Final Project Report

George Conwell, 661995055
ECSE 6680, Section 1

Background:

Gzip is a file compression and decompression technique that is lossless. It uses the DEFLATE algorithm which comprises Lempel-Ziv-77 (LZ77) and Huffman coding. LZ77 is a lossless data compression algorithm that utilizes the repetition of ASCII characters and substrings. It uses a dictionary to track characters, along with the offset and lengths of substrings inside the main string. A window "slides" along the characters, checking for repetition and adding them to the dictionary. *Figure 1* shows an example of the input string "abracadabrarray" and the window sliding along the string (window = search buffer + look ahead buffer) and the output is the formatted dictionary entries.

1	2	3	4	5	6	7	8	1	2	3	4	5	6	Output
								a	b	r	a	c	a	<0,0,a>
							a	b	r	a	c	a	d	<0,0,b>
						a	b	r	a	c	a	d	a	<0,0,r>
					a	b	r	a	c	a	d	a	b	<3,1,c>
			a	b	r	a	c	a	d	a	b	r	a	<2,1,d>
	a	b	r	a	c	a	d	a	b	r	a	r	r	<7,4,r>
c	a	d	a	b	r	a	r	r	a	y				<3,2,y>

Search Buffer — Look ahead buffer Buffer

Figure 1: LZ77 Compression Example [1]

Huffman coding is a much simpler lossless data compression technique. Where a tree is used to encode characters to reduce the bit complexity required to represent each character. The more frequently used characters are located at the top of the tree, requiring a shorter path to reach, and thus less bits to represent them. In *Figure 2,* it can be seen that the character "A" has a higher frequency than the reset of the characters.

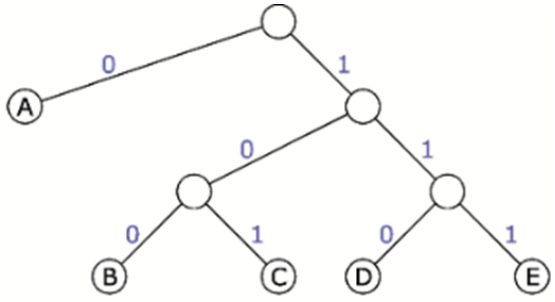

Figure 2: Huffman Coding Example [3]

Putting both these algorithms is fairly simple, where LZ77 is used as the main compression algorithm while Huffman coding is used to reduce the bit representation of the characters within the LZ77 dictionary.

Implementation:

The compression algorithm was implemented in SystemVerilog. The code operated with two modules, which meshed both algorithms to provide one unified result. *Figure 3* shows the general flow of the program, where test binary strings are inputted into an uncompressed file, the modules compress these binary strings, and outputs into a new compressed file.

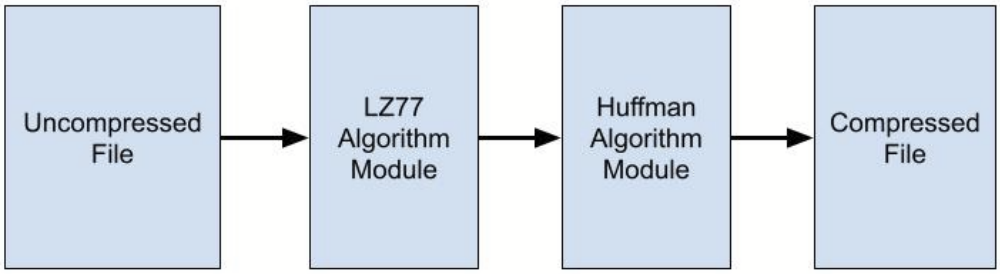

Figure 3: Gzip Implementation

The LZ77 module accepts the raw ASCII characters in binary. Then using a sliding window of 14 characters, 8 in the search buffer and 6 in the look ahead buffer, to analyze the string. The dictionary has space for 32 entries with a 3 bit offset, 3 bit length, and 8 bit ASCII character. The code used loops to properly search through the sliding window to find repetition and update the dictionary with the binary character location and lengths. This data is outputted to the Hoffman module.

The Huffman module accepts the dictionary with binary ASCII strings as input. It utilizes a case structure with the most frequently used letters in the English dictionary; it includes all 26 alphabet letters, at the top to improve running time, and . The ASCII characters are converted from an 8 bit representation to a 5 bit representation. This is much less complex than the LZ77 module, as it mostly comprises one case structure. The output of the module is a dictionary with Huffman encoded binary strings.

Conclusion:

Overall, the project was successful. The word tested was abracadabra” as seen in *Figure 4*, which was compressed from 88 bits to 66 bits. This reduced the bit complexity by 25%, while in general the reduction can range between 10% and 30%, as it depends on the repetition inherent in the inputted string.

Intel Quartus was used to analyze the hardware implementation of the SystemVerilog code. Using the Power Analyzer Tool, the total thermal power estimate for the design is 79.83 mW. As this is an FPGA software, there is no tool to determine the exact size of the hardware implementation, but this design does require 1,142 registers. Using the Timing Analyzer tool, the max frequency of the design is 134.12 MHz.

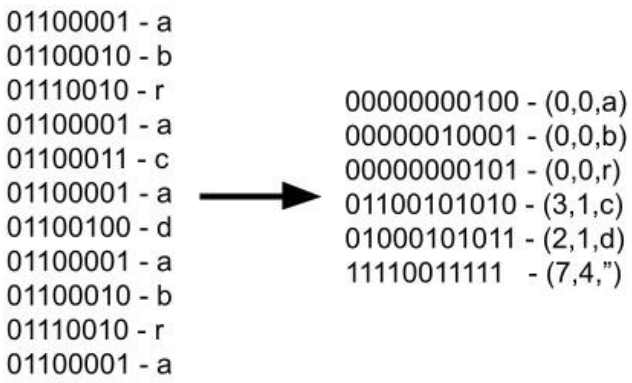

Figure 4: Compression of "abracadabra"

While this is not the most powerful compression algorithm, it still reduces bit complexity and maintains full lossless compression. It was a straightforward project, but still required difficulty in implementing in SystemVerilog, making it a valuable experience. It is likely that other algorithms could be implemented in tandem with gzip to improve compression even further.

References:

[1] S. B. Kamble, "Data compression Encoding LZ77," S.B. Kamble, 16 October 2017. [Online]. Available: https://www.youtube.com/watch?v=l7xeA6py9zM. [Accessed 23 April 2024].

[2] A. Feldspar, "An Explanation of the Deflate Algorithm," 23 August 1997. [Online]. Available: https://www.zlib.net/feldspar.html. [Accessed 23 April 2024].

[3] S. Oswal, A. Singh and K. Kumari, "DEFLATE COMPRESSION ALGORITHM," *International Journal of Engineering Research and General Science,* vol. 4, no. 1, pp. 430-436, 2016.