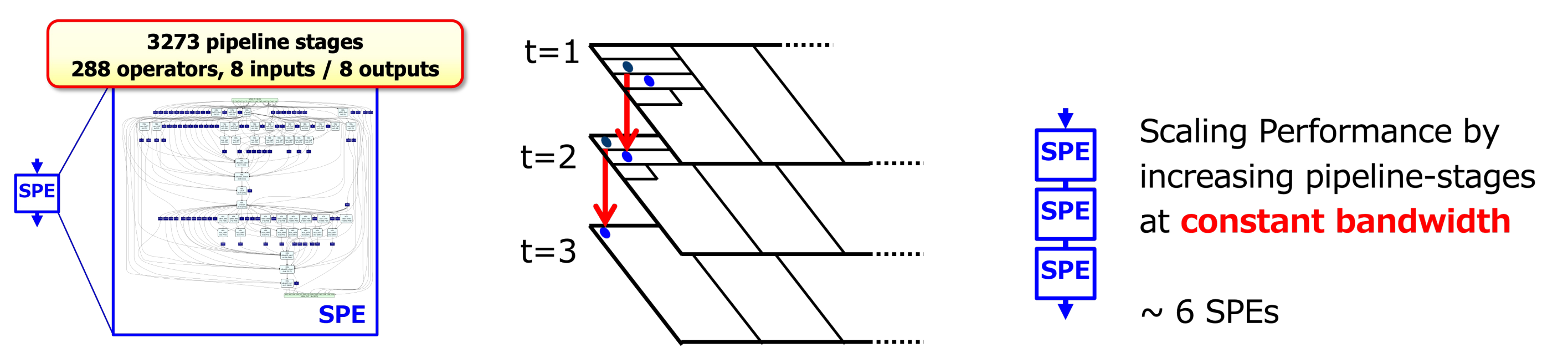
Time-Division Multiplexing of Multiple Compressed DataStreams on FPGAs

Takato Abe[1], Tomohiro Ueno[2], Norishita Fujita[3] and Taisuke Boku[3]
1.Degree Programs in Systems and information Engineering, University of Tsukuba. Japan
2.RIKEN Center for Computational Science, Japan
3.Center for Computational Sciences, University of Tsukuba, Japan

1, Background

- Stream computing is a promising computational architecture that outperforms existing architectures in terms of **memory performance** and **power consumption** [1,2,3].

Fig. 1

3273 pipeline stages
288 operators, 8 inputs / 8 outputs
SPE
SPE
t=1
t=2
t=3
SPE
SPE
SPE
Scaling Performance by increasing pipeline-stages at constant bandwidth
~ 6 SPEs
Tsunami Simulation by Stream Computing [3]

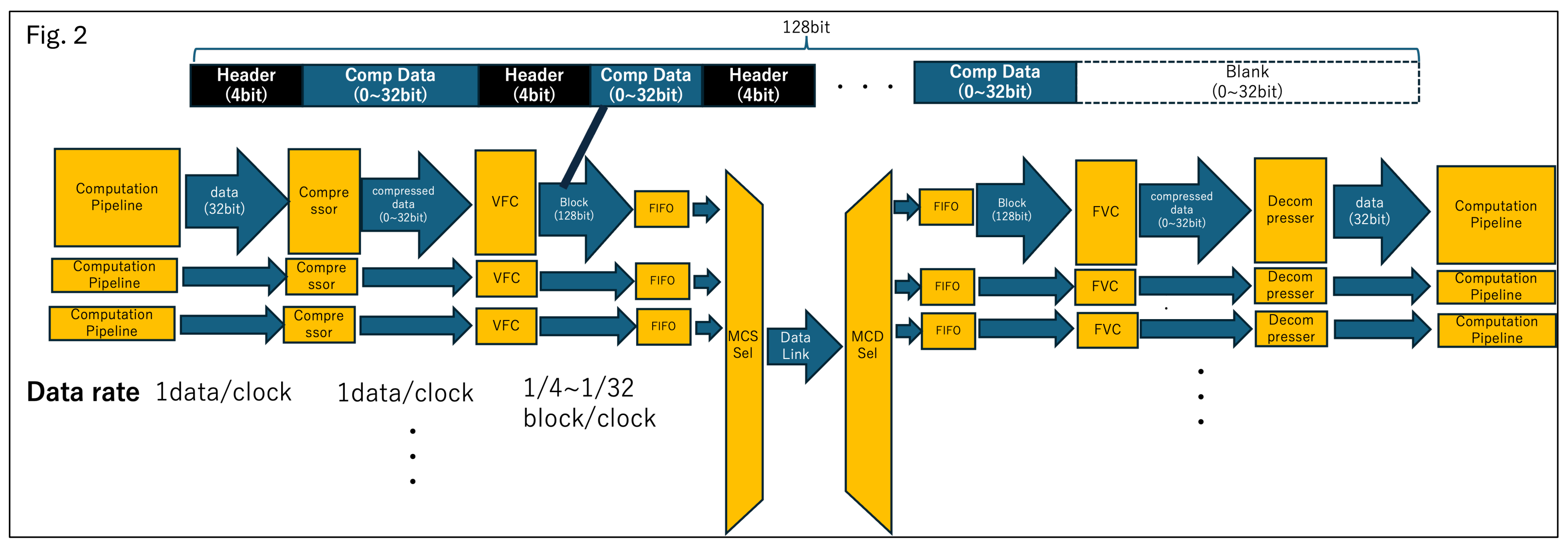
- Combining it with **low-latency stream compression** techniques can improve logical communication and memory bandwidth, thereby **improving application throughput and scalability.**
- In this study, we propose an **algorithm for allocating communication bandwidth** to each stream in a module that serializes compressed data block streams with **multiple variable write rates** in a general-purpose communication data compression platform proposed by Ueno et al. at RIKEN [4], and we compare the performance of the HDL implementations of each algorithm through logic circuit simulation.

2, Data Compression Platform

The data compression platform proposed by Ueno et al [4]. consists of the following modules:

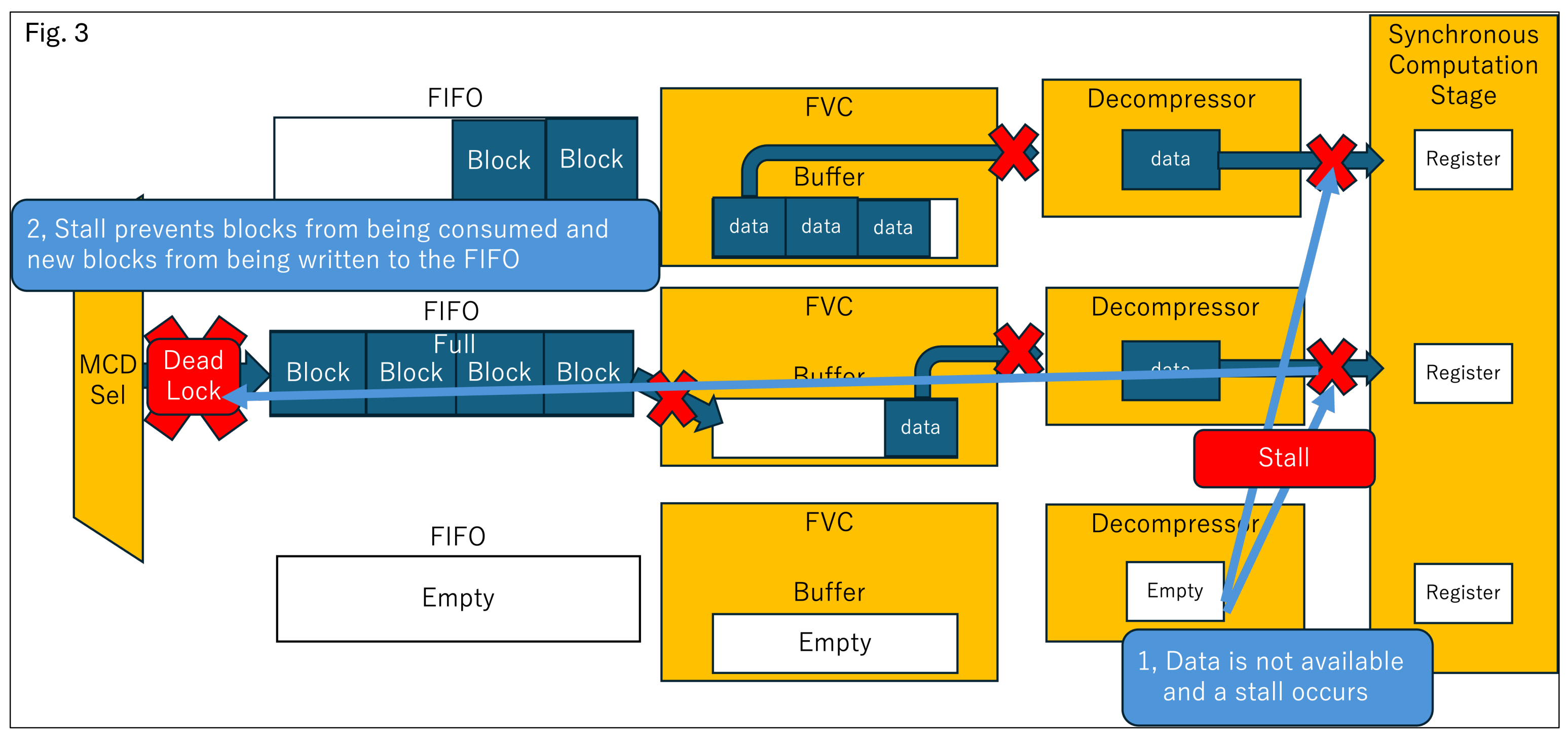
- **VariableToFixedConverter(VFC)**: Packs multiple variable-length compressed data into one fixed-length block.
  - **The clock cycles required to output** a fixed-length block **decreases** as the data **compression rate increases**.
- **MultiChannelSerializer(MCS)** : **Serializes block streams input from multiple channels** and outputs them to communication modules.
- **MultiChannelDeserializer(MCD)** : Distributes the blocks of the serialized blockstream to their respective destination channels.
- **FixedToVariabelConverter(FVC)** : Receives a block and outputs the compressed data contained in the block one per clock.

By incorporating these into the application's computational pipeline as shown in Fig. 2, we aim to improve communication throughput.

Fig. 2


3, Definition Of Problem

Stalls due to synchronization waits may result in a **decrease in operation latency** or a **deadlock** due to the FIFO becoming full(Fig. 3).

Fig. 3


Therefore, the MCS Selector needs to read blocks

- from channels **frequently** with high input data rates (i.e., **low compression ratios**)
- from channels **infrequently** with low input data rates (i.e., **high compression ratios**).

4, MCS Channel Selection Algorithm

We implement and compare the performance of five MCS algorithms.
The algorithms, except for Round-Robin, are priority-based, reading data from the FIFO of the input channel with the highest priority.

Algorithm	Primary Priority	Secondary Priority
Round-Robin (RR)	-	-
used FIFO Capacity (FC)	Used FIFO capacity on MCS	Fixed priority
StallCycle (SC)	Write stall count for each FIFO on MCS	Fixed priority
StallCycle and used FIFO Capacity (SCFC)	Write stall count for each FIFO on MCS	Used FIFO capacity on MCS
TimeStamp (TS)	Oldness of the head data in each FIFO	Fixed priority

5, HDL Implementation

- In the four methods other than RR, we use a pipelined selector tree (Fig. 5) based on 2-ch selectors (Fig. 4) to choose the FIFO with the highest priority.
  - Even if the number of channels is increased, **the operating frequency is less affected**.

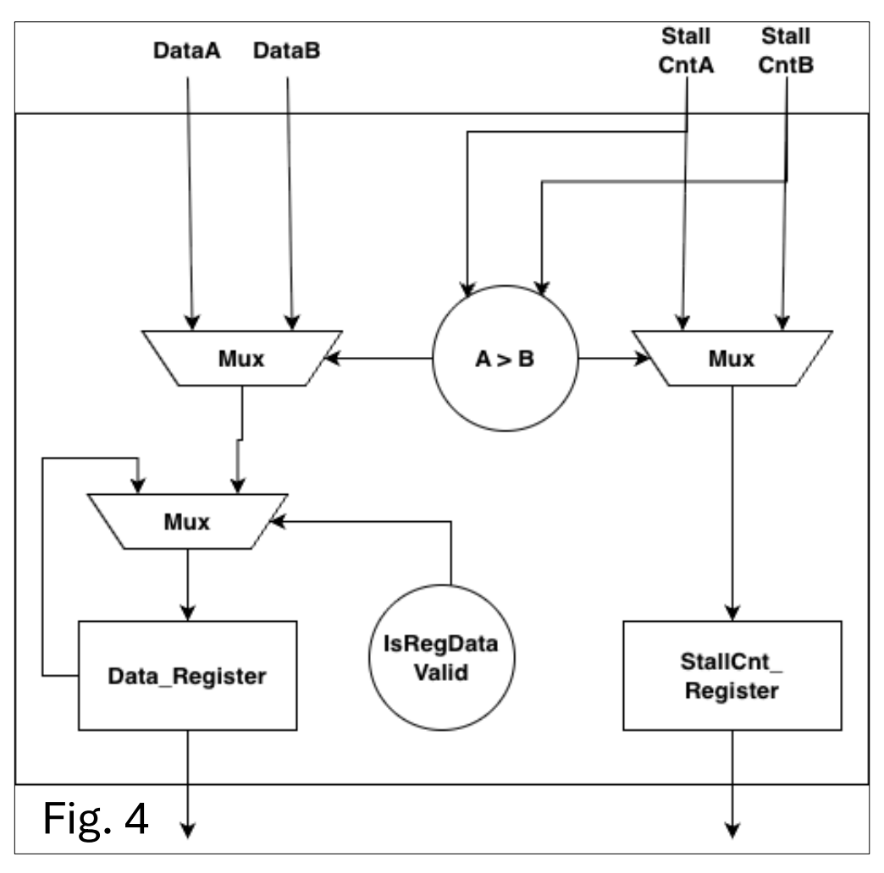


Fig. 4

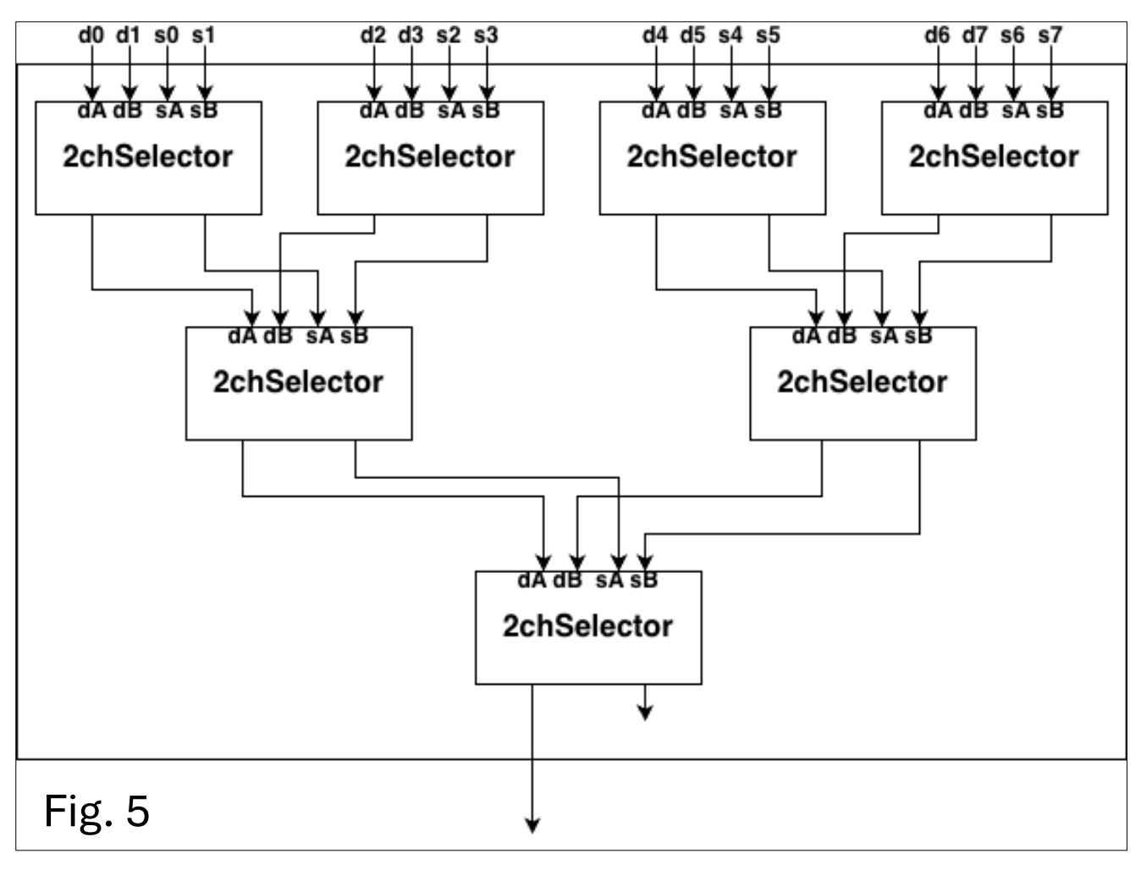
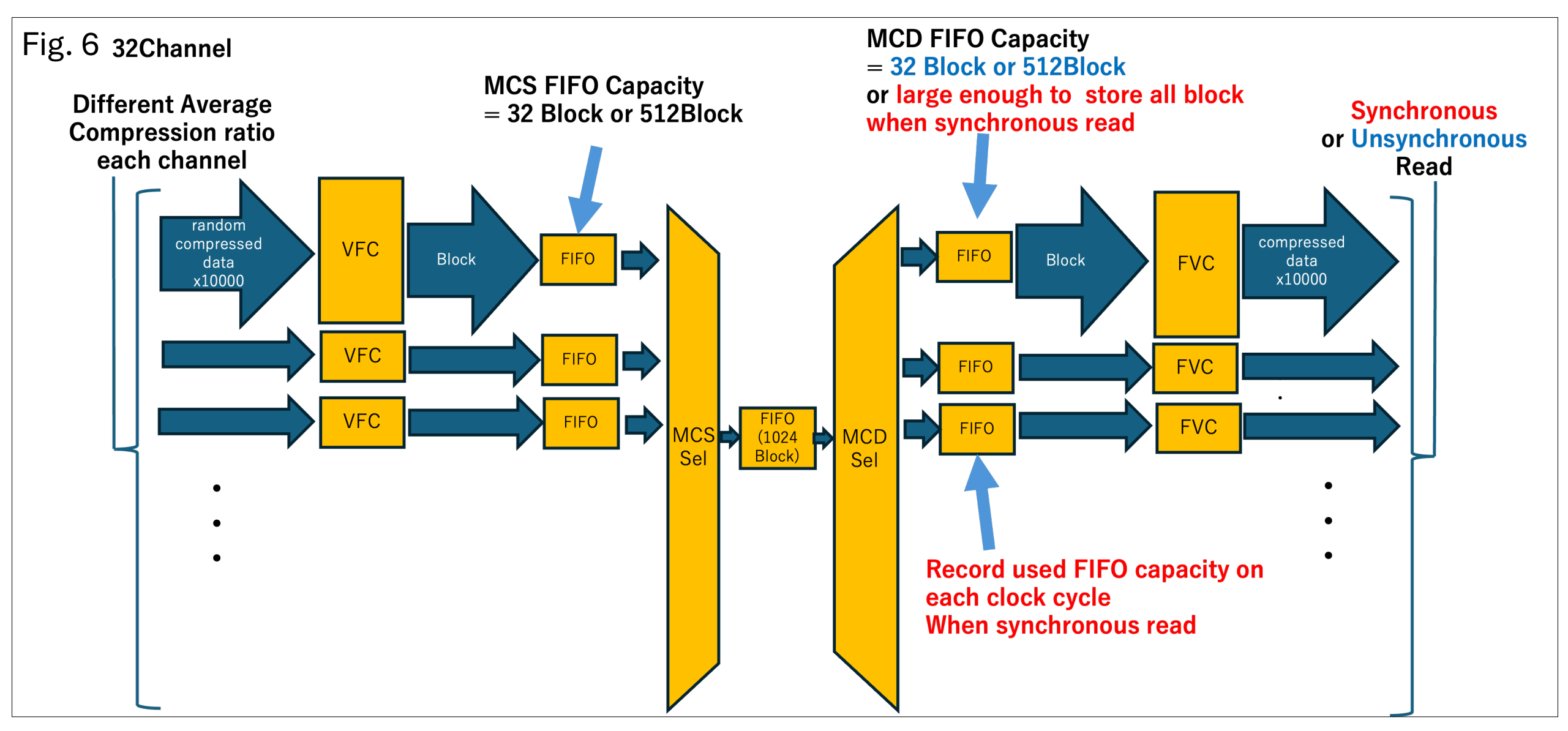


Fig. 5

- Written in Chisel, which allows the number of channels to be parameterized, and automatic construction of a tree corresponding to any number of channels.

6, Performance Evaluation

We conducted an experiment using a logic circuit simulator(Verilator) to input 10,000 words of randomly compressed data with different average compression rates for each channel to a module like Fig.6.
We evaluated the fairness of channels selection and the minimum required capacity of the MCD-side FIFO (maximum value of FIFO used capacity during the experiment) to prevent the MCD-side FIFO from becoming full and causing a deadlock.

Fig. 6

32Channel
Different Average Compression ratio each channel
MCS FIFO Capacity = 32 Block or 512Block
MCD FIFO Capacity = 32 Block or 512Block or large enough to store all block when synchronous read
Synchronous or Unsynchronous Read
random compressed data x10000
VFC
Block
FIFO
MCS Sel
FIFO (1024 Block)
MCD Sel
FIFO
Block
FVC
compressed data x10000
Record used FIFO capacity on each clock cycle When synchronous read

Number of Channels	32ch
Average Compressed Data Length	16bit
Standard deviation of the average compressed data length of each channel input data sequence	6.08bit

6.1, Fairness of data selection

- To evaluate the fairness of the proposed algorithms, we compare the time until all data have been read. In this evaluation, output channels will be read continuously and **individually**.
- The **TimeStamp implementation is the fairest** with both FIFO capacities of 32 (Fig. 7) and 512 (Fig. 8).

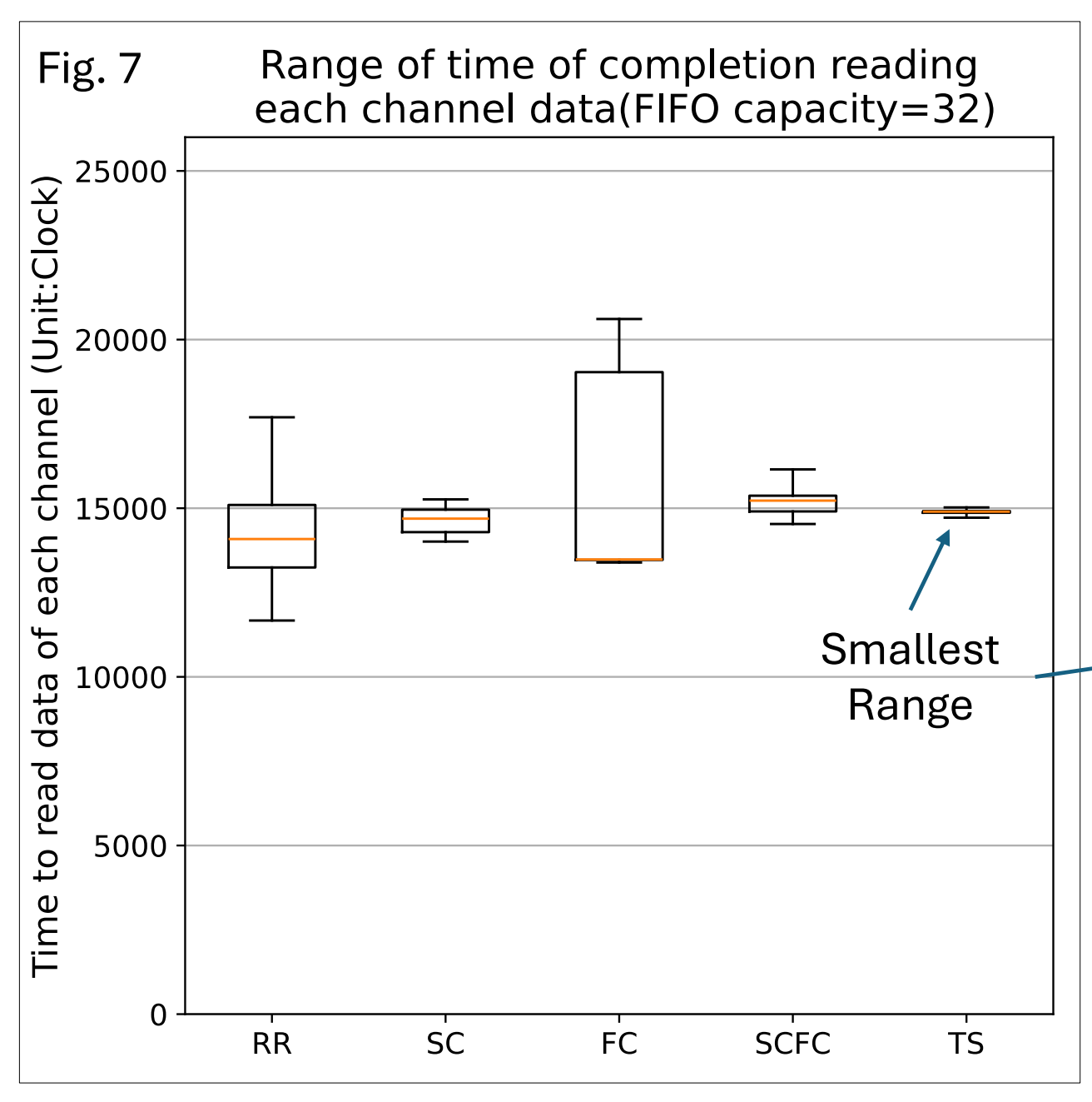
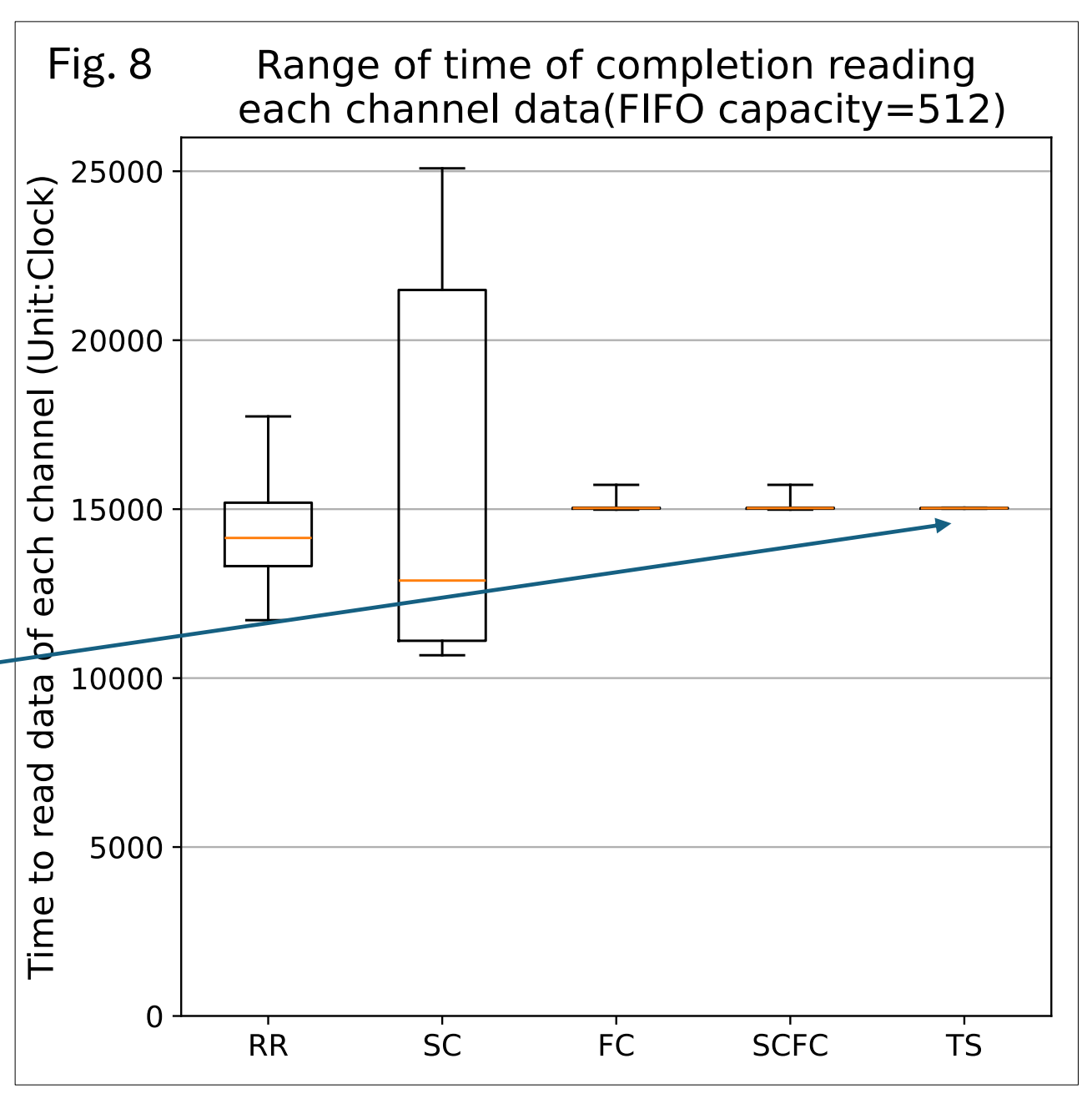
Fig. 7


Fig. 8


6.2, MCD-Side FIFO Capacity Required to Avoid Deadlock

- We evaluate the FIFO capacity required by each implementation to avoid deadlock.
- In this evaluation, **FIFOs are read only when data are available on all output FIFOs** to simulate the situation with computation logic. Deadlock may occur if the capacity of the FIFOs on the MCD side is insufficient.
- The **Timestamp implementation is also the fairest** in this evaluation for both FIFO capacities of 32 (Fig. 9) and 512 (Fig. 10).
- The TimeStamp implementation also shows a constant progression compared to the others (Fig. 11).
- Timestamp implementation's small MCD minimum FIFO capacity means that there was always little data waiting for synchronization of data reads.
  - **channels were selected fairly even when data communication was in progress.**

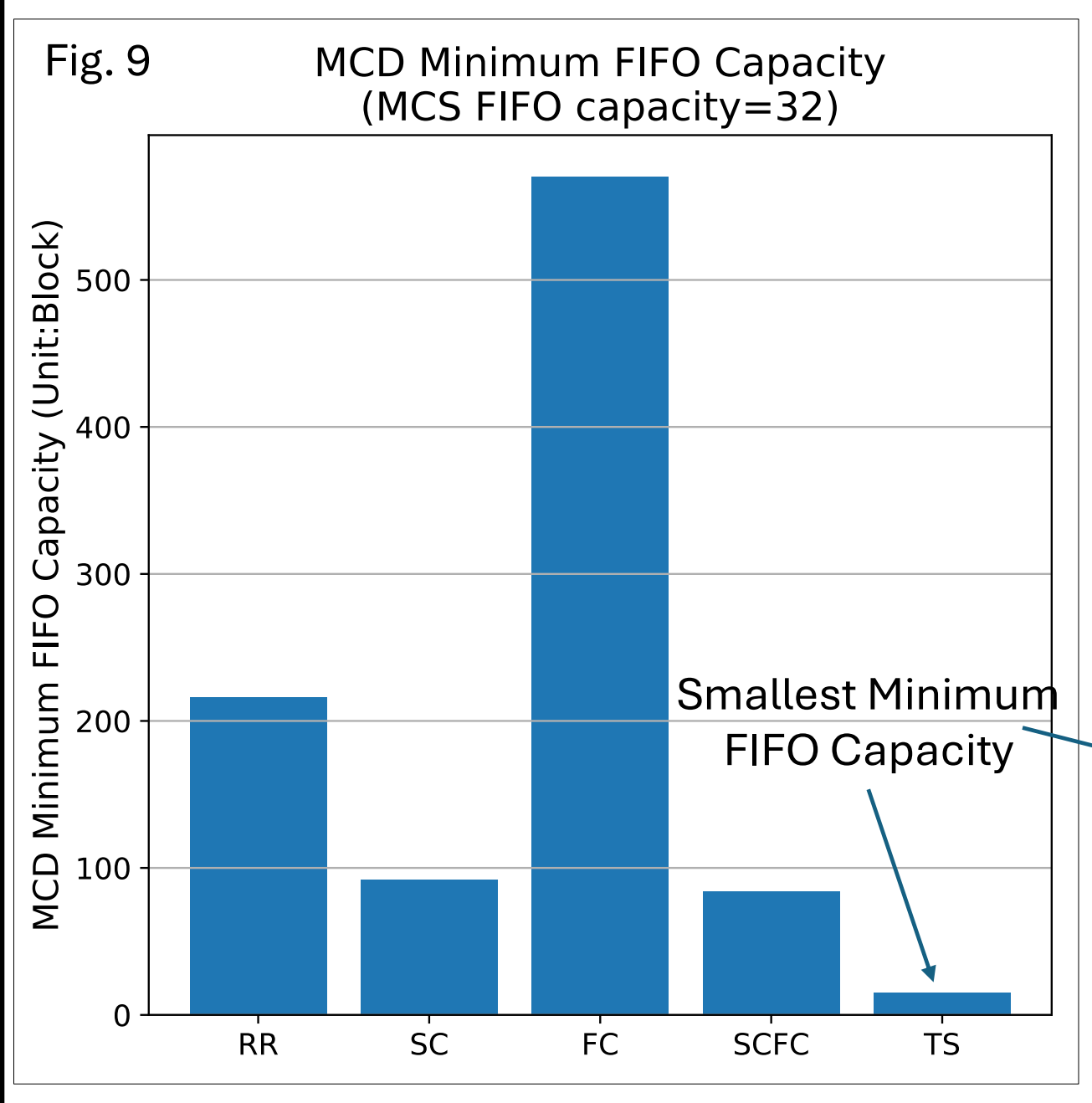
Fig. 9


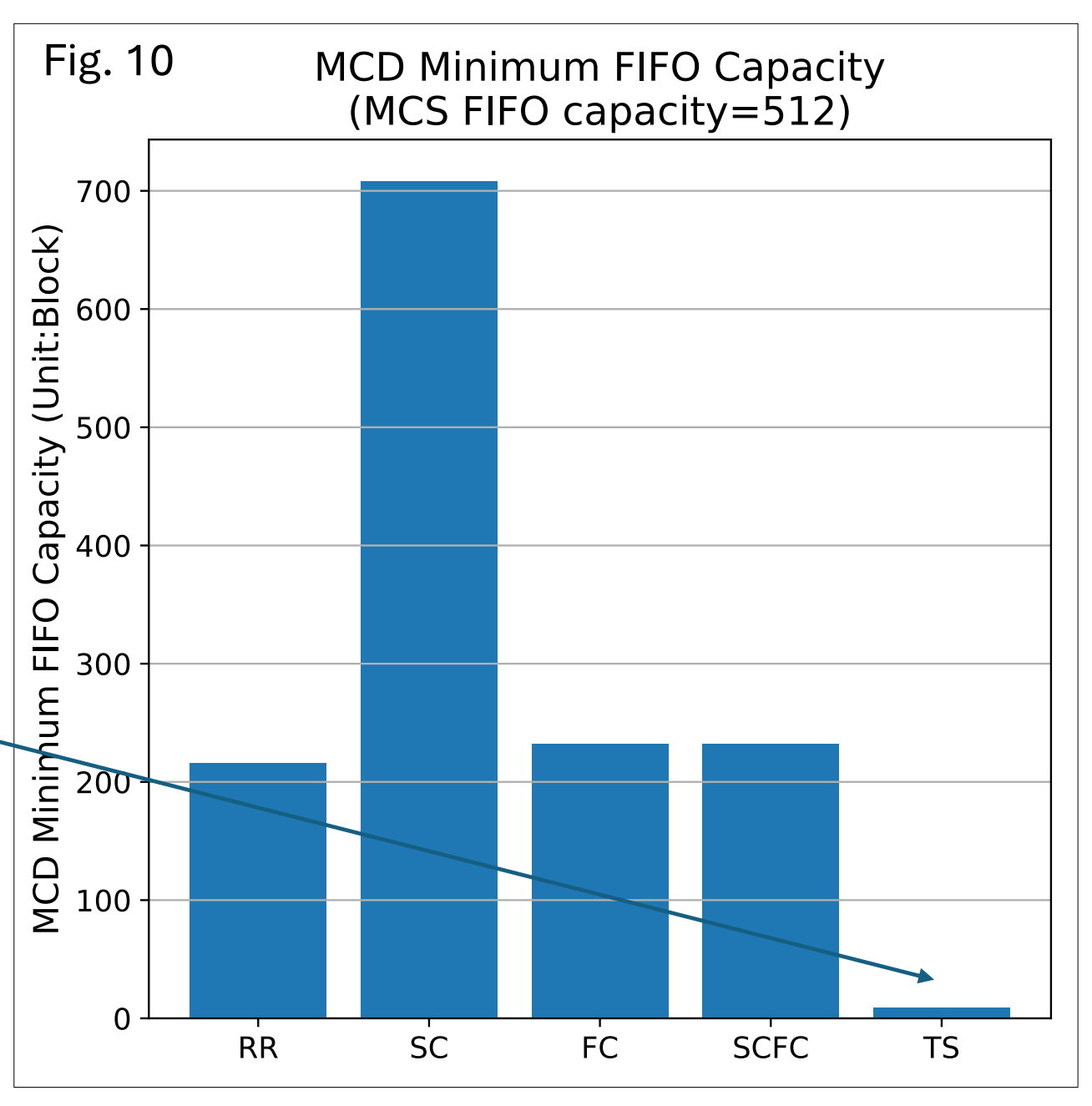
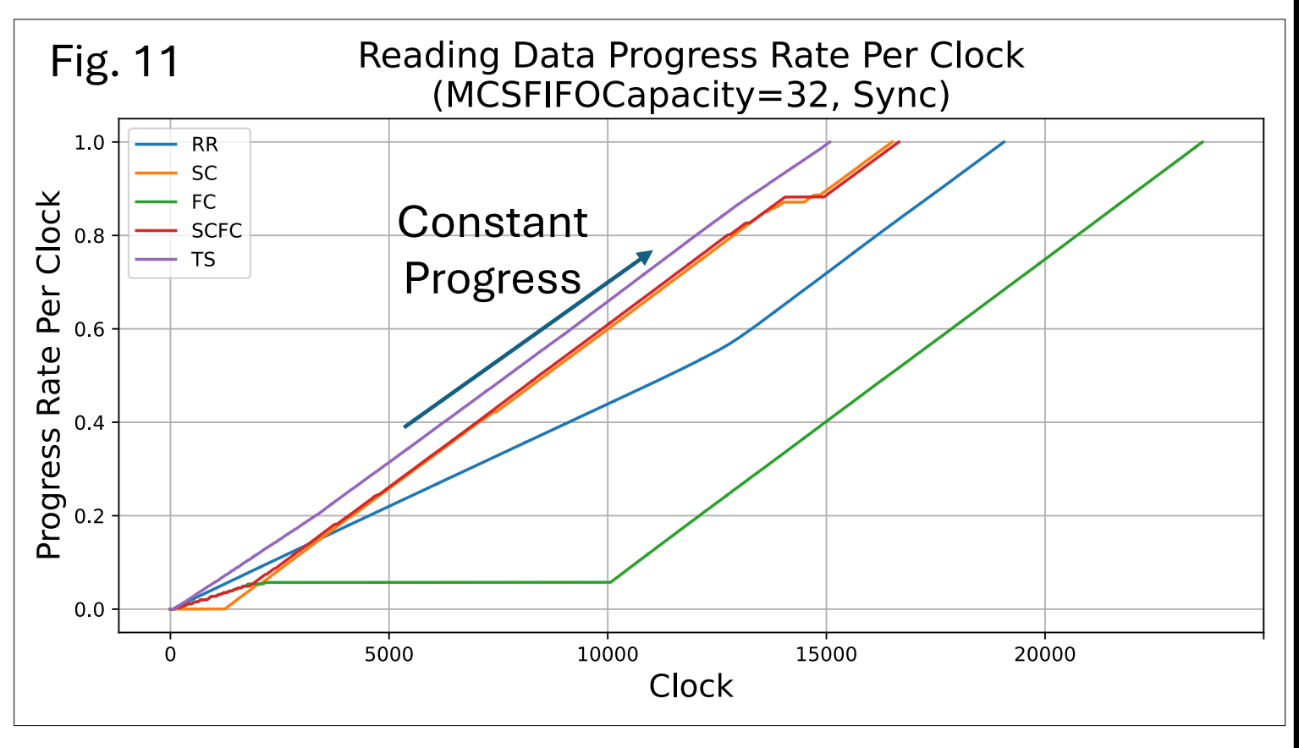
Fig. 10


Fig. 11


7, Conclusion

In this study, we proposed five channel selection algorithms (RR, FC, SC, SCFC, and TS) for serializing compressed data streams to resolve data rate imbalances that occur when the compression ratio differs for each channel in FPGA-based HPC compressed data stream processing. We implemented them in HDL and evaluated its performance on a circuit simulator. The performance evaluation showed that **TS produced the most fair data output**.
In addition, to prevent deadlock caused by the FIFO on the MCD side filling up, we investigated the FIFO size that would prevent the FIFO on the MCD side from becoming full, and found that **TS was able to prevent deadlock with the smallest FIFO capacity**.

Reference :[1]DeHon, André. "Fundamental underpinnings of reconFig.urable computing architectures." *Proceedings of the IEEE* 103.3 (2015): 355-378.**[2]** MONDIGO, Antoniette, et al. Scalability analysis of deeply pipelined tsunami simulation with multiple fpgas. IEICE TRANSACTIONS on Information and Systems, 2019, 102.5: 1029-1036.**[3]** Sano, Kentaro, Yoshiaki Hatsuda, and Satoru Yamamoto. "Multi-FPGA accelerator for scalable stencil computation with constant memory bandwidth." IEEE Transactions on Parallel and Distributed Systems 25.3 (2013): 695-705.**[4]** Ueno, Tomohiro, Kentaro Sano, and Satoru Yamamoto. "Bandwidth compression of floating-point numerical data streams for FPGA-based high-performance computing." ACM Transactions on ReconFig.urable Technology and Systems (TRETS) 10.3 (2017): 1-22.

Acknowledgements :
This work has been partially supported by Japan Science and Technology Agency (JST) as part of
Adopting Sustainable Partnerships for Innovative Research Ecosystem (ASPIRE) Grant Number JPMJAP2341.