





## Article

# Beneš Network Based Efficient Data Concentrator for Triggerless Data Acquisition Systems

Marek Gumiński <sup>1,\*</sup> , Michał Kruszewski <sup>1</sup> , Bartosz M. Zabołotny <sup>2</sup> , and Wojciech M. Zabołotny <sup>1,\*\*</sup> ,

<sup>1</sup> Warsaw University of Technology, Faculty of Electronics and Information Technology, Institute of Electronic Systems, Nowowiejska 15/19, 00-650 Warszawa, Poland

<sup>2</sup> Warsaw University of Technology, Faculty of Electronics and Information Technology, Institute of Telecommunications, Nowowiejska 15/19, 00-650 Warszawa, Poland

\*Correspondence: marek.guminski@pw.edu.pl (M.G.); wojciech.zabolotny@pw.edu.pl (W.M.Z.)

**Abstract:** The concentration of data from multiple links to a single output is an essential task performed by High-Energy Physics (HEP) Data Acquisition Systems (DAQs). At high and varying data rates combined with the large width of the concentrator's output interface, this task is non-trivial. This paper presents a concentrator based on the Beneš network, which provides efficient concentration without using a high-frequency clock internally. It warrants that empty data are eliminated and does not disturb the data time-ordering if the data rates significantly differ between inputs. Additionally, it is well suited to FPGA implementation. It is based on simple data-routing primitives and may be fully pipelined.

**Keywords:** FPGA; DAQ; Data concentration; Beneš network

## 1. Introduction

Most detectors in high-energy physics (HEP) experiments deliver massive data streams in multiple channels. Reception of this data and its delivery to the analyzing computers is the task of the readout chains. In the case of experiments using a trigger, the data must be processed locally to elaborate the level one (L1) trigger decision. Preparing it for concentration may be a side effect of this process. The data is zero suppressed – invalid or empty data words are removed from the stream. Finally, the data is buffered in memory, which may offer the data width conversion. It may be written with single data words and read with multiple words in parallel, as required by the DAQ interface.

In the triggerless readout, the situation is different. The readout system does not need to perform complex local processing of data. Extraction of interesting events is done in further stages of the DAQ ("event builder" and "event filter" [1] or "event selector" [2]). The responsibility of the readout system is different in this configuration. It should almost transparently deliver the detector data to DAQ. For triggerless DAQ, a very popular architecture is the one where the DAQ computers are located near the detectors, and data concentrating boards are the PCIe cards. The PCI Express blocks in FPGA require specific data bus width in the AXI interface. Table 1 shows the available AXI data widths depending on the speed of the link, the width of the PCIe lane, and AXI clock frequency. For PCIe 8xGen3, it is necessary to work with 256-bits wide data.

**Table 1.** Width and clock frequency of AXI interface for PCI Express blocks. Results obtained from various configurations of AMD/Xilinx DMA/Bridge Subsystem for PCIe Express (4.1).

Lane width	Maximum link speed		
	2.5 GT/s (Gen 1)	5 GT/s (Gen 2)	8 GT/s (Gen 3)
1	64 bits @ 62.5 MHz	64 bits @ 62.5 MHz	64 bits @ 125 MHz
	64 bits @ 125 MHz	64 bits @ 125 MHz	64 bits @ 250 MHz
	64 bits @ 250 MHz	64 bits @ 250 MHz	
2	64 bits @ 62.5 MHz	64 bits @ 125 MHz	64 bits @ 250 MHz
	64 bits @ 125 MHz	64 bits @ 250 MHz	128 bits @ 125 MHz
	64 bits @ 250 MHz		
4	64 bits @ 125 MHz	128 bits @ 125 MHz	128 bits @ 250 MHz
	64 bits @ 250 MHz	64 bits @ 250 MHz	256 bits @ 125 MHz
8	128 bits @ 125 MHz	256 bits @ 125 MHz	256 bits @ 250 MHz
	64 bits @ 250 MHz	128 bits @ 250 MHz	
16	128 bits @ 250 MHz	256 bits @ 250 MHz	512 bits @ 250 MHz

The detector data generated by the particle detection is usually short. For example, for the STS-XYTER (also known as SMX) [3]) the data is 24-bit long. In the concentrated stream, the data must be accompanied by metadata describing its origin, resulting in a final size of 32 bits. Hence, the data concentrator must efficiently pack 32-bit detector data into 256-bit PCIe data.

The problem may be generalized as described in the next section.

### 1.1. Formulation of the problem

The system receives the data words from  $N$  inputs at frequency  $f_{in}$ , and puts them into the records able to store  $M$  words, which are read at  $f_{out}$  frequency. The system has sufficient bandwidth. The following condition is met:

$$N \cdot f_{in} \leq M \cdot f_{out}$$

The intensity of the data stream delivered by the inputs may be different and may vary in time. It means that some links may deliver invalid (or empty) data in a particular period of the input clock. Those invalid data should be skipped not creating “holes” in the output stream. For event reconstruction in the triggerless DAQ, the data must be assigned to a particular time period<sup>1</sup>. Therefore, an essential requirement is that the concentrator disturbs the time-ordering of the input data as little as possible.

In the next chapter, the existing solutions to the concentration problem are presented, and their disadvantages are discussed.

## 2. Existing solutions for concentrators

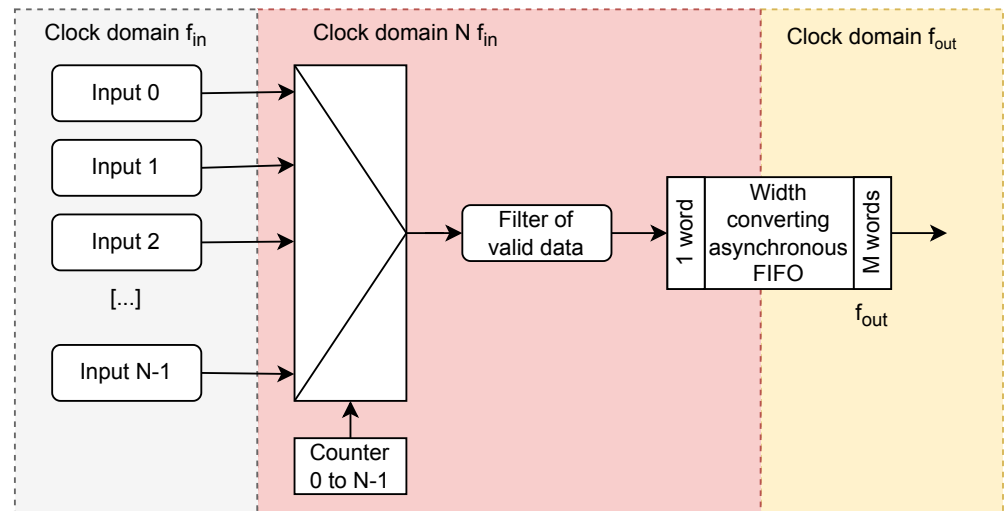
Unfortunately, finding published information about the data concentration methods used in existing FPGA-based data concentrators is difficult. The presented review is based on seldom publications and presentations, publicly available source code analysis, or the authors’ experience.

Most existing solutions may be grouped into two categories described in the following sections.

### 2.1. High speed polling

The trivial solution is based on browsing all inputs at the frequency  $f_{scan} = N \cdot f_{in}$  (see Figure 1). If valid data is found, it is copied to the asymmetric FIFO queue with an

<sup>1</sup> Of course, certain tolerance is unavoidable. Thence, some overlap between consecutive analysis periods is used.



**Figure 1.** Structure of the concentrator based on high-speed polling. The central area must work with clock frequency  $N \cdot f_{in}$ , which may be too high for FPGA.

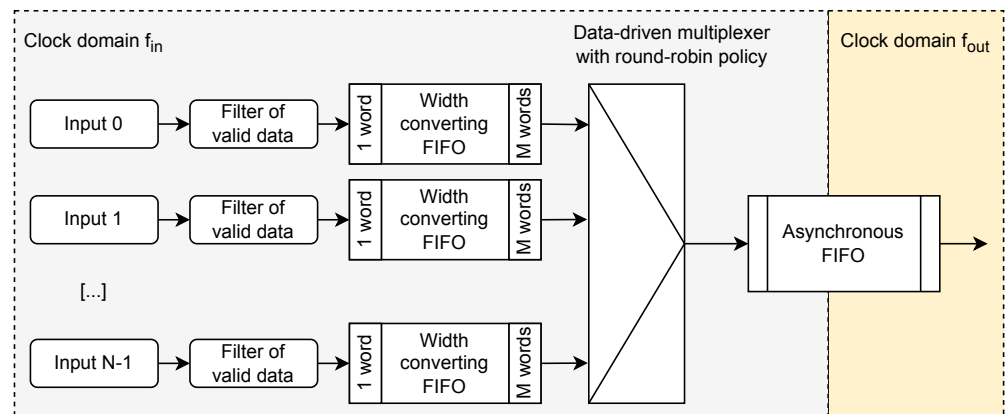
input width of 1 word and an output width of  $M$  words. That warrants that no valid data is skipped, and the output words are filled with valid data only. The only problem with that solution is that the necessary  $f_{scan}$  may be too high to be acceptable for FIFO in FPGA. Therefore this method may be used only in case of low input clock frequency or concentrating data from a small number of inputs.

This solution is used for concentrating the data from STS-XYTER2 [3] front-end ASICs transmitted through GBT Links [4] in the readout chain of the STS detector in the CBM experiment [5]. In that readout, each GBT Link transports data from 14 E-Links working with a 320 Mb/s rate. The 8b/10b encoded hit data occupy 30 bits in the E-Link. Therefore, the hit data rate in the individual E-Link is not higher than  $\frac{320 \text{ Mb/s}}{30 \text{ bits}} \approx 10.67 \text{ MHz}$ . That is a low frequency. Because  $14 \cdot 10.67 \text{ MHz} < 160 \text{ MHz}$ , the data can be safely concentrated by consecutive scanning all 14 E-Link outputs at 160 MHz.

Another example is the firmware for the CRU board [6] used by the ALICE experiment at LHC at CERN. It uses a round-robin scanning of the output of FEE links (see [6], figure 8). Then only the valid data are packed into the 256-bit wide FIFO, delivering the data to PCIe. The authors do not describe at which frequency the inputs are scanned.

## 2.2. Width conversion in input channels

A high-speed scanning may be avoided if width conversion is performed in the input channels. In this solution, the small FIFOs with one-word wide input and  $M$ -words wide output are placed in each input channel, as shown in Figure 2. This solution does not



**Figure 2.** Structure of the concentrator with width conversion in each input link.

require using a very high clock frequency. However, it has other disadvantages. If the link occupancy significantly differs between inputs, the concentration may significantly change the time-ordering of data. The data from low-rate links may get significantly delayed until  $M$  data words are collected. That problem may be solved by introducing the timeout, after which the non-empty FIFO outputs its content even if it contains less than  $M$  words. However, that modification results in inserting “holes” into the concentrated stream. Another disadvantage is the necessity to use a separate width-converting FIFO in each input channel. Those FIFOs may have limited depth, enabling implementation based on distributed RAM, but they may still increase resource consumption. Finally, that solution cannot use a simple counter-driven multiplexer periodically browsing the data. That solution requires a more complex data-driven multiplexer, which automatically selects the first input providing the complete data record after the previously serviced one (i.e., it implements the round-robin policy).

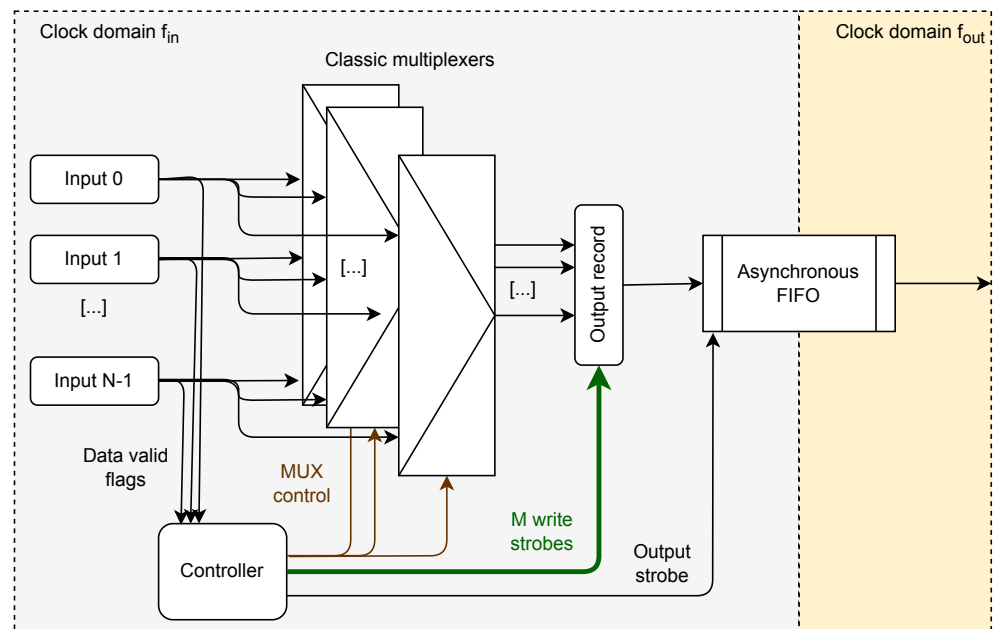
This approach seems to be used in the firmware for the FELIX board in the ATLAS experiment readout at CERN [7]. Unfortunately, the operation of the data concentrator in the FELIX firmware is not described in detail in any paper. However, the sources of that firmware are publicly available, enabling analysis of the concentrator code. The concentration is done in the CRTtoHostdm module [8] containing the asymmetric FIFO responsible for concatenating a few words and width conversion. Outputs of FIFOs from multiple channels are scanned in the CRTtoHost module [9].

### 2.3. Need for another concentration method

None of the above methods matches all the requirements described in section 1.1. Therefore, a new method is proposed in the next section.

### 3. Proposed solution - concentration with the direct routing of data

It is possible to avoid the disadvantages of the previously described solutions by directly routing data from inputs to the proper position of the output record. Such a solution is shown in Figure 3. The key functionality needed in this method is a capability

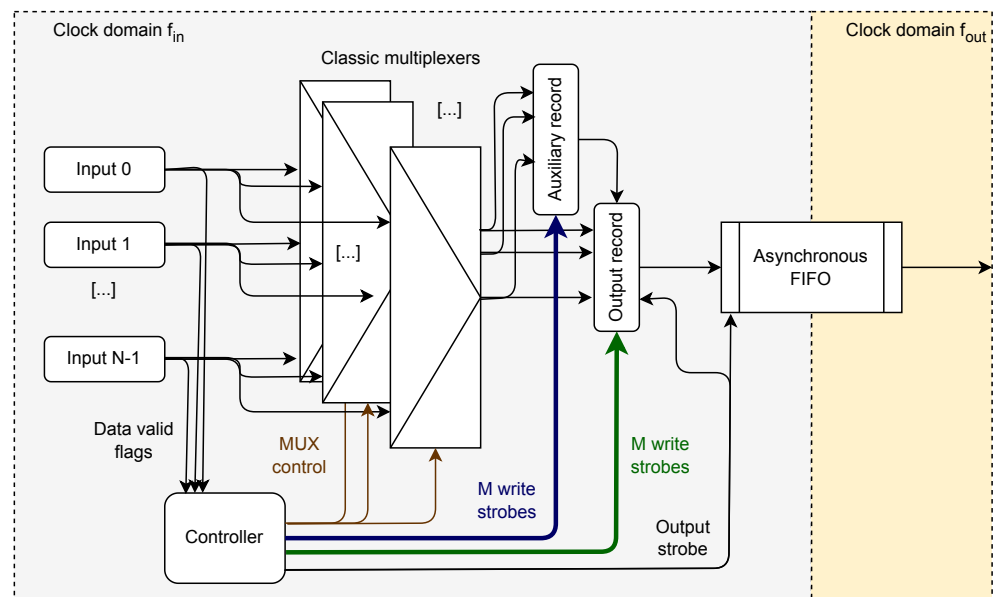


**Figure 3.** Structure of the concentrator with the direct routing of data. The controller keeps track of the current output record occupancy and routes each valid input word to the right position in the output record. When the word is completely filled, the output strobe is generated.

to write the data from each individual input to a selected position in the output record. A dedicated controller calculates the desired location of data from each input. The controller

must keep track of the occupancy of the output data. Additionally, it receives the “valid” flags from the input words. The controller starts with an output record occupancy equal to zero. If, for example, it receives valid words in three inputs, it routes them to positions 0, 1, and 2 in the output record and changes the occupancy to 3. The next valid word will be routed to position 4, and so on. When the output record is filled, the output strobe is generated, the collected words are transferred to the output FIFO, and the occupancy is set to 0.

There is, however, a problem if the concentrator receives more valid data than needed to fill the output record. Those superfluous words must be stored somewhere. For that purpose, an “auxiliary record” register is introduced. The controller generates a write strobe for both registers. The output strobe causes the transfer of the output record to FIFO and, at the same time, of the auxiliary record to the output record. The modified concentrator is shown in Figure 4.



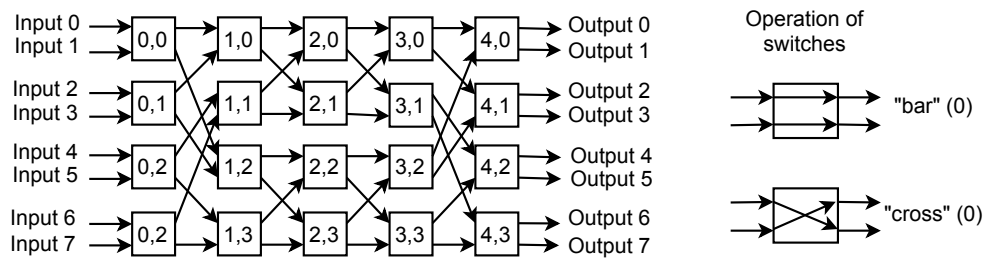
**Figure 4.** Structure of the concentrator with the direct routing of data and added auxiliary record. The controller keeps track of the current output record occupancy and routes each valid input word to the right position in the output record. When the word is completely filled, the output strobe is generated. If the number of valid words is higher than the number of empty positions in the output record, those extra words are stored in the auxiliary record. When the output strobe is generated, the content of this record is moved to the output record.

The presented concentrator should work correctly, but its implementation in FPGA is inefficient. Implementing  $M$  multiplexers routing the data words consumes many resources and generates long critical paths in the FPGA. Therefore yet another modification is needed. The multiplexers must be replaced with more efficient blocks for routing the data.

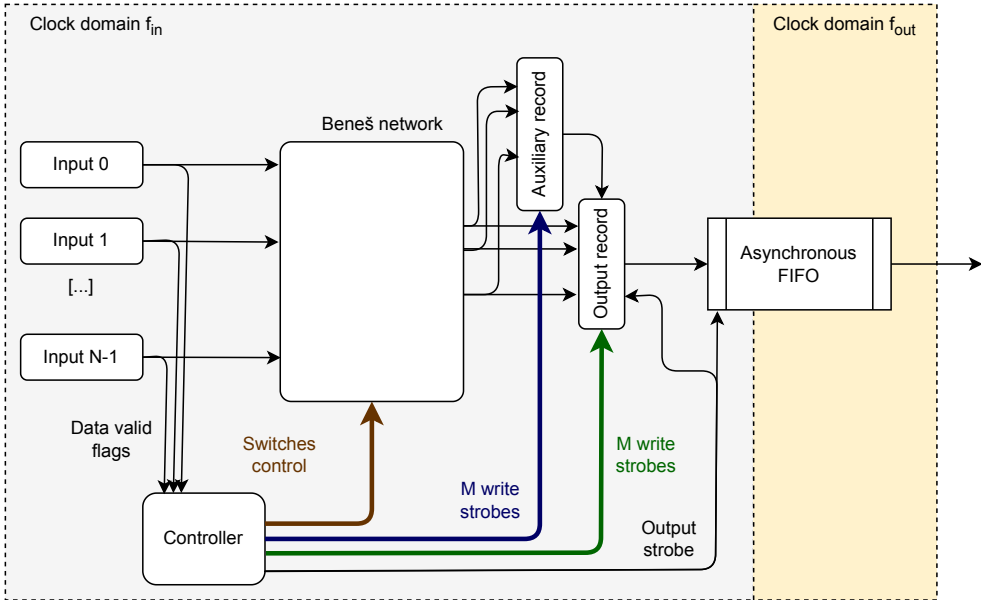
### 3.1. Concentrator based on Beneš network

A similar problem had to be solved in telecommunication networks for routing connections. Networks enabling arbitrary data permutation between their inputs and outputs are known as Beneš networks and have been described in [10]. An example of such network routing 8 inputs to 8 outputs is shown in Figure 5. The Beneš network uses simple switches with two inputs and two outputs, transmitting the data transparently or swapping them. They may be efficiently implemented in FPGA. The lengths of all data paths are the same, so this network can be efficiently pipelined, which results in a short critical path. The general scheme of the concentrator based on the Beneš network is shown in Figure 6.

The problem with the Beneš network is that its complexity quickly grows when the number of inputs and outputs increases. For example, the 4x4 Beneš network requires



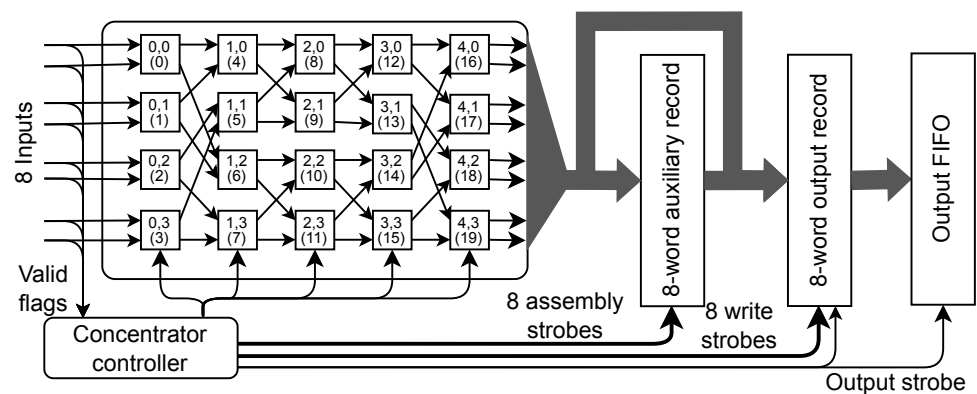
**Figure 5.** The Beneš network able to perform any permutation of 8 inputs to 8 outputs [11].



**Figure 6.** General structure of the concentrator based on the Beneš network.

6 switches in 3 layers, the 8x8 Beneš network – 20 switches in 5 layers, the 16x16 Beneš network – 56 switches in 7 layers. Generally, for  $2^N$  inputs and outputs, the network requires  $2^{N-1} \cdot (2 \cdot N - 1)$  switches in  $2 \cdot N - 1$  layers.

Additionally, finding the configuration of switches that provides the required data routing is a complex task [11]. For small networks, it is possible to use a “brute force” approach to check all possible configurations and create a table with configurations needed for all possible routings. For an 8x8 network, it is necessary to analyze  $2^{20} = 1048576$  possibilities, and find the right configuration for  $8! = 40320$  possible permutations. For a 16x16 network, the number of possible switch configurations is  $2^{56} \approx 7.2 \cdot 10^{16}$ , and the number of possible permutations is  $16! \approx 2.1 \cdot 10^{13}$ . Therefore, neither analysis of all possible configurations nor storing the right configuration for each possible permutation is viable. Therefore, an 8x8 network is used as a basis for the concentrator with the structure shown in Figure 7.



**Figure 7.** Data concentrator based on 8x8 Beneš network.

A simple C utility was written to investigate the switch settings providing different data permutations, as shown in Listing 1. It generates a simple file with lines containing the value of the switch configuration word and the data permutation it generates. This generated file is then read by the Python utility, which creates a dictionary where the key is the particular permutation, and the value is the smallest value of the switch configuration word that provides it. Of course, not all possible permutations are needed in the concentrator. What is needed is routing the valid input words to the auxiliary or output record, starting from the first free output position, and preserving their order. Therefore, the Python utility iterates over all possible occupancies of the output record, and all possible combinations of the input data valid flags and finds the corresponding switch configuration. The analysis of the generated dictionary resulted in an interesting finding. The switch configuration values generating all needed permutations are always below 0x400. It means that only the lowest 11 bits are used. The switches in layers 3 to 5 are never used. That means that the Beneš network may be significantly reduced for that particular application using a limited set of permutations, as shown in Figure 8. Of course, that also resulted in a reduction of the C utility (see Listing 2).

### 3.2. Calculation of the future occupancy

The concentrator controller is also responsible for the calculation of the future occupancy of the output record. That value is needed in the next clock period. Therefore, its calculation is separated from finding the switch configuration. The switch configuration values may be stored in a BRAM-based look-up table. Adding pipeline registers before the Beneš network may compensate for the resulting delay. That's not possible in case of future occupancy. That must be calculated in combinational logic. The implementation used in the project is shown in Listing 3.



```

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

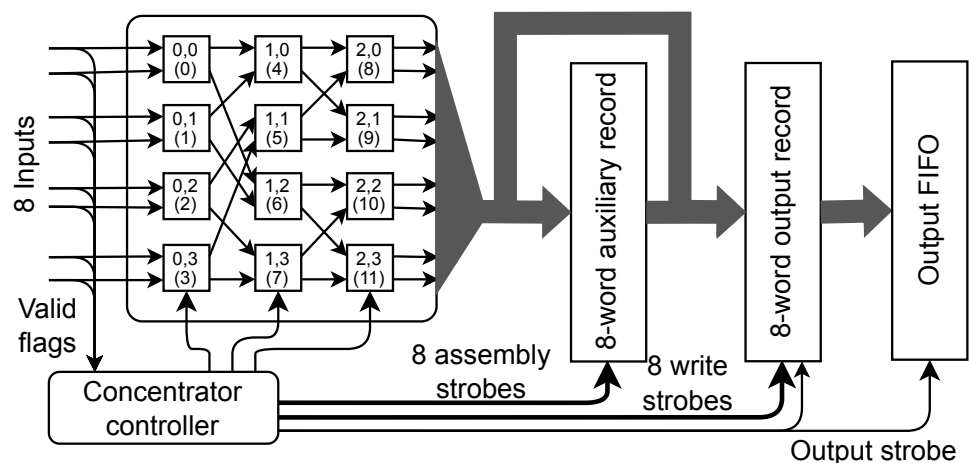
typedef uint8_t t_data;

inline void swap(t_data * i1, t_data * i2,
                t_data * o1, t_data * o2, uint32_t sw)
{
    if(sw) {
        *o1 = *i2;
        *o2 = *i1;
    } else {
        *o1 = *i1;
        *o2 = *i2;
    }
}

int main(int argc, char * argv[]) {
    uint32_t isw = 0;
    uint8_t sw[5][4];
    FILE * fout=fopen(argv[1],"wt");
    for (isw = 0; isw < (1<<20); isw++) {
        //Define the layers
        t_data l0[8],l1[8],l2[8],l3[8], l4[8], l5[8];
        //Initialize l0
        for(int i=0; i<8; i++) l0[i]=i;
        //1st layer of switches
        swap(&l0[0],&l0[1],&l1[0],&l1[4],isw & (1<<0));
        swap(&l0[2],&l0[3],&l1[1],&l1[5],isw & (1<<1));
        swap(&l0[4],&l0[5],&l1[2],&l1[6],isw & (1<<2));
        swap(&l0[6],&l0[7],&l1[3],&l1[7],isw & (1<<3));
        //2nd layer of switches
        swap(&l1[0],&l1[1],&l2[0],&l2[2],isw & (1<<4));
        swap(&l1[2],&l1[3],&l2[1],&l2[3],isw & (1<<5));
        swap(&l1[4],&l1[5],&l2[4],&l2[6],isw & (1<<6));
        swap(&l1[6],&l1[7],&l2[5],&l2[7],isw & (1<<7));
        //3rd layer of switches
        swap(&l2[0],&l2[1],&l3[0],&l3[2],isw & (1<<8));
        swap(&l2[2],&l2[3],&l3[1],&l3[3],isw & (1<<9));
        swap(&l2[4],&l2[5],&l3[4],&l3[6],isw & (1<<10));
        swap(&l2[6],&l2[7],&l3[5],&l3[7],isw & (1<<11));
        //4th layer of switches
        swap(&l3[0],&l3[1],&l4[0],&l4[2],isw & (1<<12));
        swap(&l3[2],&l3[3],&l4[1],&l4[6],isw & (1<<13));
        swap(&l3[4],&l3[5],&l4[1],&l4[3],isw & (1<<14));
        swap(&l3[6],&l3[7],&l4[5],&l4[7],isw & (1<<15));
        //5th layer of switches
        swap(&l4[0],&l4[1],&l5[0],&l5[1],isw & (1<<16));
        swap(&l4[2],&l4[3],&l5[2],&l5[3],isw & (1<<17));
        swap(&l4[4],&l4[5],&l5[4],&l5[5],isw & (1<<18));
        swap(&l4[6],&l4[7],&l5[6],&l5[7],isw & (1<<19));
        fprintf(fout, "%8.8x:%a%a%a%a%a%a%a%a\n", isw,
            (int)l5[0], (int)l5[1], (int)l5[2], (int)l5[3],
            (int)l5[4], (int)l5[5], (int)l5[6], (int)l5[7]);
    }
    fclose(fout);
    return 0;
}

```

Listing 1. C model of the 8x8 Beneš network



**Figure 8.** Data concentrator based on a reduced 8x8 Beneš network. Not all data permutations are needed to solve the concentration problem. Only three layers appeared to be sufficient for that purpose.



---

```

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

typedef uint8_t t_data;

inline void swap(t_data * i1, t_data * i2,
                t_data * o1, t_data * o2, uint32_t sw)
{
    if(sw) {
        *o1 = *i2;
        *o2 = *i1;
    } else {
        *o1 = *i1;
        *o2 = *i2;
    }
}

int main(int argc, char * argv[]) {
    uint32_t isw = 0;
    FILE * fout=fopen(argv[1],"wt");
    for (isw = 0; isw < (1<<12); isw++) {
        //Define the layers
        t_data l0[8],l1[8],l2[8],l3[8], l4[8], l5[8];
        //Initialize them to "unknown" value: 9
        for(int i=0; i<8; i++)
            l0[i]=l1[i]=l2[i]=l3[i]=l4[i]=l5[i]=9;
        for(int i=0; i<8; i++) l0[i]=i;
        //1st layer of switches
        swap(&l0[0],&l0[1],&l1[0],&l1[4],isw & (1<<0));
        swap(&l0[2],&l0[3],&l1[1],&l1[5],isw & (1<<1));
        swap(&l0[4],&l0[5],&l1[2],&l1[6],isw & (1<<2));
        swap(&l0[6],&l0[7],&l1[3],&l1[7],isw & (1<<3));
        //2nd layer of switches
        swap(&l1[0],&l1[1],&l2[0],&l2[2],isw & (1<<4));
        swap(&l1[2],&l1[3],&l2[1],&l2[3],isw & (1<<5));
        swap(&l1[4],&l1[5],&l2[4],&l2[6],isw & (1<<6));
        swap(&l1[6],&l1[7],&l2[5],&l2[7],isw & (1<<7));
        //3rd layer of switches
        swap(&l2[0],&l2[1],&l3[0],&l3[4],isw & (1<<8));
        swap(&l2[2],&l2[3],&l3[2],&l3[6],isw & (1<<9));
        swap(&l2[4],&l2[5],&l3[1],&l3[5],isw & (1<<10));
        swap(&l2[6],&l2[7],&l3[3],&l3[7],isw & (1<<11));

        fprintf(fout,"%8.8x:%d%d%d%d%d%d%d\n",isw,
                (int)l3[0],(int)l3[1],(int)l3[2],(int)l3[3],
                (int)l3[4],(int)l3[5],(int)l3[6],(int)l3[7]);
    }
    fclose(fout);
    return 0;
}

```

---

Listing 2. C model of the reduced 8x8 Beneš network

---

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library work;

entity calculate_occupancy is
    generic (
        NOF_IN_WORDS : integer range 1 to 8 := 6);
    port (
        cur_occupancy : in integer range 0 to 7;
        valid          : in std_logic_vector
                        (NOF_IN_WORDS-1 downto 0);
        next_occupancy : out integer range 0 to 7
    );
end entity calculate_occupancy;

architecture rtl of calculate_occupancy is
begin -- architecture rtl

    calc : process (cur_occupancy, valid) is
        variable tmp : integer;
    begin -- process calc
        tmp := 0;
        for i in 0 to NOF_IN_WORDS-1 loop
            if valid(i) = '1' then
                tmp := tmp+1;
            end if;
        end loop; -- i
        tmp := tmp + cur_occupancy;
        if tmp > 7 then
            tmp := tmp - 8;
        end if;
        next_occupancy <= tmp;
    end process calc;
end architecture rtl;

```

---

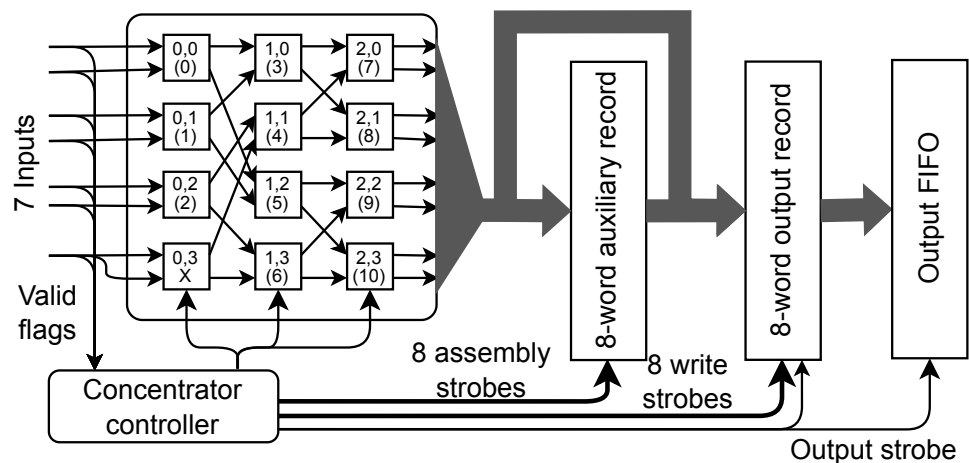
Listing 3. VHDL implementation of the combinational function calculating the future output record occupancy from the current one and the vector of valid input flags.

#### 4. Practical implementations of the concentrator

Beneš-network-based concentrator appeared to be useful in different data acquisition systems currently developed. The design may be adjusted to particular needs, as shown in this section.

The solution based on 8x8 Beneš networks is needed for the GERI board [12] based on Trenz TEC0330 PCIe card [13]. This board, when supplemented with an FMC card with 8 SFP+ cages (e.g., [14]), enables concentration of data from 8 GBT Links to the DMA system [15] connected to the 8xGen 3 PCIe bus. The DMA system uses 256-bit data which may be treated as a record containing eight 32-bit words. Thence, the solution described in the previous section may be directly applied.

If the GERI board is connected to the TFC system [16], one SFP+ cage is used for the TFC communication. In that case, a smaller 7x8 Beneš network is needed. It may be obtained from an 8x8 network. The 7th input should be connected to both inputs in the last switch in layer 0. That eliminates a need to control that switch. Its control input may be connected to a constant value. As a result, the number of switches to be controlled is reduced from 12 to 11. That configuration is shown in Figure 9.



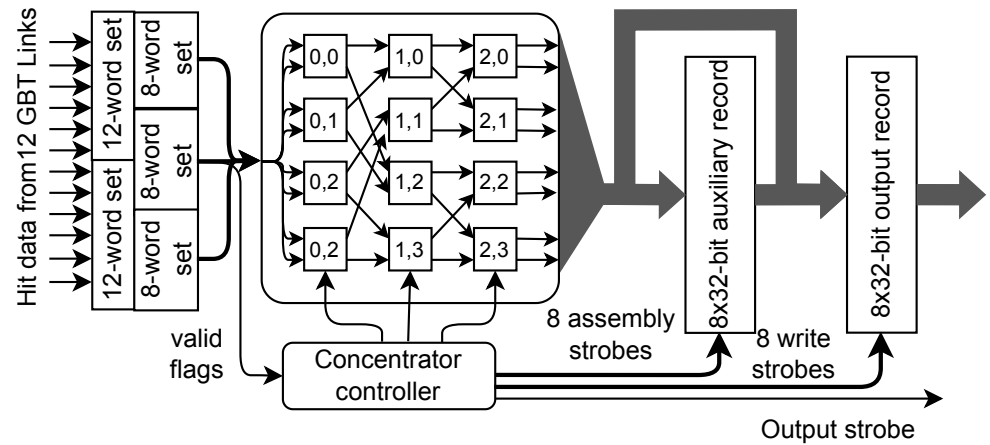
**Figure 9.** Data concentrator for 7 inputs based on a Beneš network reduced to 7x8 size.

Special solutions are needed when it is necessary to concentrate data from more than eight inputs. Such a situation occurred in designing the firmware for the new CRI2 [17] readout board for CBM [18] experiment. Currently planned hardware solutions need to concentrate data either from 9 or 12 GBT Links delivering data at 160 MHz to a 256-bit wide word at a frequency up to 250 MHz. Of course, using the Beneš network with a size limited to 8x8 requires time multiplexing the input data. However, it does not require as high frequency as the high-speed polling method described in section 2.1. For those designs, a dedicated data converter has been developed, which receives two input data sets at frequency  $f_{IN}$ , combines them, and then outputs them as three smaller sets at the frequency  $f_{OUT} = \frac{3}{2}f_{IN}$ . In the described system the Beneš network works at a frequency of 240 MHz, which is below 250 MHz.

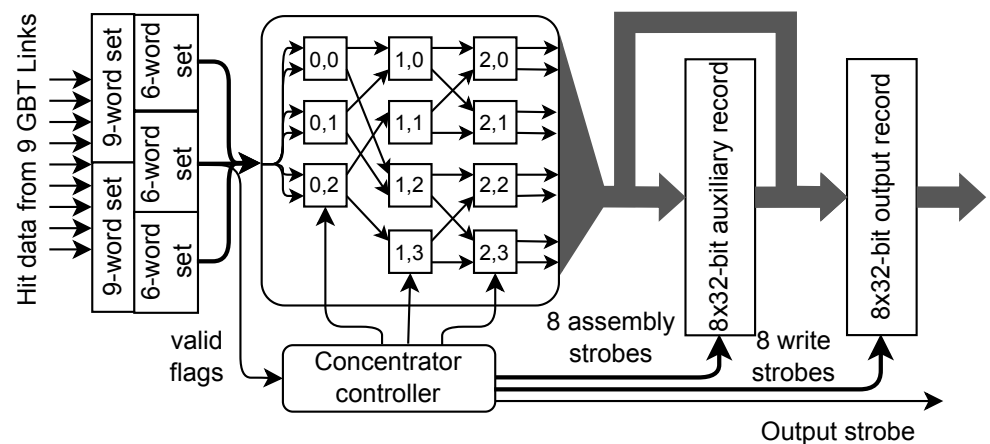
In the case of 12 input links, that converter works with an 8x8 Beneš network, as shown in Figure 10. In the case of 9 input links, the converted data consist of three 6-word sets. They are delivered to the simplified 6x8 Beneš network, as shown in Figure 11.

#### 5. Tests and results

All five described variants of concentrators have been tested in simulations. The input data were generated as 32-bit words containing consecutive numbers starting from 0x100. The user could set probability that the data word is delivered to the individual input. The tests were performed for various values of probability: very low (0.01), low (0.1), medium (0.5), high (0.9), and very high (1.0). For all tested values of the probability, all



**Figure 10.** Data concentrator for 12 inputs. The data width converter receives input sets containing 12 words at 160 MHz, concatenates two such sets, and outputs them as three 8-word sets at 240 MHz. Further concentration is performed as in Figure 8.



**Figure 11.** Data concentrator for 9 inputs. The data width converter receives input sets containing 9 words at 160 MHz, concatenates two such sets, and outputs them as three 6-word sets at 240 MHz. Further concentration is done via a Beneš network reduced to 6x8 size. The last two inputs and the fourth switch in layer 0 are removed. Only 11 switches are controlled, like in the case of a 7x8 network.

five configurations of the concentrator worked correctly. All data delivered to the inputs were transmitted exactly once, and no invalid data were inserted into the output records. The waveforms from a simulation of a 7-inputs concentrator at probability 0.5 is shown in Figure 12.

The most complex configuration with 12 inputs has also been verified in hardware. The implementation was performed in two boards:

- KCU105 [19] AMD/Xilinx board, equipped with Kintex Ultrascale XCKU040-2FFVA1156E FPGA,
- TEC0330 [13] board from Trenz Electronic equipped with Xilinx Virtex-7 XC7VX330T-2FFG1157C FPGA.

For testing in hardware, a special testbench has been prepared with the structure shown in Figure 13. The input data for the concentrator are written via PCIe to the FIFO with asymmetric port width<sup>2</sup>. Similarly, the output from the concentrator is written to the second FIFO with 256-bit wide input and 32-bit wide output connected to another PCIe-accessible register. Additional control and status registers support resetting the FIFOs and the concentrator, starting the data transfer, and reading the status of both FIFOs.

The design was successfully compiled for both selected platforms. Timing closure was obtained for 160 MHz and 240 MHz frequencies. The resulting resource consumption is shown in Table 2.

**Table 2.** Resource consumption of the 12-inputs data concentrator together with the testbench for chosen hardware platforms. Absolute and percentage (in parenthesis) consumption is given. The design was synthesized in the version where no BRAM was used for the controller. Separate values for the testbench, data width converter, and concentrator itself are given. That version uses the biggest concentrator based on the 8x8 Beneš network. For all other described configurations, the resource utilization will be lower.

	LUTs	KCU105 Flip Flops	Block RAMs	LUTs	TEC0330 Flip Flops	Block RAMs
Available	242400	484800	600	204000	408000	750
Whole testbench	7685 (3.17%)	10641 (2.19%)	36 (6.00%)	10406 (5.1%)	12339 (3.02%)	36 (4.80%)
Data width converter	745 (0.31%)	1459 (0.30%)	0 (0.0%)	742 (0.36%)	1459 (0.36%)	0 (0.0%)
Data concentrator	1038 (0.43%)	1566 (0.32%)	0 (0.0%)	1032 (0.51%)	1566 (0.38%)	0 (0.0%)

The FPGA configured with the testbench FIFO is controlled with the `uio_pci_generic` driver and a simple Python script. The script resets the FIFOs and the concentrator. Then prepares the input data sets and writes them to the first FIFO. Afterward, it starts the data transfer. Finally, it reads the concentrated data from the second FIFO. For automated tests, the input data sets are prepared similarly to the simulations. The data words containing consecutive values were written to the consecutive inputs. With the probability defined by the user, each input could be skipped.

The tests were repeated multiple times with different probabilities of skipping the input and a different number of input data sets (of course, always smaller than the capacity of the input FIFO). In all tests, the concentrated data were correctly delivered to the output FIFO without losses or duplications.

<sup>2</sup> For 12 32-bit wide inputs the necessary width of FIFO is 384 bits for data and 12 for valid flags, resulting in 396 bits. The Xilinx FIFO generator does not support that width. Therefore, a FIFO with 512-bit wide output was used. However, the minimal input width for such a FIFO is 64-bits. Therefore, the input value for that FIFO is concatenated from two PCIe-accessible registers. Writing one of them activates the FIFO write strobe.

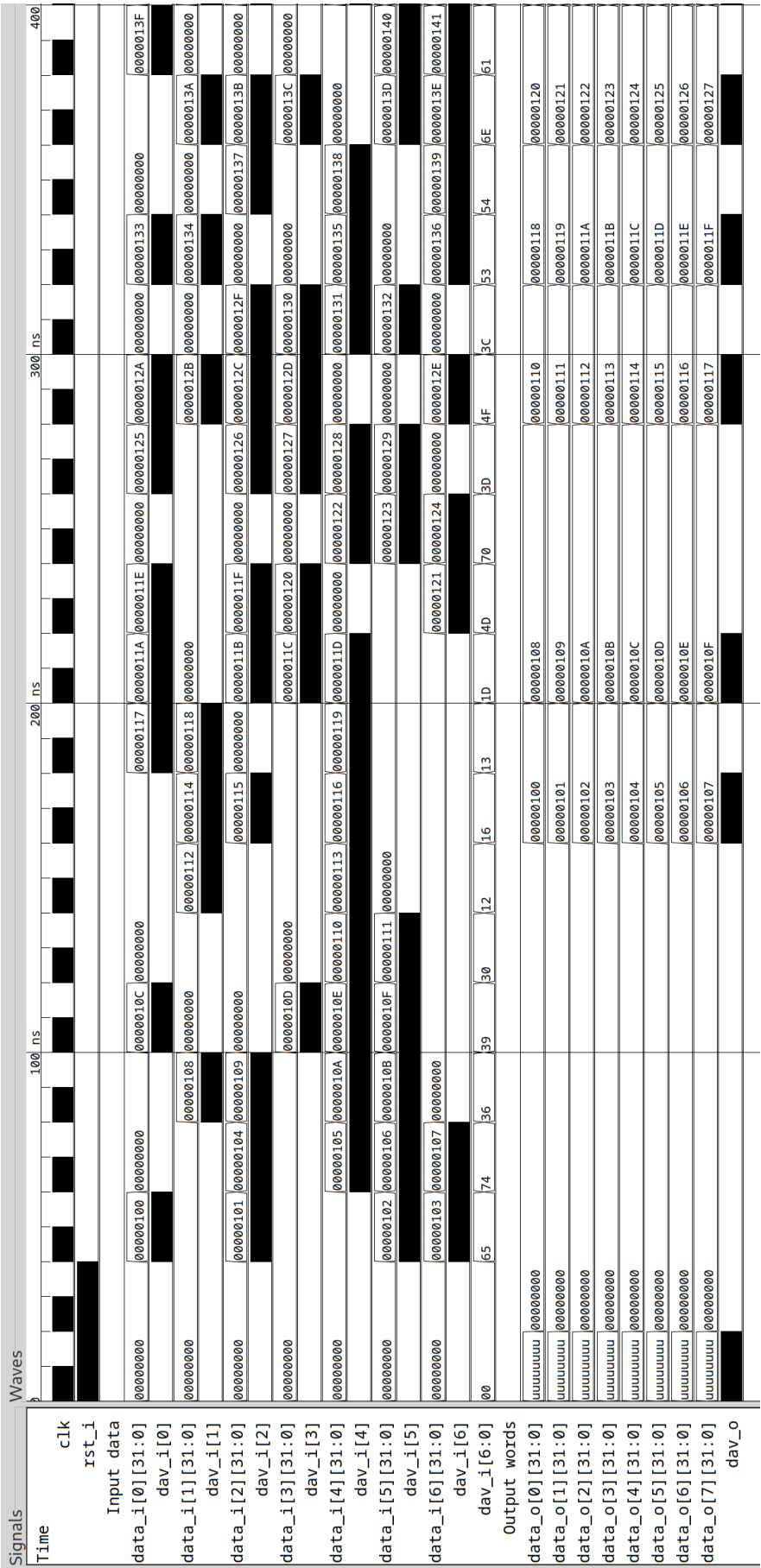
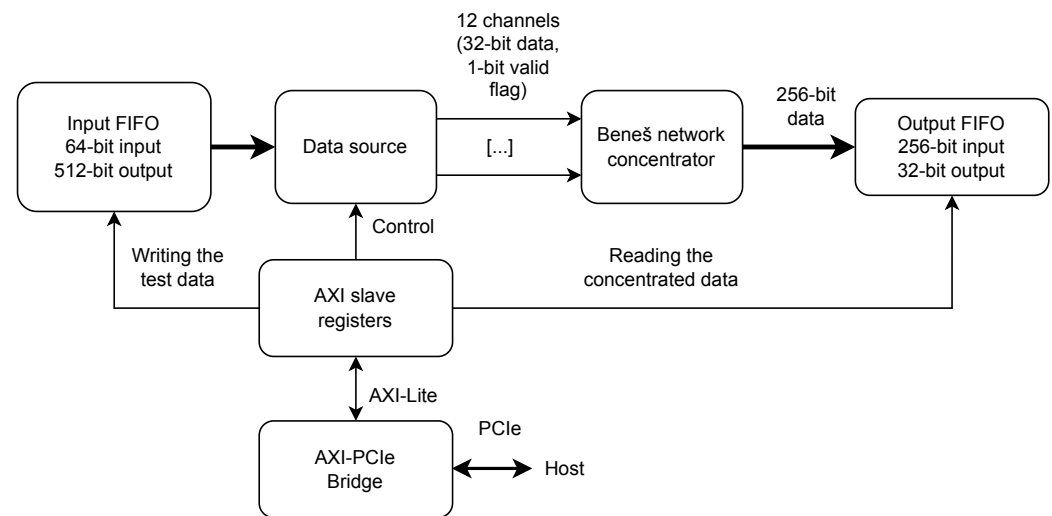


Figure 12. Results of simulation of the concentrator with 7 inputs and probability of data presence set to 0.5.



**Figure 13.** Testbench for testing the concentrator in the hardware.

## 6. Discussion and conclusions

The novelty presented concentration method eliminates the disadvantages previous solutions. It does not require scanning inputs at high clock frequency as the high-frequency polling method (see section 2.1). It does not require multiple width-converting FIFOs, does not inject empty data into the output stream, and does not disturb data time-ordering as the width conversion in the input channels (see section 2.2).

The proposed method is well-suited for FPGA implementation. The Beneš network used to route the data consists of simple blocks (2x2 switches). No complex multiplexers are needed. Pipeline registers may be added inside the switches, resulting in short critical paths and high maximum clock frequency.

The Beneš network is quite old technology, but the paper describes its usage in a new application area. Additionally, it presents a method that significantly reduced the network size and simplified its configuration due to using the precalculated switches configurations.

The solution is based on an 8x8 Beneš network, but extending it for more inputs is described and tested.

The presented solution may improve the concentration of data, especially in concentrators for triggerless DAQs where a high-speed concentration of non-continuous data streams is needed.

The design is open-source and may be freely reused. The sources are available in repository [20].

**Author Contributions:** Conceptualization, W.M.Z., B.M.Z.; software, W.M.Z. and M.G.; validation M.G.; investigation, M.K.; writing—original draft preparation, W.M.Z.; writing—review and editing, all authors; supervision, W.M.Z.; Specific technical contribution of the authors: General concept of the solution, software simulations, W.M.Z.; Concept of using the Beneš network for data routing in concentrator, B.M.Z.; Implementation in HDL, simulations, and testing in hardware M.G.; Review of previous art, M.K.; The percentage contribution of the authors is: M.G. – 40%, W.M.Z. – 35%, M.K. – 10%, B.M.Z. – 15%. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was partially supported by the statutory funds of Institute of Electronic Systems. This project has also received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 871072.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

- Colombo, T.; Amihalachioaie, A.; Arnaud, K.; Alessio, F.; Brarda, L.; Cachemiche, J.P.; Cámpora, D.; Cap, S.; Cardoso, L.; Cindolo, F.; et al. The LHCb Online system in 2020: trigger-free read-out with (almost exclusively) off-the-shelf hardware. *Journal of Physics: Conference Series* **2018**, *1085*, 032041. doi:10.1088/1742-6596/1085/3/032041.
- Cuveland, J.d.; Lindenstruth, V.; the CBM Collaboration. A First-level Event Selector for the CBM Experiment at FAIR. *Journal of Physics: Conference Series* **2011**, *331*, 022006. doi:10.1088/1742-6596/331/2/022006.
- Kasinski, K.; Szczygiel, R.; Zabolotny, W.; Lehnert, J.; Schmidt, C.; Müller, W. A protocol for hit and control synchronous transfer for the front-end electronics at the CBM experiment. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **2016**, *835*, 66–73. doi:10.1016/j.nima.2016.08.005.
- Lehnert, J.; Byszuk, A.; Emschermann, D.; Kasinski, K.; Müller, W.; Schmidt, C.; Szczygiel, R.; Zabolotny, W. GBT based readout in the CBM experiment. *Journal of Instrumentation* **2017**, *12*, C02061–C02061. doi:10.1088/1748-0221/12/02/C02061.
- Technical Design Report for the CBM Online Systems – Part I. In review.
- Bourrion, O.; Bouvier, J.; Costa, F.; Dávid, E.; Imrek, J.; Nguyen, T.; Mukherjee, S. Versatile firmware for the Common Readout Unit (CRU) of the ALICE experiment at the LHC. *Journal of Instrumentation* **2021**, *16*, P05019. doi:10.1088/1748-0221/16/05/P05019.
- Wu, W. FELIX: the New Detector Interface for the ATLAS Experiment. *IEEE Transactions on Nuclear Science* **2019**, *66*, 986–992. doi:10.1109/TNS.2019.2913617.
- FELIX firmware sources – CRTToHostdm module. <https://gitlab.cern.ch/atlas-tdaq-felix/firmware/-/blob/phase2/master/sources/CRTToHost/CRTToHostdm.vhd>. [Online; accessed 25-January-2023].
- FELIX firmware sources – CRTToHost module. <https://gitlab.cern.ch/atlas-tdaq-felix/firmware/-/blob/phase2/master/sources/CRTToHost/CRTToHost.vhd>. [Online; accessed 25-January-2023].
- Mathematical Theory of Connecting Networks and Telephone Traffic (Mathematics in science and engineering ; v. 17)*; Elsevier, 1965.
- Nikolaïdis, D.; Groumas, P.; Kouloumentas, C.; Avramopoulos, H. Novel Benes Network Routing Algorithm and Hardware Implementation. *Technologies* **2022**, *10*, 16. doi:10.3390/technologies10010016.
- Dementev, D.; Guminski, M.; Kovalev, I.; Kruszewski, M.; Kudryashov, I.; Kurganov, A.; Miedzik, P.; Murin, Y.; Pozniak, K.; Schmidt, C.J.; et al. Fast Data-Driven Readout System for the Wide Aperture Silicon Tracking System of the BM@N Experiment. *Physics of Particles and Nuclei* **2021**, *52*, 830–834. doi:10.1134/S1063779621040213.
- TEC0330 - PCIe FMC Carrier with Xilinx Virtex-7 FPGA. <https://shop.trenz-electronic.de/en/Products/Trenz-Electronic/PCIe-FMC-Carrier/TEC0330-Xilinx-Virtex-7/>. [Online; accessed 6-January-2023].
- FMC - Octal SFP/SFP+. <https://www.fastertechnology.com/store/fmc-modules/fm-s18.html>. [Online; accessed 6-January-2023].
- Zabolotny, W.M. Versatile DMA Engine for High-Energy Physics Data Acquisition Implemented with High-Level Synthesis, 2023. doi:10.20944/preprints202301.0328.v3.
- Sidorenko, V.; Fröhlich, I.; Müller, W.; Emschermann, D.; Bähr, S.; Sturm, C.; Becker, J. Prototype design of a timing and fast control system in the CBM experiment. *Journal of Instrumentation* **2022**, *17*, C05008. doi:10.1088/1748-0221/17/05/C05008.
- The readout system of the CBM experiment. [https://indico.phy.ornl.gov/event/112/contributions/566/attachments/492/1342/20211208\\_169\\_sro9\\_cbm\\_daq\\_v02.pdf](https://indico.phy.ornl.gov/event/112/contributions/566/attachments/492/1342/20211208_169_sro9_cbm_daq_v02.pdf).
- Compressed Baryonic Matter experiment at FAIR. <https://www.cbm.gsi.de/>. [Online; accessed 31-January-2023].
- Xilinx Kintex UltraScale FPGA KCU105 Evaluation Kit. <https://www.xilinx.com/products/boards-and-kits/kcu105.html>. [Online; accessed 6-January-2023].
- Beneš-network-based concentrator for triggerless DAQ systems – git repository. <https://gitlab.com/WZabISE/concentrator>. [Online; accessed 31-January-2023].