UDC 621.391.14

SYSTEM DESIGN OVERVIEW OF HAMMING PRODUCT CODES IMPLEMENTATION ON FPGA

Y.M. CHEN, X.H. REN

Belarusian State University of Informatics and Radioelectronics, Republic of Belarus

Received March 06, 2023

Abstract. This paper describes the implementation of a Hamming product codes encoder and decoder on a FPGA. The encoder and decoder design, including hardware structure and logic design.

Keywords: hardware design, Hamming product codes, FPGA, logic design.

Introduction

Based on Hamming product codes theory and using FPGA as the hardware platform, this project aims to provide a reliable and efficient data transmission solution. The main application area is on-chip interconnection or on-board interconnection, which can realize fast and reliable data transmission, thus improving the efficiency and reliability of data transmission. The main functions of the project include coding, error detection, error correction and hybrid re-request, which can effectively reduce the risk of transmission errors and lost data and ensure data integrity and reliability.

In this project, encoding, error detection and error correction are the key basic functions. Encoding converts the original data into Hamming product codes to improve the reliability of transmitted data, while error checking and error correction are performed at the receiving end to ensure the integrity of the data. This project also has a hybrid re-request function, which automatically requests data retransmission when an error occurs in data transmission, thus reducing transmission loss and improving data transmission reliability.

This project also implements the following functional features: low latency transmission through segmented transmission, which divides data into smaller parts, thereby increasing transmission speed and reducing transmission latency. The efficiency of data transmission can be further improved by increasing the clock frequency of the transmission module through clock partitioning. By utilizing the parallel computing capability of FPGAs, lower coding latency and better error detection and correction rates are achieved. These features allow the project to adapt to different application scenarios and provide an efficient and reliable solution for various data transmission needs.

In the fields of on-chip interconnect and on-board interconnect; the efficiency and reliability of data transmission are critical. The implementation of this project can provide reliable and efficient data transmission guarantee, thus contributing to the development and progress of these fields. In addition, the project can be widely used in communication, data storage, intelligent manufacturing, etc., providing efficient and reliable data transmission solutions for these fields and promoting the development and innovation in these fields.

Product codes

Product codes are serially concatenated codes, which were presented by Elias in 1954 [1]. The construction method of product codes allows us to construct long, powerful codes from short assembly codes. As shown in Figure 4, it is a schematic diagram of the code word structure of a two-dimensional Hamming product codes. The coding method uses a binary linear code $C(n,k)$, where the code word length is *n*, the message length is *k*, and the code distance is *d*. The product codes C_p of this coding

method can be expressed as $C(n,k)\times C(n,k)=C_p(n^2,k^2)$, where the code word length of the product codes is n^2, the message length is k^2, and the code distance is d^2 [2].

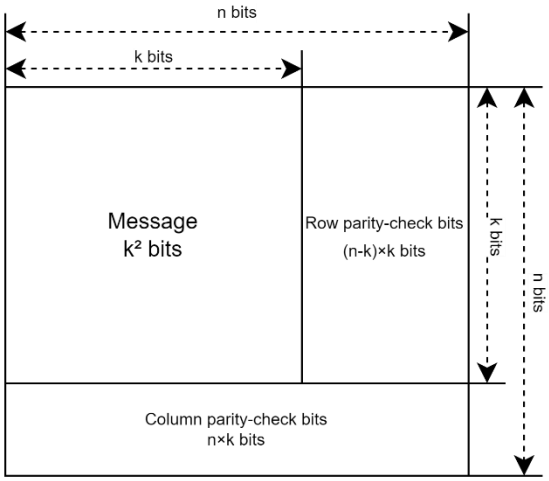

Figure 4. Codeword structure diagram

Specifically, the *k* messages of length *k* are first arranged into a matrix of *k* rows and *k* columns in a row-first order. Then each row in the matrix is encoded using binary linear coding *C* to generate *k* row parity bits of length $n-k$, and these parity bits are appended to the end of the message matrix to obtain a matrix of *k* rows and *n* columns. Next, consider this matrix as a new message matrix and encode each column in it, again using *C* for encoding, to generate n columns of column parity bits of length $n-k$. Finally, these column parity bits are appended to the end of the matrix to obtain a code word of length n^2.

System design overview

As shown in Figure 5 below, the design is mainly based on the message transmission direction and is divided into two main functional parts: transmitting and receiving. Implementing the code for the part of the virtual line is the main goal of this project [3,4].

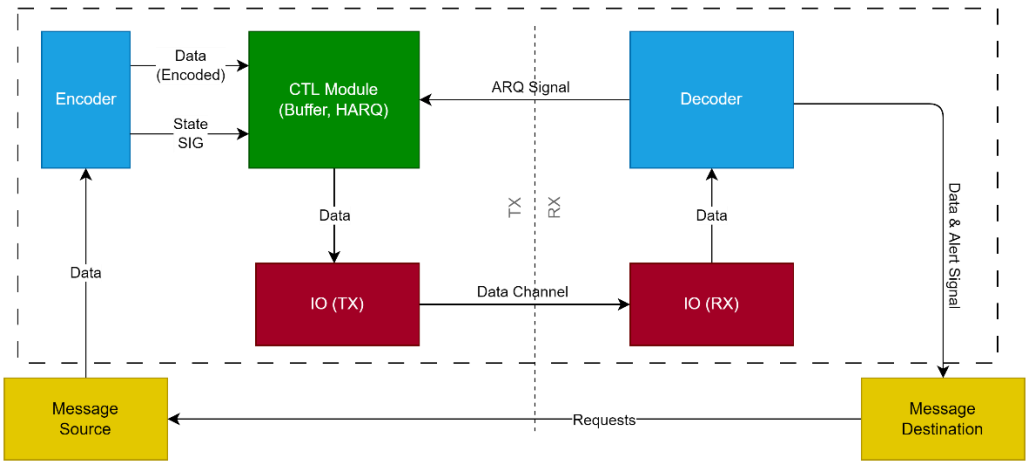

Figure 5. System overall structure diagram

Among them, the high-speed IO port is a general-purpose component for sending and receiving messages. The components unique to the sender include the message source, the encoder, and the control module. The source is responsible for generating the message to be sent, the encoder will encode the message for errors caused by the transmission, and the control module is responsible for managing the process of sending the message. In contrast, the only components that are unique to the receiver are the

message destination and the decoder. The message destination is the final point of reception of the message, and the decoder will verify the received message and correct errors if feasible. With this design, the sender can send the message to the receiver in a reliable and efficient manner. In addition, since the high-speed IO port is a very flexible and versatile component, it can be easily adapted to different clock frequencies and bit widths, increasing the scalability and flexibility of the system.

The control module of the sender is an important part of the whole system, which is not only responsible for managing the message sending process, but also enables the function of hybrid re-request. Specifically, it can cache the encoded messages and wait for the acknowledgement signal from the receiver. If the receiver fails to acknowledge in time or returns an error code, the control module sends supplementary bits or retransmits the message according to the set logic until the receiver acknowledges. This hybrid re-request mechanism can ensure transmission reliability while minimizing the number of retransmissions and improving the efficiency of the system.

The high-speed IO port serves as a clock demarcation in the system, and it can adapt higher frequency clock sources to achieve high-speed transmission. When the high-speed IO port receives a message from the sender, it transmits the message to the receiver at a high bit rate. To ensure the accuracy and reliability of the transmission, the high-speed IO port must be synchronized. By using this IO port, the whole system can achieve high-speed transmission to meet the high-bandwidth and low-latency transmission requirements.

Encoder design and implementation

The structure diagram of the encoder is shown in Figure 6, which includes four main parts: IO, control, storage (independent input-output buffer), and calculation units. The main logic control of the encoder is implemented by FSM. The calculation unit is composed of multiple parallel calculation units, whose working state is controlled by FSM state and feedback to the control unit.

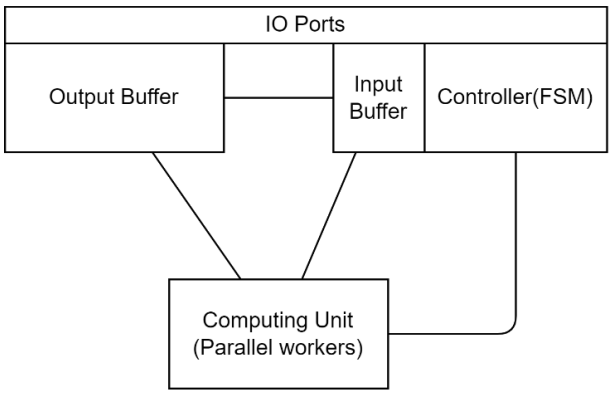

Figure 6. Encoder structure block diagram

The output of the encoder is directly transmitted to the output buffer. The control unit in this design also includes a module for implementing array transpose, which is used to transpose the data in the input buffer. By doing so, the encoder can achieve two-step encoding, thereby improving encoding efficiency.

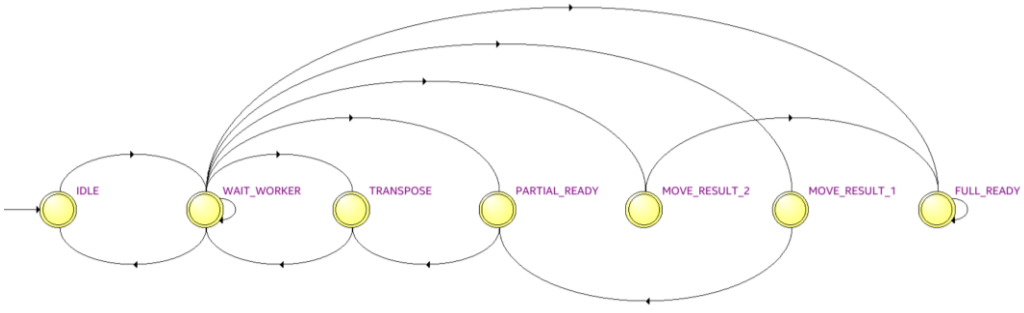

Figure 7. State transfer diagram of encoder FSM

The core of the encoder is its control unit, whose main functional logic is controlled by its internal finite state machine, as shown in Figure 7, which has the following seven states: IDLE, READY, PARTIAL_READY, WAITING_WORKER, transpose for second encoding stage (TRANSPOSE), and two states for internal data transfer (MOVE_RESULT_1 and MOVE_RESULT_2).

The operating states of the parallel computing unit are managed by its internal controller, which communicates with the encoder control unit in the form of a signal. In order to reduce the delay caused by the two-step encoding, the encoder enters the "partially completed" state after completing the row parity bit operation and can provide a code word without column parity bits for transmission.

Decoder design

The block diagram of the decoder is shown in Figure 8 below. The decoder of this project contains HDL implementations of several algorithms, so it also contains an additional logic module or set of logic modules that are responsible for adapting different algorithmic logic, depending on the construction parameters, compared to the encoder.

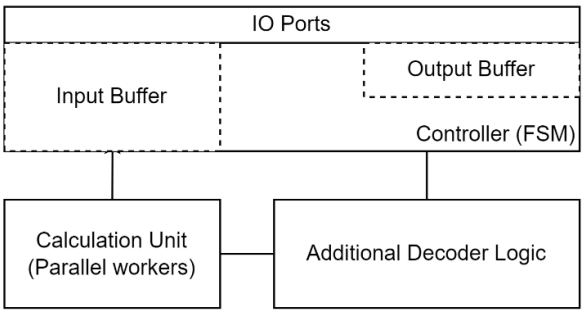

Figure 8. Decoder core structure block diagram

Similar as the encoder, the control of the decoder is controlled by a finite state machine, but its logic is more complex due to the addition of the auto-request logic. As shown in the Figure 9 below, this finite state machine includes: standby (IDLE), waiting for complete codeword (WAIT_FULL_CODE), two ongoing checks (CHK_PARTIAL, CHK_FULL) and two ready states (RDY_OK, RDY_FAIL). The first state of this finite state machine is the standby state (IDLE), where the decoder is idle and waiting for input. If all the row parity checks bits available, the decoder transitions to the check for partial codewords state (CHK_PARTIAL). If the check for partial codewords fails, the decoder transitions to a waiting state (WAIT_FULL_CODE) and waits for the missing data (column parity check bits) to arrive. Once a whole codeword available, the decoder transitions to the check for complete codewords state (CHK_FULL), the check is running in a sepreate process, which is achived by an entity installation controlled by generic mapping and generate. When the any of the tow check is successful, the decoder enters the ready state (RDY_OK), indicating that the data is not corrupted and can output it. If the checks fail, the decoder enters the ready failure state (RDY_FAIL), indicating that the data is corrupted, and the decoder cannot correct it and must wait for input again.

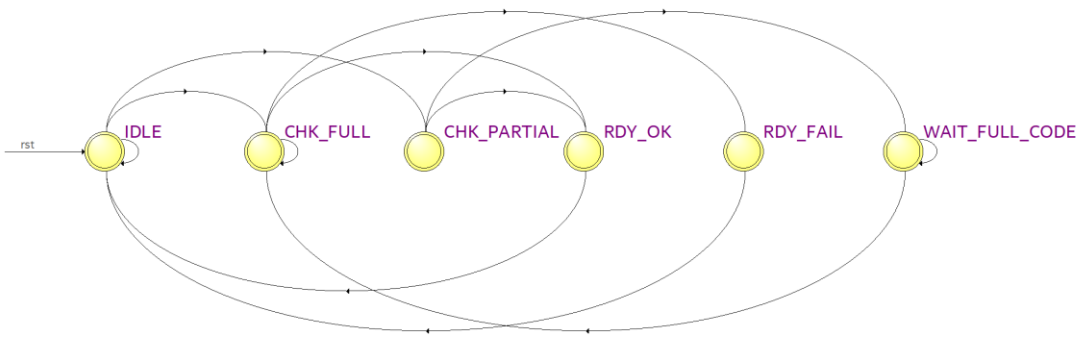

Figure 9. State transition diagram of the decoder FSM

Build and test system

The goal of this project is to implement a small communication system on FPGA in HDL that includes a transmitter, receiver, message source, message destination, and transmission channel. In order to achieve the above functionality, the project required a complete and easy-to-use tool chain for building and testing.

In the spirit of open source first, VSCode is used as the code editor, Gradle as the build environment, GHDL as the syntax checking, elaborate and emulate tool, and TypeScript, Python and GNU Octave as the auxiliary code generation tools. The project also provides a pre-built development environment in a Docker image, and a "devcontainer" configuration file to optimize development efficiency. To simplify the workflow of writing, compiling, and testing, a dedicated build script and a Gradle command wrapper were written.

Conclusion

In conclusion, this paper provides a brief description of the implementation of a Hamming product code encoder and decoder based on FPGA. Firstly, the basic principles of the algorithm are introduced, followed by a detailed discussion on the design of the encoder and decoder, including hardware structure and logic design. Then, the paper describes the building and testing system framework of the project. Through testing, the implemented encoder and decoder on FPGA demonstrate excellent performance, meeting the requirements of high-speed data transmission and low power consumption. This implementation provides a feasible solution for the design of FPGA-based encoder and decoder, with high practicality and reliability.

References

1. Elias P. // In Transactions of the IRE Professional Group on Information Theory. 1954. Vol. 4. P. 29–37.
2. Lin. S., Costello D.J. Error Control Coding, 2nd edition. USA, 2004.
3. Bo F., Ampadu P. // 2008 IEEE International SOC Conference. 2008. P. 59–62.
4. 毕涛., 刘迪., 张大为., 葛宝川. // 现代信息科技. 2023. P.58–63.