REGISTER TRANSFER LEVEL DESIGN OF COMPRESSION PROCESSOR CORE USING VERILOG HARDWARE DESCRIPTION LANGUAGE

ROSLEE BIN MOHD SABRI

UNIVERSITI TEKNOLOGI MALAYSIA

Specially dedicated to
my beloved wife and family

ACKNOWLEDGEMENTS

First and foremost, I would like to extend my deepest gratitude to my main project supervisor, Professor Dr. Mohamed Khalil bin Mohd Hani, for giving me the opportunity to work on new areas of digital system design. His constant encouragement, critics and guidance were key to bringing this project to a fruitful completion. I have learnt and gained much, not only in the field of research, but also in the lessons of life.

My sincerest appreciation goes out to all those who have contributed directly and indirectly to the completion of this research and thesis. Of particular mention is Ms Hau Yuan Wen for her guidance, advices and motivations. Without her continued support and interest, this project and thesis would not have been the same as presented here.

I would also like to recognize the support of my fellow postgraduate students. My sincere appreciation also extends to all my colleagues and others who have provided assistance at various occasions. Their views and tips are useful indeed. At the same time, the constant encouragement and camaraderie shared between all my friends during my postgraduate studies has been an enriching experience.

Finally, I would like to express my love and appreciation to my wife who has shown unrelenting care and support throughout this challenging endevour.

ABSTRACT

Throughput independent and parameterized data compression processor core was designed to tackle the needs of high-speed data compression applications. The design is based on combination of LZSS algorithm and Huffman coding, which enables it to be used in compression of a wide variety of data types. However, several design limitations exist. The design uses several technology-dependent modules that limit its hardware realization in alternative technologies. In addition, it also suffers from an abnormal functional behavior when trying to decompress data that contain sufficiently high redundancy. In view of these limitations, design enhancements are proposed. One of the proposed enhancements is to improve the design portability to any hardware implementation technology. This is accomplished through designing generic hardware to replace the technology-dependent modules and utilizing conditional compilation approach to best decide the design realization given the available resources and constraints. With this approach, IP cores designed for the targeted technology should be used to take advantage of the efficient resource utilization and proven design, which leads to faster time-to-market and minimizes the integration risks and verification efforts of large systems. However, if the design does not have access to IP cores, then generic modules can be instantiated but at the expense of development cost. Another design enhancement offers a hardware patch to fix the decompression core hardware bug. The issue was identified to originate from writing and reading the same memory location simultaneously. As a solution, the behavior of the memory controller and its supporting logic are modified to prevent this from occurring. From the design simulation results, it is concluded that the decompression core hardware bug is finally solved.

ABSTRAK

Satu perkakasan pemadatan data yang berparameter serta mempunyai kadar pemprosesan bebas telah direka untuk menangani keperluan aplikasi pemadatan data berkadar laju. Rekaan ini dicipta berasaskan kepada kombinasi algoritma LZSS and kod Huffman, yang membolehkan ia digunakan untuk memproses pelbagai jenis data. Namun, wujud beberapa kelemahan dalam rekaan tersebut. Ia menggunakan beberapa modul yang bergantung kepada teknologi khusus yang menghadkan pengrealisasian perkakas jika digunakan dalam sistem teknologi yang berbeza. Selain dari itu, rekaan tersebut juga mempunyai masalah kelakuan fungsi proses menidak-padat yang tidak normal apabila ia digunakan untuk memproses data yang mempunyai kadar ulangan yang tinggi. Justeru, beberapa cadangan membaik pulih rekaan dihasilkan untuk menangani kelemahan-kelemahan tersebut. Salah satu cadangan baik pulih yang dikemukakan membolehkan rekaan tersebut direalisasikan dalam pelbagai jenis teknologi perkakasan. Perkara ini berjaya dihasilkan menerusi rekaan perkakas generik untuk menggantikan modul yang bergantung kepada teknologi khusus dan menggunakan cara kompilasi bersyarat untuk merealisasikan rekaan berdasarkan kepada sumber dan maklumat yang ada. Menerusi cara ini, perkakas IP yang direka untuk teknologi tertentu dapat digunakan bagi memastikan hasil yang efektif dan berkesan, sekaligus memendekkan kadar masa yang diperlukan untuk memasarkan produk dan mengurangkan risiko pengalihan dan pengesahan maklumat bagi sistem-sistem yang besar. Namun, sekiranya rekaan ini tidak mempunyai akses kepada perkakas IP, modul generik boleh digunakan tetapi melibatkan kos pembangunan yang tinggi. Satu lagi cadangan baik pulih ialah menawarkan tampalan perkakasan untuk membaiki pepijat dalam proses menidak-padat. Masalah ini dikenalpasti berpunca apabila sistem memori diakses untuk menulis dan membaca pada masa yang sama. Untuk menyelesaikan perkara ini, kelakuan pengawal memori dan logik sokongannya diubahusai untuk mengelakkan masalah tersebut dari berlaku. Dari keputusan simulasi rekaan, dapat disimpulkan bahawa pepijat dalam proses menidak-padat data telah berjaya diselesaikan.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

ASIC	-	Application Sppecific Integrated Circuit
CAD	-	Computer Aided Design
CPLD	-	Complex Programmable Logic Device
CPU	-	Central Processing Unit
DSP	-	Digital Signal Processing
EDA	-	Electronic Design Automation
FIFO	-	First In First Out
FPGA	-	Field Programmable Gate Array
FSM	-	Finite State Machine
GUI	-	Graphical User Interface
HDL	-	Hardware Development Language
I/O	-	Input/Output
IP	-	Intellectual Property
LCD	-	Liquid Crystal Display
LED	-	Light Emitting Diode
LPM	-	Library of Parameterized Module
LZSS	-	Lempel Ziv Storer Szymanski
PLD	-	Programmable Logic Device
ROM	-	Read Only Memory
RTL	-	Register Transfer Level
SRAM	-	Static Random Access Memory
SoC	-	System-on-Chip
SOPC	-	System-on-Programmable-Chip
UART	-	Universal Asynchronous Receiver Transmitter
VHDL	-	Very High Speed Integrated Circuit Hardware Description Language
VLSI	-	Very Large Scale Integration

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

This project implements register-transfer-level design of a proprietary high-speed data compression and decompression processor cores using Verilog hardware description language. In addition, this project also offers enhancements aimed at improving the design portability to any hardware implementation technologies, as well as solving a hardware bug of the decompression processor core design. In the first chapter, overview of the project background is presented, followed by discussions on the problem statement, project objectives as well as the scope of work. An overview of the theory and knowledge involved is also presented. The organization of this thesis is presented at the end of the chapter.

1.1 Background

In many computing applications, getting the maximum throughput from limited resources is always desirable. For example, modern communication systems normally have limitations on its transmission medium's bandwidth utilization for data transfers. To fully utilize the available bandwidth, traffic sent through the medium should not contain any redundant information. This is not necessarily the case however, because all source information has inherent redundancies in them (Shannon, 1948). This means considerable amount of valuable resources would be wasted if these redundancies are not removed when transmitting data over the limited bandwidth medium.

In addition to efficient resource utilization, certain computing applications require fast information processing and data manipulations in order for the system to properly operate in real-time. For example, many wireless communication systems operate in time division duplex mode, where windows of finite time duration are allocated for data transfers between two communicating terminals. This means all processing must be completed and the required data must be valid within this time window to ensure proper communication takes place and to enable other data transfer windows to be allocated. Else, the communication channel will break down and all processed data will be rendered useless. Therefore, any improvements in ensuring optimal physical resources utilization must also take into account the required processing time of such improvements, so that the real-time performance of the overall system is not degraded.

A cost effective way to efficiently utilize limited physical resources in high-speed computing applications is by compressing the information processed by such applications. Essentially, data compression techniques remove the inherent redundant information in source data such that the information can be represented by fewer bits. Applying this technique in high-speed communication systems for example, allows more information to be transferred over a limited bandwidth medium compared to if original source data were to be transmitted. This would increase the effective bandwidth utilization of the transmission medium and the overall system performance.

However, data compression techniques are normally computationally intensive, which means considerable amount of processing time is required to achieve sufficient compression savings. Therefore, to effectively apply data compression techniques in high-speed computing applications, the complex processing of the data compression algorithms must be done considerably fast to enable the system operating in real-time with required performance. This means a high-speed data compression solution is needed in order to effectively utilize limited resources in any high-speed computing applications.

To fulfill this need, a proprietary high-speed data compression and decompression processor cores were designed and developed by Universiti Teknologi Malaysia (Yeem, 2002). The hardware uses data compression techniques based on combination of Lempel-Ziv-Storer-Szymanski (LZSS) algorithm and Huffman coding. It was designed as a parameterized module for easy configurability that can provide suitable compromise between constraints of hardware resources, processing speed and compression saving. In addition, both processor cores were designed to be easily integrated with any memory-mapped bus systems, which attractively lend itself for operation within virtually all modern systems utilizing some kinds of processor architecture. Initially, both compression and decompression processor cores were ported to an ALTERA programmable logic device, which is the FLEX10KE field programmable gate arrays (FPGA). As such, it uses several ALTERA intellectual-property (IP) cores to ease design and development work, and to take advantage of optimized resource utilization of the target device. The IP cores used are the Library of Parameterized Module (LPM) first-in-first-out buffers (FIFO) and dual port memories.

1.2 Problem Statement

The use of several IP modules targeted for specific technology means hardware implementation of the compression and decompression processor cores is limited to ALTERA programmable logic devices that support FIFO and dual port memories used. At best, hardware implementation of the design in other logic devices or technologies requires all IP modules to be replaced by equivalent hardware in the device or technology of interest. However, if equivalent modules are not available in target technology, or the design does not have access to IP modules because of licensing requirements, hardware implementation would be considerably difficult because the only solution is hardware redesign. Moreover, being technology-dependent means the design has less competitive advantage for commercial applications simply because potential users require highly flexible solution for procurement considerations and ease of system maintainability.

In addition to the limitation of the design's hardware portability, it is found that the functionality of the decompression processor core is not reliable. When decompressing data with sufficiently high redundant information, the restored data do not match its original source. When decompressing other sets of source data however, the decompression processor core outputs match the original source bit-by-bit. This inconsistent behavior means the decompression processor core design does not meet its required functional specifications, which renders it useless in real applications.

1.3 Objectives & Scope of Work

In view of the design limitations discussed in the previous section, the objectives of this project are:

1) To improve the compression and decompression processor core hardware portability to any programmable logic devices and/or process technologies.

2) To solve the abnormal behavior of the decompression processor core when processing highly redundant data.

3) To develop a data compression system targeted to a prototyping hardware platform for real-time design verification and performance analysis.

The scope of work of this project can be divided into three phases. The first phase involves hardware redesign of the compression and decompression processor cores using Verilog hardware description language. The parameterized nature of original design will be kept to preserve its advantage in terms of hardware re-configurability. Included in this project phase is the design of generic FIFO and dual-port memory modules to replace the ALTERA IP cores used in original design. The generic modules will enable both processor cores to be implemented in any programmable logic devices and ASIC technologies without requiring any design

modifications. In addition, a hardware patch will be designed to solve the abnormal functional behavior of the decompression processor core. With this fix, the design is expected to meet its functional specifications for any level of source data redundancies.

The second phase involves developing a stand-alone data compression system by embedding the compression and decompression processor cores in a processor-based system. Both cores will function as secondary processors that implement the required compression and decompression processing tasks to off-load the computing constraints of the system processor. This system architecture allows faster processing of the required data compression computations, so that the whole system can operate in real-time especially for high-speed applications. In this phase, the work consists of designing memory-mapped bus slave interfaces for both processor cores to enable data transfers with the system processor, building the overall system by integrating necessary components using a CAD tool, and implementing the system onto a prototyping hardware platform.

The final phase of the project involves developing an embedded firmware program that provides a mechanism for controlling and utilizing the compression and decompression processor cores inside the system. The firmware will be written in C programming language and will be an important tool in order to test the data compression system running in actual hardware and in real-time. This will enable an easier and more efficient evaluation of the design to verify its compliance to the functional specifications.

1.4 Literature Review

This section discusses the theory and background knowledge involved in this project. It provides a general overview of the compression algorithms used, which are the LZSS compression algorithm and Huffman coding, followed by discussions

on the hardware architecture and design approach of the proprietary high-speed data compression and decompression processor cores.

1.4.1 Lempel-Ziv-Storer-Szymanski (LZSS) Compression Algorithm

The main data compression technique chosen to be implemented in the design is the Lempel-Ziv-Storer-Szymanski (LZSS) algorithm. It is a lossless compression technique, which means no information is lost during the compression and decompression process. Compared to lossy compression techniques, where some information of source data is permanently lost in the process in favors for high compression savings, the LZSS algorithm generally has lower compression performance. However, for applications that cannot tolerate even a single bit of information lost, this technique can provide such guarantee. Therefore, LZSS compression algorithm actually covers larger application scope, where data compression techniques are concerned.

The LZSS algorithm is also known as a universal data compression technique. This means the algorithm can be applied to any discrete source, and its performance is comparable to certain optimal fixed code schemes designed for completely specified sources (Lempel, 1977). Using this technique, *a priori* knowledge of the source data characteristics is not required since the algorithm adaptively construct an optimal codeword representation of the source data. Coupled this with larger varieties of data compression applications it can handle, the proprietary data compression and decompression processor core design certainly has good competitive advantage for commercial high-speed computing applications.

Using this compression technique, the source data are encoded as LZSS codeword, represented as a pair of *position-length* pointer which points to parsed strings in a dictionary or encoding table. The strings are basically repeating symbols of source data that are stored inside the dictionary. The idea is to replace the representation of the repeated strings with a form that only requires fewer bits than

the original data, thus representing the source with lesser number of bits. On the decompression side, the LZSS algorithm adaptively regenerates the dictionary or encoding table based on the compressed data characteristics. Therefore, transmission of the dictionary is not required, which improves its processing speed and reduces bandwidth requirement for communication systems.

1.4.1.1 Notations and Definition

Before the exact mechanics of the coding procedures are described, we need to define terminologies used in LZSS algorithm.

Definition: The source data strings are over a finite alphabet A of α symbols, say $A = \{0, 1, \ldots, \alpha-1\}$. A string S of length $l(S) = k$ over A is an ordered k-tuple $S = s_1 s_2 \ldots s_k$ of symbols from A. To indicate a substring of S which starts at position i and ends at position j, we write $S(i, j)$. When $i \leq j$, $S(i, j) = s_i s_{i+1} \ldots s_j$, but when $i > j$, we take $S(i, j) = \Lambda$, the null string of length zero.

Definition: The concatenation of strings Q and R forms a new string $S = QR$; if $l(Q) = k$ and $l(R) = m$, then $l(S) = k + m$, $Q = S(1,k)$, and $R = S(1+1, k+m)$. For each j, $0 \leq j \leq l(S)$, $S(1, j)$ is called a prefix of S; $S(1, j)$ is a proper prefix of S if $j < l(S)$.

Definition: Given a proper prefix $S(1, j)$ of a string S and a positive integer i such that $i \leq j$, let $L(i)$ denote the largest nonnegative integer $l \leq l(S) - j$ such that $S(i, i+l-1) = S(j+1, j+l)$, and let p be a position of $S(1,j)$ for which $L(p) = max_{1 \leq i \leq j} L(i)$. The substring $S(j+1, j+L(p))$ of S is called the reproducible extension of $S(1, j)$ into S, and the integer p is called the pointer of the reproduction. For example, if $S = 00101011$ and $j = 3$, then $L(1) = 1$ since $S(j+1, j+1) = S(1,1)$ but $S(j+1, j+2) \neq S(1,2)$. Similarly, $L(2) = 4$ and $L(3) = 0$. Hence, $S(3+1, 3+4) = 0101$ is the reproducible extension of $S(1,3) = 001$ into S with pointer $p = 2$.

1.4.1.2 LZSS Encoding Process

1) Set $i = 1$, and initialize an integer h; $h_0 = 0$
2) Initialize buffer B with predefined symbols and first Ls symbols of the incoming source stream, S; $B_0 = X^{n-Ls}S(1, Ls)$, where X is the predefined symbol
3) For each i,
   a. Determine the reproducible extension of B_{i-1} $(1, n–Ls)$ into B_{i-1}.
   b. Compute the codeword, C_i, the integer h_i, and update the contents of the buffer, B_i:
      i. If $L(p)$ of the reproducible extension > 0 then
         - $C_i = 1C_{i1}C_{i2}$, where $C_{i1} = p - 1$, $C_{i2} = L(p)$
         - $h_i = h_{i-1} + L(p)$
         - $B_i = B_{i-1}$ $(1 + L(p), n)$ $S(h_i + 1, h_i + Ls)$
      ii. If $L(p)$ of the reproducible extension = 0 then
         - $C_i = 0C_{i3}$, where $C_{i3} = B_{i-1}$ $(n–Ls+1, n–Ls+1)$
         - $h_i = h_{i-1} + 1$
         - $B_i = B_{i-1}$ $(2, n)$ S $(h_i + 1, h_i + Ls)$
4) If $h_i < l(S)$, then $i = i + 1$ and go to Step 3. Else, STOP.

1.4.1.3 LZSS Decoding Process

1) Let D_i denote the content of the buffer before i-th iteration of the algorithm, where $l(D_i) = n - Ls$.
2) Set $i = 1$, and initialize the buffer D with $(n - Ls)$ predefined symbols, $D_1 = X^{n–Ls}$, X is the predefined symbol;
3) For each i,
   a. Shift the contents of the buffer, D_i:
      i. $D'_i = D'_i$ $(2, n–Ls)$ D'_i (p_i, p_i) ; if Flag = 1 (*Note 1), or
      ii. $D'_i = D'_i$ $(2, n–Ls)$ H_i ; if Flag = 0 (*Note 2)
   b. Compute the restored string, S_i:
      i. $S_i = D'_i$ $(n–Ls–l_i–1+1, n–Ls)$; if Flag = 1
      ii. $S_i = D'_i$ $(n–Ls, n–Ls)$; if Flag = 0
   c. Update the contents of the buffer: $D_{i+1} = D'_i$

4) If C_i is the last codeword, then STOP. Else $i = i + 1$ and go to Step 2.

*Note 1: Determine the p_{i-1} and l_{i-1} from the next $[\log_2 (n–Ls)]$ and the next $[\log_2(Ls)]$ bits of C_i. Apply l_{i-1}shift, while copying the contents of position p_i in the buffer into the position $n – Ls$.

*Note 2: Determine the explicit symbol (*said* H_i) from the next $l(S_i(1,1))$ bits of C_i. Shift the buffer once, while copying the H_i into the position $n–Ls$ of the buffer.

Example of the encoding and decoding process of LZSS algorithm is explained in Appendix A.

1.4.2 Huffman Coding Algorithm

Huffman coding is an entropy or statistical-based coding technique, where it requires *a priori* knowledge of the source data distribution characteristics in order to construct an optimal encoding table for better performance. It allows variable-length codeword, where lesser bits are assigned to frequently occurring symbols, and more bits are assigned to symbols that seldom occur. Effectively, the encoded data will take fewer bits to be represented since most of the frequently used symbols of the source have been replaced by shorter codes.

General procedure to construct Huffman codes is as follows:

1) Rank all symbols in order of probability of occurrence.
2) Successively combine the two symbols of the lowest probability to form a new composite symbol; eventually we will build a binary tree where each node is the probability of all nodes beneath it.
3) Trace a path to each leaf, noticing the direction at each node.

Appendix B describes the Huffman coding technique in details.

1.4.3 High-Speed Data Compression Core Design

The technique used in the design of high-speed data compression and decompression processor cores is based on combination of LZSS compression algorithm and Huffman coding. The source data to be compressed is first processed by the LZSS compression technique since the algorithm is not restricted in what type of data it can process, coupled with the fact that it requires no *a priori* knowledge of the source. LZSS codeword is then generated whenever matches between the source data and the dictionary elements are detected, where the encoded data are represented as *position-length* pair codeword. Generally, the length portion of the LZSS codeword yielded by the algorithm is non-uniformly distributed, where smaller lengths occur more frequently than longer ones (Yeem, 2002). This suggests Huffman coding be employed to further encode the length portion of LZSS codeword in order to achieve higher compression saving. In the decompression side, the whole process is performed in the reverse order. Figure 1.1 illustrates this approach.

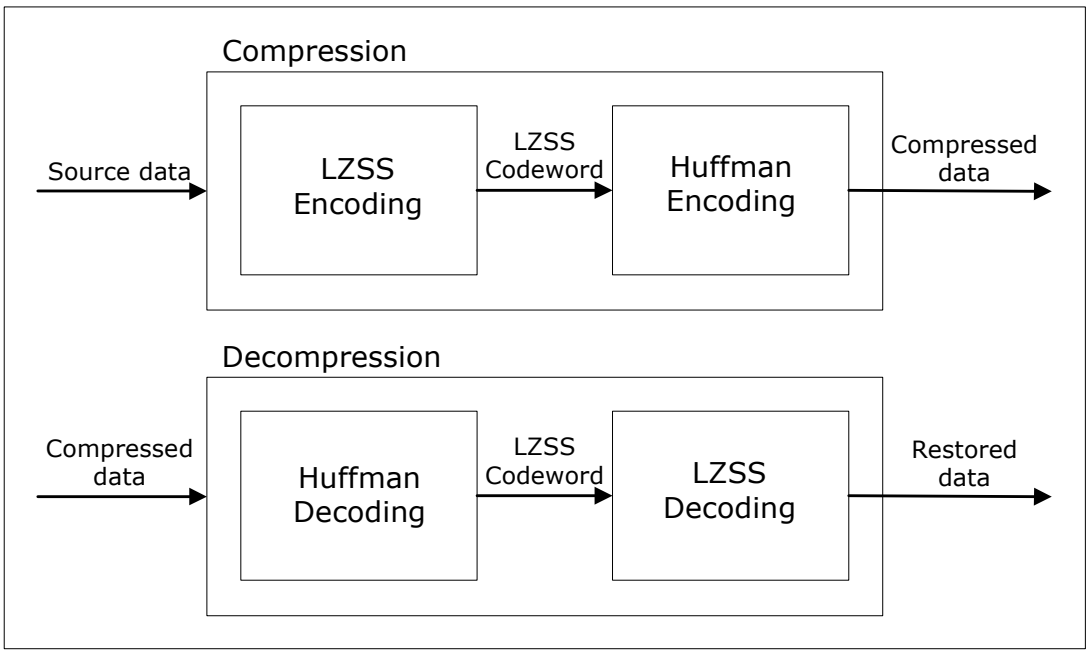

Figure 1.1: Compression and decompression approach of the processor core design

The LZSS algorithm, however, involves computationally intensive matching process during the compression stage because each input phrase has to be compared with every possible phrase in the dictionary. Furthermore, the dictionary updating process involves variable length shifting of the input source into the dictionary, since the length of longest matched phrase changes with time. If this operation is done using variable-length shifter, considerable amount of hardware resources will be consumed, which can lead to higher implementation cost because bigger (and correspondingly, more expensive) programmable logic device or ASIC silicon is needed. The design tackles these problems through systolic array architecture of the LZSS compression dictionary, where each input data is compared with every dictionary elements simultaneously, while shifting input data is done one symbol at a time through the use of a fixed-length shifter.

The Huffman coding technique also presents certain design challenges. Conventional Huffman coding requires *a priori* knowledge of the source data distribution characteristics in order to construct an optimal encoding table for better performance. However, in many real-life applications, it is difficult to determine the characteristics of source data because its probability distribution normally changes with time. Even when the source distribution statistics are available, different sources have different distribution characteristics. The encoding table must then be generated for each type of source data. Furthermore, the generated table must be transmitted along with the encoded data so that decompression can be performed correctly. This would both reduce the compression saving and increase the processing time of the hardware. The design tackles these problems by employing a predefined Huffman encoding table for both compression and decompression cores. The reason for this is two-fold; the first one is to simplify generation of the encoding table since adaptively building the table for different source data is no longer required. The second reason is to eliminate the need to transmit the encoding table to the decompression side, so that inefficient resource utilization and degradation of compression saving issues due to this encoding table transmission can be overcome.

The data compression core design also employs the reconfigurable and reusable hardware concept. This design concept promotes the use of the existing

hardware in other application domains with different processing requirements without significant modifications to the original design. As a result, it helps in speeding up development cycle of large systems and lowering the cost of implementation. The hardware design achieves this modularity through the use of scaleable hardware architecture and parameterized design approach, that resulted in configurable data compression and decompression processor cores based on suitable compromises between the constraint of resources, speed and compression saving. In addition, the design allows capability of integrating both processor cores with any external memory-mapped systems, through the use of reconfigurable bus interfaces. Table 1.1 describes the required design parameters and its effects on the generated hardware in terms of resources, speed and compression saving trade-off, as well as the suitable interfacing mechanism to the external system:

Table 1.1: Design parameters of the compression and decompression processor cores

Design Parameter	**Description**
SymbolWIDTH	Width of each input source symbol
DicLEVEL	The number of elements used to build the LZSS dictionary i.e. $2^{DicLEVEL}$
MAXWIDTH	The predefined maximum match size in parsing symbols into one string i.e. $2^{MAXWIDTH} - 1$
IniDicValue	The predefined symbol stored in each dictionary element when the dictionary is initialized.
InterfaceWIDTH	Width of the interfacing bus which is used to connect the compression/decompression hardware with an external interfacing system i.e. $2^{InterfaceWIDTH}$
PollAmount	Maximum number of data, each is $2^{InterfaceWIDTH}$, which is transferred between the compression/decompression hardware and external interfacing system within an interfacing phase.

1.5 Thesis Organization

The work in this thesis is organized into seven chapters. This first chapter presents the research background and motivation, followed by its objectives and scope of work. An overview of the theory and knowledge involved is presented, before concluding with thesis organization.

Chapter two discusses the research methodology. It starts with discussions on the design and verification approaches, followed by descriptions of the tools and techniques used to complete the research work.

Chapter three describes the design of the data compression hardware. This includes design details of both the compression and decompression processor cores, as well as their respective interfaces to external host systems.

Chapter four explains the design modifications and hardware enhancements proposed. It starts by discussing design details of the parameterized and generic memory modules, followed by discussion on the conditional compilation approach for the best compromise of hardware implementation. In addition, root cause of the decompression processor core hardware bug is described and followed by detailed explanation of the hardware modifications required to solve the issue.

Chapter five discusses the development of a processor-based, stand-alone data compression system. An overview of the system development is presented, focusing on the CAD tool used and the approach of embedding custom design with predefined IP modules. In addition, this chapter describes the details of a memory-mapped bus slave interface design to enable data transfers between the compression and decompression processor cores and the system processor. This chapter also describes the system firmware development, which will be used to test the overall system running on actual prototyping hardware platform.

Chapter six describes the design simulation and hardware test that are performed on both processor cores, as well as the complete system for functional verification and validating the system performance operating in real-time. In addition, comparison with original design is discussed to evaluate the performance of the proposed design enhancements.

Chapter seven summarizes the research work and states all deliverables of the project. Recommendations for potential future works are also given.

1.6 Summary

In this chapter, introduction to the background, theories and motivation of this project are discussed. Based on the discussions, objectives of the project are identified which leads to the scope of work necessary to achieve the desired goals. In the next chapter, research methodologies used in this project are described.