure shown in Figure 1 is the 0th reduction. Generally, at the \(k\)-th reduction, with \(0 \leq k \leq n - 2\), there are \(2^k\) sub-permutation pairs \((\pi_{k,\phi^0}, \pi_{k,\phi^0})\), \ldots, \((\pi_{k,\phi^{2^k-1}}, \pi_{k,\phi^{2^k-1}})\), where each sub-permutation has \(N/2^k + 1\) elements. The width of the binary address of these elements is given virtually by \(n - k - 1\). Note that, throughout this paper, an output address of \(n\) bits is transferred to the next sub-permutation as is, while the region of interest in the address at the \(k\)-th reduction, denoted by \(R(b(j), k)\), is reduced to \(n - k - 1\) bits, truncated by one bit at each reduction, and is expressed as follows:

\[
R(b(j), k) = (b_j^{(n-1)}, b_j^{(n-2)}, \ldots, b_j^{(n-(k+1))}) \tag{9}\]

As shown in Figure 2, the BNW is equivalent to back-to-back concatenation of two baseline networks [20]. Similarly, our routing algorithm is broken down into two parts. In the first part, the permutation is divided recursively to one-half while satisfying the SDR constraints as shown in Figure 1. Although our division process is essentially the same as that of the looping algorithm [3], it is described briefly in the following as an introduction to our parallel algorithm. This process is recursive in nature; thus, we primarily focus on the first stage to illustrate how \(\pi_0\) is divided into \(\pi_{0,\phi^0}\) and \(\pi_{0,\phi^0}\) for the two cases (i) and (ii).

(i) Select an arbitrary SE that is not set yet and set it to bar by default. In the initial state, we select \(S_{0,0}\) and set \(s_{0,0} = 0\). Let the upper duplet be \(i:j\) \((0:3\) in Figure 1). Search for an SE to which \(j\) belongs (\(j = 2\) in Figure 1) and find \(j_c\) in \(S_{0,0}\). In this case, we return to the beginning SE and a cycle ends [10]. Then, a new cycle is started from \(i\). At the end of the first cycle, we have \(\pi_{0,\phi^0} = \{1\}\) and \(\pi_{0,\phi^0} = \{\}\).

(ii) This step corresponds to \(i\); however, we state the new cycle separately for convenience. Select \(S_{0,1}\) among unset SEs and set \(s_{0,1} = 0\) again to find a new upper duplet as \(2:5\) (i.e. \(j = 5\) and \(j_c = 4\)). Here, we obtain \(\pi_{1,\phi^0} = \{1, 2\}\) and \(\pi_{0,\phi^0} = \{1, 0\}\).

(ii) Search for the neighbour SE to which \(j_c = 4\) belongs and find \(S_{0,2}\). A pair of outputs \(j = 5\) and \(j_c = 4\) shares \(S_{0,2}\), and they must be distributed to different sub-permutations due to the SDR constraints. As \(m(j) = 2\) has already been included in \(\pi_{0,\phi^0}\) at \(i^\text{th}\), \(m(j_c) = 2\) must be in \(\pi_{0,\phi^0}\). As a result, we set \(s_{0,2} = 1\). Here, we have \(\pi_{0,\phi^0} = \{1, 2, 3\}\) and \(\pi_{0,\phi^0} = \{1, 0, 2\}\). Note that \(s_{0,2}\) is not equal to \(s_{0,1}\) because the relative positions of the pair of duplets including \(j = 5\) and \(j_c = 4\) (i.e. either the upper or lower input in the respective SE) are identical [10].

(iii) Next, identify \(j_c = 6\) in \(S_{0,2}\) as the associate of \(j_c = 4\). Then, substitute \(j\) with \(j_c\) and repeat steps (ii) to (iii) until the cycle ends. When the cycle ends, rename a new cycle from (i) until all SEs are set.

Finally, we have \(\pi_{0,\phi^0} = \{1, 2, 3, 0\}\) and \(\pi_{0,\phi^0} = \{1, 0, 2, 3\}\), as shown in Figure 1. Note that \(s_{0,3} = s_{0,2}\) because the relative input positions of the pair of duplets including \(j = 6\) and \(j_c = 7\) differ [10].

The second part of our routing algorithm relies on destination-tag routing (DTR), which correlates to a superposed binary tree structure of the reverse baseline network (RBN) [20].
shown in Figure 2. Here, let $d(i)$ be a binary destination output imposed at the $i$-th input of the RBN as follows.

$$d(i) = (d_i^{(n-1)}, d_i^{(n-2)}, \ldots, d_i^0),$$  \hspace{1cm} (10)$$

where $d_i^j \in \{0, 1\}, 0 \leq i \leq N - 1$ and $0 \leq i \leq n - 1$. We label each pair of outlets of SEs in the RBN with 0 and 1, as shown in Figure 2. Subsequently, we trace a route from $d(i)$ as follows. When $d(i)$ appears at an input of $S_{p_i, q_i}$, it is forwarded to either outlet 0 or 1 of the $S_{p_i, q_i}$ depending on whether $d_i^{2n-2-p_i} = 0$ or 1. This operation is realized in $O(1)$ time. The DTR is so simple and straightforward that we omit a detailed description of its hardware implementation.

3 | DESIGN OF PARALLEL PROCESSING ELEMENTS

3.1 | Design overview

The distributed architecture of parallel processing elements (PEs) is shown in Figure 3, where the upper and lower halves are the switch body of an $N \times N$ BNW and the proposed PCU. The PCU accepts address information ($A_{in}$) slot by slot, computes routes and generates SCBs stage by stage in parallel. A PE at a stage is connected to a pair of PEs at the next stage in the same manner as its counterpart SE. A PE accepts a pair of addresses and transfers the addresses to the next PEs in the same mode (i.e. bar or cross) as the SE. The PCU has up to $O(N \log_2 N)$ PEs and is similar to a massively parallel system. Fortunately, as described in Section 3.2, the PEs can comprise several kinds of primitive logic circuits and memories, for example, comparators, multiplexers and registers, unlike [14, 21], where arithmetic units and shared memories are required.

As shown in Figure 3, there are two types of PEs, that is, $\text{PE}_{k, \bar{k}}$ in the first part, where $0 \leq k \leq n - 2$ and $0 \leq b \leq N/2 - 1$, and $\text{PE}_{k', \bar{k}'}$ in the second part, where $n - 1 \leq k' \leq 2n - 2$ and $0 \leq b \leq N/2 - 1$, according to the aforementioned two parts of routing algorithm. Note that the first stage of the PCU corresponds to the index of $k = 0$. Here, PEs in the first $k$-th ($0 \leq k \leq n - 2$) stage are divided into $2^k$ independent groups. Thus, a PE in the $k$-th stage includes $N/2^k + 1$ PEs and accepts a total of $N/2^k$ addresses, of which the bit width of the ROI becomes $n - k - 1$. PEs in a group are interconnected with $N/2^k + 1$ buses, which are shown as bold lines in the first stage of the PCU in Figure 3, to communicate within the group.

In the first part of our routing algorithm, the division process of sub-permutations is common to all PEs, although the number of PEs in a group (i.e. the size of sub-permutations) decreases with stage. In the second part, all PEs have uniform, low complexity. As a result, PEs in the first stage of the PCU have the most complex hardware. As discussed in Section 3.2, the PEs in the second part require small constant time $O(1)$. As a result, the processing time and hardware complexity are most critical in the first stage of the PCU. In the following, we focus on PEs in the first stage as in Ref. [16] and describe our parallel algorithm in detail.

3.2 | Design of PEs for full permutation

In the first stage of the PCU, there is a single PE group containing $N/2$ PEs. Here, we assign binary suffixes to the $N/2$ PEs as $\text{PE}_{i, 2^k}$ (e.g. $\text{PE}_{00}$), where $0 \leq p \leq N/2 - 1$. Note that we also re-assign binary suffixes to the status of $N/2$ SEs (e.g. $s_{00}$). The PEs are interconnected with the same number of buses $B_{p, r}$ (e.g. $B_{00}$), where $0 \leq r \leq N/2 - 1$ (Figure 4(a)). We assume a full permutation in Figure 4(a) as follows:

$$\pi_0 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 5 & 2 & 0 & 6 & 3 & 4 & 1 \end{pmatrix}. \hspace{1cm} (11)$$

A simplified functional block diagram of the PEs is shown in Figure 5, and the function of each block is described as follows. In the division process, $\text{PE}_{i, 2^k}$ puts out its own data, for example, $b(i, \bar{0})$, $b(j, \bar{0})$ and $b(j, \bar{0})$, to $B_{i, 2^k}$. Here, $b(p)$ is an extended PE suffix that is identical to $b(p)$ for full permutations and is redefined for partial permutations in Section 3.3. Each bus comprises the data information of $n - 1$ bits and a data valid flag $f_{p, r}$ of 1 bit. Although each PE can receive multiple data simultaneously, at most one pair of eligible data is accepted through the multiplexers in the bus interface (BIF) in Figure 5. Thus, the internal processing speed of the PEs remains constant independent of the PE group size. The division process for full permutations has three phases, which are described in the following.

3.2.1 | First phase

The first phase is a parallel process for the neighbour search. Here, assume $\text{PE}_{i, 2^k}$ has two destination addresses $b(i, \bar{0})$ and $b(j, \bar{0})$, as shown in Figure 5. The neighbour search is performed to find PEs with truncated addresses $b(i, \bar{0})$ and $b(j, \bar{0})$. If $b(i, \bar{0}) = b(j, \bar{0})$ holds in a PE (step ii; Section 2), the neighbour PE is identical to the original PE, and no addresses are launched onto the bus.
FIGURE 4  Bus-connected PE array and its operation: (a) Each PE accepts address information and interchanges switch control data via multiple buses in parallel; (b) Link status after first phase; (c) Link diagram in a cycle; (d) Implementing link diagram in FPGA using inverting and non-inverting gates

The 'equal to' and 'not equal to' relations are illustrated in Figures 4(b,c) by solid and dashed links, respectively. Here, we refer to both relations as link status collectively. Link status flag (fj) and neighbour PE suffixes are saved in registers, as shown in Figure 5. In practice, a pair of counter-rotating links is implemented with a portion of each bus in the third phase. The link status between two adjacent PEs is implemented in our FPGA design by an inverting or non-inverting gate, as shown in Figure 4(d), where the link originates and terminates at PE11, as the result of the following second phase. Here, neighbour search is realised in parallel, and the time complexity is O(1).

3.2.2  Second phase

The second phase is an iterative procedure to find a representative PE in each cycle. In a cycle (e.g. Figure 4(c)), each PE is correlated with each other, and a system of the relations is referred to as an initialising equation and resolved in a relatively complex manner [13]. Our solution for the initialising equation is based on a plain but unique observation. If we appoint a PE as a representative in a link diagram (e.g. PE11 in Figure 4(d)) and set the corresponding SE to the default state, the status of the other SEs is generated automatically in the link diagram, which cannot be easily implemented with only software. In other words, we borrow a concept from evolving hardware/software codesign techniques [22].

We assume the representative PE is that with the maximum suffix in a cycle (e.g. PE11 in Figure 4(b)). Our iterative solution proceeds as follows. In the initial state, each PE in a cycle is a candidate for a final representative, of which status is indicated as a double circle in Figure 6(a), where only binary suffixes of PEs, that is, bh(p), are shown for simplicity. In the first step, each PE sends its suffix to two neighbour PEs, as shown in Figure 6(b), and each PE compares its own suffix y0 to the incoming suffixes y1 and y2. Note that a similar situation occurs in the subsequent steps, and there are three possible cases.

(i) If (y0 < y1) ∪ (y0 < y2), the PE will be lost, y1 and y2 are transferred to the next neighbour PEs, for example, PE00 and PE01 in Figure 6(b), and PE10 in Figure 6(c), where lost PEs are shown by dashed circles. Note that lost PEs become transparent in the subsequent steps.

(ii) If (y0 > y1) ∩ (y0 > y2), the PE remains as a potential representative and discards both y1 and y2, for example, PE11 and PE10 in Figure 6(b).

(iii) If y0 = y1 = y2, the PE becomes a final representative and goes to the third phase, for example, PE11 in Figure 6(d).

Lost PEs bypass incoming data through simple transfer switches embedded in BIF; thus, the bypass delay is negligible. The comparison result at each PE is updated in a representative flag (fr), as shown in Figure 5. Neighbour PE suffixes are referred to by the BIF to specify a bus to receive the suffix data from. Note that each individual cycle has a maximum suffix, and there can be several representatives in a stage, for example,

Otherwise, b(j) and b(j a) are broadcast in series onto Bbh(p). The other PEs compare their own truncated addresses with the incoming b(j) and b(j a) using the comparators in the BIF. For example, as shown in Figure 4(b), PE00 first broadcasts 11 from its upper inlet to B00, and the other PEs compare their truncated addresses to 11. PE10 discovers PE00 as a neighbour because it has 110 at its upper inlet. Note that 111 and 110 are at upper inlets, and they must be in different sub-permutations; therefore, s10 = to s00 (Figure 4(c)). Similarly, when PE00 launches 10 from its lower inlet, PE11 discovers PE00 as a neighbour because it has 100 at its upper inlet. Thus, s11 = to s00.

FIGURE 5  Simplified functional diagram of PE with binary suffix b(p)
FIGURE 6 Iterative steps to determine representative PE in the second phase: (a) Each PE is eligible for representative in the initial state; (b) The PE receives two suffixes and compares its suffix to them and surviving PEs are reduced to one-half and lost PEs become transparent and bypass incoming data; (c) The competition process is repeated in subsequent iterations; (d) Maximum suffixes return to the originating PE in the final iteration.

$S_{0,0}$ and $S_{0,3}$ in Figure 2. As a result, the number of candidates for a representative is reduced to one-half after each iteration; thus, the second phase for the first stage completes in at most $O(\log_2 N)$ time.

### 3.2.3 Third phase

The third phase is performed to determine the status of each SE in each cycle according to the direction of a representative and pass addresses on to the next stage. For example, PE$_{11}$ in Figure 4(d) sets an initial value of $s_{11} = 0$, which also serves as a trigger to the link diagram. Note that each link between neighbour PEs has already been established over buses in the beginning of the third phase. In addition, inverting and non-inverting gates, which reside in an SCB gen. in Figure 5, have been configured according to $f_j$. Here, a pair of input addresses $b(j)$ and $b(j_0)$ is transferred to PEs in the next stage via a $2 \times 2$ switch (Figure 5), which is set to bar or cross by an SCB. The propagation delay in the link diagram is very small compared to a clock period; thus, the time complexity of the third phase is estimated as $O(1)$.

In summary, the total processing time in the first stage is $O(\log_2 N)$. Generally, the total processing time in the $k$-th reduction, where $0 \leq k \leq n - 2$, is $O(\log_2 N/2^k)$, because the size of permutation in the $k$-th reduction is given by $N/2^k$. It is evident that the first stage has a maximum processing time, which decides the maximum throughput of the pipelined PCU.

### 3.3 Unified parallel algorithm for full and partial permutations

Here, we begin by observing how an idle connection affects our algorithm. We assume a partial permutation as follows:

$$\pi_0 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \infty & 5 & 2 & 0 & 6 & 3 & 4 & 1 \end{pmatrix},$$

where $\infty$ indicates an idle address. This partial permutation is similar to the full permutation in the previous Section, for example, Equation (11), except the first element that is changed to $\infty$. As shown in Figures 7(a,b), there are no links between PE$_{00}$ and PE$_{10}$ due to the idle address. Thus, no representative PE can be determined because $y_1$ and $y_2$ (step (iii); Section 3.2.2) are missing due to the link disconnection. In spite of the situation, we have two design options to overcome this problem. One option is a structural adaptation to loop back the disconnected links at the two end PEs (Figure 7(c)). This loop-back mechanism allows all PEs to have a pair of neighbour PEs, similar to full permutations. Although the loop-back causes no change in the comparison operation, it nearly doubles the length of a cycle and increases processing time.
The other option is a behavioural adaptation to mask the disconnection by blocking the received suffix data at the two end PEs. For this purpose, we redefine $\bar{b}(p)$ as follows: If a PE has a single neighbour PE like PE$_{010}$ and PE$_{110}$, which reside on both ends in Figure 7(b), $\bar{b}(p)$ is obtained by

$$
\bar{b}(p) \left( 1, b_{p}^{p-1}, b_{p}^{p-2}, \ldots, b_{p}^{1} \right).
$$

If a PE has two neighbour PEs like PE$_{011}$ and PE$_{111}$, which reside in between two end nodes, $\bar{b}(p)$ is obtained by

$$
\bar{b}(p) \left( 0, b_{p}^{p-1}, b_{p}^{p-2}, \ldots, b_{p}^{1} \right).
$$

It is evident from Equations (13) and (14) that the two end PEs have greater suffixes than intermediate PEs; thus, they block intermediate PE numbers and survive competitions to the final iteration. Eventually, they exchange suffixes over the available links, and either becomes a final representative. Here, we must add a pair of termination conditions to step (iii) (Section 3.2.2) as follows. Note that $\bar{b}(y)_{\text{MSB}} = 1$ indicates that PE$_{\bar{y}}$ is an end PE.

(i) If $(\bar{b}(y)_{\text{MSB}} = 1) \cap (y_{0} < y_{1})$, one end PE becomes lost and halts, for example, PE$_{100}$ in Figure 7(d).

(ii) If $(\bar{b}(y)_{\text{MSB}} = 1) \cap (y_{0} > y_{1})$, the other end PE becomes a final representative and goes to the third phase, for example, PE$_{110}$ in Figure 7(d).

Note that the latter design incurs very little additional processing time and hardware while maintaining the comparison operation for full permutation. In this study, we developed a PCU using the latter design option to realising high speed and low hardware complexity.

4 | FPGA DESIGN AND EXPERIMENTAL RESULTS

4.1 | Design environment and alternatives

FPGAs are an efficient hardware target for rapid prototyping [23]. In this study, we used a Xilinx midrange FPGA (XC6SLX45), which has 6,822 configurable logic blocks (CLBs) operating at 100 MHz. We also used the ISE Design Suite 14.7 development tool for Windows 10, where the code for PEs was written in VHDL with the IEEE Std_Logic_1164ALL package. Note that the synthesis options were set to defaults, that is, speed priority mode, normal optimisation effort etc. Here, a constraint file was used to allow internal signals, for example, counter outputs, $f_{i}$ and SCB, to be output to I/O pins to monitor their logic status and measure delay time. In addition, the input addresses were generated within the FPGA. Note that we needed to redesign the FPGA when the permutations changed; however, external digital pattern generators were not required. We referred to a device utilisation summary to analyse hardware utilisation, for example we estimated the total hardware amount according to the number of occupied slices in the summary report.

Note that each PE in a given stage operates synchronously to a common clock. In our algorithm, three processing steps operate synchronously. The first step is an initializing process, where destination addresses are imported and flags and registers are reset. The second step is composed of the three phases described in Section 3. The third step is a terminating process, where SCBs and addresses are exported. However, to increase processing speed, we introduced asynchronous hardware operation in part as follows, while we made no change to the parallel algorithm described in Section 3. In the first part of the PCU, the iterative procedure to find a representative (Section 3.2), can operate asynchronously by referring to the $f_{i}$, flags and comparison results. Here, we added an additional set of buses in Figure 4(a) for this purpose, and each pair of input addresses at PEs was processed simultaneously. In the second part of the PCU, DTR was also implemented asynchronously by referring to a specified routing bit given by Equation (10). Note that these asynchronous operations were described easily in VHDL code using if-statements, and the synchronous operations were realised using when-statements.

4.2 | Experimental results and discussion

Figure 8 shows the number of occupied slices in the first and second parts of the PCU for $N = 4$ to 32, denoted by HW$_{1}$ and HW$_{2}$, respectively. As expected in Section 2, the first part accounts for a large portion of the PCU hardware, whereas the second part requires a significantly less amount of hardware than the first part. We estimate the hardware amount of a single PE in the first part (Figure 5) as $O(\log_{2}N)$ because most parts of a PE have a dimension of $\log_{2}N$. Recall that a single stage comprises $N/2$ PEs; thus, the hardware complexity per stage is $O(N\log_{2}N)$. Therefore, the total hardware complexity over $\log_{2}N - 1$ stages, denoted by HW$_{1}$, is $O(N(\log_{2}N)^{2})$. Similarly, the total hardware complexity of the second part, denoted by HW$_{2}$, is also given by $O(N(\log_{2}N)^{2})$. As shown in
### Conclusion

In this paper, we have proposed a new design for parallel and distributed PEs to set up partial permutations in Beneš networks based on a novel parallel algorithm. Our algorithm realises both full and partial permutations in a unified manner with little overhead time and additional hardware costs. Specifically, the proposed design reduces hardware complexity from \( O(N^3) \) to \( O(N \log_2 N)^2 \) due to its distributed architecture and reduces time complexity from \( O((\log_2 N)^2) \) to \( O(\log_2 N) \) due to its pipeline architecture. In addition, we introduced in part asynchronous operation to further increase speed, and the time complexity is reduced to \( O(\log_2 N) \). However, both processes of \( O(\log_2 N) \) complexity were reduced to \( O(\log_2 N^{0.16}) \) due to the asynchronous operation, as predicted in Section 4.1.

Figure 10 shows the total processing time in clock cycles for the first stage. The conventional algorithm [15] for the first stage requires \( O(\log_2 N) \) clock times, while the proposed design demonstrated \( O(1) \) clock time until \( N = 128 \), because the most time-consuming process for the proposed algorithm completes within a single clock cycle as shown in Figure 9. In fact, the proposed algorithm requires five clocks, whose breakdown is as follows: one clock for the initializing step, three clocks for the first to third phases, and the last clock for the terminating step described in Section 4.1. For example, in the \( N = 16 \) case, our design is more than three times faster than the previous method, and the performance difference increases with increasing \( N \). Note that the number of clock cycles for \( N > 128 \) is estimated to increase by only one clock cycle due to the asynchronous operation. Consequently, the number of clock cycles of the proposed design is at least several times faster than the previous method [15].
to 32. The experiments results demonstrate that the proposed design outperforms a conventional method by at least several times in terms of hardware and processing time complexities. We expect that the proposed algorithm and design will extend the application areas of Beneš networks and stimulate parallel processing techniques implemented using this hardware.

ORCID
Labson Koloko https://orcid.org/0000-0001-9081-5729

REFERENCES
1. El-Bawab, T.S.: Optical switching (Springer, New York, NY 10013, USA 2010), Chapter 9 written by D.K. Hunter, pp. 275–306 https://www.springer.com/gp/book/9780387261416
2. Waksman, A.: A permutation network. J. ACM. 15(1), 159—163 (1968)
3. Oh, D., Parhi, K.K.: Low-complexity switch network for reconfigurable LDPC decoders. IEEE Trans. Very Large Scale Integr. Syst. 18(1), 85–94 (2010)
4. Antonescu, M., Ştefan, G.M.: Multi-function scan circuit. International Semiconductor Conference (CAS), Sinaia, Romania, pp. 123–126 (2020)
5. Opferman, D.C., Tsao-wu, N.T.: On a class of rearrangeable switching networks part I: Control algorithm. Bell Syst. Tech. J. 50(5), 1579–1600 (1971)
6. Lee, K.Y.: A new Benes network control algorithm. IEEE Trans. Comput. C-36(6), 768–772 (1987)
7. Aweya, J.: Switch/Router Architectures: Systems with Crossbar Switch Fabrics. CRC Press, Boca Raton, Florida, Chapter 2, pp. 47–108 (2019)
8. Lu, E., Zheng, S.Q.: Parallel routing algorithms for nonblocking electronic and photonic switching networks. IEEE Trans. Parallel Distrib. Syst. 16(8), 702–713 (2005)
9. Lev, G.F., Pippenger, N., Valiant, L.G.: A fast parallel algorithm for routing in permutation networks. IEEE Trans. Comput. C-30(2), 93–100 (1981)
10. Nassimi, D., Sahni, S.: Parallel Algorithms to set up the Benes permutation network. IEEE Trans. Comput. C-31(2), 148–153 (1982)
11. Jain, T., Schneider, K.: Routing partial permutations in interconnection networks based on radix sorting. Proceedings of 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip, Lille, France, pp. 1–10 (2018)
12. Lee, C.-Y., Oruc, A.Y.: A fast parallel algorithm for routing unicast assignments in Benes networks. IEEE Trans. Parallel Distrib. Syst. 6(3), 329–334 (1995)
13. Lee, T.T., Liew, S.Y.: Parallel routing algorithms in Benes-Clos networks. IEEE Trans. Commun. 50(11), 1841–1847 (2002)
14. Jiang, Y., Yang, M.: Hardware implementation of parallel algorithm for setting up Benes networks. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and applications, Las Vegas pp. 10–16 (2016)
15. Jiang, Y., Yang, M.: Hardware design of parallel switch setting algorithm for Benes networks. Int. J. High Perform. Syst. Archit. 7(1), 26–40 (2017)
16. Kai, Y., Hamada, K., Miao, Y., Obara, H.: Design of partially-asynchronous parallel processing elements for setting up Benes networks in O(log2N) time. Proceedings of the International Conference on Photonics in Switching, Pisa, Italy pp. 1–2 (2009)
17. Obara, H.: Cascaded versus parallel architectures of two-stage optical crossbar switches with an extra set of inputs and outputs. IET Optoelectron 12(4), 196–201 (2018)
18. Hall, P.: On representatives of subsets. J. London Math. Soc. 10, 26–30 (1935)
19. Hwang, F.K.: The Mathematical Theory of Nonblocking Switching Networks, 2nd ed. World Scientific, Singapore, p. 67 (2004)
20. Wu, C.-L., Feng, T.-Y.: On a class of multistage interconnection networks. IEEE Trans. Comput. C-29(8), 694–702 (1980)
21. Zheng, S.Q., Gumaste, A., Lu, E.: A practical fast parallel routing architecture for Clos networks. Symposium on Architecture for Networking and Communications Systems, San Jose, CA, pp. 21–30 (2006)
22. Teich, J.: Hardware/software codesign: The past, the present, and predicting the future. Proc. IEEE 100(5), 1411–1430 (2012)
23. Tan, X., et al.: OpenFPGA: An open-source framework for agile prototyping customizable FPGAs. IEEE Micro 40(4), 41–48 (2020)
24. XILINX: Spartan-6 Family Overview. https://www.xilinx.com/support/documentation/data_sheets/ds160.pdf, (2020). accessed December 2020

How to cite this article: Koloko, L., Matsumoto, T., Obara, H.: Design and implementation of fast and hardware-efficient parallel processing elements to set full and partial permutations in Beneš networks. J. Eng. 312–320 (2021). https://doi.org/10.1049/tje2.12037