BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF CONTROL AND INSTRUMENTATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DATA STREAM COMPRESSION AND DECOMPRESSION METHODS.

REAL TIME KOMPRESE A DEKOMPRESE INFORMACÍ V DATOVÝCH TOCÍCH

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — **Oleksandr Makedonenko**
AUTOR PRÁCE

SUPERVISOR — **Ing. Soběslav Valach**
VEDOUCÍ PRÁCE

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: Oleksandr Makedonenko ***ID:*** 175207
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Real time komprese a dekomprese informací v datových tocích

POKYNY PRO VYPRACOVÁNÍ:

Cílem projektu je snížit alespoň přechodně bezeztrátově datový tok v komunikačním kanále a to tak, aby se mohly přenášet servisní informace a nekritické datové toky. Příkladem nechť je datová linka o maximální propustnosti dat 1Gbit/s saturovaná na 98-100% své nominální kapacity.
1) Prostudovat metody bezeztrátové komprese dat a signálů.
2) Vybrat vhodnou metodu, která bude využitelná pro hradlové pole s přiměřenými nároky na spotřebu zdrojů a definovanými parametry.
3) Implementovat alespoň jednu metodu pro přijímač a vysílač.
4) Zaměřit se na řešení okrajových podmínek (rozpojení přenosové trasy, výpadek, chyba).
5) Demonstrace výsledků na hardwarové platformě.

DOPORUČENÁ LITERATURA:

1. Joe-Ming Cheng: Contributions to binary adaptive-coding, sliding window hardware compression, Huffman coding redundancy bounds, and hybrid arithmetic coding. Sptember 9, 2011

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 20.5.2019

Vedoucí práce: Ing. Soběslav Valach
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

Abstrakt

Cílem tento práce je prostudovat metody bezztrátové komprese a zmenšit datový tok ve komunikačním kanále, prováděním bezztrátové algoritmu komprese, který může být použity pro FPGA desku s teoretickým dosahem rychlosti 1Gbit/s.

Klíčová slova

Bezeztrátová, komprese, algoritmus, LZW, LZ77, LZ78, Huffman, Deflate, FPGA, MyHDL

Abstract

Goal of this work is to study lossless compression methods and to reduce data flow in communication channel by implementing lossless compression algorithm that can be useable on FPGA board with theoretical achievement of speed 1 Gbit/s.

Keywords

Lossless, compression, algorithm, LZW, LZ77, LZ78, Huffman, Deflate, FPGA, MyHDL

Bibliografická citace:

MAKEDONENKO, Oleksandr. *Real time komprese a dekomprese informací v datových tocích.* Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/119332. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Soběslav Valach.

Prohlášení

„Prohlašuji, že svou diplomovou (bakalářskou) práci na téma “Real time komprese a dekomprese informací v datových tocích” jsem vypracoval samostatně pod vedením vedoucí/ho diplomové (bakalářské) práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové (bakalářské) práce dále prohlašuji, že v souvislosti s vytvořením této diplomové (bakalářské) práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **13. května 2019**

……
podpis autora

ACKNOWLEDGMENTS

I would like to thank my supervisor Ing. Soběsalv Valach for help and tips during my work. And I would like to thank my relatives for moral and material support.

Table of contents

List of abbreviations

FPGA	…	field-programmable gate array
Gbit	…	gigabit
LZ77	…	Lempel-Ziv 77
LZ78	…	Lempel-Ziv 78
LZW	…	Lempel-Ziv- Welch
VHDL	…	(Very high-speed integrated circuits) Hardware Description Language
NYT	…	Not Yet Transmitted

List of figures

List of tables

1. INTRODUCTION

The goal of this work is to reduce data flow in the communication channel in order to transmit service information and non-critical data flows. To accomplish this goal data stream compression will be used. For this reason, several lossless data compression methods will be studied and compared, then one will be chosen and implemented on a FPGA board.

Data compression is a very important tool in information technologies. As this work will demonstrate, data compression is very useful for transmitting data. Very often data in different solutions struggle form limitations in transfer speed. Data compression can encode data based on predictable patterns and order, to reduce amount of data needed to transmit the same amount of information. By decreasing total amount of data needed to be sent, we can decrease amount of time needed for transmitting same amount of information.

Data compression is a software solution for this problem, that's why this is a very cheap and efficient solution for increasing amount of data flow in channel, as well as effective one.

A chosen compression algorithm would be testes in simulation environment, with a goal achieving speed of 1 Gbit/s.

2. COPMRESSION METHODS DEFINITION AND CLASSIFICATION

Data compression is a process of data stream processing which generate new data steam that has a smaller size [1]. By term compression algorithm we usually refer to two algorithms compression and decompression (reconstruction) [2].

Compression algorithm analyze data stream X and generate smaller data steam Y, which can be transmitted. Decompression algorithm takes Y as an input and generate reconstructed data stream X' [2]. We can divide all compression methods into two groups: lossy, when X' is different from X, and lossless, when X' is identical as X.

2.1 Lossy compression

Lossy compression methods by definition are not capable to fully reconstruct original data. By using to we accept the loss of some amount of information [3]. This type of compression is used when data loss is not critical and its very common in multimedia file compression, like videos, images and audio. The loss of data can be unnoticed due to human perception.

Also, lossy methods are not universal for every type of data and they are not suitable for critical data, when we cannot tolerate loss of a single bit. Further this type of compression will not be considered, because it's unsuitable for purposes of transmitting service data.

2.2 Lossless compression

Opposite of lossy compression, lossless compression encode data in a way, that after decompression output file would be identical to original. This is very useful, because we gain result as a reduced size of data stream and do not modify information carrying by this data [2]. Disadvantage of this methods is that possibilities to compress data is limited, there is some point when further data compression is not possible without a loss of information. Basically, we cannot compress data stream to a single bit.

3. LOSSLESS COMPRESSION TYPES

There are a bunch of methods that are used for data compression. Further will be described several methods that are commonly used in data compression.

3.1 Entropy coding type

Entropy in information theory determining uncertainty of the system, therefore how unpredictable the system is. The concept of this principle was described by Claude Shannon in 1948 in the work “A Mathematical Theory of Communication” [4]. Also, similar principle was proposed by Robert Fano in the work “The transmission of information” [5].

The idea of this concept is that the information carried in data is a randomness. Data that can be easily predicted, has small amount of information. For example, the stream of bits “0000” carry less information than then “0110”. We can simplify patterns in this data to reduce its actual size, but not affect information. In the case of stream of “0” we can just say that this is a repeat of the same bit. In contrast, in case of more complex second example there is no pattern and we have no choice, but to acknowledge every bit in the stream. Entropy as a value indicates minimum average number of bits per symbol required for encoding (compressing) the string.

C. Shannon and R. Fano proposed compression method – Shannon-Fano coding. This method encodes more frequent symbol with lower number of bits. Before compression whole data package must be analyzed for calculation frequency of the symbols in data, depending on this prefix-free code for every symbol will be generated. David Huffman improved this method to be more optimal and today we know this method as Huffman coding [6].

Data compressed by this method has non-consistent byte length, but this is not a problem during decompression, because symbols are encoded in prefix-free code. This means that we can imagine code for symbols as a binary tree. The flow of this method is that compression depends on pre-compression data analyze and for reconstruction decompression algorithm need to have the results of this analysis as well or it need to have prefix codes generated by compression algorithm. This data must be transmitted

with compressed data, we can provide it in form of “header” before actual compressed data.

For continues data stream that can’t be analyzed before compression, adaptive Huffman coding can be used. In this method symbol codes are generated during compression, without pre-compression analysis. After each symbol adapt code according to Huffman code properties [7][8].

3.2 Dictionary type

Compression method that uses dictionary first was proposed by Jacob Ziv and Abraham Lempel in 1977 – the LZ77 compression algorithm [9]. The idea is to build dictionary that can encode data sequence Algorithm tries to find same symbol sequences in in the data stream and replace repeated sequences in the stream with a “link” that refer to the same previous data sequence, encoding repeated sequence with its length and offset, pointing where the same sequence was already occurred. This algorithm analyze data in the window with finite length, so windows in every step loses data that protentional can be used for compression. Decompressor repeats steps of compressor, following “links” and rebuilding data stream.

There is a family of compression algorithms based on LZ77 – LZ family of compression algorithms. LZ77 was improved by its creatures in LZ78 algorithm. LZ78 can build and remember dictionary without window limitations, like in LZ77. LZ78 remembers sequences found in data and replace same sequences with indexes for its dictionary [10]. Based on this algorithm was created very popular LZW (Lempel-Ziv-Welch) algorithm. The main difference between LZ78 and LZW is that LZ78 build every index in the dictionary from the entry and make new index with a longer sequence only when reference to one of the previous indexes is occurred. LZW already need to have all possible symbols in the dictionary. In every step algorithm take one symbol ahead and tries to find same sequence in the dictionary and makes new entry in the dictionary [11].

Very spread algorithm that is using LZ family compression is the Deflate, developed by Phil Katz [12]. This algorithm uses combination of LZ77 and Huffman coding.

4. LOSSLESS COMPRESSION METHODS IN DETAILS

In this section will be described how selected lossless compression algorithms work in details, with provided examples.

4.1 Huffman coding

As mentioned before in section 3.1, Huffman encoding uses information entropy to reduce usage of data space by generating prefix-free code, that can be represented by binary tree.

Steps to encode data in Huffman coding:

1. Count frequency for every symbol in sequence.
2. Create a leaf node for each symbol and add them to the queue.
3. Repeat, while there is more than node in the queue:
    a. Take the two nodes with the lowest frequency, make priority on new created nodes.
    b. Create a new node with these two nodes as children, with frequency equal to the sum of the two nodes' frequencies.
    c. Add this node to the queue.
4. The remaining node is the root node.

For example, to encode string "BOOKKEEPER", after counting frequency, we can see that symbols "R", "P" and "O" has same lowest frequency of 1. With "B" and "R", we can create new node, with sum of frequency of 2. Prioritizing created node, we connect nodes with the lowest frequency, it's new created node (2) and symbol "P" (1) node, new node connecting them would have frequency of 3. We can see that the lowest frequency nodes are "K" and "O", they create new node of frequency 4. The lowest fervency of 3 now have older created node and symbol "E", both connected in new node of 6. Now we need only connect two nodes with root node [6][1].

Without compression string "BOOKKERPPER" requires 80 bits to transmit, after Huffman encoding only 25 bits.

Table 4-1: Huffman encoding example of string "BOOKKEEPER"

Value	Frequency	Code
E	3	01
K	2	10
O	2	11
P	1	001
R	1	0001
B	1	0000

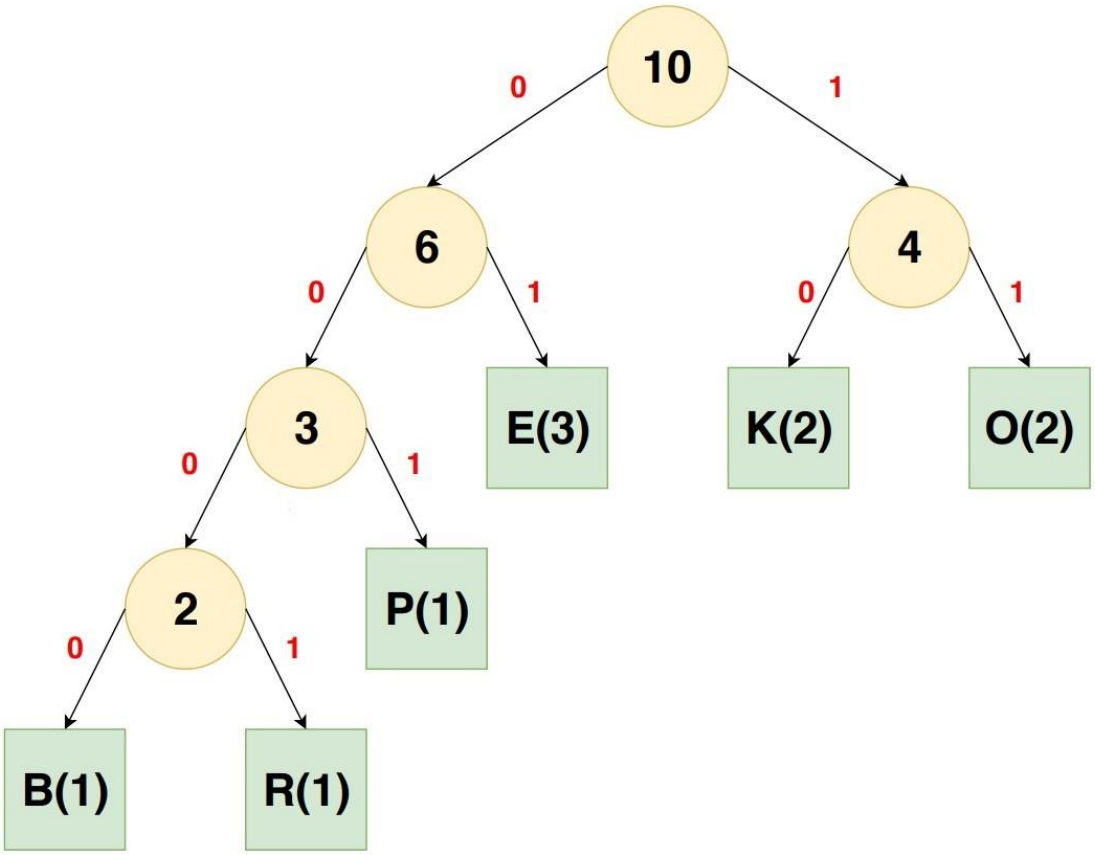

Figure 4-1: Huffman binary tree example of string "BOOKKEEPER"

For decoding process, information about symbols frequencies is required. After rebuilding binary tree, we can decode compress binary data stream just following nodes in the tree.

By using same-length byte encoding we trade efficiency of encoding for purpose of more easily byte representation. This algorithm is a useful tool because it can be sued for reduction number of used bits, that are just rudiment in the data. And as it was proven by Huffman himself, this algorithm can generate the most efficient way to encode symbols.

4.2 Adaptive Huffman coding

Adaptive (dynamic) Huffman coding do not require pre-compression data analysis, like it was mentioned before in section 4.1. Adaptive Huffman coding can adapt existing binary tree according to new data from the input. This binary tree can be updated every step for receiver, that's why dative Huffman coding can be used for real time streaming of compressed data.

Eventually at the end of data stream binary tree generated by adaptive Huffman encoder, would not differ from not-adaptive Huffman encoder. But data output would be different. In this compression algorithm for building binary tree is used special leaf – NYT (Not Yet Transmitted). NYT symbol is a symbol that was not encountered yet, when we transmit NYT symbol, in a first-place algorithm transmit code of NYT leaf in the binary tree, so decompression algorithm will understand that following bits are uncompressed byte. When encountered symbol that was already encoded in the binary tree, algorithm would increase its weight. Nodes with the same weight called blocks. Also, every node has a number, order of numbers is from top to right to left. Root node has number 256.

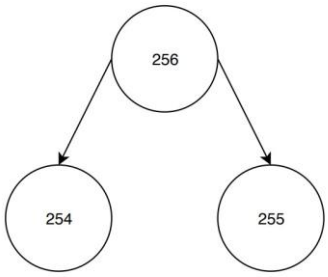

Figure 4-2: Nodes numeration example

Every step binary tree must be updated in according to these steps:

1. If the current character is not encountered, add two child nodes to the NYT: one for the next NYT, the other for the encountered symbol. Increase the weight of the new leaf and the old NYT and go to step 4. If the current character is not NYT, go to the symbol leaf.
2. If this node does not have the greatest weight in the block, change it with the node having the largest number (this means replacing weights and corresponding symbols, but not numbers)
3. Weight gain for current node
4. If it is not the root node, go to the parent node then go to step 2. If it is root, end.

4.3 **LZ77**

In this type of compression algorithm is used slidable window divided by dictionary and buffer, algorithm compare string in buffer with strings in dictionary, it would generate a code referring to previous data sequence, with offset of steps in the dictionary for referring to position of the strings' start, size of this string and symbol after this string in the buffer. If there is no coincidence in the dictionary, offset and size would be 0.

Because first codes would be always uncompressed symbols, decompression algorithm can easily rebuild staring sequence of the compressed data, when it would encounter encoded compressed string, it would already have in the dictionary [9].

This algorithm is very suitable for long repeated sequences, its performance is highly depending on window size.

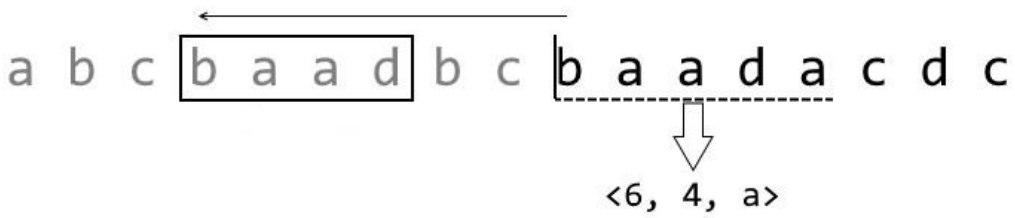

Figure 4-3: LZ77 example

4.4 Lempel-Ziv-Weich algorithm

LZW algorithm builds dictionary on its memory to make a reference to data sequence that previously occurred in the stream, replacing it with one byte instead. Because all information needed for compression provided by compressed data itself, decompression algorithm can reconstruct data by rebuilding dictionary same as compression algorithm.

Steps to compress data by LZW algorithm:

1. Add to table all symbols in the data stream, encoding it like data without compression.
2. Find the longest string in the dictionary that matches current input
3. Send code referring to this string from the dictionary to the output, remove this string from the input.
4. Create new entry in the dictionary, with new string = old string + next symbol
5. Repeat from step 2

As an example, compressing string “ABBBAABBBA” steps are following - adding “A” and “B” with their uncompressed encoding to the dictionary. Read first input - “A”, encode it with 41, add new entry “AB” to the dictionary. Read “B” from the input, encode with 42, add “BB” to the dictionary. Now we can see that string “BB” is identical to following symbol sequence, we encode it according to the dictionary with 81 and make new entry to the dictionary with sting adding to used string next symbol, encode this string “BBA” with 83 [11].

The size of string “ABBBAABBBA” without compression – 10 bytes, after LZW compression – 6 bytes

Table 4-2: LZW encoding example

Value	Code
A	41
B	42
AB	81
BB	82
BBA	83
AA	84
ABB	85

Input	A	B	B	B	A	A	B	B	B	A
Compressed	A	B	82	A	81	83				

Decompression can be tricky, decompression algorithm needs to make new entry in the dictionary to the rules identical to compression, but sometimes can be a situation when decompression algorithm can encounter a code for a sequence that not in the dictionary yet. For example: if we would decomposers sequence form previous example, after reading “B” algorithm would encounter code 82, but at that time there is only code 81 in the dictionary. To resolve this issue, algorithm consider that code 82 can’t be sequence encoded as 81 and code 82 can only have already known symbols. Code 82 first symbol must be “B”, because it was made form second step where “B” is the first symbol.

This algorithm has high performance in the data with repeated sequences, unlike LZ77 algorithm, it’s not limited to the size of window, basically LZW algorithm do not forget previously repeated strings. Downside of this algorithm is a high sensitivity to transmission errors.

4.5 Deflate compression algorithm

Deflate algorithm is a combination of LZ77, Huffman encoding and adaptive Huffman encoding. Compression process completed in two steps. First repeated sequences elimination with LZ77 algorithm. Then bit reduction by Huffman coding. Data

encoded by Deflate algorithm with blocks. Each block has a header of 3 bits. First bit indicates if this block is the last one in the data stream. Other two describe method used for bit reduction: 00 – no compression (copy-paste), 01 – static Huffman, 10 adaptive Huffman. Compression type is chosen by algorithm for each block individually, depending on data complexity [12].

Depending on these two bits following bits encoding would differ. For no compression mode, next 4 bits represent the length of the block by following uncompressed data. In static Huffman mode, data is encoded by predetermined Huffman binary tree, so there is no need to transmit information needed for Huffman binary tree reconstructing. This block must end with zero byte – end of block symbol. For adaptive Huffman encoded block, following 14 bits encoded with information for rebuilding Huffman binary tree [12]:

5 Bits: HLIT, # of Literal/Length codes - 257 (257 - 286)

5 Bits: HDIST, # of Distance codes - 1 (1 - 32)

4 Bits: HCLEN, # of Code Length codes - 4 (4 - 19)

This algorithm is slightly spread and used in many solutions, his main advantage is good ratio of performance and speed. Also, need to mention its flexibility – this compression method can be sued for big virality of data inputs.

The downside of this compression method is its complexity, demanding a big portion of FPGA resources, especially for decompression. But because of its flexibility, it’s possible to trade performance or/and speed for less resources demand. Also, because all data encoded by this compression algorithm are separated in blocks, makes it resistant to transmission error. Even if error is occurred, only corrupted bytes will be vulnerable, but algorithm would continue decompression process, without effect on the rest data in the stream.

5. PRACTICAL IMPLEMENATION

For practical demonstration was chosen Deflate compression algorithm, due to high performance and flexibility of this compression algorithm. Demonstration implemented in FPGA simulation, provided by python library MyHDL v0.10.0. Also, MyHDL library can generate code in Verilog or VHDL for usage on real FPGA.

Note: MyHDL v0.10.0 library has compatibility issues with Python 3.7, because Python 3.7 promoted "async" to reserved keyword, and this causes the conflict. There are two options to resolve this conflict: use Python 3.6 or replace every "async" word in MyHDL library files with "isasync" (preferred).

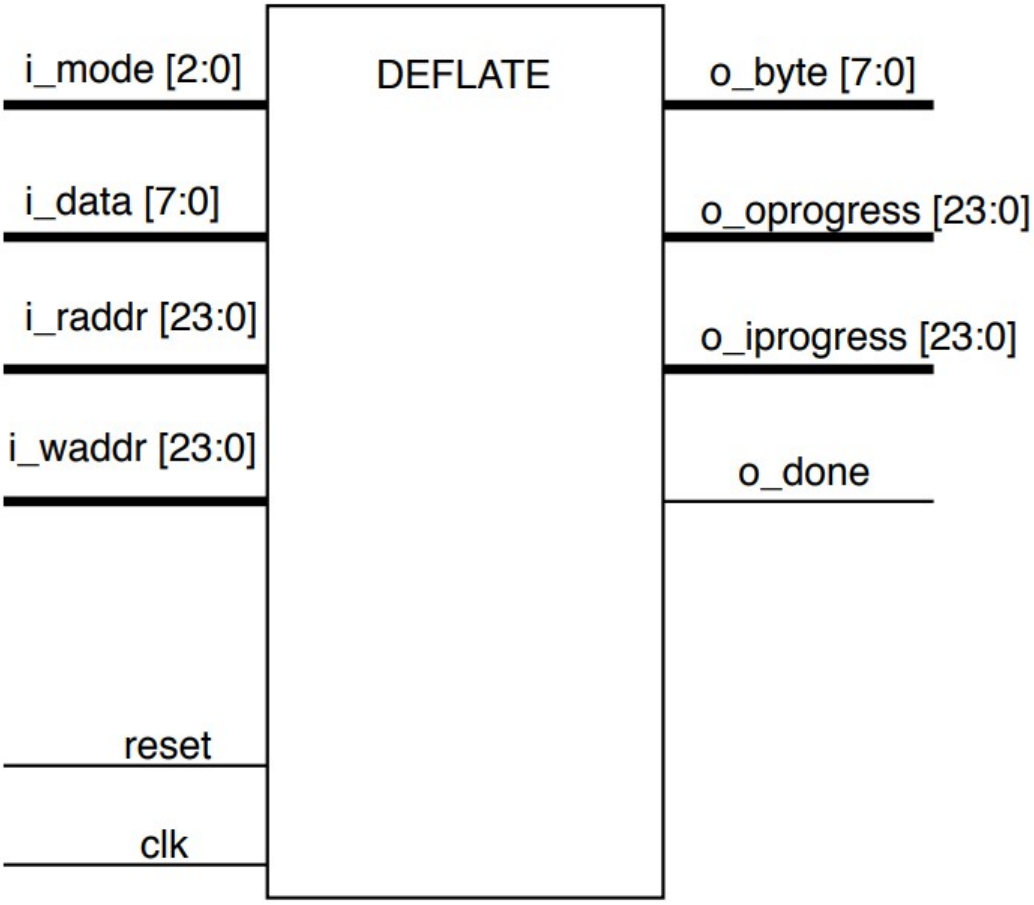

Figure 5-1: DEFLATE logic block

Inputs of Deflate logic block:

- i_mode [2:0] – determine in which mode DEFLATE block need to work. Has 5 states:
  - IDLE [000] - waits, no operation
  - WRITE [001] - reads from input i_data, i_raddr and i_waddr
  - READ [010] – puts byte in output o_byte
  - STARTC [011] – initialize compression state
  - STARTD [100] – initialize decompression state
- i_data [7:0] – data to process
- i_raddr [23:0] – output counter
- i_waddr [23:0] – input counter

Outputs of Deflate logic block:

- o_byte [7:0] – output of processed byte
- o_oprogress [23:0] – output progress
- o_iprogress [23:0] – input progress
- o_done – notice process completion

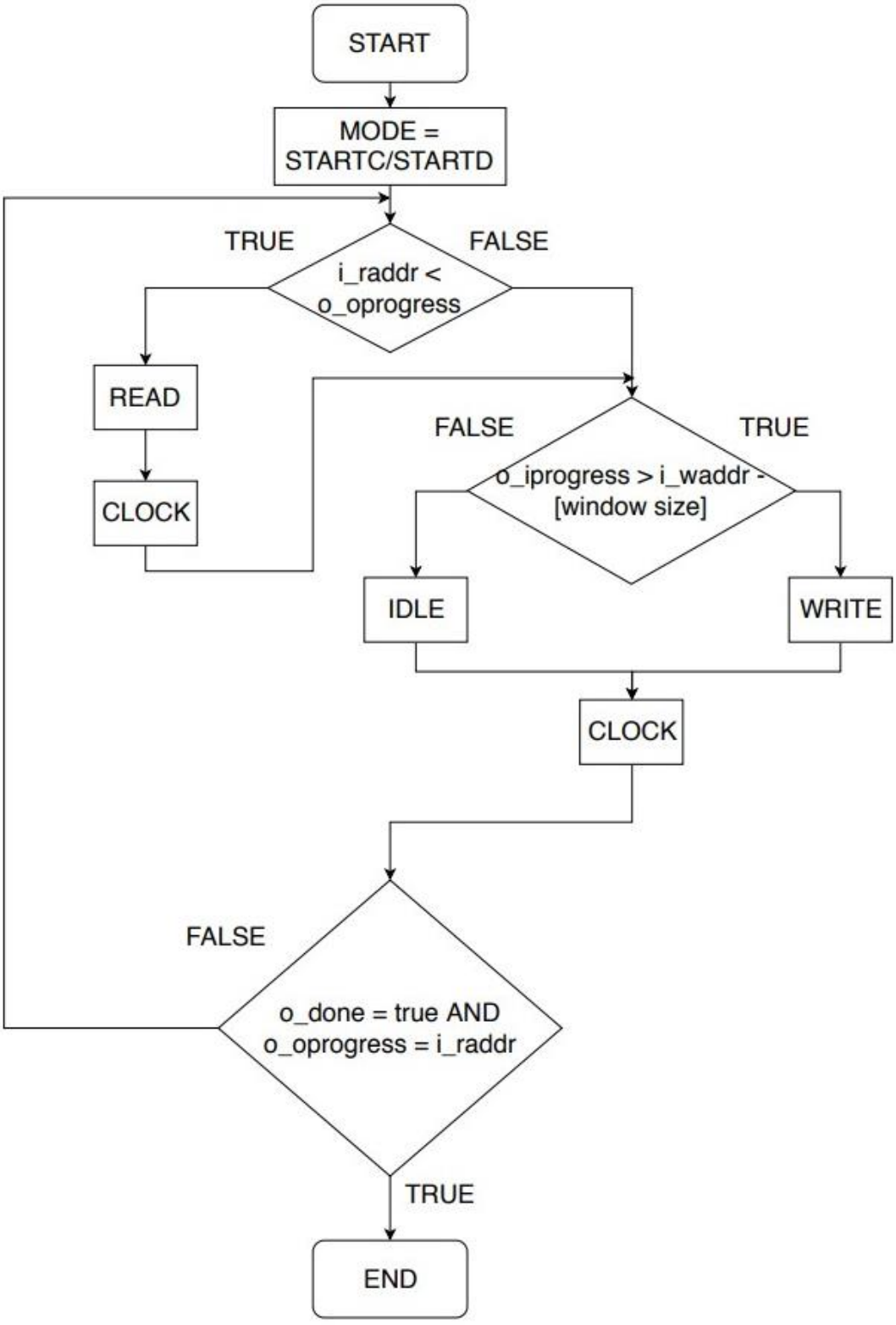

Figure 5-2: DEFLATE work block scheme

For verification workability and speed of DEFLATE compression was tested in simulation environment provided by MyHDL library. For testing was used several data sets: 1 – repeating string, 2- repeating string, with incrementing digits, 3 – repeating string, with random digits in low range.

Table 5-1: DEFLATE compression testing results

No. of string	Type of process	Input [Bytes]	Output [Bytes]	I/O ratio [-]	Cycles [-]	Speed at 250 MHz [Gbit/s]	Speed at 300 MHz [Gbit/s]	Speed at 350 MHz [Gbit/s]
1	Compression	1599	303	5.277	3605	0.887	1.065	1.242
	Decompression	303	1599	0.189	37431	0.023	0.019	0.023
2	Compression	2389	652	3.664	7188	0.665	0.798	0.931
	Decompression	652	2389	0.273	34170	0.046	0.039	0.046
3	Compression	1599	377	4.241	4320	0.740	0.888	1.036
	Decompression	377	1599	0.236	37489	0.020	0.024	0.028

According to test results 300 MHz would be enough for some case to obtain speed 1 Gbit/s, however 350 MHz would be more preferred. Decompression process is much slower compering to compression, but this would not cause problems, because even if decompression algorithm didn't finish their work, it can be feed with more data, that would be stored in RAM. Error handling is also implemented, testing it with corrupted compressed data, would result in loss of corrupted bytes, but process would not stop in most cases. This algorithm achieved decent compression ratio.

6. CONCLUSION

The goal of this work was to reduce data flow in transmission channel of service data, by implementing lossless data compression method and demonstrating its capabilities in FPGA simulation environment.

Lossless compression algorithm could be useful tool in transmission data, due to cheap and easy implementation with a remarkable improving transmission result.

In this work several compression methods and principals were described for better understanding of issue and solution. Such compression methods principles as entropy coding and dictionary coding were studied, with more detail studying of Huffman coding, adaptive Huffman coding, LZ77 algorithm and LZW algorithm.

For main goal of studying was chosen a Deflate compression algorithm, which is a combination of the LZ77 compression algorithm and static and dynamic Huffman coding.

In simulation environment Deflate compression algorithm achieved decent result in compression ratio. Also, it was possible to achieve speed of 1 Gbit/s at 300 MHz, but for 350 MHz is more preferred.

It is possible that an error would occur in communication link, in this situation Deflate decompression algorithm can continue work, only skipping corrupted bytes.

As a result of this work, deflate algorithm implementation on FPGA board theoretically would be real, as well as achieving speed of 1 Gbit/s, in addition MyHDL library could generate Verilog or VHDL code for implementation on a real FPGA board.

Literature

[1] SALOMON, David. "*Data compression: the complete reference*." 3rd ed. New York: Springer, 2004. ISBN 0-387-40697-2.

[2] SAYOOD, Khalid. "*Introduction to data compression*." 5th ed. Cambridge: Morgan Kaufmann Publishers, 2018. Morgan Kaufmann series in multimedia information and systems. ISBN 978-0-12-809474-7.

[3] NELSON, Mark a Jean-Loup GAILLY. "*The data compression book*." 2nd ed. New York: M&T Books, 1996. ISBN 1558514341.

[4] SHANNON, Claude. "A mathematical theory of communication." *Bell System Technical Journal*. 1948, vol. 27 pp. 379–423, 623–656.

[5] FANO, Robert. "*The transmission of information*," Massachusetts, 1949. Technical Report. Research Laboratory of Electronics, Massachusetts Institute of Technology.

[6] HUFFMAN, David Albert. "A method for construction of minimum-redundancy codes," *Proceedings IRE*. 1952, vol. 40, pp. 1098–1101.

[7] FALLER, N., "An adaptive system for data compression," in Record of the 7th Asilomar Conference on Circuits, Systems, and Computers, 1973, pp. 593–597.

[8] GALLAGER, Robert. "Variations on a theme by Huffman. " *IEEE Transactions on Information Theory*. 1978, vol. 24, no. 6, pp. 668-674.

[9] LEMPEL, Abraham a Jacob ZIV, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*. 1977 vol. 23, no. 3, pp. 337–343.

[10] LEMPEL, Abraham a Jacob ZIV, "Compression of individual sequences via variable-rate coding," *IEEE transactions on Information Theory*. 1978 vol. 24, no. 5, pp. 530–536.

[11] WELCH, Terry, "A technique for high-performance data compression," *Computer*, 1984, vol. 17, no. 6, pp. 8–19.

[12] DEUTSCH, Peter, "*DEFLATE Compressed Data Format Specification version 1.3*," 1996, RFC 1951. [Online]. Available: https://tools.ietf.org/html/rfc1951

Attachments

Attachment 1: CD with electronic version of this work, python code of HDL implementation of the Deflate algorithm.