IEICE TRANSACTIONS

on Information and Systems

DOI:10.1587/transinf.2024EDP7033

Publicized:2024/08/07

This advance publication article will be replaced by the finalized version after proofreading.

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY

The Institute of Electronics, Information and Communication Engineers

Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

Applying Run-Length Compression to the Configuration Data of SLM Fine-Grained Reconfigurable Logic

Souhei TAKAGI[†], ***Nonmember***, **Takuya KOJIMA**[††], ***Member***, **Hideharu AMANO**[†], **Morihiro KUGA**[†††], ***and*** **Masahiro IIDA**[†††], ***Fellows***

SUMMARY SLM (Scalable Logic Module) is a fine-grained reconfigurable logic developed by Kumamoto University. Its small configuration information size characterizes it, resulting in a smaller area for logic cells. We have been developing an SoC-type FPGA called SLMLET to take advantage of SLM. It keeps multiple sets of configuration data in the memory module inside the chip in a compressed form and exchanges them quickly. This paper proposes a simple run-length compression technique called TLC(Tag Less Compression). It achieved a 1.01-3.06 compression ratio, is embedded in the prototype of the SLMLET, and is available now. Then, we propose DMC (Duplication Module Compression), which uses repeatedly appearing patterns in the SLM configuration data. The DMC achieves a better compression ratio for complicated designs that are hard to compress with TLC.

key words: *SLM reconfigurable logic, Run-Length Compression, FPGA*

1. Introduction

In recent years, FPGA has rapidly advanced, achieving high functionality and performance, and is widely utilized in applications such as AI accelerators. In IoT (Internet of Things), hardware that offloads anonymization and protocol processing is necessary, and System-on-Chip (SoC) type FPGAs with embedded CPUs are used. However, recent SoC FPGAs are often over-spec and tend to consume a lot of power.

To address this problem, we have been developing a tiny SoC FPGA called SLMLET[1], which provides a unique FPGA core called SLM(Scalable Logic Module)[2], RISC-V CPU, HyperBus interface, and DMA/switching function as shown in Figure 1. The characteristic of SLM is its small configuration data size. Leveraging this, SLMLET compresses multiple configuration data and stores them in the chip's internal memory. While SLMLET does not accommodate much logic, it effectively utilizes small-area SLM by swapping configuration data for each task. This can also reduce power consumption.

Research on compression and decompression of configuration data was conducted in early FPGAs and was even integrated into some commercial FPGAs. However, recent FPGAs have increased logic capacity, and the frequency of configuration changes has decreased, reducing the necessity for compression. Consequently, research on configuration data compression is seldom pursued nowadays. Furthermore, this study aims to reduce the already small configuration data for SLM, aiming to store more configuration data sets within the chip. This differs from conventional FPGA configuration data compression in its conditions.

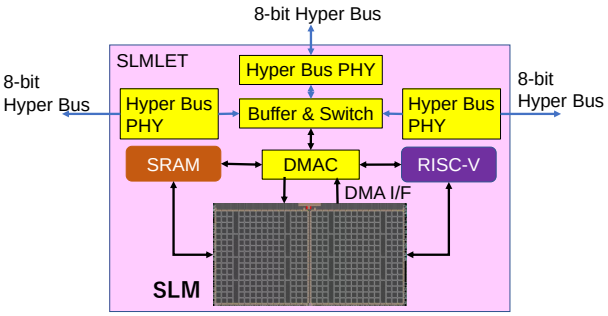

Fig. 1: SLMLET Overview

This paper proposes a simple run-length TLC(Tag Less Compression) technique. Although TLC achieves a good compression ratio for simple designs, it cannot reduce the size of complicated design configurations. So, we propose DMC (Duplication Module Compression), which uses repeatedly appearing patterns in the SLM configuration data. The early stage of the study was presented in the poster paper[3] only including an idea of TLC.

The paper is organized as follows. In Section 2, we review the configuration methods and related work. Section 3 describes SLM and how to integrate in SLMLET-1. In Section 4, we propose TLC and DMC, and evaluate the compression ratio. Section 5 is for hardware implementation, and Section 6 concludes the paper.

2. Compression Methods

2.1 Run Length Compression

Compression methods are classified into two categories: lossless compression, which allows complete restoration of the original data, and lossy compression, which does not. For the compression of configuration data, lossless compression is required [4]. Lossy compression is commonly used for

[†]Department of Information & Computer Science, Keio University, Yokohama, 223-8522, Japan
[††]Graduate School of Information Science and Technology, University of Tokyo, Tokyo, 113-8656, Japan
[†††]Department of Computer Science and Electrical Engineering, Kumamoto University, Kumamoto 860-8555, Japan

data like images or audio data, where complete restoration of the original data may not be necessary, achieving high compression ratios.

However, for FPGA configuration data, compression must be reversible. Two standard reversible compression methods are Run Length Compression, which we focus on here, and the Lempel Ziv method. Lempel Ziv methods are popularly used for file compression. Although a high compression ratio is achieved, the complexity makes it difficult to decompress the configuration data with the logic in the chip [5]. Thus, Lempel Ziv is only used to compare the compression ratio in this paper.

Run-length Compression works by registering patterns and their corresponding prefixes in advance. During compression, when a target pattern appears, it is replaced with the registered prefix, reducing the data size. During decompression, the prefix part is replaced with the original data, restoring the uncompressed data. This method is effective for data with consecutive occurrences of specific values [6].

The number of compression patterns can be increased by adjusting the length of prefixes corresponding to run lengths. However, this may lead to a decrease in compression ratio proportional to the length of the prefixes. Particularly, for parts of the data that do not match any compression pattern, if the prefix and data parts are always output as a set, the data size may become larger than the original data by the length of the prefix.

Regarding prefixes, ensuring complete differentiation during the expansion process is crucial. Failure to distinguish prefixes from other parts may result in incorrect restoration. Depending on the characteristics of the original data and the compression algorithm, it is essential to eliminate such possibilities during the design phase.

One method to distinguish prefixes is to reserve a part of the compressed data as a code section. For example, when compressing to 32 bits, reserving the initial bits as a code section is one way to achieve this. While this method allows flexibility in determining compression pattern content for any original data, it may lead to inefficiency due to including code section data, even for parts that do not match the pattern.

Here, as an example of run-length expression, we introduce a simple method called FPC (Frequent Pattern Compressor). FPC is designed for Run Length compression of 32-bit integer data. It targets situations where zeros are likely to appear consecutively near the beginning of the data. FPC achieves compression by replacing consecutive zero parts with a 3-bit prefix. The most efficient prefix corresponds to data with 18 or more leading zeros, compressing 32-bit original data to 17 bits [7].

Table 1 shows the prefixes and compression patterns used in FPC. The method efficiently handles cases where a sequence of zeros is longer than 18 bits. The unused prefix and the uncompressed category provide additional flexibility.

Table 1: Prefixes and Compression Patterns in FPC

Prefix	Pattern encoded	Data size after encoding
101	18 bits or more leading zero	17 bits
100	17 bits leading zero	18 bits
011	16 bits leading zero	19 bits
010	15 bits leading zero	20 bits
001	14 bits leading zero	21 bits
000	13 bits leading zero	22 bits
111	Uncompressed	35 bits
110	(Unused prefix)	

2.2 Compression for FPGA configuration data

The compression of FPGA configuration data has been a research topic since the 1990s due to the large amount of configuration data, often containing many repetitive patterns. Xilinx supported configuration data compression and decompression solution[8]. It uses Lampel-Zip for saving storage, and the decompression is done with dedicated hardware called System Ace[9].

However, recent commercial FPGAs, especially those used for embedded purposes, typically load configuration data serially from external flash memory. FPGAs used as accelerators often transfer data rapidly from the host via PCIe, and both scenarios usually have sufficient external memory space. As a result, the necessity for compression and decompression has diminished. Consequently, recent research on compression and decompression methods often targets specialized FPGAs.

Studies by [4] and [10] propose compression and decompression algorithms for the Xilinx XC6200 FPGA. The XC6200 is a unique FPGA that allows the Configuration Register, where configuration data is stored, to be mapped to the host's memory. By rewriting this register, the hardware configuration can be changed rapidly. The proposed method utilizes the XC6200's configuration data, composed of address and data pairs, and employs a run-length compression method.

In [6], a compression and decompression circuit for sending configuration data to a small original FPGA via JTAG is proposed. The use of serial transmission allows for flexible run-length compression using a 4-bit prefix. While this work provides detailed hardware data, the proposed method is more complex than ours.

[5] employs a sophisticated compression and decompression circuit using a dictionary-based algorithm in the LZ family, achieving high compression rates. However, this method requires substantial hardware resources. [11] presents a compression method for the FDP2009-II-SOPC, a new FPGA. Similar to the XC6200, this FPGA uses configuration data consisting of addresses and data, and the proposed compression method takes advantage of this characteristic. Notably, the architecture of this FPGA is designed with compression of configuration data in mind.

[12] proposes a configuration data compression method for CGRAs (Coarse-grained Reconfigurable Arrays). The method aggregates the configuration of identical processing elements when they occur consecutively, achieving significant compression ratios.

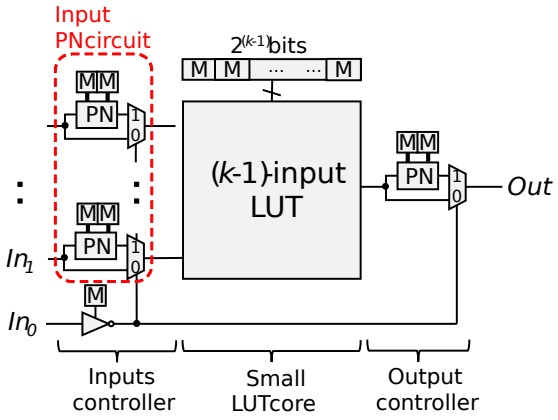

Fig. 2: Structure of the SLM

3. SLM

Kumamoto University developed SLM (Scalable Module Logic) in 2014 as a new configuration method for FPGAs. Based on the LUTs adopted in existing FPGAs, it reduces the configuration information's size to enhance the logic cell's area efficiency. An SLM with K inputs is composed of a K-1 input LUT and w+1 Programmable NAND (PN) circuits. In this case, while a K-input LUT requires 2^k configuration information, an SLM with the same number of inputs can make do with 2^{k-1} + 2w+2.

In SLM, specific operations are performed on partial functions obtained by expanding the logical function through Shannon expansion, as expressed below:

$$f = \bar{x} \cdot f_{(x=0)} + x \cdot f_{(x=1)}$$

The allowable operations involve the inversion or assignment of constants for any input variable or output. As a result,

$$f_{(x=0)} = f_{(x=1)}$$

SLM implementation is possible only for logical functions where this equality holds.

The array structure of the SLM is shown in Figure 3. Each tile provides LB (Logic Block), Connection Block on Top side (CBT), Connection Block on Right side (CBR) and SB (Switch Block). The detail of the structure is shown in the paper[13].

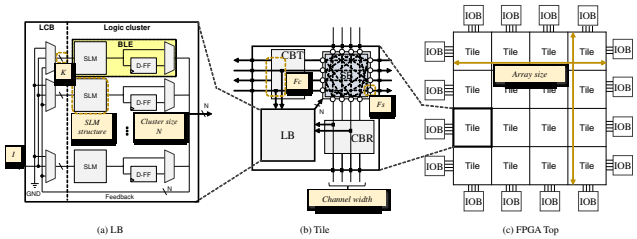

Fig. 3: Structure of Logic Block, Tile and Array

3.1 The SLM in the SLMLET-1

SLMLET-1, the prototype of SLMLET provides two FPGA-IPs, each of which is shown in Fig. 4. The tile size of each FPGA-IP is 16 × 16, thus, SLMLET-1 provides 256 × 2 tiles. Added to the fundamental structure, it provides 4 DSP tiles including multipliers. Furthermore, LB includes a 4-bit ripple carry adder, and all adders are cascaded by carry chains. Here, the LB is further divided into LCB (Local Connection Block), BLEs (Basic Logic Elements), and CYINT (CarrY-IN selecTor). IOB (Input Output Block) includes 10 IOEs (Input Output Elements) for the input/output port of the FPGA-IP.

In SLMLET-1, configuration data sets are stored in the local memory and serially transferred to two FPGA-IPs in parallel. This action is triggered by RISC-V software, while the configuration operation itself is controlled by the hardware. Since each FPGA-IP requires 103,880 bits and SLMLET works at 200MHz clock, it takes 520μsec to configure two FPGA-IPs. It is much faster than those of the commercial FPGAs. Also, the total configuration data is about 2/3 of the common FPGAs with 5-LUTs[2].

Figure 4 also shows the path for the configuration. Arrows indicate the order of configuration from `CONF_IN` to `CONF_OUT`. In other words, inside the memory, configuration data are arranged in the reverse order of the arrows. In other words, when reading the configuration data from the beginning, the first part encountered is the Tile 1616 section.

3.2 Compression Strategy

While research on compressing and decompressing FPGA configuration data has been conducted [6][10][4][5], these methods are architecture-dependent and may not be suitable

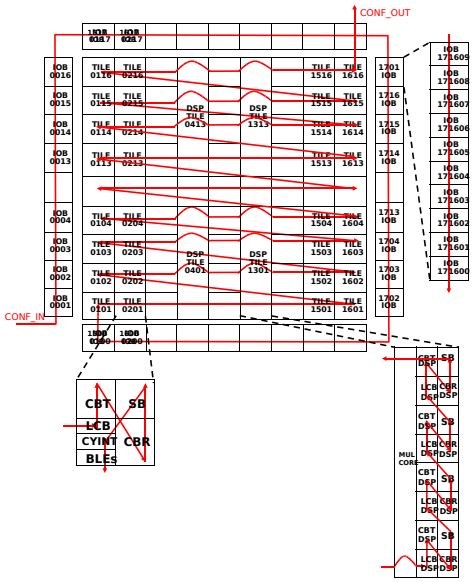

Fig. 4: Structure of an FPGA-IP and configuration path

for SLM whose structure is different from traditional FPGAs. On the other hand, there is a commonality between the configuration data of SLM and the one of existing FPGAs. This commonality lies in the high proportion of zeros. This characteristic is particularly pronounced in the configuration data of simple circuits, and for such cases, a specialized Run Length encoding of consecutive zeros is expected. For this aim, we proposed a general compression strategy called Tag Less Compression or TLC, and have implemented it in SLMLET-1.

However, for complex circuits, the compression ratio of TLC becomes low. Thus, we must make use of the architectural features of the SLM. In the configuration data, LCB, SB, and LB modules occupy a significant portion of the configuration data, around 70% of the entire array. Taking advantage of this, we propose using a run-length compression called Duplicated Module Compression or DMC, which provides a dictionary for modules that appear multiple times. The LCB, responsible for connections within the logic block, has fewer duplications and a higher proportion of zeros. Therefore, we use the TLC for this part.

4. Compression Methods

4.1 Tag Less Compression

For SLMLET, compression can be done outside the chip, while on-the-fly decompression needs to be done inside the chip. Thus, the compression method with simple decompressing hardware is required. Also, the compressed configuration data must be efficiently stored in the memory for RISC-V CPU. Considering these requirements, we propose a TLC (Tag Less Compression) compression method.

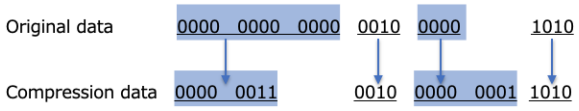

Fig. 5: Tag Less Compression (TLC)

TLC is a type of run-length compression that targets patterns with n-bit continuous zeros. Here, we explain with a 4-bit version shown in Figure 5). When more than four continuous zeros are found in the original data, '0000' is first outputted. After that, the number of consecutive patterns of '0000' is specified by 4 bits. In the example of Figure 5, the pattern '0000 0000 0000' is compressed into '0000 0011'. '0011' shows the number of continuous '0000's. To represent the consecutive occurrence of "0000" up to 15 times, a 4-bit pattern is used. This allows the representation of patterns with up to 15 consecutive occurrences using 8 bits. Patterns that are not the target of compression are outputted as they are. On the contrary, the worst-case scenario is when "0000" appears only once. In this case, the conversion of a 4-bit string to 8 bits would result in a decrease in compression efficiency as shown in Figure 5. One of the features of this method is that it does not use tags. Data are handled in 4-bit units by not using tags and are easy to handle with most CPUs. This ability facilitates both compression and decompression implementations. The problem with TLC is that it can increase the data amount if only one '0000' frequently appears.

4.2 Duplicated Module Compression

TLC has been implemented in SLMLET-1 and is available in the actual chip. However, as shown later, the compression ratio of TLC is low when the complicated design is implemented. For further compression, we need to use the features of SLM configuration data.

The FPGA-IP consists of tiles and IOBs as shown in Figure 3. The configuration data of the tiles accounts for a significant portion of the entire bitstream data. Therefore, we aimed to compress the configuration data of the tiles and proposed DMC (Duplicated Module Compression).

Each tile, as shown in Figure 3, comprises multiple modules. Switch Blocks (SB) and Logic Blocks (LB) exhibit duplications with the same modules in other tiles. Hence, we performed a run-length compression by creating dictionaries for these modules. Additionally, while Local Connection Blocks (LCB) rarely show duplications with the same modules in other tiles, they have a high percentage of zeros. Therefore, we applied a similar approach to TLC.

The DMC uses a dictionary that associates configuration data with prefixes. For example, we show the case of LB, which has 21-bit configuration data. For each configuration data for an LB, we provide a variable length prefix. Here, we provide a directory with n entries, as shown in Table 2. Here, let n be nine. List the top nine configuration data with high duplicate frequencies for LB and assign them variable prefixes in sequence, starting with 01, 001, 0001, ... 00000001. Furthermore, since the compression speed is not crucial in this paper, the LB part is read in advance to create the dictionary. We replace the target configuration data with the corresponding prefix in the compression step in the server which is used for SLM design. Compared with the time for the place&route step of SLM design, the time for the compression in the server is negligible.

In the decompression step, when the top bit is '1', it shows uncompressed data, so output the following 21-bit data as it is. This top bit is added as needed. When the first bit is '0,' we scan the bit pattern until we find '1' and get the prefix. Then, we refer to the dictionary and change the prefix with the corresponding 21-bit configuration data.Since the configuration data is serially sent to the FPGA-IP, we used the variable length prefix. This approach aims to improve compression rates by inserting the shortest '1' as a marker for patterns not in the dictionary. It means outputting a "1" before the configuration data. We assigned lengths based on the descending order of occurrence frequencies for other patterns. Additionally, for LB's configuration data handled in 21-bit units, we limited the prefix length to 10 bits ($n = 9$). This restriction is based on the criterion that the compression rate improves as long as the target configuration data appears

at least twice.

The same procedure was applied to the SB by creating dictionaries. Since SB's configuration data is handled in 45-bit units, we limited the prefix length to 22 bits.

Table 2: An example of a dictionary for LB

prefix	configuration data
01	001100010001010000000
001	000001011000010000010
⋮	⋮
000000001	001111101111110000100

Next, Table 3 illustrates the relationship between control signals and selected signals in the configuration of LCB. Each LCB has 20 outputs, each assigned a 5-bit control signal. Despite half of the control signals being allocated to GND on the format, they are not used in practice. We attempted compression focusing on this feature because nearly half of LCB's outputs select GND. Specifically, we utilized the unused control signals from 10001 onwards. During compression, when GND is selected or when 00000 occurs consecutively, we replace them with 5-bit data ranging from 10001 to 11111, depending on the occurrence count. As shown in 4, 10001 represents two consecutive occurrences, and 11111 represents 16 consecutive occurrences. Since these 5-bit data are not used for the original control signals, they will not appear in LCB's configuration data for other purposes. Therefore, during decompression, if the corresponding bit sequence appears, outputting 00000 for the corresponding number of times allows for restoration. This processing is similar to that of 5-bit TLC.

Table 3: Relation of Configuration data and selected signal on LCB

Configuration data	Selected signal
00000	GND
00001	GI[0]
00010	GI[1]
⋮	⋮
01100	GI[11]
01101	LI[0]
01110	LI[1]
01111	LI[2]
10000	LI[3]
10001	GND
10010	GND
⋮	⋮
11111	GND

Note that we did not compress CBT and CBR, because they do not occupy a large part of configuration data and have a small degree of similarity.

4.3 Compression Ratio Evaluation

We measured the compression ratio for circuits implemented on the actual SLM reconstruction logic in the evaluation process. The targeted SLM, shown in Figure 3 has 256 tiles, and the entire configuration can realize a logic circuit equivalent to 5k gates. In SLMLET-1, two FPGA-IP were implemented and the same decompression circuits are provided and work in parallel.

Table 4: Relation of compressed data and original configuration data on LCB

Compressed data	original configuration data
00000	00000
00001	00001
00010	00010
⋮	⋮
10000	10000
10001	Repeat 00000 2 times
10010	Repeat 00000 3 times
⋮	⋮
11111	Repeat 00000 16 times

We compared the compression ratios for the configuration data of circuits implemented on SLM, including anonymization circuits (ldp, dp), multiplication (mul), communication protocol-related circuits (mac), and encryption circuits (aes). Additionally, we calculated the compression ratio for circuits from publicly available benchmarks LGSynth[14] and iwls05[15]. The tools used to generate configuration data are freeware and tools developed at Kumamoto University [16]. As an example of Lampel-Ziv compression, we used gzip command on Linux. Note that gzip is difficult to implement in a small chip, and it shows the upper limit of the compression. Also, we selected FPC[7] shown in Section 2 as an example of another run-length compression that can be implemented in a small chip. The compression ratio is calculated by dividing the original data's size by the compressed data's size. In other words, a larger value indicates a better compression efficiency.

Table 5: Compression Ratio Measurement Results

Original Data	Compression Ratio			
	DMC	TLC	gzip	FPC
dp	1.31	1.22	1.43	1.05
dp-chain	1.33	1.25	1.45	1.06
mac	2.06	2.48	3.61	1.39
mul	2.45	3.06	5.52	1.49
ldp	1.59	1.62	1.31	1.97
0.aes	1.22	1.04	1.28	0.97
1.aes	1.20	1.01	1.21	0.96
LGSynth				
s298	2.55	3.26	6.53	1.52
s1494	2.56	3.28	6.89	1.53
iwls05				
s9234	1.70	1.65	2.04	1.20
s13207	1.61	1.67	1.94	1.18

As shown in Table 5 and Figure 6, TLC tends to achieve higher compression ratios for simple designs whose configuration data includes a lot of consecutive zeros. Sometimes,

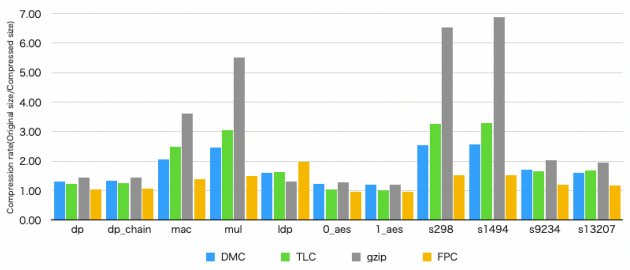

Fig. 6: Compression Ratio with SLM Configuration Data

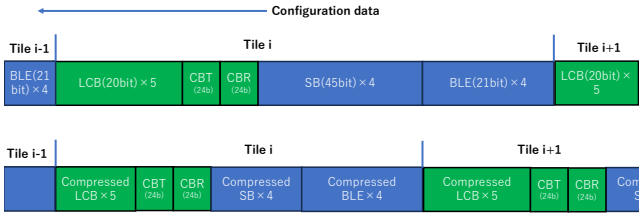

Fig. 7: The configuration data format without and with DMC compression

it is better than DMC because it does not have tags, which sometimes becomes the overhead. Compared to the FPC, it achieves a constantly better compression ratio.

On the other hand, it cannot compress the configuration data for complicated designs like AES at all. On the other hand, DMC, though not outperforming TLC in simple circuits like mac and mul, demonstrates stable compression ratios even for circuits where TLC struggles. The compression ratio is smaller than that of gzip, but closes or better in some cases. If the target designs are simple, we can save the local memory and more number of configuration data sets according to the ratio shown in Table 5.

5. Implementation on Hardware

5.1 DMC Implementation

The configuration compression proposed here aims to store the configuration data set in the local memory as much as possible. The time of initial transfer of the configuration data set from the outside chip to the local memory is reduced in correspondence with the compression rates shown in Table 5. However, in SLMLET, dynamic reconfiguration is done with the inner-chip configuration. The inner-chip transfer time remains constant regardless of compression, as shown in Section 3.1, at 520μsec. This value is smaller than that of conventional FPGAs due to the following factors: the initially small configuration data of SLM, the higher frequency of chip-internal transfers at 200MHz compared to the clock for normal configuration data settings, and the ability to perform in parallel with two FPGA modules. It is not the purpose of this proposal to further reduce this value by compressing it.

The decompression must be done on the fly when the configuration data is sent to the FPGA-IP so as not to degrade the high-speed configuration. The configuration data is stored in the local memory of SLMLET Tile-by-Tile as shown in Figure 7, and transferred to the SLM module in a bit-serial manner. The upper data format is the original one,

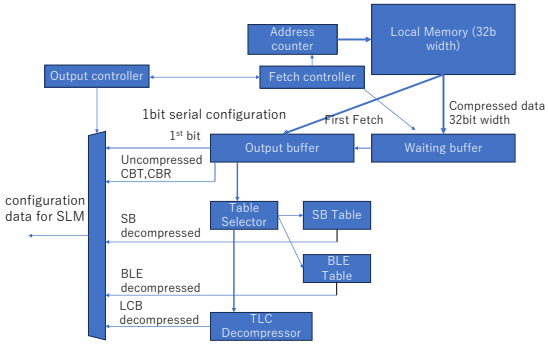

Fig. 8: Block diagram of the decompression circuit

and the lower one is the DMC compressed format. Note that the configuration data for I/O modules and DSPs are omitted in the figure. The transferred order is shown by the red arrow in Figure 3.

In Figure 8, an overview of the decompression circuit is presented.

The 32-bit data in the local memory is fetched to the waiting buffer and serially transferred to the output buffer. The output buffer is a shift register, but all bits can be observed in parallel. Based on the several bits in the output buffer, the appropriate table is selected and the referred data from the table is serially transferred. When the LCB is decompressed, the TLC decompressor shown in Figure 9 is used.

In the case of TCL decompression, 00000 is the mark of the compression and the following 5-bit shows the continuous number of 00000. Thus, the 00000 in the register is iteratively outputted until the counter becomes zero as shown in Figure 9a). Otherwise, the number in the register is serially outputted as it is.

32-bit data is read out from the local memory, while the decompressed data is transferred serially with the same clock. Therefore, the decompression hardware can operate with ample time margin.

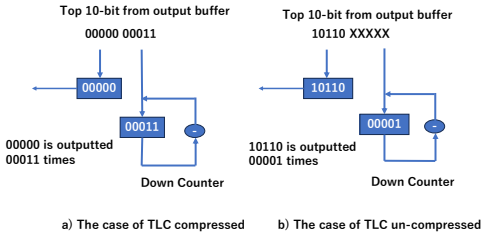

Fig. 9: Operation of TLC decompression

Table 6: Area of Decompression Circuits

Algorithm	Area (μm^2)
DMC	708.9
TLC(SLMLET)	664.8
FPC	5147

5.2 Evaluation of Circuit Area

The logic circuits for decompressing the data from TLC and DMC were described in Verilog HDL and synthesized for the NANGATE 45nm process. The logical synthesis was performed using Synopsys Design Compiler N-2017.09-SP1. Although the decompression circuit is integrated into the configuration setting circuit of the SLMLET-1 chip for practical use, in this evaluation, the circuit was designed to operate independently at 1GHz for assessment purposes. Under the same conditions, the circuit area was evaluated alongside the FPC (Frequent Pattern Compression) circuit used in the literature [17]. Note that all tables of the DMC decompressor are implemented with registers and the area is included in the Table 6.

As shown in Table 6, DMC, in addition to compressing the LBC portion equivalent to TLC, involves table references, but the increase in area is not significantly high. The local memory size of SLMLET-1 is 20684 μm. SLMLET-1 is designed with 55nm process, while the above evaluation is done with 45nm process. However, considering the difference in the process size, the area of the decompression circuits is sufficiently small.

6. Conclusion

In this paper, we aimed to further compress the configuration data of SLM, which originally had excellent area efficiency, to enable the storage of multiple configuration data in the chip's memory. Compared to existing methods that switch logic circuits by reading from external memory, using configuration data stored within the chip results in shorter required times. As the decompression needs to be performed online, the decompression algorithm also requires to be simple.

To address this, we proposed compression methods, TLC and DMC, based on the assumption of SLM reconstruction logic. We implemented these algorithms in C and evaluated the compression ratios. For the decompression circuit, we simulated its operation using Verilog, set a target delay for 1GHz operation, and evaluated the area using Synopsys' Design Compiler with the NANGATE45nm process.

As a result, we found that TLC and DMC decompression circuits could be implemented with relatively small circuit areas of 664.8 μm^2 and 708.9 μm^2, respectively.

Acknowledgment

This work was supported by JST CREST Grant Number JPMJCR19K1, Japan. CAD is supported by Synopsys through VDEC.

References

[1] M. Amagasaki, R. Araki, M. Iida, and T. Sueyoshi, "SLM: A scalable logic module architecture with less configuration memory," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol.99, no.12, pp.2500–2506, 2016.
[2] Q. Zhao, K. Yanagida, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi, "A logic cell architecture exploiting the Shannon expansion for the reduction of configuration memory," 2014 24th International Conference on Field Programmable Logic and Applications (FPL), pp.1–6, IEEE, 2014.
[3] S.Takagi, N.Niwa, Y.Yanai, H.Amano, M.Amagasaki, Y.Nakazato, and M.Iida, "Tag-less compression for FPGA configuration data," Proc. of SASIMI 2022, pp.1–2, 2022.
[4] S. Hauck and W.D. Wilson, "Runlength compression techniques for FPGA configurations," Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines (Cat. No. PR00375), pp.286–287, IEEE, 1999.
[5] A. Dandalis and V.K. Prasanna, "Configuration compression for FPGA-based embedded systems," Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays, pp.173–182, 2001.
[6] R. Jia, F. Wang, R. Chen, X.G. Wang, and H.G. Yang, "JTAG-based bitstream compression for FPGA configuration," 2012 IEEE 11th International Conference on Solid-State and Integrated Circuit Technology, pp.1–3, IEEE, 2012.
[7] A. Alameldeen and D. Wood, "Frequent pattern compression: A significance-based compression scheme for l2 caches," tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 2004.
[8] A. Khu, "Xilinx FPGA Configuration Data Compression and Decompression," tech. rep., Sept. 2001.
[9] E. Thacker, "System Ace: Configuration Solution for Xilinx FPGAs," tech. rep., Sept. 2001.
[10] S. Hauck, Z. Li, and E. Schwabe, "Configuration compression for the Xilinx XC6200 FPGA," Proceedings. IEEE Symposium on FPGAs for Custom Computing Machines (Cat. No. 98TB100251), pp.138–146, IEEE, 1998.
[11] X. Jing, W. Yabin, C. Liguang, W. Jian, W. Yuan, L. Jinmei, and T. Jiarong, "Fast configuration architecture of FPGA suitable for bitstream compression," 2009 IEEE 8th International Conference on ASIC, pp.126–130, IEEE, 2009.
[12] K. Tanigawa, T. Kawasaki, and T. Hironaka, "A coarse-grained reconfigurable architecture with low cost configuration data compression mechanism," Proceedings. 2003 IEEE International Conference on Field-Programmable Technology (FPT)(IEEE Cat. No. 03EX798), pp.311–314, IEEE, 2003.
[13] M.Kuga, Q.Zhao, Y.Nakazato, M.Amagasaki, and M.Iida, "An eF-

PGA Generation Suite with Customizable Architecture and IDE," IEICE Transactions Fundamentals, vol.E106-A, no.3, pp.1–17, 2023.
[14] S. Yang, "Logic synthesis and optimization benchmarks user guide: Version 3.0," tech. rep., MCNC Technical Report, Jan. 1991.
[15] C. Albrecht, "IWLS 2005 Benchmarks," tech. rep., June 2005.
[16] Q. Zhao, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi, "Towards Open-HW: A Platform to Design, Share and Deploy FPGA Accelerators in Low Cost," IPSJ Transactions on System LSI Design Methodology, vol.10, pp.63–70, 2017.
[17] A. Alameldeen and D. Wood, "Frequent pattern compression: A significance-based compression scheme for l2 caches," tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 2004.

Souhei Takagi received the B.E. degree in Department of Information Engineering from Keio University, Japan in 2022. He is currently a M.S. student in the Department of Information and Computer Science, Keio University.

Takuya KOJIMA received his B.E., M.E., and Ph.D. degrees from Keio University, Japan, in 2017, 2019, and 2021, respectively. Since 2021, he has been an assistant professor at The University of Tokyo, Japan. He is currently a JST PRESTO researcher as a concurrent post. His research interests include optimization methods, especially for reconfigurable devices, and heterogeneous computing with 3-D stacked LSI. He is a member of IEEE and IEICE.

Masahiro Iida received his B.E. degree in Electronic Engineering from Tokyo Denki University in 1988. He was a research engineer at Mitsubishi Electric Engineering Co., Ltd. from 1988 to 2003. He received his M.E. degree in Computer Science from the Kyushu Institute of Technology in 1997. He received his D.E. degree from Kumamoto University, Japan, in 2002. He was an associate professor at Kumamoto University until 2015. From 2002 to 2005, he held an additional position as a researcher at PRESTO, Japan Science and Technology Corporation (JST). He has been a professor with the Faculty of Advanced Science and Technology at Kumamoto University since January 2016. His current research interests include high-performance, low-power computer architectures, reconfigurable computing systems, and VLSI devices and design methodologies. He is a senior member of IPSJ and IEICE, and a member of IEEE.

Morihiro Kuga received his B.E.degree in Electronics Engineering fromFukuoka University in 1987. Further, hereceived his M.E. and D.E. degrees in In-formation Systems from Kyushu University in 1989 and 1992, respectively. From1992 to 1998, he was a lecturer at KyushuInstitute of Technology. He has been anassociate professor at Kumamoto Univer-sity since 1998. He has been belonging to the Faculty of Advanced Science andTechnology since 2016. His research interests include parallel processing, computer architecture, reconfigurable system, and VLSIsystem design. He is a member of IEICE and IPSJ.

Hideharu AMANO received Ph.D degree from the Department of Electronic Engineering, Keio University, Japan in 1986. He is currently a professor in the Department of Information and Computer Science, Keio University. His research interests include the area of parallel architectures and reconfigurable systems.