CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

The Implementation of a Reed Solomon Code Encoder /Decoder

A graduate project submitted in partial fulfillment of the requirements
For the degree of Master of Science in
Electrical Engineering

By

Qiang Zhang

May 2014

The graduate project of Qiang Zhang is approved:

Dr. Ali Amini | Date

Dr. Sharlene Katz | Date

Dr. Nagi El Naga, Chair | Date

California State University, Northridge

Acknowledgements

I would like to thank my project professor and committee: Dr. Nagi El Naga, Dr. Ali Amini, and Dr. Sharlene Katz for their time and help.

I would also like to thank my parents for their love and care. I could not have finished this project without them, because they give me the huge encouragement during last two years.

Dedication

To my parents

Table of Contents

List of Figures

Abstract

The Implementation of a Reed Solomon Code

Encoder /Decoder

By

Qiang Zhang

Master of Science in Electrical Engineering

This project details the design and implementation of a (255,223) Reed-Solomon code encoder and decoder of an error detection and correction subsystem. The encoder encodes a 223-byte data block and generates a 255-byte code block to be transmitted on a digital communication channel. This code uses Galois field arithmetic GF(2^8), which can correct up to 16 short bursts of errors.

The encoder is much simpler to design than is the decoder. The decoder design includes multiple modules in order to calculate the syndrome and find the error locations and error values. In order to find the coefficients of the error location polynomial, Berlekamp's iterative algorithm and an improved Berlekamp-Massey algorithm is introduced and used. Chien's searching algorithm is also used to search for the error locations. Modeling, simulation, and verification of the system design is done using Verilog HDL.

Chapter 1 Introduction

1.1 Introduction

In 1948, Claude Shannon's channel coding theorem resulted in a mathematical proof demonstrating that codes with special properties can help reach reliable data communications over a noisy channel if the channel has a capacity larger than the data rate of the message source. Shannon's work brought new field research for digital communications: error control coding theory. At that time, his work could prove only the existence of the code, but no mathematical abstracts were used to help people construct such codes. With the error handling capability, the simplest first code, which was constructed by appending the parity check bit to the message bits, could detect a single bit error, but not correct it.

Engineers have labored to find more-powerful codes, vital due to the high volume of data exchanged. The new codes had to be designed with enough redundancy to detect or detect-and-correct multiple bit errors up to a certain probability related to the characteristic of the digital channel. They also needed to make the code rates (messages' length/code length in bits) very close to one, so that the codes would be easy to implement in hardware.

Cyclic codes satisfy mathematical structures that permit the design of higher order correcting codes and easy encoding and syndrome calculations by using simple shift registers. The systematic codes are popular in practical applications. A systematic code is defined as a code that appends bits/bytes to the message words, and there is no need to change them to form the code words.

The Bose-Chaudhuri-Hocquenghen (BCH) codes are perhaps the best studied class of random-error-correcting cyclic codes. The BCH codes are a subset of cyclic codes. They can be divided into binary BCH codes and nonbinary BCH codes. As a special case of nonbinary BCH codes, Reed-Solomon codes are by far the most successful forward-error-correction codes in practice today. In Reed-Solomon codes, the code digits are elements of a finite field (Galois Field) with an order of 2^m, which can explain the fact that all digital systems are using actual binary symbols (0s and 1s) at the hardware level to represent any type of data (information), so the representation of elements from $GF(2^m)$ is straightforward.

1.2 Objectives

The objective of this project is to design and implement a (255,223) Reed-Solomon code error detection and correction subsystem. The first part introduces the basic knowledge of several important codes and then mainly describes the general concepts and procedures for encoding and decoding a Reed-Solomon code.

Since Reed-Solomon codes can correct multiple and long burst errors with a relatively high code rate, they are very useful in digital communication. For the Reed-Solomon code (255,223), the code rate is 223/255=0.8745, and the code can successfully detect and correct up to 16 erroneous bytes in each block of 255 bytes. In a block of $255 \times 8 = 2040$ bits transmitted bit by bit, this code can detect and correct up to $16 \times 8 = 128$ bits of information. If more than 16 erroneous bytes (every 2040 bits) occurred during the transmission, this Reed-Solomon code would fail. The code is defined over $GF(2^8)$, which is proven by the large spread of the popular 8-bit data buses adopted in the industry. The encoder and decoder are designed and implemented in the software of Modelsim and Altera Quartus Ⅱ by using Verilog HDL.

1.3 Project Outline

This project includes seven chapters and three appendices. The second chapter describes two important codes: the Hamming code and BCH code. Chapter three introduces the general concepts of Reed-Solomon codes. Chapter four is the detailed Reed-Solomon code encoding process. Chapter five presents the decoding procedure of Reed-Solomon code. Chapter six shows the simulation of the encoder and decoder in the software Modelsim and Altera Quartus ii. The last chapter, the seventh chapter, is the conclusion of this project. The detailed programming used is in the appendices.

This project uses Modelsim and Quartus Ⅱ as software applications. The design, modeling, simulation, and verification use Verilog HDL.

Chapter 2 Hamming Code and Bose-Chaudhuri-Hocquenghem Code

2.1 Hamming Codes

The digital channel for error control coding is shown in Figure 2.1 below:

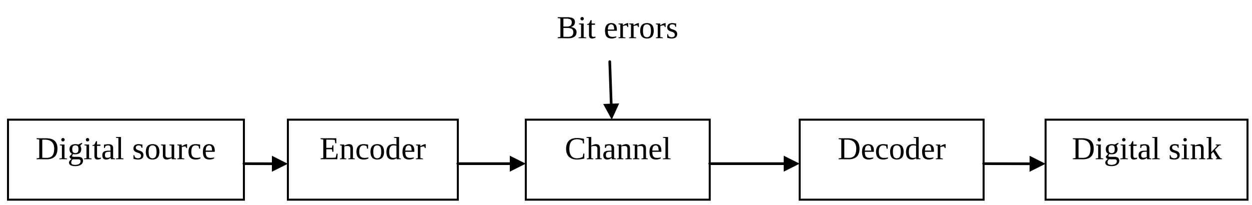


Figure 2.1 Digital channel for error control coding

The Hamming codes hold an important position in error control codes history, as they were the first class of linear block code for error correction. A cyclic Hamming code is a code with a generator polynomial that is a primitive polynomial P(x) of degree m. This code has the following properties:

Code length:	$n=2^m - 1$
Number of parity check digits:	$n-k=m$
Number of information digits:	$k=2^m-m-1$
Error-correcting capability:	$t=1$
Minimum distance:	$d_{min}=3$

A Hamming code is a single-error-correcting code. The double-error-detecting and single-error-correcting Hamming codes are very important for future coding theory and research.

2.2 Bose-Chaudhuri-Hocquenghem (BCH) Codes

The BCH codes were discovered by Hocquenghem in 1959 and independently by Bose and Chaudhuri in 1960. They are the most extensive and powerful codes by far. The BCH codes are cyclic codes, which were first defined in binary symbols and then generalized to codes in p^m symbols (where m is any positive integer and p is any prime) by Gorenstein and Zierler in 1960. Later, Peterson's algorithm was generalized and refined by Gorenstein and Zierler, Chien, Berlekamp, Forney, and Massey.[3]

Binary BCH codes will be discussed in this section and the nonbinary BCH codes in the next chapter.

For any positive integer t and m ($t < 2^m-1$), there exists a BCH code with the following parameters:

Block length: $n = 2^m - 1$

Number of parity-check digits: $n - k \leq mt$

Minimum Distance: $d \geq 2t + 1$

This code is called t-error-correcting BCH code, which has the capability of correcting any combination of t or fewer errors in a block of $n = 2^m - 1$ digits. First let α be a primitive element of the Galois field $GF(2^m)$. Then consider the following sequence of consecutive powers of α:

$$\alpha, \alpha^2, \alpha^3, \cdots, \alpha^{2t} .$$

Let $m_i(X)$ be the minimum polynomial of α^i. Then the generator polynomial of the t-error-correcting BCH code is the least common multiple of $m_1(X), m_2(X), \cdots, m_{2t}(X)$, which can be written as following:

$$g(X) = LCM[m_1(X), m_2(X), \cdots, m_{2t}(X)] .$$

Therefore, $\alpha, \alpha^2, \alpha^3, \cdots, \alpha^{2t}$ are roots of g(X), such that $g(\alpha^i) = 0$ for i=1, 2, …, 2t. If i is an even integer, it can be expressed as $i = i'2^b$, in which i' is an odd integer and $b \geq 1$, and $m_i(X) = m_{i'}(X)$ .Which means every even power of α in the sequence of $\alpha, \alpha^2, \alpha^3, \cdots, \alpha^{2t}$ has the same minimum polynomial as some previous odd power of α in the sequence. Then the generator polynomial is reduced to the following:

$$g(X) = \mathrm{LCM}[m_1(X), m_3(X), \cdots, \ m_{2t-1}(X)] .$$

The degree of g(X) is at most mt because of the degree of each minimum polynomial is m or less. Which means n-k (the number of parity-check digits) is at most mt.

Chapter 3 Reed-Solomon Codes

3.1 Reed-Solomon Codes

First let us turn our attention to nonbinary BCH codes. A t-error-correcting BCH code of block length $n= q^m -1$ is a (n, k) cyclic code whose generator polynomial g(x) has coefficients from GF (q) and roots

$$\beta, \beta^2, \cdots, \beta^{2t}$$

in $GF(q^m)$ an extension field of GF(q). In order to construct a nonbinary BCH code minimal polynomials over GF(q) are required and the generator polynomial of the code is

$$g(x) = LCM[m_1(X), m_2(X), \cdots, m_{2t}(X)],$$

where $m_i(x)$ is the minimal polynomial over GF(q) of β^i. When q=2, the minimal polynomials are binary and we get the binary BCH codes.

The Reed-Solomon codes are the most important class of nonbinary BCH codes, both the symbols and the generator polynomial roots lie in the field GF(q). A t-error-correcting Reed-Solomon code has the following algebraic structure:

Block length: $n = q - 1$

The number of parity-check digits: $n - k = 2t$

The minimum code word distance: $d_{min} = 2t + 1$

When $q=2^m$, the symbols and roots lie in $GF(2^m)$. The generator polynomial of a t-error-correcting Reed-Solomon code is the least-degree polynomial that has $\beta, \beta^2, \cdots, \beta^{2t}$ as roots, where β is an element from $GF(2^m)$. And the minimum polynomial is $m_\beta = x + \beta$, which is the least-degree polynomial that has β as a root.

Therefore, the generator polynomial of a Reed-Solomon code is given by:

$$g(x) = (x+\beta)(x+ \beta^2)\cdots(x+ \beta^{2t})$$

Since all the factors are different, there is no need to take the least common multiple of the factors.

3.2 Properties of Reed-Solomon Codes

A Reed-Solomon (RS)-code is written as RS (n, k) with s-bit symbols, where k is the number of data symbols of s-bit each and the addition of 2t parity check symbols will make n-symbol code word.

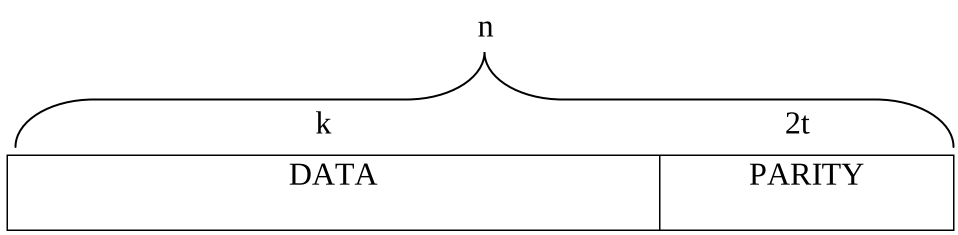


Figure 3.1 The structure of the transmitted information

Block length: n= 255

The number of parity check digits: n – k = 2t = 32

The minimum codeword distance: $d_{min} = 2t + 1 = 33$

The error correction capability: t = 16

For example Consider an RS code (255,223) with s=8. Here 223 are the information data and the code word bytes are 255. Therefore the parity will be 32 bytes.

As n=255 k=223

$$n - k = 255 – 223 = 32 = 2t$$

$$t=16$$

It can correct up to 16 symbol errors, and the larger the value of t, the larger the errors that can be corrected. Since the symbol s is given, the maximum codeword length is $n = 2^s – 1$. Therefore, if $s = 8$, the number of codeword bytes is 255.

Reed-Solomon codes can correct errors in the wide range of systems in digital communication. RS codes are widely used in Compact Discs, DVDs, barcodes, and wireless and mobile communications.

Chapter 4 Reed-Solomon Code Encoding

4.1 Reed-Solomon Encoder

The function for RS-Encoder is shown as following:

$$g(X) = g_0 + g_1X + g_2X^2 + g_3X^3 + \cdots + g_{2t-1}X^{2t-1} + g_{2t}X^{2t}$$

The number of parity symbols is 2t, which is equal to the degree of the function, and the roots of the polynomial are

$$\alpha, \alpha^2, \ldots., \alpha^{2t}$$

Since Reed-Solomon codes are cyclic codes, encoding in systematic form is similar to the binary encoding procedure. The encoding procedures are as follows:

Let the input message be : $m = (m_0, m_1, m_2, \ldots, m_{k-1})$ (as we know n-k=2t)

Then $m(X) = m_0 + m_1X + m_2X^2 + \ldots + m_{k-1}X^{k-1}$

If we multiply m(X) by X^{2t}, we can get as following:

$X^{2t}m(X) = m_0X^{2t} + m_1X^{2t+1} + m_2X^{2t+2} + \ldots + m_{k-1}X^{n-1}$

In order to find the parity check digits, the polynomial $X^{2t}m(X)$ is divided by g(X) to obtain the remainder b(X) as follows:

$$X^{2t}m(X) = a(X)g(X) + b(X)$$

In the equation, a(x) is the quotient, and b(X) is the remainder of the division.

Therefore the remainder b(X) can be written as: $b(X) = X^{2t}m(X) \bmod g(X)$

Then the resulting codeword polynomial $U(X) = b(X) + X^{n-k}\, m(X)$

4.2 The hardware concept of RS-encoder

For the (255,223) Reed Solomon code where the symbols of the code are elements of GF (2^8) and have the following characteristics:

Degree of the polynomial: $m = 8$

Block length: $n = 2^8 - 1 = 255$

Number of parity check digits: $n - k = 2t = 32$

Error correction capability: $t = 16$

However, each symbol is represented by eight binary digits or one byte. Also, each data block contains 223 information symbols.

This code is capable of correcting up to sixteen short burst errors of one byte or any burst error combination of up to a total length of eight bytes, providing that they only affect a maximum of sixteen individual symbols.

The elements of GF (2^8) are generated by primitive polynomial of degree 8:

$$p(x) = 1 + X^2 + X^3 + X^4 + X^8$$

So, the elements can be represented in an 8-tuple with 8 components being 0 or 1 and represent code word. The zero element of GF(2^8) appears as an all zero 8-tuple.

If α is a primitive element in GF(2^m), then the root of p(x) is only the first thirty two powers of α and are the roots of the generator polynomial as the following:

$$p(\alpha) = 1 + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^8 = 0$$

$$\text{or } 1 + \alpha^2 + \alpha^3 + \alpha^4 = \alpha^8$$

Only the first 32 powers of α are the roots of the generator polynomial g(X), so the generator polynomial g(X) is:

$$g(X) = (X + \alpha)(X + \alpha^2)(X + \alpha^3)(X + \alpha^4)(X + \alpha^5)\cdots(X + \alpha^{32}) \text{ or}$$

$g(X) = 45 + 216X + 239X^2 + 24X^3 + 253X^4 + 104X^5 + 27X^6 + 40X^7 + 107X^8 + 50X^9$

$+ 163X^{10} + 210X^{11} + 227X^{12} + 134X^{13} + 224X^{14} + 158X^{15} + 119X^{16} + 13X^{17} + 158X^{18} + X^{19} +$

$238X^{20} + 164X^{21} + 82X^{22} + 43X^{23} + 15X^{24} + 232X^{25} + 246X^{26} + 142X^{27}$

$+ 50 X^{28} + 189X^{29} + 29X^{30} + 232X^{31} + X^{32}$

As a result, the coefficients of g(X) used in the encoder multiplication are as follows:

$g_0 = 45 = \alpha^{18}$ $g_1 = 216 = \alpha^{251}$ $g_2 = 239 = \alpha^{215}$ $g_3 = 24 = \alpha^{28}$ $g_4 = 253 = \alpha^{80}$

$g_5 = 104 = \alpha^{107}$ $g_6 = 27 = \alpha^{248}$ $g_7 = 40 = \alpha^{53}$ $g_8 = 107 = \alpha^{84}$ $g_9 = 50 = \alpha^{194}$

$g_{10} = 163 = \alpha^{91}$ $g_{11} = 210 = \alpha^{59}$ $g_{12} = 227 = \alpha^{176}$ $g_{13} = 134 = \alpha^{99}$ $g_{14} = 224 = \alpha^{203}$

$g_{15} = 158 = \alpha^{137}$ $g_{16} = 119 = \alpha^{43}$ $g_{17} = 13 = \alpha^{104}$ $g_{18} = 158 = \alpha^{137}$ $g_{19} = 1 = \alpha^0$

$g_{20} = 238 = \alpha^{44}$ $g_{21} = 164 = \alpha^{149}$ $g_{22} = 82 = \alpha^{148}$ $g_{23} = 43 = \alpha^{218}$ $g_{24} = 15 = \alpha^{75}$

$g_{25} = 232 = \alpha^{11}$ $g_{26} = 246 = \alpha^{173}$ $g_{27} = 142 = \alpha^{254}$ $g_{28} = 50 = \alpha^{194}$ $g_{29} = 189 = \alpha^{109}$

$g_{30} = 29 = \alpha^8$ $g_{31} = 232 = \alpha^{11}$ $g_{32} = 1$

In the encoder part, since (255,223) RS code is defined over $GF(2^8)$, therefore every element can be presented as its natural basis as its linear combination of $a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_7\alpha^7$.

Take α^8 as an example:

$$\alpha^8(a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_7\alpha^7)$$

$$= a_0\alpha^8 + a_1\alpha^9 + a_2\alpha^{10} + \cdots + a_7\alpha^{15}$$

$$= a_0(1+ \alpha^2 + \alpha^3 + \alpha^4) + a_1(\alpha^5 + \alpha^4 + \alpha^3 + \alpha) + a_2(\alpha^6 + \alpha^5 + \alpha^4 + \alpha^2) + a_3(\alpha^7 + \alpha^6 + \alpha^5 + \alpha^3)$$

$$+ a_4(\alpha^7 + \alpha^6 + \alpha^3 + \alpha^2 + 1) + a_5(\alpha^7 + \alpha^2 + \alpha + 1) + a_6(\alpha^4 + \alpha + 1) + a_7(\alpha^5 + \alpha^2 + \alpha)$$

$$= \alpha^7(a_5 + a_4 + a_3) + \alpha^6(a_4 + a_3 + a_2) + \alpha^5(a_7 + a_3 + a_2 + a_1) + \alpha^4(a_6 + a_2 + a_1 + a_0)$$

$$+ \alpha^3(a_4 + a_3 + a_1 + a_0) + \alpha^2(a_7 + a_5 + a_4 + a_2 + 0) + \alpha(a_7 + a_6 + a_5 + a_1) + (a_6 + a_5 + a_4+a_0)$$

So every multiplier can be converted into adders, which are more convenient and fast to implement.

It should be known that the dimension of Reed-Solomon code can be shortened to meet the requirement of different applications. The (255,223) RS code is commonly used today due to its right symbol size that exactly matches the 8-bit data byte. And also in order to match any system specification, the 255 code length can be truncated or shortened if the major in the encoder and decoder hardware circuitry does not change.

The concept of the RS encoder is mainly a circuit that divides by g(x). The encoder is designed by using a feedback shift register. It is known that the feedback shift register contains the 32 parity check digits after the last message digit is loaded, and the message digits are shifted sequentially. The 32 parity check digits are appended to the 223 message digits to form the corresponding codeword as which is transmitted over the channel. There are some shift registers, adders, multipliers, and switches, which can show the working of the system.[20]

There are three steps in the encoding procedure. Firstly, multiply the message polynomial m(X) by X^{2t} (2t = n-k). Secondly, divide $X^{2t}m(X)$ by g(X) to get the remainder b(X). Thirdly, form the code word U(X), which is equal to $b(X) + X^{2t} m(X)$. All these three steps can be done with a division circuit which is a linear (n-k)-stage shift register with feedback connections based on the generator polynomial g(X).[2]

The encoding operation is implemented as follows:

Step 1: turn on the gate, the k information digits $m_0, m_1, m_2, \cdots, m_{k-1}$ are shifted into the circuit and the communication channel at the same time. Shifting the message m(X) in to the circuit form the front end is equivalent to pre-multiply m(X) by X^{2t}. Once the complete message has entered the circuit, the n-k digits in the digits in the register form the remainder and hence they are the parity check digits.

Step 2: turn off the gate to break the feedback connection.

Step 3: shift the parity check digits out and send them into the channel. These n-k parity check digits with the k information digits form a complete code vector.

The working of the system is shown in Figure 4.1.

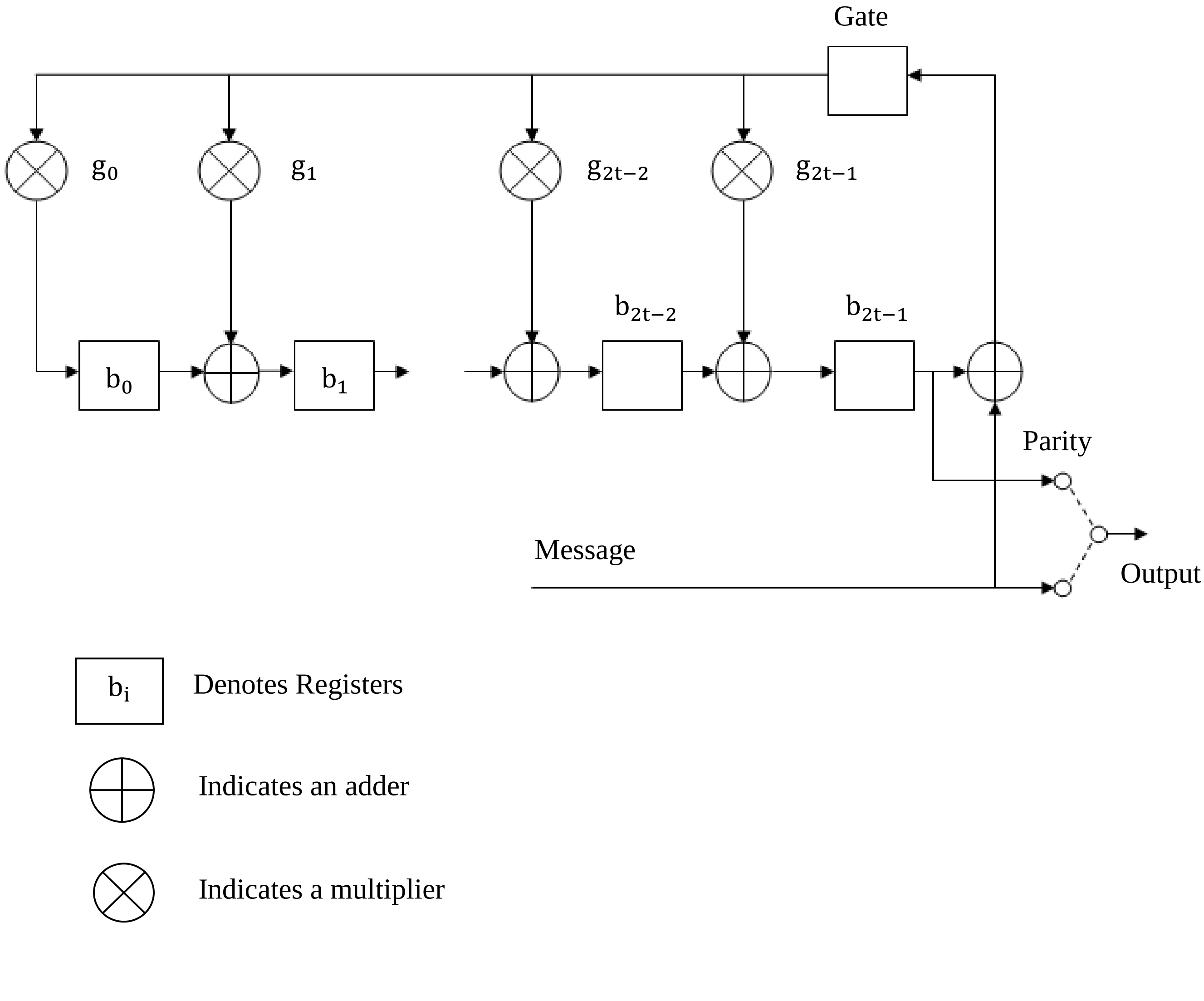


Figure 4.1 Encoding circuit for RS code

Chapter 5 Reed-Solomon Code Decoding

5.1 The Procedure of Reed-Solomon Code Decoding

The RS-Decoding procedure is shown in Figure 5.1 below:

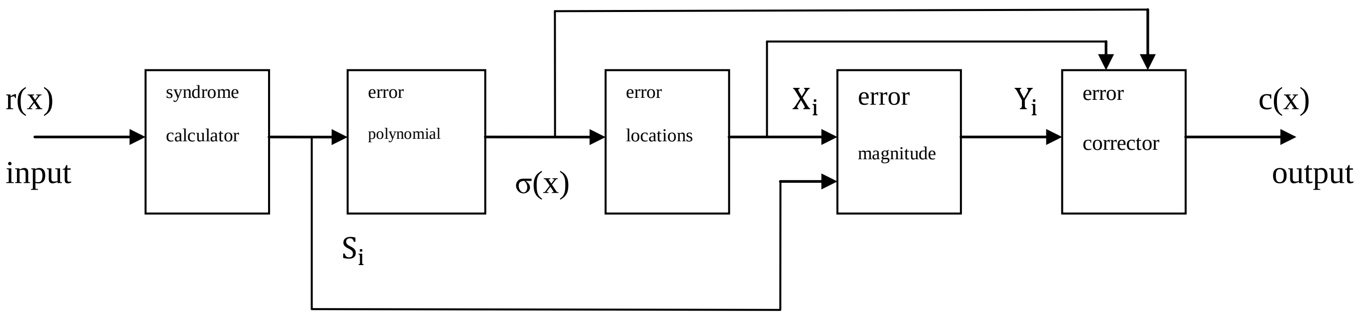


Figure 5.1 The diagram of RS-Decoding procedure

The description is as follows:

The r(x) is the code word received by the decoder. During transmission, some errors also can occur, so r(x) is the c(x) in addition to the e(x), which is the error polynomial.

The RS decoder checks for the error and attempts to correct it.

$$r(x) = c(x) + e(x)$$

There are four steps in the decoding process.

First, let r(X) be the received code polynomial. The syndromes' computation is as follows:

$S_1 = r(\alpha) = e_{j1}\alpha^{j1} + e_{j2}\alpha^{j2} + \ldots + e_{jv}\alpha^{jv}$

$S_2 = r(\alpha^2) = e_{j1}\,(\alpha^{j1})^2 + e_{j2}\,(\alpha^{j2})^2 + \ldots + e_{jv}\,(\alpha^{jv})^2$

.

.

.

$S_{2t} = r(\alpha^{32}) = e_{j1}\,(\alpha^{j1})^{32} + e_{j2}\,(\alpha^{j2})^{32} + \ldots + e_{jv}\,(\alpha^{jv})^{32}$

After the syndrome calculation, the second step is to find the coefficients of the error location polynomial. The Berlekamp's iterative algorithm is used here. Since 2t = n – k = 255 – 223 = 32, so there are 32 iterations to be performed which are given below:

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	S_1	0	0
1				
2				
…				
32				

It is known that the algorithm will yield the minimum-degree polynomial, which will satisfy the first μ Newton's equations.[5] In this table, the first two rows are the initial starting points of the algorithm, and d_μ is the discrepancy value from the previous row. If $d_\mu = 0$, then the $\sigma^{(\mu)}(X)$ will be entered as in the previous step, and if $d_\mu \neq 0$, a correction factor must be added, and the entry will be as follows:

$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) + d_\mu d_p^{-1} X^{(\mu-p)} \sigma^{(p)}(X)$

In the above equation, p is a previous iteration step such that $d_p \neq 0$ and $p - l_p$, which is from the last column has the largest value, and for the last two columns, the next entries are:

$l_{\mu+1} = \max[0, (\mu - l_\mu) - (p - l_p)] + l_\mu,$

and for the last column is just a simple subtraction.[5]

In order to get the last entry for the current iteration step, we have to find the new discrepancy value $d_{\mu+1}$ as follows:

$$d_{\mu+1} = S_{\mu+2} + \sigma_1^{(\mu+1)} S_{\mu+2-1} + \cdots + \sigma_{l_{\mu+1}}^{(\mu+1)} S_{\mu+2-l_{\mu+1}}$$

At step 32, which means after the last step of the iteration process, we find the correct error location polynomial from column $\sigma^{(\mu)}(X)$ in the table only if there are fewer than 16 errors in the data.

The third step of the decoding process is to find the roots of the error location polynomial. Chien's searching algorithm is used in this step. First, load the registers with the coefficients of the error location polynomial. Then, at each clock pulse, every register is multiplied by the value held in the feedback loop. All the registers are XOR-ed together with the unity element of $GF(2^8)$ after each clock cycle. A root has been found when the XOR result equals 0. The feedback values are chosen as the powers of α (the primitive element), so all the nonzero elements of $GF(2^8)$ are eventually checked as roots of the error location polynomial $\sigma(X)$ after 254 trials, and the trials resulting in a root are saved as the inverses of the error positions.[6] The block diagram of Chien's searching algorithm is shown in Figure 5.2.

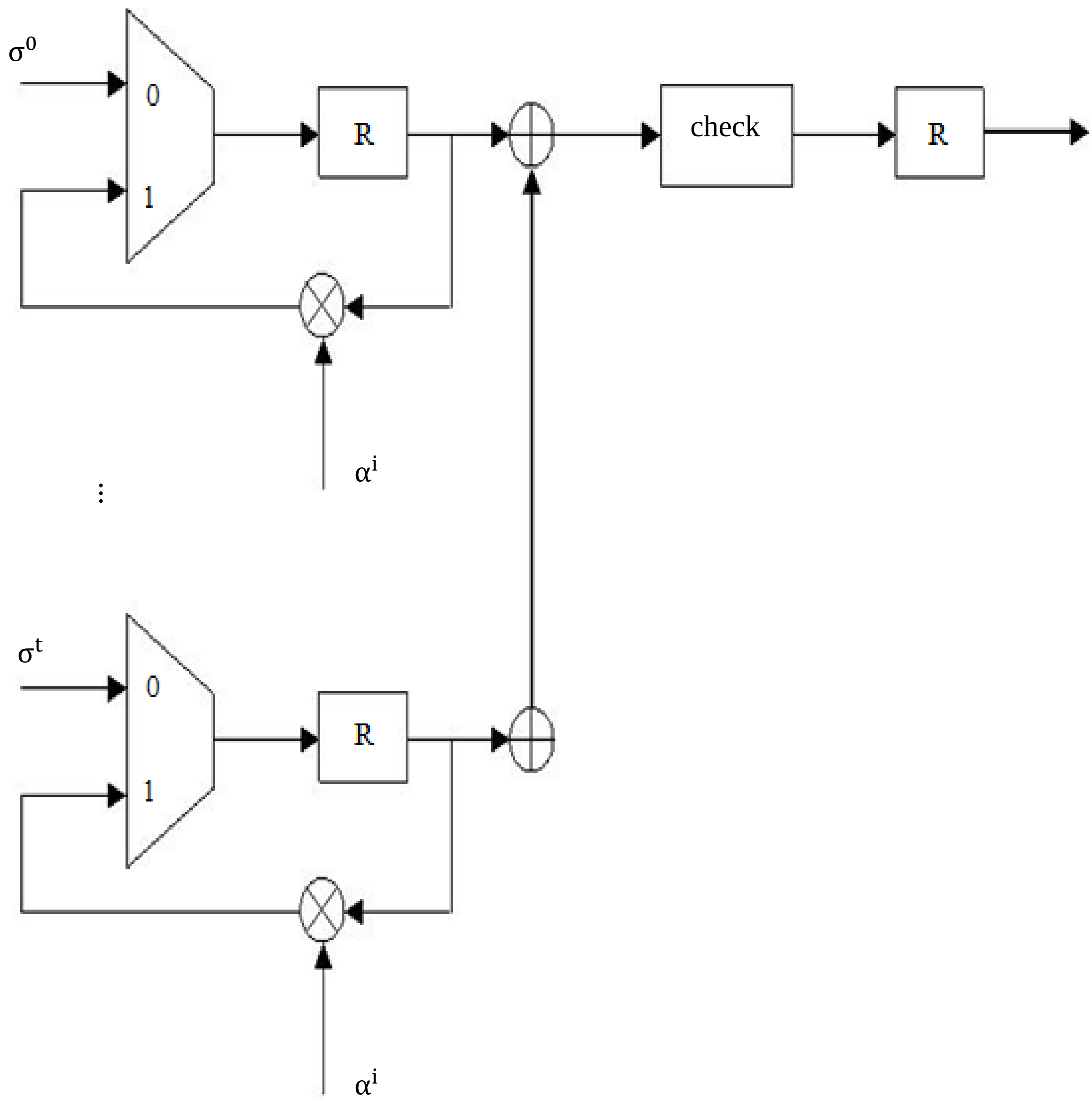


Figure 5.2 The block diagram of Chien’s searching algorithm

The last step in the decoding process is to find the error values and do the error correction by first constructing Z(X), which is shown as follows:

$Z(X) = 1 + (S_1 + \sigma_1)X + (S_2 + \sigma_1 S_1 + \sigma_2)X^2 + \cdots + (S_v + \sigma_1 S_{v-1} + \sigma_2 S_{v-2} + \cdots + \sigma_v)X^v$

And the error value is:

$$e_{j_l} = \frac{Z(\beta_l^{-1})}{\prod_{\substack{i=1 \\ i \neq l}}^{v} (1 + \beta_i \beta_l^{-1})} \qquad \text{where } \beta_l = \alpha^{jl}$$

The decoding is completed by taking r(X) – e(X) after we get the error pattern e(X).

5.2 Detailed Reed-Solomon Decoding

The code word $c(x) = c_0 + c_1 x + c_2 x^2 + \cdots c_{n-1}x^{n-1}$

The error $e(x) = e_0 + e_1 x + e_2 x^2 + \cdots e_{n-1}x^{n-1}$

The received vector $r(x) = r_0 + r_1 x + r_2 x^2 + \cdots r_{n-1}x^{n-1}$

In order to get the code word c(x), as we known, c(x) = r(x) - e(x)

The decoding steps are as follows:

Firstly, find the syndrome s(x) and the key equation. Next, find the error location polynomial σ(x) and the error-value evaluator ω(x). Then, using Chien's searching algorithm to find the error location number x_i, after ward evaluate y_i (the error-value polynomial). We thus find the information according to r(x) and y_i.

5.2.1 Calculate the Syndrome

The first step is calculating the syndrome s(x), since $s(x) = s_1 + s_2 x + s_3 x^2 + \cdots s_{2t}x^{2t-1}$

The coefficients $s_i = r(\alpha^i) = \sum_{j=0}^{n-1} r_j (\alpha^i)^j$

If $s_i = 0$, there is no error; if $s_i \neq 0$, the error occurs, the system has to do the error correction.

$s_i = r(\alpha^i) = e(\alpha^i) + c(\alpha^i)$, where $c(\alpha^i) = 0$.

Therefore $s_i = e(\alpha^i) = \sum_{j=0}^{n-1} e_j(\alpha^i)^j = e_0 + e_1\alpha^i + \cdots + e_{n-1}(\alpha^i)^{n-1}$

Let $e(x) = y_1x_1 + y_2x_2 + \cdots + y_ex_e$, where y_i and x_i are stand for the value and the error location of the i'th error respectively.

Since $(\alpha^i)^j = (\alpha^j)^i$, which can be written as $(\alpha^i)^j = (x_j)^i$, therefore $s_i = \sum_{j=1}^{t} y_i(x_j)^i$ which gives the following:

$$s_1 = y_1x_1 + y_2x_2 + \cdots + y_tx_t$$

$$s_2 = y_1(x_1)^2 + y_2(x_2)^2 + \cdots + y_t(x_t)^2$$

$$\vdots$$

$$s_{2t} = y_1(x_1)^{2t} + y_2(x_2)^{2t} + \cdots + y_t(x_t)^{2t}$$

5.2.2 Finding the Key Equation

The next step is finding the key equation. In order to get the error value y_i and the error location x_i, let the error location polynomial be $\sigma(x)$, and $\sigma(x) = \prod_{i=1}^{t}(1 - xx_i)$

Since $x = x_i^{-1}$, therefore $\sigma(x_i^{-1}) = \prod_{i=1}^{t}(1 - x_i^{-1}x_i)$

If we want to get error locations, we have to solve the equation $\sigma(x) = 0$.

$$\sigma(x) = \prod_{i=1}^{t}(1 - xx_i) = 1 + \sigma_1x + \sigma_2x^2 + \cdots + \sigma_tx^t = 0$$

$$\sigma(x_i{}^{-1}) = \prod_{i=1}^{t}(1 - x_i{}^{-1}x_i) = 1 + \sigma_1 x_i{}^{-1} + \sigma_2 x_i{}^{-2} + \cdots + \sigma_t x_i{}^{-t} = 0$$

Both sides of the equation multiply by $y_i(x_i)^{t+j}$, we get the following,

$$y_i x_i^{t+j} + \sigma_1 y_i x_i^{t+j-1} + \cdots + \sigma_{t-1} y_i x_i^{j+1} + \sigma_t y_i x_i^{j} = 0$$

Take the summation and we get

$\sum_{i=1}^{t} y_i x_i^{t+j} + \sigma_1 \sum_{i=1}^{t} y_i x_i^{t+j-1} + \cdots + \sigma_{t-1} \sum_{i=1}^{t} y_i x_i^{j+1} + \sigma_t \sum_{i=1}^{t} y_i x_i^{j} = 0$, where i and j are both equal to 1, 2, ⋯ , t.

Since we know, $s_{t+j} = \sum_{i=1}^{t} y_i x_i^{t+j}$ which is derived from $s_i = \sum_{j=1}^{t} y_i (x_j)^i$.

Hence we can get $s_{j+t} + \sigma_1 s_{j+t-1} + \sigma_2 s_{j+t-2} + \cdots + \sigma_t s_j = 0$ (where j=1, 2,⋯, t), which can be expanded as the following,

$$\sigma_1 s_t + \sigma_2 s_{t-1} + \cdots + \sigma_t s_1 = -s_{t+1}$$

$$\sigma_1 s_{t+1} + \sigma_2 s_t + \cdots + \sigma_t s_2 = -s_{t+2}$$

$$\vdots$$

$$\sigma_1 s_{2t-1} + \sigma_2 s_{2t-2} + \cdots + \sigma_t s_t = -s_{2t}$$

The value of σ_i can be determined by solving the above equations.

Assume $s(x) = \sum_{i=0}^{\infty} s_i x^i = s_0 + s_1 x + s_2 x^2 + \cdots$

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2 + \cdots + \sigma_t x^t ,$$

$$\omega(x) = s(x)\sigma(x)$$

$\omega(x) = s_0 + (s_1 + s_0\sigma_1)x + (s_2 + s_1\sigma_1 + s_0\sigma_2)x^2 + \cdots + (s_t + s_{t-1}\sigma_1 + s_{t-2}\sigma_2 +$

$\cdots + s_0\sigma_t)x^t + (s_{t+1} + s_t\sigma_1 + s_{t-1}\sigma_2 + \cdots + s_1\sigma_t)x^{t+1} + \cdots$

Since $s_0 = 1$, therefore $\omega(x) = 1 + \omega_1 x + \omega_2 x^2 + \cdots + \omega_t x^t + \omega_{t+1}x^{t+1} + \cdots$

The $\omega(x)$ is the error value evaluator polynomial, and the key equation is as following:

$s(x)\sigma(x) = \omega(x) \bmod x^{2t+1}$.

5.2.3 Berlekamp-Massey Algorithm and Improved Berlekamp-Massey Algorithm

The Berlekamp-Massey(BM) iteration algorithm is adopted to solve the key equation due to the complication of solving it directly.

First, let D(j) be the lowest power of $\sigma^j(x)$ achieved from the (j+1)th iteration if $\omega^j(x)$ and $\sigma^j(x)$ satisfy the key equation; d(j) is the difference between the (j+1)th and jth iteration, if $\omega^j(x)$ and $\sigma^j(x)$ do not satisfy the key equation and d(j) satisfies the following equation:

$$s(x)\sigma^j(x) = \omega^j(x) + d_j x^{j+1} \bmod x^{j+2}$$

Since $s(x)\sigma^j(x) = (s_0 + s_1 x + s_2 x^2 + \cdots)(1 + \sigma_1^j x + \sigma_2^j x^2 + \cdots)$

$$= s_0 + \left(s_1 + s_0\sigma_1^j\right)x + \cdots + (s_{j+1} + s_j\sigma_1^j + s_{j-1}\sigma_2^j + \cdots + s_0\sigma_{j+1}^j)x^{j+1}$$

Therefore d(j) can be written as $d(j) = s_{j+1} + \sum_{i=1}^{\partial(\sigma^j(x))} s_{j+1-i}\sigma_i^j$, where σ_i^j is the coefficient of x^i in $\sigma^j(x)$. And then we can get:

$$\sigma^{j+1}(x) = \sigma^j(x) - d_j d_i^{-1} x^{j-i}\sigma^j(x)$$

$$\omega^{j+1}(x) = \omega^j(x) - d_j d_i^{-1} x^{j-i}\omega^j(x)$$

$$D(j+1) = \max(D(j), j - i + D(j))$$

The BM algorithm can be described as the following steps:

Firstly, set the initial values as following:

$$\sigma^{-1}(x) = 1, \omega^{-1}(x) = 0, \sigma^{0}(x) = 1, \omega^{0}(x) = 1, D(-1) = 0, d_{-1} = 1, D(0) = 0, d_{0} = s_{1}$$

Secondly, determine the value of d_j:

if $d_j = 0$, calculate d_{j+1} and do the next iteration.

When $d_j = 0$, $\sigma^{j+1}(x) = \sigma^{j}(x)$,

$\omega^{j+1}(x) = \omega^{j}(x)$,

$D(j + 1) = D(j)$.

If $d_j \neq 0$, i - D(i) that in row i is the greatest among all the rows which are in front of row j.

After that, we can calculate $\sigma^{j+1}(x)$ and $\omega^{j+1}(x)$ to solve for the j+1th step.

The last step is to repeat the above procedures, and we can get σ(x) and ω(x) after 2t iterations in the end.

An improved Berlekamp-Massey algorithm is adopted since the calculation in the original is so complicated and requires the inverse operation to be used.[10] The improved BM algorithm does not need inverse operations, and the iteration steps are shown below:

Firstly, set the initial values as following:

$$\sigma^{0}(x) = 1, \omega^{0}(x) = 1, \lambda^{0}(x) = 1, \beta^{0}(x) = 1, l_{0} = 0, \gamma_{0} = 1$$

The iteration equation is $\delta_{k+1} = \sum_{j=0}^{l_k} s_{k+1-j}\sigma_j^k$

Where $\sigma^{k+1}(x) = \gamma_k\sigma^k(x) - \delta_{k+1}\lambda^k(x)x$

$\omega^{k+1}(x) = \gamma_k\omega^k(x) - \delta_{k+1}\beta^k(x)x$.

If $\delta_{k+1} = 0$ or $2l_k > k$, then $\lambda^{k+1}(x) = x\lambda^k(x)$, $\beta^{k+1}(x) = x\beta^k(x)$

$$l_{k+1} = l_k, \gamma_{k+1} = \gamma_k$$

And if $\delta_{k+1} \neq 0$ and $2l_k \leq k$, then $\lambda^{k+1}(x) = \sigma^k(x)$

$$\beta^{k+1}(x) = \omega^k(x)$$

$$l_{k+1} = k + 1 - l_k$$

$$\gamma_{k+1} = \delta_{k+1}$$

5.2.4 Chien's Searching Algorithm

Chien's searching algorithm is adopted to search for error locations since after we solve the error location polynomial $\sigma(x)$ with the roots of $\sigma(x)$ we can determine the error locations x_i.

The error location polynomial $\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2 + \cdots + \sigma_t x^t$

The received vector polynomial $r(x) = r_0 + r_1 x + r_2 x^2 + \cdots + r_{n-1} x^{n-1}$

Then we have $\sigma(\alpha^{-i}) = \sigma_0 + \sigma_1 \alpha^{-i} + \sigma_2 \alpha^{-2i} + \cdots + \sigma_i \alpha^{-ti}$

If $\sigma(\alpha^{-i}) = 0$, r_i is corrupted with error; if $\sigma(\alpha^{-i}) \neq 0$, r_i with no error corrupted.

5.2.5 Calculate the Error Value

The error values can be obtained by plug the error locations into the syndrome calculation.

Let k be the number of errors, so k≤t, and $s_i = \sum_{j=1}^{k} y_j x_j^i$

Since $s_0 = 1$, then $s(x) = \sum_{i=1}^{\infty} s_i x^i = s_0 + s_1 x + s_2 x^2 + \cdots = 1 + \sum_{i=1}^{\infty}(\sum_{j=1}^{k} y_j x_j^i) x^i$

$$=1 + (\sum_{j=1}^{k} y_j (\sum_{i=1}^{\infty} (x_j x)^i)$$

If $|x_j x| < 1$, we have $\sum_{i=1}^{\infty}(x_j x)^i = \frac{x_j x}{1 - x_j x}$

Therefore $s(x) = 1 + (\sum_{j=1}^{k} y_j \frac{x_j x}{1 - x_j x})$

According to Forney's algorithm, in the end $y_i = \frac{-x_i \omega(x_i^{-1})}{\sigma'(x_i^{-1})}$

By this point, the error locations x_i and the error values y_i are obtained, so the error correction can be performed, and we can get the correct codeword c(x) = r(x) - e(x).

Chapter 6 Modeling and Simulations of Reed-Solomon Encoder and Decoder

6.1 Modeling of the Encoder

The design top level block diagram of the encoder is shown in Figure 6.1, in which "x[7⋯0]" is the 8-bit input data; "enable" is the control signal that enables the encoder; "data" is the message signal; "clk" is the clock signal; "rst_n" is the reset signal; "y[7⋯0]" is the 8-bit output data.

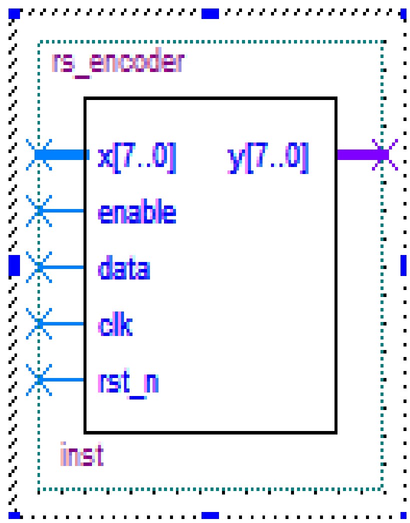


Figure 6.1 The top level block diagram of the Encoder

The RTL description of the encoder is shown in Appendix A, and the Verilog HDL files for the encoder are included in Appendix B.

6.2 Modeling of the Decoder

The following block diagrams of different modules are obtained from the software Altera Quartus Ⅱ after compilation.

The block diagram of the top level decoder module is shown in Figure 6.2, in which "x[7⋯0]" is the 8-bit long input data; "enable" is the control signal that enables the decoder; "k[7⋯0]" is the 8-bit input signal; "clk" is the clock signal; "rst_n" is the reset signal;

“error[7⋯0]” is the 8-bit output error data; “with_error” is the output data with error; “valid” is the output data.

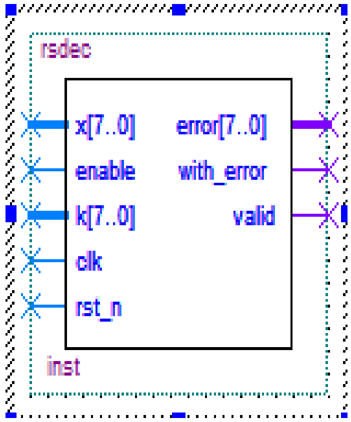


Figure 6.2 Block Diagram of the Top Level Decoder Module

The block diagram of the syndrome calculation module is shown in Figure 6.3, in which “u[7⋯0]” is the 8-bit long input data; “enable” is the control signal that enables the syndrome calculation; “shift” is the shifted signal; “init” is the start flag of this module; “clk” is the clock signal; “rst_n” is the reset signal; “y0[7⋯0]” to “y31[7⋯0]” are thirty two 8-bit long output syndromes.

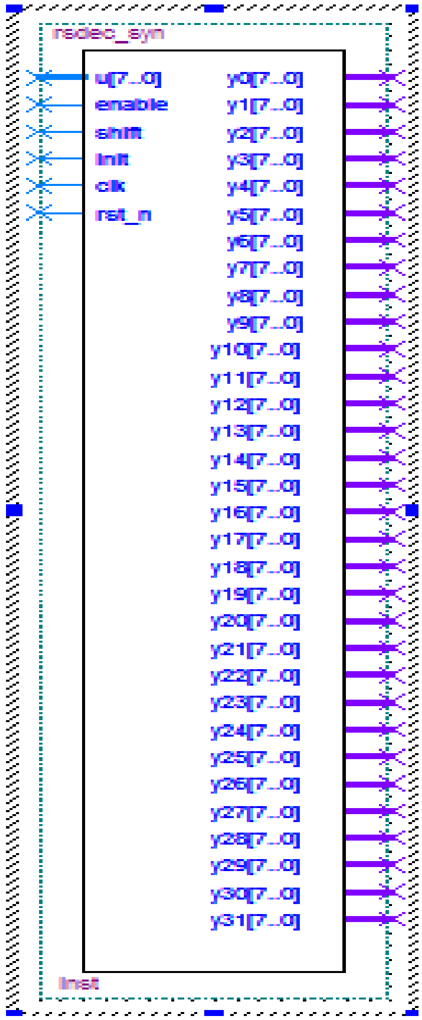


Figure 6.3 Block Diagram of the Syndrome Calculation Module

The block diagram of the multiplier is shown in Figure 6.4, in which “a[7⋯ 0]” is the 8-bit input data; “b[7⋯ 0]” is the 8-bit long input data; “y[7⋯ 0]” is the 8-bit long output data.

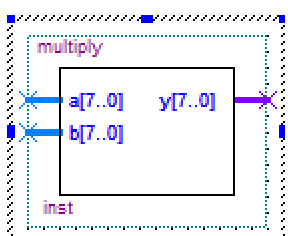


Figure 6.4 Block Diagram of the Multiplier

The block diagram of error location polynomial calculation is shown in Figure 6.5, in which "syndrome0[7⋯ 0]" to "syndrome31[7⋯ 0]" are the 32 syndromes; D[7⋯ 0] is the 8-bit long input signal; "count" is the counted input signal; "phase0" and "phase32" are the input phase signal; "enable" is the control signal that enables the Berlekamp-Massey algorithm; "clk" is the clock signal; "rst_n" is the reset signal; "lambda_out[7 ⋯ 0]" is the output σ(x); "omega_out[7⋯ 0]" is the output ω(x); "D[7⋯ 0]" is the 8-bit long output data; "a", "b", "c", "d" and "e" are the input data; "y[7⋯ 0]" is the output data.

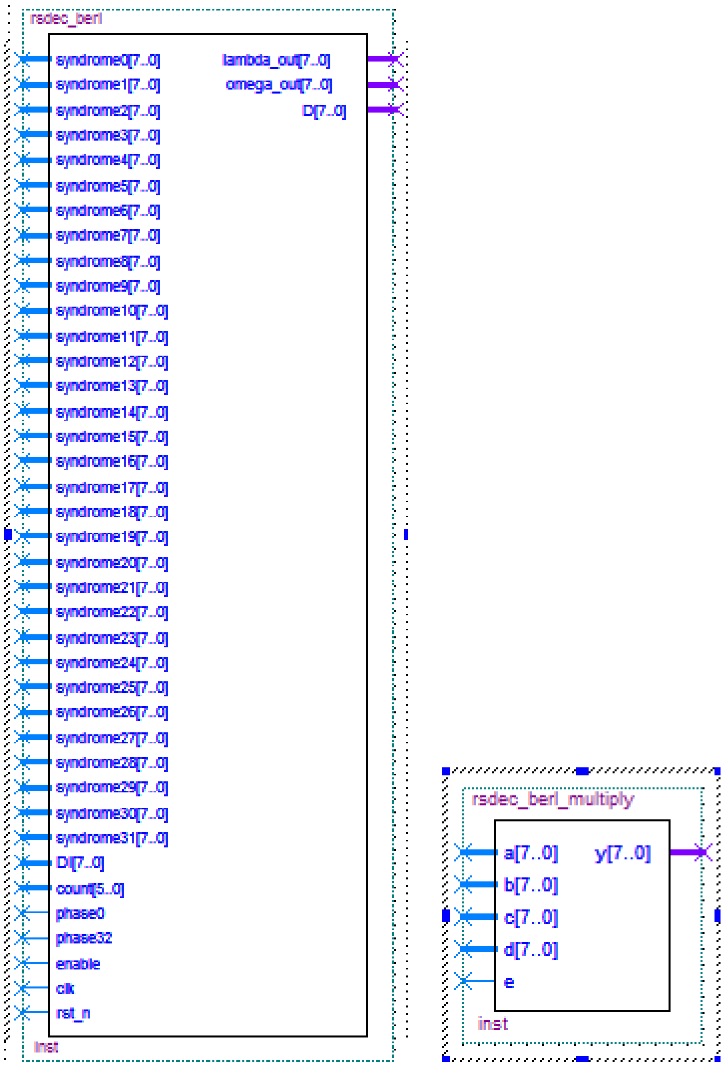


Figure 6.5 The Block Diagram of Error Location Polynomial Calculation

(Berlekamp-Massey Algorithm)

The block diagram of Chien's searching algorithm is shown in Figure 6.6. In the figure, "lambda[7⋯0]" is the input σ(x); "omega[7⋯0]" is the input ω(x); D[7⋯0] is the input data; "search" is the input search signal; "load" is the loaded signal; "shorten" is the shortened signal; "clk" is the clock signal; "rst_n" is the reset signal; "error[7⋯0]" is the output error data; "alpha[7⋯0]" and "even[7⋯0]" are the 8-bit long output data.

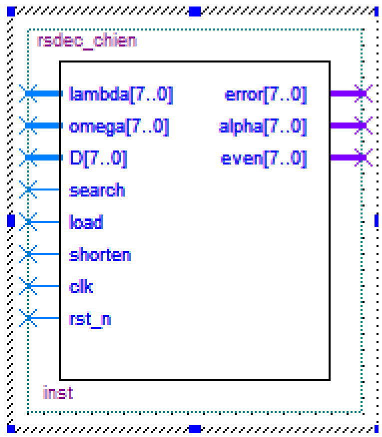


Figure 6.6 Block Diagram of Chien's Searching Algorithm

The block diagram of inverse calculation is shown in Figure 6.7. In this figure, "x[7⋯0]" is the 8-bit long input data, and "y[7⋯0]" is the 8-bit long output data.

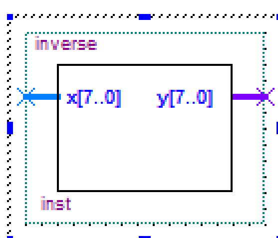


Figure 6.7 Block Diagram of Inverse Calculation

The block diagram of power calculation is shown in Figure 6.8, in which "address[7⋯0]" is the input address, and "data[7⋯0]" is the 8-bit long output data.

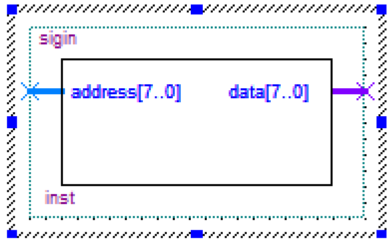


Figure 6.8 Block Diagram of Power Calculation

All the different modules Verilog HDL files for the decoder are included in Appendix C.

6.3 Simulation of the Encoder

The software Altera Quartus Ⅱ and Modelsim are used to perform compilation and synthesis. In the test module, an input of decimal numbers from 1 to 255 is created. The input is transmitted into the encoder, the encoder calculates, and the received output is the code word.

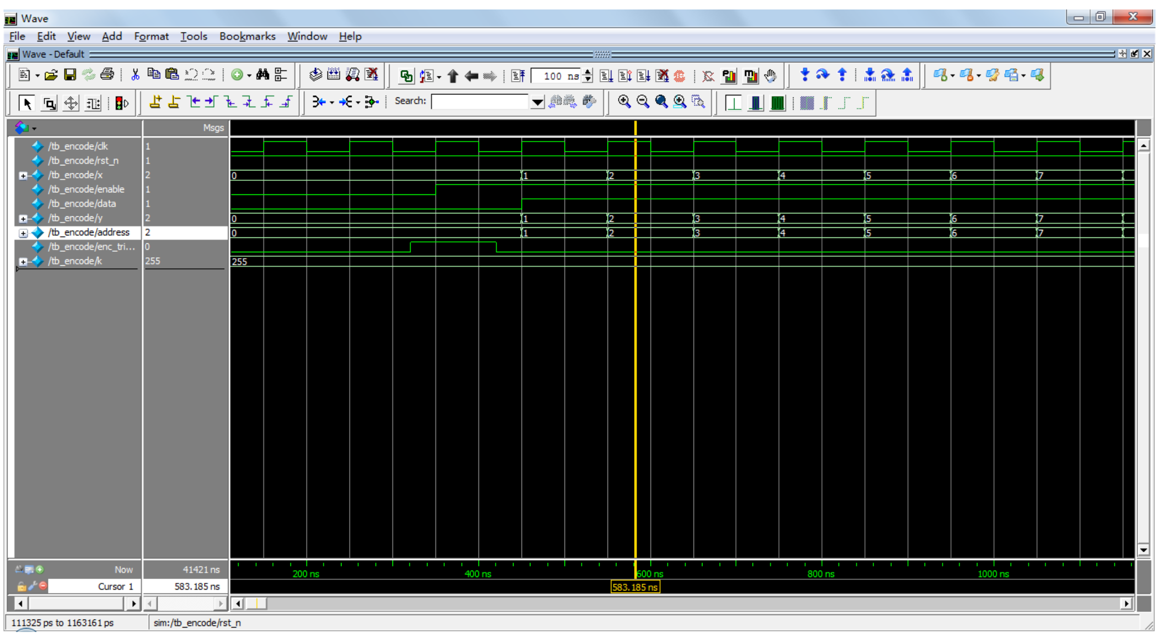


Figure 6.9 Simulation Waveform for the Encoder part 1

In Figure 6.9, signal ‘clk’ is 1, ‘rst_n’ is 1, ‘enable’ is 1, it is clear to see that the encoder output y is 2 which is same as the input x.

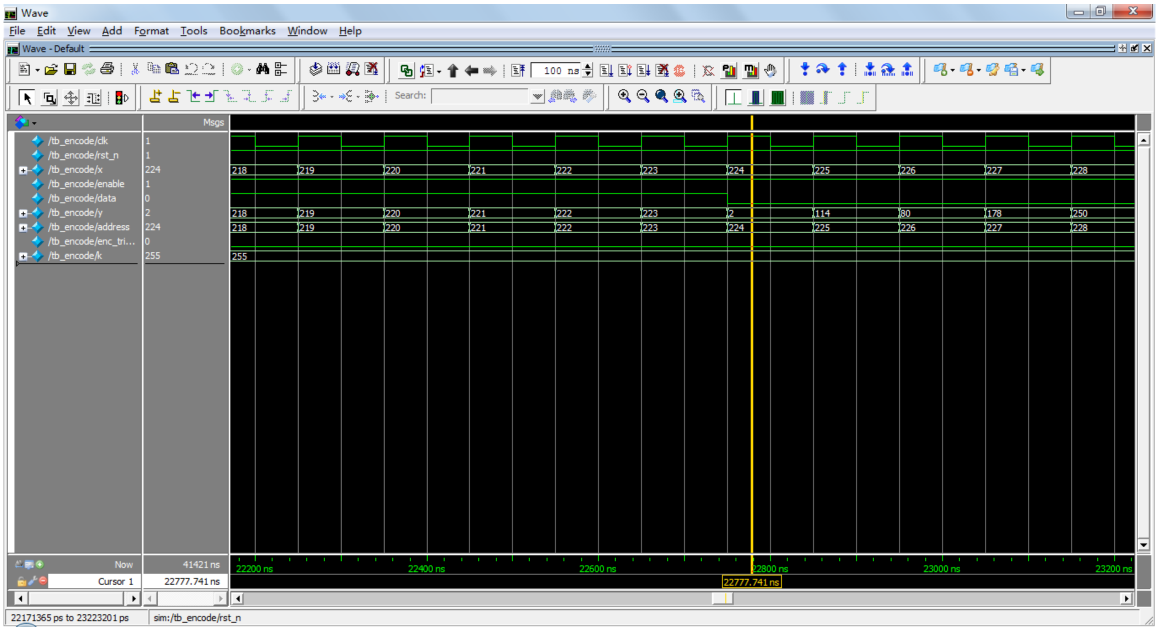


Figure 6.10 Simulation Waveform for the Encoder part 2

In Figure 6.10, signal ‘clk’ is 1, ‘rst_n’ is 1, ‘enable’ is 1, the input signal x is 224, and after the encoder, the output y is 2. From this point (include this point), the following numbers indicates the parity check digits as shown in Figure 6.3 as following:

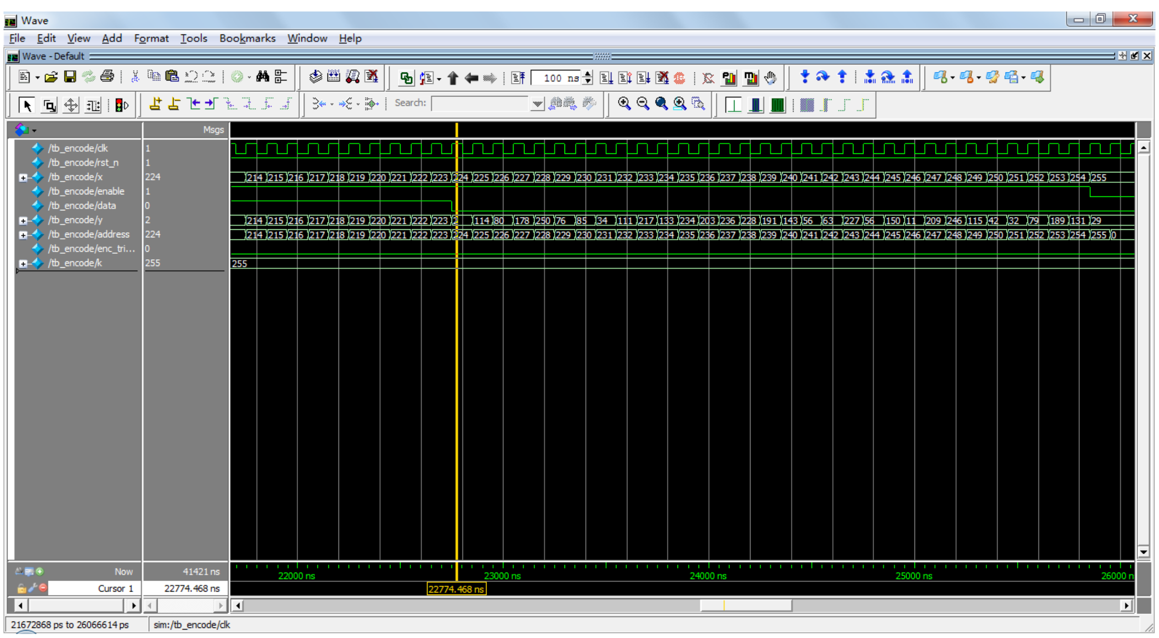


Figure 6.11 Simulation Waveform for the Encoder part 3

In Fugure 6.11, the numbers in output y: 2, 114, 80, 178, 250, 76, 85, 34, 111, 217, 133, 234, 203, 236, 228, 191, 143, 56, 63, 227, 56, 150, 11, 209, 246, 115, 42, 32, 79, 189, 131, 29. These are the parity check digits for the (255,223) Reed-Solomon Code.

The hardware design of the encoder is realized using EP2S15F484C3 of the Stratix II. The analysis is shown in Figure 6.12.

Flow Summary

Flow Status	Successful - Wed Mar 19 23:36:15 2014
Quartus II Version	9.0 Build 132 02/25/2009 SJ Full Version
Revision Name	rs_encoder
Top-level Entity Name	rs_encoder
Family	Stratix II
Met timing requirements	Yes
Logic utilization	2 %
Combinational ALUTs	279 / 12,480 (2 %)
Dedicated logic registers	256 / 12,480 (2 %)
Total registers	256
Total pins	20 / 343 (6 %)
Total virtual pins	0
Total block memory bits	0 / 419,328 (0 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2S15F484C3
Timing Models	Final

Figure 6.12 Flow Summary of the Encoder

The encoder was successfully designed, as proven from the above figures and the flow summary.

6.4 Simulation of the Decoder

In the decoder part, the software Altera Quartus II and Modelsim are used to perform the simulation. A test bench is created to do the simulation. The Verilog HDL files created are shown in the Appendix C.

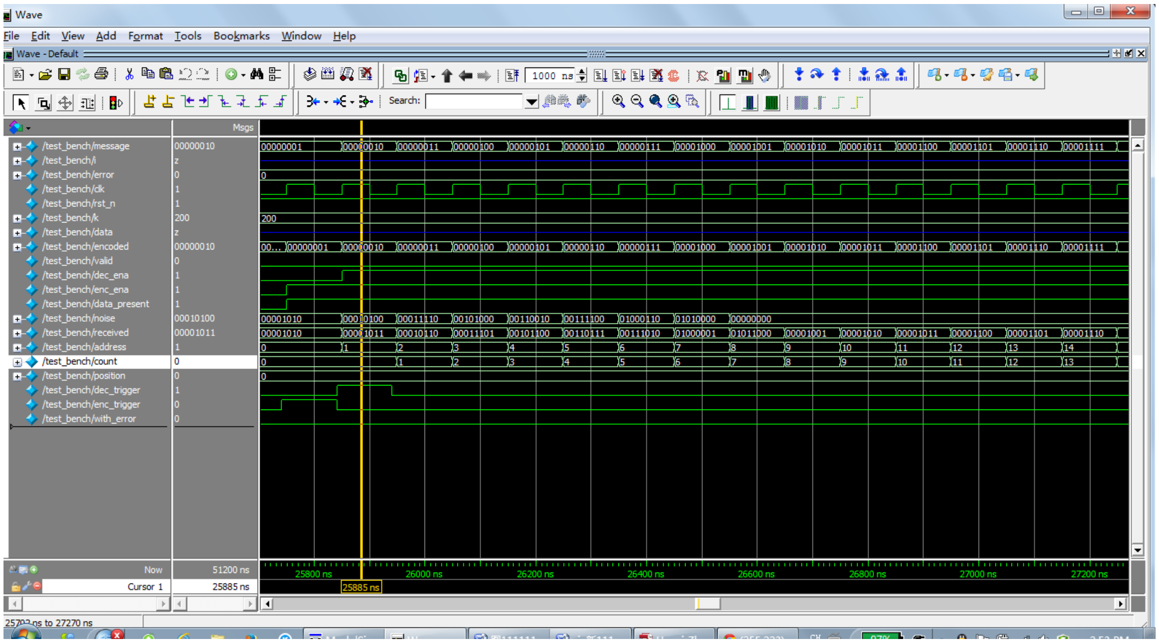


Figure 6.13 Simulation Waveform for the Decoder part 1

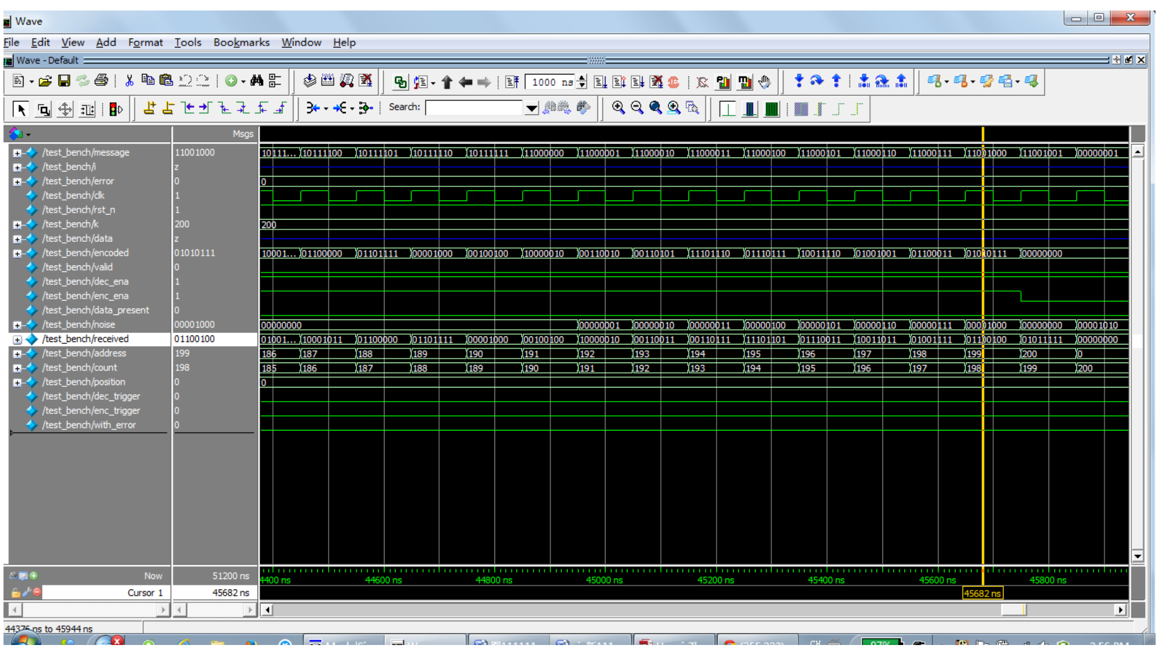


Figure 6.14 Simulation Waveform for the Decoder part 2

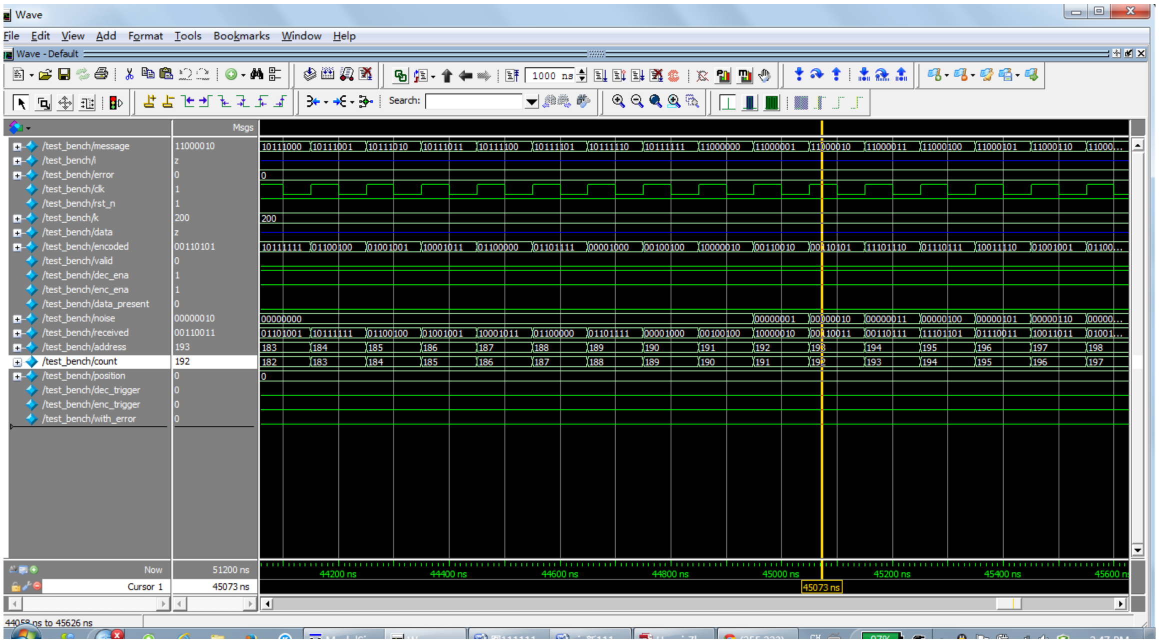


Figure 6.15 Simulation Waveform for the Decoder part 3

In Figure 6.15, the received signal is equal to the encoded signal plus the noise. The encoded data is 00110101, the noise is 00000010, and the received data is 00110111 in the next clock, which satisfies the requirement for the decoder. And the decoder can correct less than or equal to 16 errors.

The hardware design of the decoder is realized using EP2S15F484C3 of the Stratix II. The analysis is shown in figure 6.16 in the next page.

Flow Summary

Flow Status	Successful - Thu Mar 20 00:28:41 2014
Quartus II Version	9.0 Build 132 02/25/2009 SJ Full Version
Revision Name	rsdec
Top-level Entity Name	rsdec
Family	Stratix II
Met timing requirements	Yes
Logic utilization	38 %
Combinational ALUTs	2,691 / 12,480 (22 %)
Dedicated logic registers	2,116 / 12,480 (17 %)
Total registers	2116
Total pins	29 / 343 (8 %)
Total virtual pins	0
Total block memory bits	0 / 419,328 (0 %)
DSP block 9-bit elements	0 / 96 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2S15F484C3
Timing Models	Final

Figure 6.16 Flow Summary of the Decoder

The decoder was successfully designed, as proven from the above figures and the flow summary.

Chapter 7 Conclusion

7.1 Conclusion

In digital data communication systems, transferred information is always liable to corruption by noise within the digital communication channel. Therefore, error detection and correction codes are widely used in the modern digital communication field and data storage systems.

In this project, the design, modeling, simulation and verifications of a (255,223) Reed-Solomon code encoder and decoder is presented. RS codes are very useful in digital communication systems due to their capability of correcting multiple errors as well as burst errors with a high code rate. For the (255,223) RS code presented in this project, the code rate is 223/255 = 0.8745, and it can detect and correct up to 16 erroneous bytes in every block of 255 bytes.

The encoder is implemented using a linear feedback shift register which divides by g(X) in order to get the remainder, and then form the code word.

The decoder uses Berlekamp's iteration algorithm for finding the coefficients of the error location polynomial and Chien's searching algorithm for finding its roots.

The design of both the encoder and decoder are modeled and simulated using Verilog HDL running on Altera Quartus Ⅱ and ModelSim. Eventually, the correction of the design has been verified by writing and running a test bench.

Reference

[1] Roth, Ron. *Introduction to Coding Theory*. Cambridge: Cambridge UP, 2006. Print.

[2] Lin, Shu, and Daniel J. Costello. *Error Control Coding: Fundamentals and Applications*. Upper Saddle River, NJ: Pearson-Prentice Hall, 2004. Print.

[3] Lin, Shu. *An Introduction to Error Correcting Codes*. Englewood Cliffs, NJ: Prentice-Hall, 1970. Print.

[4] E.R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1967.

[5] Wicker, Stephen B., and Vijay K. Bhargava. *Reed-Solomon Codes and Their Applications*. Piscataway, NJ: IEEE, 1994. Print.

[6] Blahut, Richard E. *Theory and Practice of Error Control Codes*. Reading, MA: Assison-Wesley Pub., 1983. Print.

[7] Reed, I. S., and G. Solomon. "*Polynomial Codes Over Certain Finite Fields*." Journal of the Society for Industrial and Applied Mathematics 8.2 (1960): 300. Print.

[8] Gallager, Robert G. *Information Theory and Reliable Communication*. Wein: SpringerVerlag, 1972. Print.

[9] Odenwalder, Joseph P. *Error Control Coding Handbook Final Report*. San Diego, CA: Linkabit, 1976. Print.

[10] Wenyi Jin and Marc Fossorier, *Towards Maximum Lilelihood Soft Decision Decoding of the (255, 239) Reed Solomon Code*, IEEE, 423-428, 2008.

[11] Markus Mehnert, Dorothea Freiin von Droste, and Daniel Schiel, *VHDL Implementation of a (255, 191) Reed Solomon Coder for DVB-H*, IEEE, 2006.

[12] David Gozalvez, David Gomez-Barquero, and Narcis Gardona, *Performance Evaluation of the MPE-iFEC Sliding RS Encoding for DVB-H Streaming Services*, IEEE, 2008.

[13] David Taggart, Rajendra Kumar, and Nick Wagner, *PCWFM Performance Enhancement Using Reed Solomon Channel Coding*, IEEE, 1337-1346, 2004.

[14] Shirish S Karande and Hayder Radha, *Partial Reed Solomon Codes for Erasure Channels*, IEEE, 82-85, 2003.

[15] H.M. Shao and I.S. Reed, *A Systolic VLSI Design of a Pipeline Reed-Solomon Decoder*, TDA Progress Report, 42-76J, 99-113, 1983.

[16] S. Reed and M.T. Shih, *VLSI Design of Inverse-Free Berlekamp-Massey Algorithm*, Computers and Digital Techniques, IEE Proceedings E, Vol. 138, Issue 5, 295-298, 1991.

[17] R.T. Chien, *Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes*, IEEE Transactions on Information Theory, Vol. 10, No. 10, 357-363, 1964.

[18] D. Dabiri and I.F. Blake, *Fast Parallel Algorithms for Decoding Reed-Solomon Codes Based on Remainder Polynomials*, IEEE Transactions on Information Theory, Vol. 41, No. 4, 873-885, 1995.

[19] J.L. Massey, *Shift-Register Synthesis and BCH Decoding*, IEEE Transactions on Information Theory, No. 15, 122-127, 1968.

[20] K.Y. Liu, *Architecture for VLSI Design of Reed-Solomon Encoder*. IEEE Transactions on Information Theory, Vol. IT28, No. 6, 869-874, 1982

[21] C.E. Shannon, *A Mathematical Theory of Communication*, The Bell System Technical Journal, Vol. 27, 379-423, 623-656, 1948.

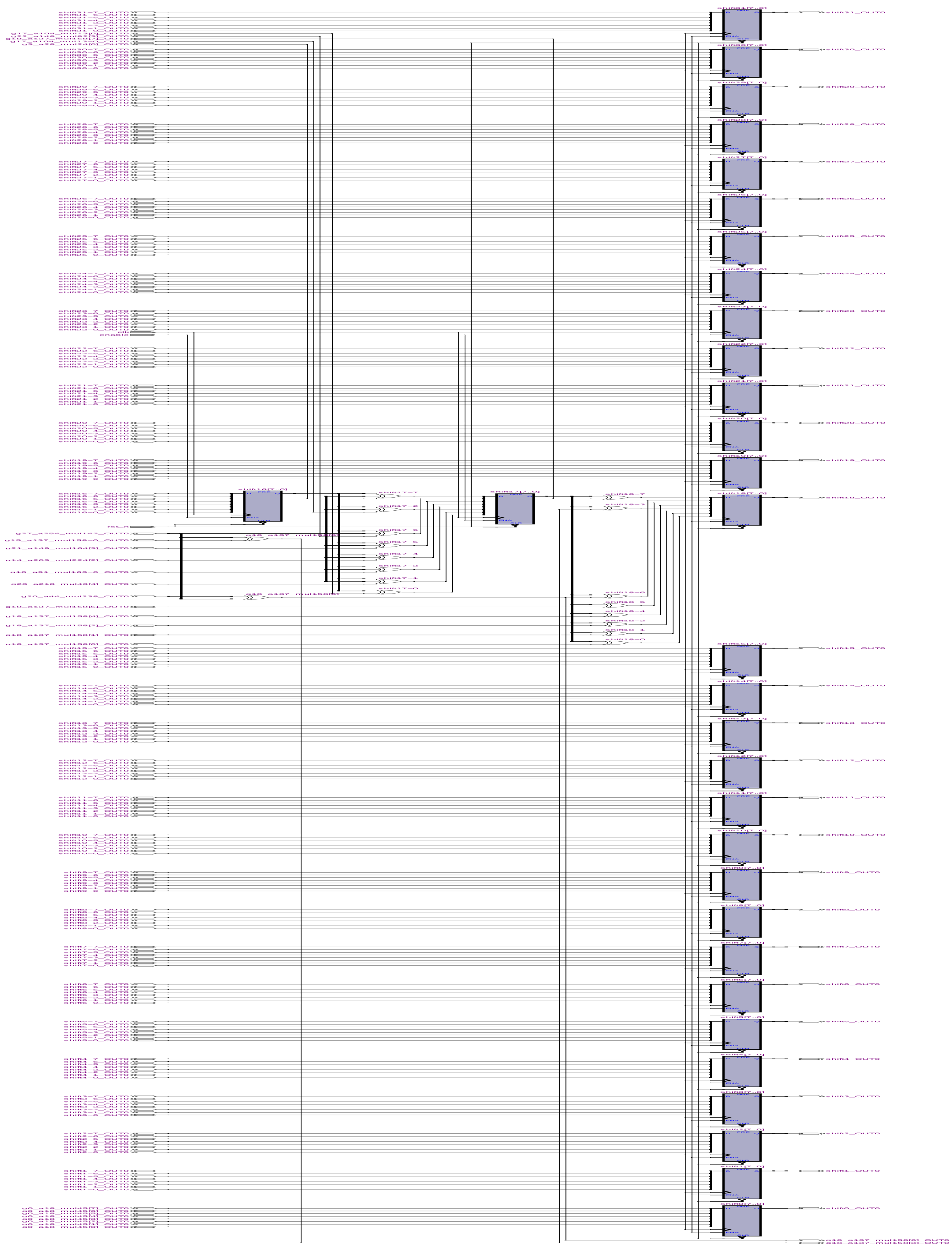
[22] O. Ben-Haim, A. Reichman, and D. Wulich, *Iterative Decoding of 8-DPSK with Reed Solomon Code*, IEEE, 106-109, 2004.

[23] G.D. Forney, *On Decoding BCH Codes*, IEEE Transactions on Information Theory, 549-557.

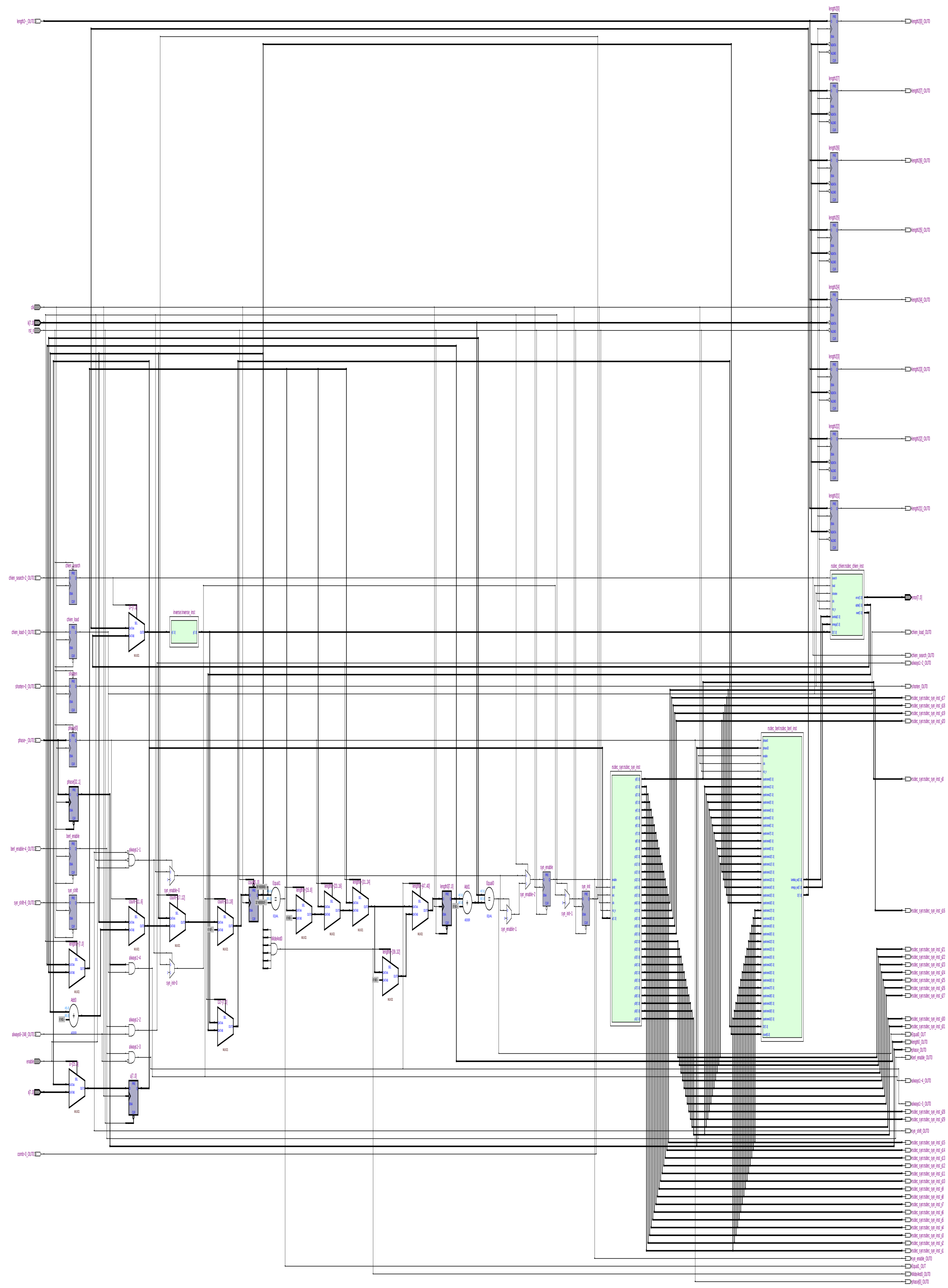
[24] G.D. Forney, *Generalized Minimum Distance Decoding*, IEEE Transactions on Information Theory, Vol. IT-12, No. 2, 125-131.

Appendix A The RTL Description for the (255,223) RS Encoder/Decoder

A-1 The RTL Description for (255,223) RS Encoder



A-2 The RTL Description for (255,223) RS Decoder



Appendix B The Verilog HDL files for the (255,223) RS Encoder

rs-encoder.v

```
module rs_encoder (clk, rst_n, din, din_en, din_syn, dout, dout_en, dout_syn);
  input   wire              clk;                      // clock input
  input   wire              rst_n;                    // active low asynchronous reset
  input   wire  [7:0]        din;                     // data input
  input   wire              din_en;                   // data input valid
  input   wire              din_syn;                  // synchronous for counter
  output  reg               dout_en;
  output  reg               dout_syn;
  output  reg   [7:0]        dout;                    // data output
//g(X) = 45+216X+239X2 +24X3 +253X4 +104X5 +27X6 +40X7 +107X8 +50X9 +163X10
+210X11
//     +227X12 +134X13 +224X14 +158X15 +119X16 +13X17 +158X18 +X19 +238X20
+164X21
//     +82X22 +43X23 +15X24 +232X25 +246X26 +142X27 +50X28 +189X29 +29X32
+232X31 +X32

  reg        [7:0]        cnt_255;

// declaration of shift registers
  reg        [7:0]        shift0;
  reg        [7:0]        shift1;
  reg        [7:0]        shift2;
  reg        [7:0]        shift3;
  reg        [7:0]        shift4;
  reg        [7:0]        shift5;
  reg        [7:0]        shift6;
  reg        [7:0]        shift7;
  reg        [7:0]        shift8;
  reg        [7:0]        shift9;
  reg        [7:0]        shift10;
  reg        [7:0]        shift11;
  reg        [7:0]        shift12;
  reg        [7:0]        shift13;
  reg        [7:0]        shift14;
  reg        [7:0]        shift15;
  reg        [7:0]        shift16;
  reg        [7:0]        shift17;
  reg        [7:0]        shift18;
  reg        [7:0]        shift19;
  reg        [7:0]        shift20;
  reg        [7:0]        shift21;
  reg        [7:0]        shift22;
```

```
reg     [7:0]    shift23;
reg     [7:0]    shift24;
reg     [7:0]    shift25;
reg     [7:0]    shift26;
reg     [7:0]    shift27;
reg     [7:0]    shift28;
reg     [7:0]    shift29;
reg     [7:0]    shift30;
reg     [7:0]    shift31;

reg     [7:0]    xor_feedback;          // xor feedback from shift registers


wire    [7:0]    g0_a18_mul45;          // coefficient g0  of the generator
polynomial multiply with feed back
wire    [7:0]    g1_a251_mul216;        // coefficient g1  of the generator
polynomial multiply with feed back
wire    [7:0]    g2_a215_mul239;        // coefficient g2  of the generator
polynomial multiply with feed back
wire    [7:0]    g3_a28_mul24;          // coefficient g3  of the generator
polynomial multiply with feed back
wire    [7:0]    g4_a80_mul253;         // coefficient g4  of the generator
polynomial multiply with feed back
wire    [7:0]    g5_a107_mul104;        // coefficient g5  of the generator
polynomial multiply with feed back
wire    [7:0]    g6_a248_mul27;         // coefficient g6  of the generator
polynomial multiply with feed back
wire    [7:0]    g7_a53_mul40;          // coefficient g7  of the generator
polynomial multiply with feed back
wire    [7:0]    g8_a84_mul107;         // coefficient g8  of the generator
polynomial multiply with feed back
wire    [7:0]    g9_a194_mul50;         // coefficient g9  of the generator
polynomial multiply with feed back
wire    [7:0]    g10_a91_mul163;        // coefficient g10 of the generator
polynomial multiply with feed back
wire    [7:0]    g11_a59_mul210;        // coefficient g11 of the generator
polynomial multiply with feed back
wire    [7:0]    g12_a176_mul227;       // coefficient g12 of the generator
polynomial multiply with feed back
wire    [7:0]    g13_a99_mul134;        // coefficient g13 of the generator
polynomial multiply with feed back
wire    [7:0]    g14_a203_mul224;       // coefficient g14 of the generator
polynomial multiply with feed back
wire    [7:0]    g15_a137_mul158;       // coefficient g15 of the generator
polynomial multiply with feed back
```

```
  wire          [7:0]              g16_a43_mul119;                              // coefficient g16 of the generator
polynomial multiply with feed back
  wire          [7:0]              g17_a104_mul13;                              // coefficient g17 of the generator
polynomial multiply with feed back
  wire          [7:0]              g18_a137_mul158;                              // coefficient g18 of the generator
polynomial multiply with feed back
  wire          [7:0]              g19_a0_mul1;                              // coefficient g19 of the generator
polynomial multiply with feed back
  wire          [7:0]              g20_a44_mul238;                              // coefficient g20 of the generator
polynomial multiply with feed back
  wire          [7:0]              g21_a149_mul164;                              // coefficient g21 of the generator
polynomial multiply with feed back
  wire          [7:0]              g22_a148_mul82;                              // coefficient g22 of the generator
polynomial multiply with feed back
  wire          [7:0]              g23_a218_mul43;                              // coefficient g23 of the generator
polynomial multiply with feed back
  wire          [7:0]              g24_a75_mul15;                              // coefficient g24 of the generator
polynomial multiply with feed back
  wire          [7:0]              g25_a11_mul232;                              // coefficient g25 of the generator
polynomial multiply with feed back
  wire          [7:0]              g26_a173_mul246;                              // coefficient g26 of the generator
polynomial multiply with feed back
  wire          [7:0]              g27_a254_mul142;                              // coefficient g27 of the generator
polynomial multiply with feed back
  wire          [7:0]              g28_a194_mul50;                              // coefficient g28 of the generator
polynomial multiply with feed back
  wire          [7:0]              g29_a109_mul189;                              // coefficient g29 of the generator
polynomial multiply with feed back
  wire          [7:0]              g30_a8_mul29;                              // coefficient g30 of the generator
polynomial multiply with feed back
  wire          [7:0]              g31_a11_mul232;                              // coefficient g31 of the generator
polynomial multiply with feed back


// multiple xor_feedback with 45: g0
  assign g0_a18_mul45[7]  = xor_feedback[2] ^ xor_feedback[4] ^ xor_feedback[5] ^
xor_feedback[6];
  assign g0_a18_mul45[6]  = xor_feedback[1] ^ xor_feedback[3] ^ xor_feedback[4] ^
xor_feedback[5];
  assign g0_a18_mul45[5]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[4];
  assign g0_a18_mul45[4]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3];
  assign g0_a18_mul45[3]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[4] ^
xor_feedback[5] ^ xor_feedback[6];
  assign g0_a18_mul45[2]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[6];
```

```
  assign g0_a18_mul45[1]  = xor_feedback[1] ^ xor_feedback[4] ^ xor_feedback[6] ^
xor_feedback[7];
  assign g0_a18_mul45[0]  = xor_feedback[0] ^ xor_feedback[3] ^ xor_feedback[5] ^
xor_feedback[6] ^ xor_feedback[7];

  // multiple xor_feedback with 216: g1
  assign g1_a251_mul216[7]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[4];
  assign g1_a251_mul216[6]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[7];
  assign g1_a251_mul216[5]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[6];
  assign g1_a251_mul216[4]  = xor_feedback[1] ^ xor_feedback[5];
  assign g1_a251_mul216[3]  = xor_feedback[1] ^ xor_feedback[2];
  assign g1_a251_mul216[2]  = xor_feedback[2] ^ xor_feedback[4] ^ xor_feedback[7];
  assign g1_a251_mul216[1]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[6];
  assign g1_a251_mul216[0]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[5];

  // multiple xor_feedback with 239: g2
  assign g2_a215_mul239[7]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[3] ^ xor_feedback[6];
  assign g2_a215_mul239[6]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[5] ^ xor_feedback[7];
  assign g2_a215_mul239[5]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[4] ^
xor_feedback[6];
  assign g2_a215_mul239[4]  = xor_feedback[0] ^ xor_feedback[3] ^ xor_feedback[5];
  assign g2_a215_mul239[3]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[6];
  assign g2_a215_mul239[2]  = xor_feedback[1] ^ xor_feedback[5] ^ xor_feedback[6] ^
xor_feedback[7];
  assign g2_a215_mul239[1]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[5];
  assign g2_a215_mul239[0]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[7];

  // multiple xor_feedback with 24: g3
  assign g3_a28_mul24[7]  = xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[7];
  assign g3_a28_mul24[6]  = xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[6];
  assign g3_a28_mul24[5]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[5];
  assign g3_a28_mul24[4]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[4] ^
xor_feedback[7];
  assign g3_a28_mul24[3]  = xor_feedback[0] ^ xor_feedback[4] ^ xor_feedback[6] ^
xor_feedback[7];
  assign g3_a28_mul24[2]  = xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[6] ^
xor_feedback[7];
```

```
  assign g3_a28_mul24[1]  = xor_feedback[5] ^ xor_feedback[6];
  assign g3_a28_mul24[0]  = xor_feedback[4] ^ xor_feedback[5];

  // multiple xor_feedback with 253: g4
  assign g4_a80_mul253[7]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[3] ^ xor_feedback[5] ^ xor_feedback[6];
  assign g4_a80_mul253[6]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[7];
  assign g4_a80_mul253[5]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[6] ^ xor_feedback[7];
  assign g4_a80_mul253[4]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[5] ^ xor_feedback[6] ^ xor_feedback[7];
  assign g4_a80_mul253[3]  = xor_feedback[0] ^ xor_feedback[3] ^ xor_feedback[4] ^
xor_feedback[7];
  assign g4_a80_mul253[2]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[5] ^
xor_feedback[7];
  assign g4_a80_mul253[1]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[7];
  assign g4_a80_mul253[0]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[6] ^ xor_feedback[7];

  // multiple xor_feedback with 104: g5
  assign g5_a107_mul104[7]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[5];
  assign g5_a107_mul104[6]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[4];
  assign g5_a107_mul104[5]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[7];
  assign g5_a107_mul104[4]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[6] ^
xor_feedback[7];
  assign g5_a107_mul104[3]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[6] ^ xor_feedback[7];
  assign g5_a107_mul104[2]  = xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[4] ^
xor_feedback[6] ^ xor_feedback[7];
  assign g5_a107_mul104[1]  = xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[6] ^
xor_feedback[7];
  assign g5_a107_mul104[0]  = xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[5] ^
xor_feedback[6];

  // multiple xor_feedback with 27: g6
  assign g6_a248_mul27[7]  = xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[6];
  assign g6_a248_mul27[6]  = xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[5];
  assign g6_a248_mul27[5]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[4];
  assign g6_a248_mul27[4]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3];
  assign g6_a248_mul27[3]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[6];
```

```
  assign g6_a248_mul27[2]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[5] ^ xor_feedback[6];
  assign g6_a248_mul27[1]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[5] ^
xor_feedback[6];
  assign g6_a248_mul27[0]  = xor_feedback[0] ^ xor_feedback[4] ^ xor_feedback[5] ^
xor_feedback[7];

  // multiple xor_feedback with 40: g7
  assign g7_a53_mul40[7]  = xor_feedback[2] ^ xor_feedback[4] ^ xor_feedback[6] ^
xor_feedback[7];
  assign g7_a53_mul40[6]  = xor_feedback[1] ^ xor_feedback[3] ^ xor_feedback[5] ^
xor_feedback[6];
  assign g7_a53_mul40[5]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[5] ^ xor_feedback[7];
  assign g7_a53_mul40[4]  = xor_feedback[1] ^ xor_feedback[3] ^ xor_feedback[4] ^
xor_feedback[6] ^ xor_feedback[7];
  assign g7_a53_mul40[3]  = xor_feedback[0] ^ xor_feedback[3] ^ xor_feedback[4] ^
xor_feedback[5] ^ xor_feedback[7];
  assign g7_a53_mul40[2]  = xor_feedback[3];
  assign g7_a53_mul40[1]  = xor_feedback[4] ^ xor_feedback[6];
  assign g7_a53_mul40[0]  = xor_feedback[3] ^ xor_feedback[5] ^ xor_feedback[7];

  // multiple xor_feedback with 107: g8
  assign g8_a84_mul107[7]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[5] ^ xor_feedback[6] ^ xor_feedback[7];
  assign g8_a84_mul107[6]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[6];
  assign g8_a84_mul107[5]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[7];
  assign g8_a84_mul107[4]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[6];
  assign g8_a84_mul107[3]  = xor_feedback[0] ^ xor_feedback[3] ^ xor_feedback[4] ^
xor_feedback[6];
  assign g8_a84_mul107[2]  = xor_feedback[1] ^ xor_feedback[3] ^ xor_feedback[4] ^
xor_feedback[6];
  assign g8_a84_mul107[1]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[6] ^ xor_feedback[7];
  assign g8_a84_mul107[0]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[5] ^ xor_feedback[6] ^ xor_feedback[7];

  // multiple xor_feedback with 50: g9
  assign g9_a194_mul50[7]  = xor_feedback[2] ^ xor_feedback[3];
  assign g9_a194_mul50[6]  = xor_feedback[1] ^ xor_feedback[2];
  assign g9_a194_mul50[5]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[7];
  assign g9_a194_mul50[4]  = xor_feedback[0] ^ xor_feedback[6] ^ xor_feedback[7];
```

```
  assign g9_a194_mul50[3]  = xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[5] ^
xor_feedback[6] ^ xor_feedback[7];
  assign g9_a194_mul50[2]  = xor_feedback[1] ^ xor_feedback[3] ^ xor_feedback[4] ^
xor_feedback[5] ^ xor_feedback[6];
  assign g9_a194_mul50[1]  = xor_feedback[0] ^ xor_feedback[4] ^ xor_feedback[5];
  assign g9_a194_mul50[0]  = xor_feedback[3] ^ xor_feedback[4];

  // multiple xor_feedback with 163: g10
  assign g10_a91_mul163[7]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[5] ^ xor_feedback[6];
  assign g10_a91_mul163[6]  = xor_feedback[1] ^ xor_feedback[3] ^ xor_feedback[4] ^
xor_feedback[5] ^ xor_feedback[7];
  assign g10_a91_mul163[5]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[6];
  assign g10_a91_mul163[4]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[5];
  assign g10_a91_mul163[3]  = xor_feedback[1] ^ xor_feedback[5] ^ xor_feedback[6];
  assign g10_a91_mul163[2]  = xor_feedback[2] ^ xor_feedback[6];
  assign g10_a91_mul163[1]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[4] ^ xor_feedback[6] ^ xor_feedback[7];
  assign g10_a91_mul163[0]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[5] ^ xor_feedback[6] ^ xor_feedback[7];

  // multiple xor_feedback with 210: g11
  assign g11_a59_mul210[7]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[6];
  assign g11_a59_mul210[6]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[5] ^ xor_feedback[7];
  assign g11_a59_mul210[5]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[6] ^ xor_feedback[7];
  assign g11_a59_mul210[4]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[5] ^ xor_feedback[6];
  assign g11_a59_mul210[3]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[5] ^ xor_feedback[6];
  assign g11_a59_mul210[2]  = xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[5] ^
xor_feedback[6];
  assign g11_a59_mul210[1]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[5] ^ xor_feedback[6];
  assign g11_a59_mul210[0]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[5] ^ xor_feedback[7];

  // multiple xor_feedback with 227: g12
  assign g12_a176_mul227[7]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[4] ^ xor_feedback[7];
  assign g12_a176_mul227[6]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[6] ^ xor_feedback[7];
```

```
  assign g12_a176_mul227[5]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[5] ^
xor_feedback[6];
  assign g12_a176_mul227[4]  = xor_feedback[1] ^ xor_feedback[4] ^ xor_feedback[5];
  assign g12_a176_mul227[3]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3];
  assign g12_a176_mul227[2]  = xor_feedback[4] ^ xor_feedback[7];
  assign g12_a176_mul227[1]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[6];
  assign g12_a176_mul227[0]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[3] ^ xor_feedback[5];

  // multiple xor_feedback with 134: g13
  assign g13_a99_mul134[7]  = xor_feedback[0] ^ xor_feedback[4];
  assign g13_a99_mul134[6]  = xor_feedback[3];
  assign g13_a99_mul134[5]  = xor_feedback[2] ^ xor_feedback[7];
  assign g13_a99_mul134[4]  = xor_feedback[1] ^ xor_feedback[6] ^ xor_feedback[7];
  assign g13_a99_mul134[3]  = xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[6];
  assign g13_a99_mul134[2]  = xor_feedback[0] ^ xor_feedback[3] ^ xor_feedback[5] ^
xor_feedback[7];
  assign g13_a99_mul134[1]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[6];
  assign g13_a99_mul134[0]  = xor_feedback[1] ^ xor_feedback[5];

  // multiple xor_feedback with 224: g14
  assign g14_a203_mul224[7]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[4] ^ xor_feedback[6];
  assign g14_a203_mul224[6]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[5] ^ xor_feedback[7];
  assign g14_a203_mul224[5]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[6];
  assign g14_a203_mul224[4]  = xor_feedback[1] ^ xor_feedback[3] ^ xor_feedback[5] ^
xor_feedback[7];
  assign g14_a203_mul224[3]  = xor_feedback[1] ^ xor_feedback[7];
  assign g14_a203_mul224[2]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[4];
  assign g14_a203_mul224[1]  = xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[4] ^
xor_feedback[6];
  assign g14_a203_mul224[0]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[5] ^ xor_feedback[7];

  // multiple xor_feedback with 158: g15
  assign g15_a137_mul158[7]  = xor_feedback[0] ^ xor_feedback[3] ^ xor_feedback[7];
  assign g15_a137_mul158[6]  = xor_feedback[2] ^ xor_feedback[6];
  assign g15_a137_mul158[5]  = xor_feedback[1] ^ xor_feedback[5] ^ xor_feedback[7];
  assign g15_a137_mul158[4]  = xor_feedback[0] ^ xor_feedback[4] ^ xor_feedback[6];
  assign g15_a137_mul158[3]  = xor_feedback[0] ^ xor_feedback[5] ^ xor_feedback[7];
  assign g15_a137_mul158[2]  = xor_feedback[0] ^ xor_feedback[3] ^ xor_feedback[4] ^
xor_feedback[6];
  assign g15_a137_mul158[1]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[5];
```

```
  assign g15_a137_mul158[0]  = xor_feedback[1] ^ xor_feedback[4];

  // multiple xor_feedback with 119: g16
  assign g16_a43_mul119[7]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[6];
  assign g16_a43_mul119[6]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[5];
  assign g16_a43_mul119[5]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[4];
  assign g16_a43_mul119[4]  = xor_feedback[0] ^ xor_feedback[3];
  assign g16_a43_mul119[3]  = xor_feedback[1] ^ xor_feedback[3] ^ xor_feedback[6];
  assign g16_a43_mul119[2]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[5] ^ xor_feedback[6] ^ xor_feedback[7];
  assign g16_a43_mul119[1]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[5];
  assign g16_a43_mul119[0]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[7];

  // multiple xor_feedback with 13: g17
  assign g17_a104_mul13[7]  = xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[7];
  assign g17_a104_mul13[6]  = xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[6] ^
xor_feedback[7];
  assign g17_a104_mul13[5]  = xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[5] ^
xor_feedback[6];
  assign g17_a104_mul13[4]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[5];
  assign g17_a104_mul13[3]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[5] ^ xor_feedback[7];
  assign g17_a104_mul13[2]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[5] ^
xor_feedback[6] ^ xor_feedback[7];
  assign g17_a104_mul13[1]  = xor_feedback[1] ^ xor_feedback[6] ^ xor_feedback[7];
  assign g17_a104_mul13[0]  = xor_feedback[0] ^ xor_feedback[5] ^ xor_feedback[6];

  // multiple xor_feedback with 158: g18
  assign g18_a137_mul158  = g15_a137_mul158;

  // multiple xor_feedback with 1: g19
  assign g19_a0_mul1  = xor_feedback;

  // multiple xor_feedback with 238: g20
  assign g20_a44_mul238[7]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[5];
  assign g20_a44_mul238[6]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[4];
  assign g20_a44_mul238[5]  = xor_feedback[0] ^ xor_feedback[3];
  assign g20_a44_mul238[4]  = xor_feedback[2];
  assign g20_a44_mul238[3]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[5] ^
xor_feedback[7];
```

```
  assign g20_a44_mul238[2]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[5] ^ xor_feedback[6];
  assign g20_a44_mul238[1]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[7];
  assign g20_a44_mul238[0]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[6];

  // multiple xor_feedback with 164: g21
  assign g21_a149_mul164[7]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[7];
  assign g21_a149_mul164[6]  = xor_feedback[1] ^ xor_feedback[3] ^ xor_feedback[6] ^
xor_feedback[7];
  assign g21_a149_mul164[5]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[5] ^
xor_feedback[6] ^ xor_feedback[7];
  assign g21_a149_mul164[4]  = xor_feedback[1] ^ xor_feedback[4] ^ xor_feedback[5] ^
xor_feedback[6];
  assign g21_a149_mul164[3]  = xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[5];
  assign g21_a149_mul164[2]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[7];
  assign g21_a149_mul164[1]  = xor_feedback[2] ^ xor_feedback[4] ^ xor_feedback[6];
  assign g21_a149_mul164[0]  = xor_feedback[1] ^ xor_feedback[3] ^ xor_feedback[5];

  // multiple xor_feedback with 82: g22
  assign g22_a148_mul82[7]  = xor_feedback[1] ^ xor_feedback[3] ^ xor_feedback[5];
  assign g22_a148_mul82[6]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[7];
  assign g22_a148_mul82[5]  = xor_feedback[1] ^ xor_feedback[3] ^ xor_feedback[6] ^
xor_feedback[7];
  assign g22_a148_mul82[4]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[5] ^
xor_feedback[6] ^ xor_feedback[7];
  assign g22_a148_mul82[3]  = xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[6];
  assign g22_a148_mul82[2]  = xor_feedback[1] ^ xor_feedback[2];
  assign g22_a148_mul82[1]  = xor_feedback[0] ^ xor_feedback[3] ^ xor_feedback[5] ^
xor_feedback[7];
  assign g22_a148_mul82[0]  = xor_feedback[2] ^ xor_feedback[4] ^ xor_feedback[6];

  // multiple xor_feedback with 43: g23
  assign g23_a218_mul43[7]  = xor_feedback[2] ^ xor_feedback[4];
  assign g23_a218_mul43[6]  = xor_feedback[1] ^ xor_feedback[3];
  assign g23_a218_mul43[5]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[7];
  assign g23_a218_mul43[4]  = xor_feedback[1] ^ xor_feedback[6];
  assign g23_a218_mul43[3]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[5];
  assign g23_a218_mul43[2]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[7];
  assign g23_a218_mul43[1]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[4] ^
xor_feedback[6];
```

assign g23_a218_mul43[0] = xor_feedback[0] ^ xor_feedback[3] ^ xor_feedback[5];

// multiple xor_feedback with 15: g24
assign g24_a75_mul15[7] = xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[6] ^ xor_feedback[7];
assign g24_a75_mul15[6] = xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[6] ^ xor_feedback[7];
assign g24_a75_mul15[5] = xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[6];
assign g24_a75_mul15[4] = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[7];
assign g24_a75_mul15[3] = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[5];
assign g24_a75_mul15[2] = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[5] ^ xor_feedback[6];
assign g24_a75_mul15[1] = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[6] ^ xor_feedback[7];
assign g24_a75_mul15[0] = xor_feedback[0] ^ xor_feedback[5] ^ xor_feedback[6] ^ xor_feedback[7];

// multiple xor_feedback with 232: g25
assign g25_a11_mul232[7] = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[6];
assign g25_a11_mul232[6] = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[5];
assign g25_a11_mul232[5] = xor_feedback[0] ^ xor_feedback[4] ^ xor_feedback[7];
assign g25_a11_mul232[4] = xor_feedback[3] ^ xor_feedback[6];
assign g25_a11_mul232[3] = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[5] ^ xor_feedback[6] ^ xor_feedback[7];
assign g25_a11_mul232[2] = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[7];
assign g25_a11_mul232[1] = xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[4];
assign g25_a11_mul232[0] = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[7];

// multiple xor_feedback with 246: g26
assign g26_a173_mul246[7] = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[7];
assign g26_a173_mul246[6] = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[6];
assign g26_a173_mul246[5] = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[5];
assign g26_a173_mul246[4] = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[4] ^ xor_feedback[7];
assign g26_a173_mul246[3] = xor_feedback[2] ^ xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[6];
assign g26_a173_mul246[2] = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[7];

```
  assign g26_a173_mul246[1]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[6] ^ xor_feedback[7];
  assign g26_a173_mul246[0]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[6];

  // multiple xor_feedback with 142: g27
  assign g27_a254_mul142[7]  = xor_feedback[0];
  assign g27_a254_mul142[6]  = xor_feedback[7];
  assign g27_a254_mul142[5]  = xor_feedback[6];
  assign g27_a254_mul142[4]  = xor_feedback[5];
  assign g27_a254_mul142[3]  = xor_feedback[0] ^ xor_feedback[4];
  assign g27_a254_mul142[2]  = xor_feedback[0] ^ xor_feedback[3];
  assign g27_a254_mul142[1]  = xor_feedback[0] ^ xor_feedback[2];
  assign g27_a254_mul142[0]  = xor_feedback[1];

  // multiple xor_feedback with 50: g28
  assign g28_a194_mul50 = g9_a194_mul50;

  // multiple xor_feedback with 189: g29
  assign g29_a109_mul189[7]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[7];
  assign g29_a109_mul189[6]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[6] ^
xor_feedback[7];
  assign g29_a109_mul189[5]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[5] ^
xor_feedback[6] ^ xor_feedback[7];
  assign g29_a109_mul189[4]  = xor_feedback[0] ^ xor_feedback[4] ^ xor_feedback[5] ^
xor_feedback[6] ^ xor_feedback[7];
  assign g29_a109_mul189[3]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[5] ^ xor_feedback[6] ^ xor_feedback[7];
  assign g29_a109_mul189[2]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[4] ^ xor_feedback[5] ^ xor_feedback[6];
  assign g29_a109_mul189[1]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[5];
  assign g29_a109_mul189[0]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[4];

  // multiple xor_feedback with 29: g30
  assign g30_a8_mul29[7]  = xor_feedback[3] ^ xor_feedback[4] ^ xor_feedback[5];
  assign g30_a8_mul29[6]  = xor_feedback[2] ^ xor_feedback[3] ^ xor_feedback[4];
  assign g30_a8_mul29[5]  = xor_feedback[1] ^ xor_feedback[2] ^ xor_feedback[3] ^
xor_feedback[7];
  assign g30_a8_mul29[4]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[2] ^
xor_feedback[6];
  assign g30_a8_mul29[3]  = xor_feedback[0] ^ xor_feedback[1] ^ xor_feedback[3] ^
xor_feedback[4];
```

```
  assign g30_a8_mul29[2]  = xor_feedback[0] ^ xor_feedback[2] ^ xor_feedback[4] ^
xor_feedback[5] ^ xor_feedback[7];
  assign g30_a8_mul29[1]  = xor_feedback[1] ^ xor_feedback[5] ^ xor_feedback[6] ^
xor_feedback[7];
  assign g30_a8_mul29[0]  = xor_feedback[0] ^ xor_feedback[4] ^ xor_feedback[5] ^
xor_feedback[6];

  // multiple xor_feedback with 29: g31
  assign g31_a11_mul232 = g25_a11_mul232;

//---------------------------------------------------------------------------
// process for counter
//---------------------------------------------------------------------------
  always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
      cnt_255 <= 8'd0;
    end
    else if (din_en) begin
      if(din_syn)
        cnt_255 <= 8'd0;
      else begin
        cnt_255 <= (cnt_255 < 8'd255)? (cnt_255 + 1'b1): 8'd0;
      end
    end
    else begin
      cnt_255 <= cnt_255;
    end
  end

//---------------------------------------------------------------------------
// process for shift register
//---------------------------------------------------------------------------
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      shift0  <= 8'd0;
      shift1  <= 8'd0;
      shift2  <= 8'd0;
      shift3  <= 8'd0;
      shift4  <= 8'd0;
      shift5  <= 8'd0;
      shift6  <= 8'd0;
      shift7  <= 8'd0;
      shift8  <= 8'd0;
      shift9  <= 8'd0;
      shift10 <= 8'd0;
      shift11 <= 8'd0;
```

```
  shift12 <= 8'd0;
  shift13 <= 8'd0;
  shift14 <= 8'd0;
  shift15 <= 8'd0;
  shift16 <= 8'd0;
  shift17 <= 8'd0;
  shift18 <= 8'd0;
  shift19 <= 8'd0;
  shift20 <= 8'd0;
  shift21 <= 8'd0;
  shift22 <= 8'd0;
  shift23 <= 8'd0;
  shift24 <= 8'd0;
  shift25 <= 8'd0;
  shift26 <= 8'd0;
  shift27 <= 8'd0;
  shift28 <= 8'd0;
  shift29 <= 8'd0;
  shift30 <= 8'd0;
  shift31 <= 8'd0;
end
else if (din_en) begin
  shift0  <= g0_a18_mul45;
  shift1  <= shift0  ^ g1_a251_mul216;
  shift2  <= shift1  ^ g2_a215_mul239;
  shift3  <= shift2  ^ g3_a28_mul24;
  shift4  <= shift3  ^ g4_a80_mul253;
  shift5  <= shift4  ^ g5_a107_mul104;
  shift6  <= shift5  ^ g6_a248_mul27;
  shift7  <= shift6  ^ g7_a53_mul40;
  shift8  <= shift7  ^ g8_a84_mul107;
  shift9  <= shift8  ^ g9_a194_mul50;
  shift10 <= shift9  ^ g10_a91_mul163;
  shift11 <= shift10 ^ g11_a59_mul210;
  shift12 <= shift11 ^ g12_a176_mul227;
  shift13 <= shift12 ^ g13_a99_mul134;
  shift14 <= shift13 ^ g14_a203_mul224;
  shift15 <= shift14 ^ g15_a137_mul158;
  shift16 <= shift15 ^ g16_a43_mul119;
  shift17 <= shift16 ^ g17_a104_mul13;
  shift18 <= shift17 ^ g18_a137_mul158;
  shift19 <= shift18 ^ g19_a0_mul1;
  shift20 <= shift19 ^ g20_a44_mul238;
  shift21 <= shift20 ^ g21_a149_mul164;
  shift22 <= shift21 ^ g22_a148_mul82;
  shift23 <= shift22 ^ g23_a218_mul43;
```

```
      shift24 <= shift23 ^ g24_a75_mul15;
      shift25 <= shift24 ^ g25_a11_mul232;
      shift26 <= shift25 ^ g26_a173_mul246;
      shift27 <= shift26 ^ g27_a254_mul142;
      shift28 <= shift27 ^ g28_a194_mul50;
      shift29 <= shift28 ^ g29_a109_mul189;
      shift30 <= shift29 ^ g30_a8_mul29;
      shift31 <= shift30 ^ g31_a11_mul232;
    end
  end
  assign dout = (cnt_255 < 8'd224)? din :  shift31;
  assign dout_en = din_en;
  assign dout_syn = din_syn;

//-------------------------------------------------------------------------
// process for shift register
//-------------------------------------------------------------------------
  always @ (din or shift15) begin
    xor_feedback = (cnt_255 > 8'd223) ? 8'd0 : din^shift31;
  end

endmodule
```

sigin.v

```
module sigin (data, clk, en, rst_n);
  input   wire                clk;                          // clock input
  input   wire                rst_n;                         // active low asynchronous reset
  input   wire                en;                           // enable signal
  output  reg      [7:0]        data;                          // data output

  always @(posedge clk or negedge rst_n) begin
    if(~rst_n) begin
      data <= 8'd0;
    end
    else if(en) begin
      data <= data + 8'd1;
    end
  end
endmodule
```

tb_encode.v

```
timescale 1ns/1ps
module tb_encode;
  reg                    clk;                    // clock input
  reg                    rst_n;                  // active low asynchronous reset
  wire      [7:0]          din;
  reg                    din_en,din_syn;
  wire      [7:0]          dout;
  reg       [7:0]          address;
  reg                    enc_trigger;
  reg       [7:0]          k;

  always #50 clk = ~clk;

  //instantiate sigin
  sigin
    sigin_inst (
      .clk  (clk  ),
      .rst_n(rst_n),
      .en   (din_en),
      .data (din  )
      );
  //instantiate encoder
  rs_encoder
    rs_encoder_inst (
      .clk     (clk     ),
      .rst_n   (rst_n   ),
      .din     (din     ),
      .din_en  (din_en  ),
      .din_syn (din_syn ),
      .dout_en (dout_en ),
      .dout_syn(dout_syn),
      .dout    (dout    )
      );

  always @(posedge clk or negedge rst_n) begin
    if(~rst_n) begin
      din_en  <= 0;
      din_syn <= 0;
      address <= 0;
    end
    else begin
      if(enc_trigger && address == 0) begin
        din_en <= 1;
      end
```

```
      else if (address == k-1) begin
        din_en <= 0;
      end
      else
        din_en <= din_en;

      if(address == k-1) begin
        din_syn <= 1;
      end
      else begin
        din_syn <= 0;
      end
      if (din_en)
        address <= address + 1;
      else
        address <= 0;
    end
  end

  initial begin
    clk = 0;
    rst_n = 1;
    enc_trigger = 0;
    k = 255;
    #1 rst_n = 0;
    #20 rst_n = 1;
    #200;
    #100 enc_trigger = 1;
    #100 enc_trigger = 0;
    #30400 $finish;
  end
endmodule
```

Appendix C The Verilog HDL files for the (255,223) RS Decoder

rsdec.v

```
module rsdec(x, error, with_error, enable, valid, k, clk, rst_n);
	input enable, clk, rst_n;
	input [7:0] k, x;
	output [7:0] error;
	wire [7:0] error;
	output with_error, valid;
	reg with_error, valid;

	wire [7:0] s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, s14, s15, s16, s17, s18,
s19, s20, s21, s22, s23, s24, s25, s26, s27, s28, s29, s30, s31;
	wire [7:0] lambda, omega, alpha;
	reg [5:0] count;
	reg [32:0] phase;
	wire [7:0] D0, D1, DI;
	reg [7:0] D, D2;
	reg [7:0] u, length0, length1, length2, length3;
	reg syn_enable, syn_init, syn_shift, berl_enable;
	reg chien_search, chien_load, shorten;

	always @ (chien_search or shorten)
		valid = chien_search & ~shorten;

	rsdec_syn rsdec_syn_inst (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, s14, s15,
s16, s17, s18, s19, s20, s21, s22, s23, s24, s25, s26, s27, s28, s29, s30, s31,
		u, syn_enable, syn_shift&phase[0], syn_init, clk, rst_n);
	rsdec_berl rsdec_berl_inst (lambda, omega,
		s0, s31, s30, s29, s28, s27, s26, s25, s24, s23, s22, s21, s20, s19, s18, s17, s16,
s15, s14, s13, s12, s11, s10, s9, s8, s7, s6, s5, s4, s3, s2, s1,
		D0, D2, count, phase[0], phase[32], berl_enable, clk, rst_n);
	rsdec_chien rsdec_chien_inst (error, alpha, lambda, omega,
		D1, DI, chien_search, chien_load, shorten, clk, rst_n);
	inverse inverse_inst (DI, D);

	always @ (posedge clk or negedge rst_n)
	begin
		if (~rst_n)
		begin
			syn_enable <= 0;
			syn_shift <= 0;
			berl_enable <= 0;
			chien_search <= 1;
			chien_load <= 0;
```

```
            length0 <= 0;
            length2 <= 255 - k;
            count <= -1;
            phase <= 1;
            u <= 0;
            shorten <= 1;
            syn_init <= 0;
        end
        else
        begin
            if (enable & ~syn_enable & ~syn_shift)
            begin
                syn_enable <= 1;
                syn_init <= 1;
            end
            if (syn_enable)
            begin
                length0 <= length1;
                syn_init <= 0;
                if (length1 == k)
                begin
                    syn_enable <= 0;
                    syn_shift <= 1;
                    berl_enable <= 1;
                end
            end
            if (berl_enable & with_error)
            begin
                if (phase[0])
                begin
                    count <= count + 1;
                    if (count == 31)
                    begin
                        syn_shift <= 0;
                        length0 <= 0;
                        chien_load <= 1;
                        length2 <= length0;
                    end
                end
                phase <= {phase[31:0], phase[32]};
            end
            if (berl_enable & ~with_error)
                if (&count)
                begin
                    syn_shift <= 0;
                    length0 <= 0;
```

```
                        berl_enable <= 0;
                    end
                    else
                        phase <= {phase[31:0], phase[32]};
                if (chien_load & phase[32])
                begin
                    berl_enable <= 0;
                    chien_load <= 0;
                    chien_search <= 1;
                    count <= -1;
                    phase <= 1;
                end
                if (chien_search)
                begin
                    length2 <= length3;
                    if (length3 == 0)
                        chien_search <= 0;
                end
            if (enable) u <= x;
            if (shorten == 1 && length2 == 0)
                shorten <= 0;
            end

    end

    always @ (chien_search or D0 or D1)
        if (chien_search) D = D1;
        else D = D0;

    always @ (DI or alpha or chien_load)
        if (chien_load) D2 = alpha;
        else D2 = DI;

    always @ (length0) length1 = length0 + 1;
    always @ (length2) length3 = length2 - 1;
    always @ (syn_shift or s0 or s1 or s2 or s3 or s4 or s5 or s6 or s7 or s8 or s9 or s10 or s11
or s12 or s13 or s14 or s15 or s16 or s17 or s18 or s19 or s20 or s21 or s22 or s23 or s24 or s25 or
s26 or s27 or s28 or s29 or s30 or s31)
            if (syn_shift && (s0 | s1 | s2 | s3 | s4 | s5 | s6 | s7 | s8 | s9 | s10 | s11 | s12 | s13 | s14
| s15 | s16 | s17 | s18 | s19 | s20 | s21 | s22 | s23 | s24 | s25 | s26 | s27 | s28 | s29 | s30 | s31)!= 0)
                with_error = 1;
            else with_error = 0;

endmodule
```

berlekamp.v

```
module rsdec_berl (lambda_out, omega_out, syndrome0, syndrome1, syndrome2, syndrome3,
syndrome4, syndrome5, syndrome6, syndrome7, syndrome8, syndrome9, syndrome10,
syndrome11, syndrome12, syndrome13, syndrome14, syndrome15, syndrome16, syndrome17,
syndrome18, syndrome19, syndrome20, syndrome21, syndrome22, syndrome23, syndrome24,
syndrome25, syndrome26, syndrome27, syndrome28, syndrome29, syndrome30, syndrome31,
D, DI, count, phase0, phase32, enable, clk, rst_n);
	input clk, rst_n, enable, phase0, phase32;
	input [7:0] syndrome0;
	input [7:0] syndrome1;
	input [7:0] syndrome2;
	input [7:0] syndrome3;
	input [7:0] syndrome4;
	input [7:0] syndrome5;
	input [7:0] syndrome6;
	input [7:0] syndrome7;
	input [7:0] syndrome8;
	input [7:0] syndrome9;
	input [7:0] syndrome10;
	input [7:0] syndrome11;
	input [7:0] syndrome12;
	input [7:0] syndrome13;
	input [7:0] syndrome14;
	input [7:0] syndrome15;
	input [7:0] syndrome16;
	input [7:0] syndrome17;
	input [7:0] syndrome18;
	input [7:0] syndrome19;
	input [7:0] syndrome20;
	input [7:0] syndrome21;
	input [7:0] syndrome22;
	input [7:0] syndrome23;
	input [7:0] syndrome24;
	input [7:0] syndrome25;
	input [7:0] syndrome26;
	input [7:0] syndrome27;
	input [7:0] syndrome28;
	input [7:0] syndrome29;
	input [7:0] syndrome30;
	input [7:0] syndrome31;
	input [7:0] DI;
	input [5:0] count;
	output [7:0] lambda_out;
	output [7:0] omega_out;
	reg [7:0] lambda_out;
```

```
reg [7:0] omega_out;
output [7:0] D;
reg [7:0] D;

integer j;
reg init, delta;
reg [4:0] L;
reg [7:0] lambda[31:0];
reg [7:0] omega[31:0];
reg [7:0] A[30:0];
reg [7:0] B[30:0];
wire [7:0] tmp0;
wire [7:0] tmp1;
wire [7:0] tmp2;
wire [7:0] tmp3;
wire [7:0] tmp4;
wire [7:0] tmp5;
wire [7:0] tmp6;
wire [7:0] tmp7;
wire [7:0] tmp8;
wire [7:0] tmp9;
wire [7:0] tmp10;
wire [7:0] tmp11;
wire [7:0] tmp12;
wire [7:0] tmp13;
wire [7:0] tmp14;
wire [7:0] tmp15;
wire [7:0] tmp16;
wire [7:0] tmp17;
wire [7:0] tmp18;
wire [7:0] tmp19;
wire [7:0] tmp20;
wire [7:0] tmp21;
wire [7:0] tmp22;
wire [7:0] tmp23;
wire [7:0] tmp24;
wire [7:0] tmp25;
wire [7:0] tmp26;
wire [7:0] tmp27;
wire [7:0] tmp28;
wire [7:0] tmp29;
wire [7:0] tmp30;
wire [7:0] tmp31;

always @ (tmp1) lambda_out = tmp1;
always @ (tmp3) omega_out = tmp3;
```

```
	always @ (L or D or count)
		// delta = (D != 0 && 2*L <= i);
		if (D != 0 && count >= {L, 1'b0}) delta = 1;
		else delta = 0;

	rsdec_berl_multiply rsdec_berl_multiply_x0 (tmp0, B[30], D, lambda[0], syndrome0,
phase0);
	rsdec_berl_multiply rsdec_berl_multiply_x1 (tmp1, lambda[31], DI, lambda[1],
syndrome1, phase0);
	rsdec_berl_multiply rsdec_berl_multiply_x2 (tmp2, A[30], D, lambda[2], syndrome2,
phase0);
	rsdec_berl_multiply rsdec_berl_multiply_x3 (tmp3, omega[31], DI, lambda[3],
syndrome3, phase0);
	multiply multiply_x4 (tmp4, lambda[4], syndrome4);
	multiply multiply_x5 (tmp5, lambda[5], syndrome5);
	multiply multiply_x6 (tmp6, lambda[6], syndrome6);
	multiply multiply_x7 (tmp7, lambda[7], syndrome7);
	multiply multiply_x8 (tmp8, lambda[8], syndrome8);
	multiply multiply_x9 (tmp9, lambda[9], syndrome9);
	multiply multiply_x10 (tmp10, lambda[10], syndrome10);
	multiply multiply_x11 (tmp11, lambda[11], syndrome11);
	multiply multiply_x12 (tmp12, lambda[12], syndrome12);
	multiply multiply_x13 (tmp13, lambda[13], syndrome13);
	multiply multiply_x14 (tmp14, lambda[14], syndrome14);
	multiply multiply_x15 (tmp15, lambda[15], syndrome15);
	multiply multiply_x16 (tmp16, lambda[16], syndrome16);
	multiply multiply_x17 (tmp17, lambda[17], syndrome17);
	multiply multiply_x18 (tmp18, lambda[18], syndrome18);
	multiply multiply_x19 (tmp19, lambda[19], syndrome19);
	multiply multiply_x20 (tmp20, lambda[20], syndrome20);
	multiply multiply_x21 (tmp21, lambda[21], syndrome21);
	multiply multiply_x22 (tmp22, lambda[22], syndrome22);
	multiply multiply_x23 (tmp23, lambda[23], syndrome23);
	multiply multiply_x24 (tmp24, lambda[24], syndrome24);
	multiply multiply_x25 (tmp25, lambda[25], syndrome25);
	multiply multiply_x26 (tmp26, lambda[26], syndrome26);
	multiply multiply_x27 (tmp27, lambda[27], syndrome27);
	multiply multiply_x28 (tmp28, lambda[28], syndrome28);
	multiply multiply_x29 (tmp29, lambda[29], syndrome29);
	multiply multiply_x30 (tmp30, lambda[30], syndrome30);
	multiply multiply_x31 (tmp31, lambda[31], syndrome31);

	always @ (posedge clk or negedge rst_n)
	begin
		// for (j = t-1; j >=0; j--)
```

```
//	if (j != 0) lambda[j] += D * B[j-1];
if (~rst_n)
begin
	for (j = 0; j < 32; j = j + 1) lambda[j] <= 0;
	for (j = 0; j < 31; j = j + 1) B[j] <= 0;
	for (j = 0; j < 32; j = j + 1) omega[j] <= 0;
	for (j = 0; j < 31; j = j + 1) A[j] <= 0;
	L = 0;
	D = 0;
end
else if (~enable)
begin
	lambda[0] <= 1;
	for (j = 1; j < 32; j = j +1) lambda[j] <= 0;
	B[0] <= 1;
	for (j = 1; j < 31; j = j +1) B[j] <= 0;
	omega[0] <= 1;
	for (j = 1; j < 32; j = j +1) omega[j] <= 0;
	for (j = 0; j < 31; j = j + 1) A[j] <= 0;
	L = 0;
	D = 0;
end
else
begin
	if (~phase0)
	begin
		if (~phase32) lambda[0] <= lambda[31] ^ tmp0;
		else lambda[0] <= lambda[31];
		for (j = 1; j < 32; j = j + 1)
			lambda[j] <= lambda[j-1];
	end

// for (j = t-1; j >=0; j--)
//	if (delta) B[j] = lambda[j] *DI;
//	else if (j != 0) B[j] = B[j-1];
//	else B[j] = 0;
	if (~phase0)
	begin
		if (delta)		B[0] <= tmp1;
		else if (~phase32) B[0] <= B[30];
		else B[0] <= 0;
		for (j = 1; j < 31; j = j + 1)
			B[j] <= B[j-1];
	end

// for (j = t-1; j >=0; j--)
```

```
//	if (j != 0) omega[j] += D * A[j-1];
		if (~phase0)
		begin
			if (~phase32) omega[0] <= omega[31] ^ tmp2;
			else omega[0] <= omega[31];
			for (j = 1; j < 32; j = j + 1)
				omega[j] <= omega[j-1];
		end

	// for (j = t-1; j >=0; j--)
	//	if (delta) A[j] = omega[j] *DI;
	//	else if (j != 0) A[j] = A[j-1];
	//	else A[j] = 0;
		if (~phase0)
		begin
			if (delta)	A[0] <= tmp3;
			else if (~phase32) A[0] <= A[30];
			else A[0] <= 0;
			for (j = 1; j < 31; j = j + 1)
				A[j] <= A[j-1];
		end

	// if (delta) L = i - L + 1;
		if ((phase0 & delta) && (count != -1)) L = count - L + 1;

	//for (D = j = 0; j < t; j = j + 1)
	//	D += lambda[j] * syndrome[t-j-1];
		if (phase0)
			D = tmp0 ^ tmp1 ^ tmp2 ^ tmp3 ^ tmp4 ^ tmp5 ^ tmp6 ^ tmp7 ^
tmp8 ^ tmp9 ^ tmp10 ^ tmp11 ^ tmp12 ^ tmp13 ^ tmp14 ^ tmp15 ^ tmp16 ^ tmp17 ^ tmp18 ^
tmp19 ^ tmp20 ^ tmp21 ^ tmp22 ^ tmp23 ^ tmp24 ^ tmp25 ^ tmp26 ^ tmp27 ^ tmp28 ^ tmp29 ^
tmp30 ^ tmp31;

	end
end

endmodule


module rsdec_berl_multiply (y, a, b, c, d, e);
	input [7:0] a, b, c, d;
	input e;
	output [7:0] y;
	wire [7:0] y;
	reg [7:0] p, q;
```

```
	always @ (a or c or e)
		if (e) p = c;
		else p = a;
	always @ (b or d or e)
		if (e) q = d;
		else q = b;

	multiply x0 (y, p, q);

endmodule

module multiply (y, a, b);
	input [7:0] a, b;
	output [7:0] y;
	reg [7:0] y;
	always @ (a or b)
	begin
		y[0] = (a[0] & b[0]) ^ (a[1] & b[7]) ^ (a[2] & b[6]) ^ (a[2] & b[7]) ^ (a[3] & b[5])
^ (a[3] & b[6]) ^ (a[3] & b[7]) ^ (a[4] & b[4]) ^ (a[4] & b[5]) ^ (a[4] & b[6]) ^ (a[4] & b[7]) ^
(a[5] & b[3]) ^ (a[5] & b[4]) ^ (a[5] & b[5]) ^ (a[5] & b[6]) ^ (a[5] & b[7]) ^ (a[6] & b[2]) ^
(a[6] & b[3]) ^ (a[6] & b[4]) ^ (a[6] & b[5]) ^ (a[6] & b[6]) ^ (a[6] & b[7]) ^ (a[7] & b[1]) ^
(a[7] & b[2]) ^ (a[7] & b[3]) ^ (a[7] & b[4]) ^ (a[7] & b[5]) ^ (a[7] & b[6]);
		y[1] = (a[0] & b[1]) ^ (a[1] & b[0]) ^ (a[1] & b[7]) ^ (a[2] & b[6]) ^ (a[3] & b[5])
^ (a[4] & b[4]) ^ (a[5] & b[3]) ^ (a[6] & b[2]) ^ (a[7] & b[1]) ^ (a[7] & b[7]);
		y[2] = (a[0] & b[2]) ^ (a[1] & b[1]) ^ (a[1] & b[7]) ^ (a[2] & b[0]) ^ (a[2] & b[6])
^ (a[3] & b[5]) ^ (a[3] & b[7]) ^ (a[4] & b[4]) ^ (a[4] & b[6]) ^ (a[4] & b[7]) ^ (a[5] & b[3]) ^
(a[5] & b[5]) ^ (a[5] & b[6]) ^ (a[5] & b[7]) ^ (a[6] & b[2]) ^ (a[6] & b[4]) ^ (a[6] & b[5]) ^
(a[6] & b[6]) ^ (a[6] & b[7]) ^ (a[7] & b[1]) ^ (a[7] & b[3]) ^ (a[7] & b[4]) ^ (a[7] & b[5]) ^
(a[7] & b[6]);
		y[3] = (a[0] & b[3]) ^ (a[1] & b[2]) ^ (a[2] & b[1]) ^ (a[2] & b[7]) ^ (a[3] & b[0])
^ (a[3] & b[6]) ^ (a[4] & b[5]) ^ (a[4] & b[7]) ^ (a[5] & b[4]) ^ (a[5] & b[6]) ^ (a[5] & b[7]) ^
(a[6] & b[3]) ^ (a[6] & b[5]) ^ (a[6] & b[6]) ^ (a[6] & b[7]) ^ (a[7] & b[2]) ^ (a[7] & b[4]) ^
(a[7] & b[5]) ^ (a[7] & b[6]) ^ (a[7] & b[7]);
		y[4] = (a[0] & b[4]) ^ (a[1] & b[3]) ^ (a[2] & b[2]) ^ (a[3] & b[1]) ^ (a[3] & b[7])
^ (a[4] & b[0]) ^ (a[4] & b[6]) ^ (a[5] & b[5]) ^ (a[5] & b[7]) ^ (a[6] & b[4]) ^ (a[6] & b[6]) ^
(a[6] & b[7]) ^ (a[7] & b[3]) ^ (a[7] & b[5]) ^ (a[7] & b[6]) ^ (a[7] & b[7]);
		y[5] = (a[0] & b[5]) ^ (a[1] & b[4]) ^ (a[2] & b[3]) ^ (a[3] & b[2]) ^ (a[4] & b[1])
^ (a[4] & b[7]) ^ (a[5] & b[0]) ^ (a[5] & b[6]) ^ (a[6] & b[5]) ^ (a[6] & b[7]) ^ (a[7] & b[4]) ^
(a[7] & b[6]) ^ (a[7] & b[7]);
		y[6] = (a[0] & b[6]) ^ (a[1] & b[5]) ^ (a[2] & b[4]) ^ (a[3] & b[3]) ^ (a[4] & b[2])
^ (a[5] & b[1]) ^ (a[5] & b[7]) ^ (a[6] & b[0]) ^ (a[6] & b[6]) ^ (a[7] & b[5]) ^ (a[7] & b[7]);
		y[7] = (a[0] & b[7]) ^ (a[1] & b[6]) ^ (a[1] & b[7]) ^ (a[2] & b[5]) ^ (a[2] & b[6])
^ (a[2] & b[7]) ^ (a[3] & b[4]) ^ (a[3] & b[5]) ^ (a[3] & b[6]) ^ (a[3] & b[7]) ^ (a[4] & b[3]) ^
(a[4] & b[4]) ^ (a[4] & b[5]) ^ (a[4] & b[6]) ^ (a[4] & b[7]) ^ (a[5] & b[2]) ^ (a[5] & b[3]) ^
(a[5] & b[4]) ^ (a[5] & b[5]) ^ (a[5] & b[6]) ^ (a[5] & b[7]) ^ (a[6] & b[1]) ^ (a[6] & b[2]) ^
```

```
(a[6] & b[3]) ^ (a[6] & b[4]) ^ (a[6] & b[5]) ^ (a[6] & b[6]) ^ (a[7] & b[0]) ^ (a[7] & b[1]) ^
(a[7] & b[2]) ^ (a[7] & b[3]) ^ (a[7] & b[4]) ^ (a[7] & b[5]);
          end
endmodule
```

chien-search.v

```
module rsdec_chien_scale0 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0];
		y[1] = x[1];
		y[2] = x[2];
		y[3] = x[3];
		y[4] = x[4];
		y[5] = x[5];
		y[6] = x[6];
		y[7] = x[7];
	end
endmodule

module rsdec_chien_scale1 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[7];
		y[1] = x[0] ^ x[7];
		y[2] = x[1] ^ x[7];
		y[3] = x[2];
		y[4] = x[3];
		y[5] = x[4];
		y[6] = x[5];
		y[7] = x[6] ^ x[7];
	end
endmodule

module rsdec_chien_scale2 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[6] ^ x[7];
```

```
        y[1] = x[6];
        y[2] = x[0] ^ x[6];
        y[3] = x[1] ^ x[7];
        y[4] = x[2];
        y[5] = x[3];
        y[6] = x[4];
        y[7] = x[5] ^ x[6] ^ x[7];
    end
endmodule

module rsdec_chien_scale3 (y, x);
    input [7:0] x;
    output [7:0] y;
    reg [7:0] y;

    always @ (x)
    begin
        y[0] = x[5] ^ x[6] ^ x[7];
        y[1] = x[5];
        y[2] = x[5] ^ x[7];
        y[3] = x[0] ^ x[6];
        y[4] = x[1] ^ x[7];
        y[5] = x[2];
        y[6] = x[3];
        y[7] = x[4] ^ x[5] ^ x[6] ^ x[7];
    end
endmodule

module rsdec_chien_scale4 (y, x);
    input [7:0] x;
    output [7:0] y;
    reg [7:0] y;

    always @ (x)
    begin
        y[0] = x[4] ^ x[5] ^ x[6] ^ x[7];
        y[1] = x[4];
        y[2] = x[4] ^ x[6] ^ x[7];
        y[3] = x[5] ^ x[7];
        y[4] = x[0] ^ x[6];
        y[5] = x[1] ^ x[7];
        y[6] = x[2];
        y[7] = x[3] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
    end
endmodule
```

```
module rsdec_chien_scale5 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[3] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[1] = x[3];
		y[2] = x[3] ^ x[5] ^ x[6] ^ x[7];
		y[3] = x[4] ^ x[6] ^ x[7];
		y[4] = x[5] ^ x[7];
		y[5] = x[0] ^ x[6];
		y[6] = x[1] ^ x[7];
		y[7] = x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
	end
endmodule

module rsdec_chien_scale6 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[1] = x[2];
		y[2] = x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[3] = x[3] ^ x[5] ^ x[6] ^ x[7];
		y[4] = x[4] ^ x[6] ^ x[7];
		y[5] = x[5] ^ x[7];
		y[6] = x[0] ^ x[6];
		y[7] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
	end
endmodule

module rsdec_chien_scale7 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[1] = x[1] ^ x[7];
		y[2] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
```

```
		y[3] = x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[4] = x[3] ^ x[5] ^ x[6] ^ x[7];
		y[5] = x[4] ^ x[6] ^ x[7];
		y[6] = x[5] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
	end
endmodule

module rsdec_chien_scale8 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[1] = x[0] ^ x[6];
		y[2] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[3] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[4] = x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[5] = x[3] ^ x[5] ^ x[6] ^ x[7];
		y[6] = x[4] ^ x[6] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[7];
	end
endmodule

module rsdec_chien_scale9 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[7];
		y[1] = x[5] ^ x[7];
		y[2] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[3] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[4] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[5] = x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[6] = x[3] ^ x[5] ^ x[6] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[6];
	end
endmodule

module rsdec_chien_scale10 (y, x);
	input [7:0] x;
```

```
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[6];
		y[1] = x[4] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[3] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[4] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[5] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[6] = x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[5] ^ x[7];
	end
endmodule

module rsdec_chien_scale11 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[5] ^ x[7];
		y[1] = x[3] ^ x[5] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6];
		y[3] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[4] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[5] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[6] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[7] = x[0] ^ x[1] ^ x[4] ^ x[6];
	end
endmodule

module rsdec_chien_scale12 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[4] ^ x[6];
		y[1] = x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[3] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6];
		y[4] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
```

```
		y[5] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[6] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[7] = x[0] ^ x[3] ^ x[5];
	end
endmodule

module rsdec_chien_scale13 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[3] ^ x[5];
		y[1] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[2] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[3] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[4] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6];
		y[5] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[6] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[7] = x[2] ^ x[4] ^ x[7];
	end
endmodule

module rsdec_chien_scale14 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[2] ^ x[4] ^ x[7];
		y[1] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[2] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[3] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[4] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[5] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6];
		y[6] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[7] = x[1] ^ x[3] ^ x[6];
	end
endmodule

module rsdec_chien_scale15 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
```

```
	always @ (x)
	begin
		y[0] = x[1] ^ x[3] ^ x[6];
		y[1] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[3] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[4] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[5] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[6] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6];
		y[7] = x[0] ^ x[2] ^ x[5] ^ x[7];
	end
endmodule

module rsdec_chien_scale16 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[2] ^ x[5] ^ x[7];
		y[1] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[3] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[4] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[5] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[6] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[7] = x[1] ^ x[4] ^ x[6] ^ x[7];
	end
endmodule

module rsdec_chien_scale17 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[1] ^ x[4] ^ x[6] ^ x[7];
		y[1] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6];
		y[2] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[3] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[4] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[5] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[6] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
```

```
		y[7] = x[0] ^ x[3] ^ x[5] ^ x[6];
	end
endmodule

module rsdec_chien_scale18 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[3] ^ x[5] ^ x[6];
		y[1] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[2] = x[1] ^ x[2] ^ x[3] ^ x[4];
		y[3] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[4] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[5] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[6] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[7] = x[2] ^ x[4] ^ x[5] ^ x[7];
	end
endmodule

module rsdec_chien_scale19 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[2] ^ x[4] ^ x[5] ^ x[7];
		y[1] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[2] ^ x[3];
		y[3] = x[1] ^ x[2] ^ x[3] ^ x[4];
		y[4] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[5] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[6] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[7] = x[1] ^ x[3] ^ x[4] ^ x[6];
	end
endmodule

module rsdec_chien_scale20 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
```

```
	begin
		y[0] = x[1] ^ x[3] ^ x[4] ^ x[6];
		y[1] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[2] ^ x[7];
		y[3] = x[0] ^ x[1] ^ x[2] ^ x[3];
		y[4] = x[1] ^ x[2] ^ x[3] ^ x[4];
		y[5] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[6] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[7] = x[0] ^ x[2] ^ x[3] ^ x[5] ^ x[7];
	end
endmodule

module rsdec_chien_scale21 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[2] ^ x[3] ^ x[5] ^ x[7];
		y[1] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[6];
		y[3] = x[0] ^ x[1] ^ x[2] ^ x[7];
		y[4] = x[0] ^ x[1] ^ x[2] ^ x[3];
		y[5] = x[1] ^ x[2] ^ x[3] ^ x[4];
		y[6] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[7] = x[1] ^ x[2] ^ x[4] ^ x[6] ^ x[7];
	end
endmodule

module rsdec_chien_scale22 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[1] ^ x[2] ^ x[4] ^ x[6] ^ x[7];
		y[1] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[2] = x[0] ^ x[5];
		y[3] = x[0] ^ x[1] ^ x[6];
		y[4] = x[0] ^ x[1] ^ x[2] ^ x[7];
		y[5] = x[0] ^ x[1] ^ x[2] ^ x[3];
		y[6] = x[1] ^ x[2] ^ x[3] ^ x[4];
		y[7] = x[0] ^ x[1] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
	end
```

```
endmodule

module rsdec_chien_scale23 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[1] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[2] = x[4] ^ x[7];
		y[3] = x[0] ^ x[5];
		y[4] = x[0] ^ x[1] ^ x[6];
		y[5] = x[0] ^ x[1] ^ x[2] ^ x[7];
		y[6] = x[0] ^ x[1] ^ x[2] ^ x[3];
		y[7] = x[0] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
	end
endmodule

module rsdec_chien_scale24 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[1] = x[1] ^ x[2] ^ x[3] ^ x[4];
		y[2] = x[3] ^ x[6] ^ x[7];
		y[3] = x[4] ^ x[7];
		y[4] = x[0] ^ x[5];
		y[5] = x[0] ^ x[1] ^ x[6];
		y[6] = x[0] ^ x[1] ^ x[2] ^ x[7];
		y[7] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
	end
endmodule

module rsdec_chien_scale25 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
```

```
		y[1] = x[0] ^ x[1] ^ x[2] ^ x[3];
		y[2] = x[2] ^ x[5] ^ x[6] ^ x[7];
		y[3] = x[3] ^ x[6] ^ x[7];
		y[4] = x[4] ^ x[7];
		y[5] = x[0] ^ x[5];
		y[6] = x[0] ^ x[1] ^ x[6];
		y[7] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
	end
endmodule

module rsdec_chien_scale26 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[1] = x[0] ^ x[1] ^ x[2] ^ x[7];
		y[2] = x[1] ^ x[4] ^ x[5] ^ x[6];
		y[3] = x[2] ^ x[5] ^ x[6] ^ x[7];
		y[4] = x[3] ^ x[6] ^ x[7];
		y[5] = x[4] ^ x[7];
		y[6] = x[0] ^ x[5];
		y[7] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
	end
endmodule

module rsdec_chien_scale27 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[1] = x[0] ^ x[1] ^ x[6];
		y[2] = x[0] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[3] = x[1] ^ x[4] ^ x[5] ^ x[6];
		y[4] = x[2] ^ x[5] ^ x[6] ^ x[7];
		y[5] = x[3] ^ x[6] ^ x[7];
		y[6] = x[4] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[4];
	end
endmodule
```

```
module rsdec_chien_scale28 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[4];
		y[1] = x[0] ^ x[5];
		y[2] = x[2] ^ x[3] ^ x[4] ^ x[6];
		y[3] = x[0] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[4] = x[1] ^ x[4] ^ x[5] ^ x[6];
		y[5] = x[2] ^ x[5] ^ x[6] ^ x[7];
		y[6] = x[3] ^ x[6] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[7];
	end
endmodule

module rsdec_chien_scale29 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[7];
		y[1] = x[4] ^ x[7];
		y[2] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[7];
		y[3] = x[2] ^ x[3] ^ x[4] ^ x[6];
		y[4] = x[0] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[5] = x[1] ^ x[4] ^ x[5] ^ x[6];
		y[6] = x[2] ^ x[5] ^ x[6] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[6];
	end
endmodule

module rsdec_chien_scale30 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[6];
		y[1] = x[3] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[6] ^ x[7];
```

```
		y[3] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[7];
		y[4] = x[2] ^ x[3] ^ x[4] ^ x[6];
		y[5] = x[0] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[6] = x[1] ^ x[4] ^ x[5] ^ x[6];
		y[7] = x[0] ^ x[1] ^ x[5] ^ x[7];
	end
endmodule

module rsdec_chien_scale31 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[5] ^ x[7];
		y[1] = x[2] ^ x[5] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[3] ^ x[5] ^ x[6];
		y[3] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[6] ^ x[7];
		y[4] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[7];
		y[5] = x[2] ^ x[3] ^ x[4] ^ x[6];
		y[6] = x[0] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[7] = x[0] ^ x[4] ^ x[6] ^ x[7];
	end
endmodule

module rsdec_chien (error, alpha, lambda, omega, even, D, search, load, shorten, clk, rst_n);
	input clk, rst_n, load, search, shorten;
	input [7:0] D;
	input [7:0] lambda;
	input [7:0] omega;
	output [7:0] even, error;
	output [7:0] alpha;
	reg [7:0] even, error;
	reg [7:0] alpha;

	wire [7:0] scale0;
	wire [7:0] scale1;
	wire [7:0] scale2;
	wire [7:0] scale3;
	wire [7:0] scale4;
	wire [7:0] scale5;
	wire [7:0] scale6;
	wire [7:0] scale7;
	wire [7:0] scale8;
	wire [7:0] scale9;
```

```
wire [7:0] scale10;
wire [7:0] scale11;
wire [7:0] scale12;
wire [7:0] scale13;
wire [7:0] scale14;
wire [7:0] scale15;
wire [7:0] scale16;
wire [7:0] scale17;
wire [7:0] scale18;
wire [7:0] scale19;
wire [7:0] scale20;
wire [7:0] scale21;
wire [7:0] scale22;
wire [7:0] scale23;
wire [7:0] scale24;
wire [7:0] scale25;
wire [7:0] scale26;
wire [7:0] scale27;
wire [7:0] scale28;
wire [7:0] scale29;
wire [7:0] scale30;
wire [7:0] scale31;
wire [7:0] scale32;
wire [7:0] scale33;
wire [7:0] scale34;
wire [7:0] scale35;
wire [7:0] scale36;
wire [7:0] scale37;
wire [7:0] scale38;
wire [7:0] scale39;
wire [7:0] scale40;
wire [7:0] scale41;
wire [7:0] scale42;
wire [7:0] scale43;
wire [7:0] scale44;
wire [7:0] scale45;
wire [7:0] scale46;
wire [7:0] scale47;
wire [7:0] scale48;
wire [7:0] scale49;
wire [7:0] scale50;
wire [7:0] scale51;
wire [7:0] scale52;
wire [7:0] scale53;
wire [7:0] scale54;
wire [7:0] scale55;
```

```
wire [7:0] scale56;
wire [7:0] scale57;
wire [7:0] scale58;
wire [7:0] scale59;
wire [7:0] scale60;
wire [7:0] scale61;
wire [7:0] scale62;
wire [7:0] scale63;
reg [7:0] data0;
reg [7:0] data1;
reg [7:0] data2;
reg [7:0] data3;
reg [7:0] data4;
reg [7:0] data5;
reg [7:0] data6;
reg [7:0] data7;
reg [7:0] data8;
reg [7:0] data9;
reg [7:0] data10;
reg [7:0] data11;
reg [7:0] data12;
reg [7:0] data13;
reg [7:0] data14;
reg [7:0] data15;
reg [7:0] data16;
reg [7:0] data17;
reg [7:0] data18;
reg [7:0] data19;
reg [7:0] data20;
reg [7:0] data21;
reg [7:0] data22;
reg [7:0] data23;
reg [7:0] data24;
reg [7:0] data25;
reg [7:0] data26;
reg [7:0] data27;
reg [7:0] data28;
reg [7:0] data29;
reg [7:0] data30;
reg [7:0] data31;
reg [7:0] a0;
reg [7:0] a1;
reg [7:0] a2;
reg [7:0] a3;
reg [7:0] a4;
reg [7:0] a5;
```

```
reg [7:0] a6;
reg [7:0] a7;
reg [7:0] a8;
reg [7:0] a9;
reg [7:0] a10;
reg [7:0] a11;
reg [7:0] a12;
reg [7:0] a13;
reg [7:0] a14;
reg [7:0] a15;
reg [7:0] a16;
reg [7:0] a17;
reg [7:0] a18;
reg [7:0] a19;
reg [7:0] a20;
reg [7:0] a21;
reg [7:0] a22;
reg [7:0] a23;
reg [7:0] a24;
reg [7:0] a25;
reg [7:0] a26;
reg [7:0] a27;
reg [7:0] a28;
reg [7:0] a29;
reg [7:0] a30;
reg [7:0] a31;
reg [7:0] l0;
reg [7:0] l1;
reg [7:0] l2;
reg [7:0] l3;
reg [7:0] l4;
reg [7:0] l5;
reg [7:0] l6;
reg [7:0] l7;
reg [7:0] l8;
reg [7:0] l9;
reg [7:0] l10;
reg [7:0] l11;
reg [7:0] l12;
reg [7:0] l13;
reg [7:0] l14;
reg [7:0] l15;
reg [7:0] l16;
reg [7:0] l17;
reg [7:0] l18;
reg [7:0] l19;
```

```
reg [7:0] l20;
reg [7:0] l21;
reg [7:0] l22;
reg [7:0] l23;
reg [7:0] l24;
reg [7:0] l25;
reg [7:0] l26;
reg [7:0] l27;
reg [7:0] l28;
reg [7:0] l29;
reg [7:0] l30;
reg [7:0] l31;
reg [7:0] o0;
reg [7:0] o1;
reg [7:0] o2;
reg [7:0] o3;
reg [7:0] o4;
reg [7:0] o5;
reg [7:0] o6;
reg [7:0] o7;
reg [7:0] o8;
reg [7:0] o9;
reg [7:0] o10;
reg [7:0] o11;
reg [7:0] o12;
reg [7:0] o13;
reg [7:0] o14;
reg [7:0] o15;
reg [7:0] o16;
reg [7:0] o17;
reg [7:0] o18;
reg [7:0] o19;
reg [7:0] o20;
reg [7:0] o21;
reg [7:0] o22;
reg [7:0] o23;
reg [7:0] o24;
reg [7:0] o25;
reg [7:0] o26;
reg [7:0] o27;
reg [7:0] o28;
reg [7:0] o29;
reg [7:0] o30;
reg [7:0] o31;
reg [7:0] odd, numerator;
wire [7:0] tmp;
```

```
integer j;

 rsdec_chien_scale0  rsdec_chien_scale0_inst (scale0, data0);
 rsdec_chien_scale1  rsdec_chien_scale1_inst (scale1, data1);
 rsdec_chien_scale2  rsdec_chien_scale2_inst (scale2, data2);
 rsdec_chien_scale3  rsdec_chien_scale3_inst (scale3, data3);
 rsdec_chien_scale4  rsdec_chien_scale4_inst (scale4, data4);
 rsdec_chien_scale5  rsdec_chien_scale5_inst (scale5, data5);
 rsdec_chien_scale6  rsdec_chien_scale6_inst (scale6, data6);
 rsdec_chien_scale7  rsdec_chien_scale7_inst (scale7, data7);
 rsdec_chien_scale8  rsdec_chien_scale8_inst (scale8, data8);
 rsdec_chien_scale9  rsdec_chien_scale9_inst (scale9, data9);
rsdec_chien_scale10 rsdec_chien_scale10_inst (scale10, data10);
rsdec_chien_scale11 rsdec_chien_scale11_inst (scale11, data11);
rsdec_chien_scale12 rsdec_chien_scale12_inst (scale12, data12);
rsdec_chien_scale13 rsdec_chien_scale13_inst (scale13, data13);
rsdec_chien_scale14 rsdec_chien_scale14_inst (scale14, data14);
rsdec_chien_scale15 rsdec_chien_scale15_inst (scale15, data15);
rsdec_chien_scale16 rsdec_chien_scale16_inst (scale16, data16);
rsdec_chien_scale17 rsdec_chien_scale17_inst (scale17, data17);
rsdec_chien_scale18 rsdec_chien_scale18_inst (scale18, data18);
rsdec_chien_scale19 rsdec_chien_scale19_inst (scale19, data19);
rsdec_chien_scale20 rsdec_chien_scale20_inst (scale20, data20);
rsdec_chien_scale21 rsdec_chien_scale21_inst (scale21, data21);
rsdec_chien_scale22 rsdec_chien_scale22_inst (scale22, data22);
rsdec_chien_scale23 rsdec_chien_scale23_inst (scale23, data23);
rsdec_chien_scale24 rsdec_chien_scale24_inst (scale24, data24);
rsdec_chien_scale25 rsdec_chien_scale25_inst (scale25, data25);
rsdec_chien_scale26 rsdec_chien_scale26_inst (scale26, data26);
rsdec_chien_scale27 rsdec_chien_scale27_inst (scale27, data27);
rsdec_chien_scale28 rsdec_chien_scale28_inst (scale28, data28);
rsdec_chien_scale29 rsdec_chien_scale29_inst (scale29, data29);
rsdec_chien_scale30 rsdec_chien_scale30_inst (scale30, data30);
rsdec_chien_scale31 rsdec_chien_scale31_inst (scale31, data31);
 rsdec_chien_scale0  rsdec_chien_scale0_inst2 (scale32, o0);
 rsdec_chien_scale1  rsdec_chien_scale1_inst2 (scale33, o1);
 rsdec_chien_scale2  rsdec_chien_scale2_inst2 (scale34, o2);
 rsdec_chien_scale3  rsdec_chien_scale3_inst2 (scale35, o3);
 rsdec_chien_scale4  rsdec_chien_scale4_inst2 (scale36, o4);
 rsdec_chien_scale5  rsdec_chien_scale5_inst2 (scale37, o5);
 rsdec_chien_scale6  rsdec_chien_scale6_inst2 (scale38, o6);
 rsdec_chien_scale7  rsdec_chien_scale7_inst2 (scale39, o7);
 rsdec_chien_scale8  rsdec_chien_scale8_inst2 (scale40, o8);
 rsdec_chien_scale9  rsdec_chien_scale9_inst2 (scale41, o9);
rsdec_chien_scale10 rsdec_chien_scale10_inst2 (scale42, o10);
rsdec_chien_scale11 rsdec_chien_scale11_inst2 (scale43, o11);
```

```
rsdec_chien_scale12 rsdec_chien_scale12_inst2 (scale44, o12);
rsdec_chien_scale13 rsdec_chien_scale13_inst2 (scale45, o13);
rsdec_chien_scale14 rsdec_chien_scale14_inst2 (scale46, o14);
rsdec_chien_scale15 rsdec_chien_scale15_inst2 (scale47, o15);
rsdec_chien_scale16 rsdec_chien_scale16_inst2 (scale48, o16);
rsdec_chien_scale17 rsdec_chien_scale17_inst2 (scale49, o17);
rsdec_chien_scale18 rsdec_chien_scale18_inst2 (scale50, o18);
rsdec_chien_scale19 rsdec_chien_scale19_inst2 (scale51, o19);
rsdec_chien_scale20 rsdec_chien_scale20_inst2 (scale52, o20);
rsdec_chien_scale21 rsdec_chien_scale21_inst2 (scale53, o21);
rsdec_chien_scale22 rsdec_chien_scale22_inst2 (scale54, o22);
rsdec_chien_scale23 rsdec_chien_scale23_inst2 (scale55, o23);
rsdec_chien_scale24 rsdec_chien_scale24_inst2 (scale56, o24);
rsdec_chien_scale25 rsdec_chien_scale25_inst2 (scale57, o25);
rsdec_chien_scale26 rsdec_chien_scale26_inst2 (scale58, o26);
rsdec_chien_scale27 rsdec_chien_scale27_inst2 (scale59, o27);
rsdec_chien_scale28 rsdec_chien_scale28_inst2 (scale60, o28);
rsdec_chien_scale29 rsdec_chien_scale29_inst2 (scale61, o29);
rsdec_chien_scale30 rsdec_chien_scale30_inst2 (scale62, o30);
rsdec_chien_scale31 rsdec_chien_scale31_inst2 (scale63, o31);

always @ (shorten or a0 or l0)
	if (shorten) data0 = a0;
	else data0 = l0;

always @ (shorten or a1 or l1)
	if (shorten) data1 = a1;
	else data1 = l1;

always @ (shorten or a2 or l2)
	if (shorten) data2 = a2;
	else data2 = l2;

always @ (shorten or a3 or l3)
	if (shorten) data3 = a3;
	else data3 = l3;

always @ (shorten or a4 or l4)
	if (shorten) data4 = a4;
	else data4 = l4;

always @ (shorten or a5 or l5)
	if (shorten) data5 = a5;
	else data5 = l5;

always @ (shorten or a6 or l6)
```

```
        if (shorten) data6 = a6;
        else data6 = l6;

always @ (shorten or a7 or l7)
        if (shorten) data7 = a7;
        else data7 = l7;

always @ (shorten or a8 or l8)
        if (shorten) data8 = a8;
        else data8 = l8;

always @ (shorten or a9 or l9)
        if (shorten) data9 = a9;
        else data9 = l9;

always @ (shorten or a10 or l10)
        if (shorten) data10 = a10;
        else data10 = l10;

always @ (shorten or a11 or l11)
        if (shorten) data11 = a11;
        else data11 = l11;

always @ (shorten or a12 or l12)
        if (shorten) data12 = a12;
        else data12 = l12;

always @ (shorten or a13 or l13)
        if (shorten) data13 = a13;
        else data13 = l13;

always @ (shorten or a14 or l14)
        if (shorten) data14 = a14;
        else data14 = l14;

always @ (shorten or a15 or l15)
        if (shorten) data15 = a15;
        else data15 = l15;

always @ (shorten or a16 or l16)
        if (shorten) data16 = a16;
        else data16 = l16;

always @ (shorten or a17 or l17)
        if (shorten) data17 = a17;
        else data17 = l17;
```

```
always @ (shorten or a18 or l18)
        if (shorten) data18 = a18;
        else data18 = l18;

always @ (shorten or a19 or l19)
        if (shorten) data19 = a19;
        else data19 = l19;

always @ (shorten or a20 or l20)
        if (shorten) data20 = a20;
        else data20 = l20;

always @ (shorten or a21 or l21)
        if (shorten) data21 = a21;
        else data21 = l21;

always @ (shorten or a22 or l22)
        if (shorten) data22 = a22;
        else data22 = l22;

always @ (shorten or a23 or l23)
        if (shorten) data23 = a23;
        else data23 = l23;

always @ (shorten or a24 or l24)
        if (shorten) data24 = a24;
        else data24 = l24;

always @ (shorten or a25 or l25)
        if (shorten) data25 = a25;
        else data25 = l25;

always @ (shorten or a26 or l26)
        if (shorten) data26 = a26;
        else data26 = l26;

always @ (shorten or a27 or l27)
        if (shorten) data27 = a27;
        else data27 = l27;

always @ (shorten or a28 or l28)
        if (shorten) data28 = a28;
        else data28 = l28;

always @ (shorten or a29 or l29)
```

```
		if (shorten) data29 = a29;
		else data29 = l29;

	always @ (shorten or a30 or l30)
		if (shorten) data30 = a30;
		else data30 = l30;

	always @ (shorten or a31 or l31)
		if (shorten) data31 = a31;
		else data31 = l31;

	always @ (posedge clk or negedge rst_n)
	begin
		if (~rst_n)
		begin
			l0 <= 0;
			l1 <= 0;
			l2 <= 0;
			l3 <= 0;
			l4 <= 0;
			l5 <= 0;
			l6 <= 0;
			l7 <= 0;
			l8 <= 0;
			l9 <= 0;
			l10 <= 0;
			l11 <= 0;
			l12 <= 0;
			l13 <= 0;
			l14 <= 0;
			l15 <= 0;
			l16 <= 0;
			l17 <= 0;
			l18 <= 0;
			l19 <= 0;
			l20 <= 0;
			l21 <= 0;
			l22 <= 0;
			l23 <= 0;
			l24 <= 0;
			l25 <= 0;
			l26 <= 0;
			l27 <= 0;
			l28 <= 0;
			l29 <= 0;
			l30 <= 0;
```

```
l31 <= 0;
o0 <= 0;
o1 <= 0;
o2 <= 0;
o3 <= 0;
o4 <= 0;
o5 <= 0;
o6 <= 0;
o7 <= 0;
o8 <= 0;
o9 <= 0;
o10 <= 0;
o11 <= 0;
o12 <= 0;
o13 <= 0;
o14 <= 0;
o15 <= 0;
o16 <= 0;
o17 <= 0;
o18 <= 0;
o19 <= 0;
o20 <= 0;
o21 <= 0;
o22 <= 0;
o23 <= 0;
o24 <= 0;
o25 <= 0;
o26 <= 0;
o27 <= 0;
o28 <= 0;
o29 <= 0;
o30 <= 0;
o31 <= 0;
a0 <= 1;
a1 <= 1;
a2 <= 1;
a3 <= 1;
a4 <= 1;
a5 <= 1;
a6 <= 1;
a7 <= 1;
a8 <= 1;
a9 <= 1;
a10 <= 1;
a11 <= 1;
a12 <= 1;
```

```
		a13 <= 1;
		a14 <= 1;
		a15 <= 1;
		a16 <= 1;
		a17 <= 1;
		a18 <= 1;
		a19 <= 1;
		a20 <= 1;
		a21 <= 1;
		a22 <= 1;
		a23 <= 1;
		a24 <= 1;
		a25 <= 1;
		a26 <= 1;
		a27 <= 1;
		a28 <= 1;
		a29 <= 1;
		a30 <= 1;
		a31 <= 1;
	end
	else if (shorten)
	begin
		a0 <= scale0;
		a1 <= scale1;
		a2 <= scale2;
		a3 <= scale3;
		a4 <= scale4;
		a5 <= scale5;
		a6 <= scale6;
		a7 <= scale7;
		a8 <= scale8;
		a9 <= scale9;
		a10 <= scale10;
		a11 <= scale11;
		a12 <= scale12;
		a13 <= scale13;
		a14 <= scale14;
		a15 <= scale15;
		a16 <= scale16;
		a17 <= scale17;
		a18 <= scale18;
		a19 <= scale19;
		a20 <= scale20;
		a21 <= scale21;
		a22 <= scale22;
		a23 <= scale23;
```

```
        a24 <= scale24;
        a25 <= scale25;
        a26 <= scale26;
        a27 <= scale27;
        a28 <= scale28;
        a29 <= scale29;
        a30 <= scale30;
        a31 <= scale31;
end
else if (search)
begin
        l0 <= scale0;
        l1 <= scale1;
        l2 <= scale2;
        l3 <= scale3;
        l4 <= scale4;
        l5 <= scale5;
        l6 <= scale6;
        l7 <= scale7;
        l8 <= scale8;
        l9 <= scale9;
        l10 <= scale10;
        l11 <= scale11;
        l12 <= scale12;
        l13 <= scale13;
        l14 <= scale14;
        l15 <= scale15;
        l16 <= scale16;
        l17 <= scale17;
        l18 <= scale18;
        l19 <= scale19;
        l20 <= scale20;
        l21 <= scale21;
        l22 <= scale22;
        l23 <= scale23;
        l24 <= scale24;
        l25 <= scale25;
        l26 <= scale26;
        l27 <= scale27;
        l28 <= scale28;
        l29 <= scale29;
        l30 <= scale30;
        l31 <= scale31;
        o0 <= scale32;
        o1 <= scale33;
        o2 <= scale34;
```

```
        o3 <= scale35;
        o4 <= scale36;
        o5 <= scale37;
        o6 <= scale38;
        o7 <= scale39;
        o8 <= scale40;
        o9 <= scale41;
        o10 <= scale42;
        o11 <= scale43;
        o12 <= scale44;
        o13 <= scale45;
        o14 <= scale46;
        o15 <= scale47;
        o16 <= scale48;
        o17 <= scale49;
        o18 <= scale50;
        o19 <= scale51;
        o20 <= scale52;
        o21 <= scale53;
        o22 <= scale54;
        o23 <= scale55;
        o24 <= scale56;
        o25 <= scale57;
        o26 <= scale58;
        o27 <= scale59;
        o28 <= scale60;
        o29 <= scale61;
        o30 <= scale62;
        o31 <= scale63;
end
else if (load)
begin
        l0 <= lambda;
        l1 <= l0;
        l2 <= l1;
        l3 <= l2;
        l4 <= l3;
        l5 <= l4;
        l6 <= l5;
        l7 <= l6;
        l8 <= l7;
        l9 <= l8;
        l10 <= l9;
        l11 <= l10;
        l12 <= l11;
        l13 <= l12;
```

```
l14 <= l13;
l15 <= l14;
l16 <= l15;
l17 <= l16;
l18 <= l17;
l19 <= l18;
l20 <= l19;
l21 <= l20;
l22 <= l21;
l23 <= l22;
l24 <= l23;
l25 <= l24;
l26 <= l25;
l27 <= l26;
l28 <= l27;
l29 <= l28;
l30 <= l29;
l31 <= l30;
o0 <= omega;
o1 <= o0;
o2 <= o1;
o3 <= o2;
o4 <= o3;
o5 <= o4;
o6 <= o5;
o7 <= o6;
o8 <= o7;
o9 <= o8;
o10 <= o9;
o11 <= o10;
o12 <= o11;
o13 <= o12;
o14 <= o13;
o15 <= o14;
o16 <= o15;
o17 <= o16;
o18 <= o17;
o19 <= o18;
o20 <= o19;
o21 <= o20;
o22 <= o21;
o23 <= o22;
o24 <= o23;
o25 <= o24;
o26 <= o25;
o27 <= o26;
```

```
                o28 <= o27;
                o29 <= o28;
                o30 <= o29;
                o31 <= o30;
                a0 <= a31;
                a1 <= a0;
                a2 <= a1;
                a3 <= a2;
                a4 <= a3;
                a5 <= a4;
                a6 <= a5;
                a7 <= a6;
                a8 <= a7;
                a9 <= a8;
                a10 <= a9;
                a11 <= a10;
                a12 <= a11;
                a13 <= a12;
                a14 <= a13;
                a15 <= a14;
                a16 <= a15;
                a17 <= a16;
                a18 <= a17;
                a19 <= a18;
                a20 <= a19;
                a21 <= a20;
                a22 <= a21;
                a23 <= a22;
                a24 <= a23;
                a25 <= a24;
                a26 <= a25;
                a27 <= a26;
                a28 <= a27;
                a29 <= a28;
                a30 <= a29;
                a31 <= a30;
            end
    end

    always @ (l0 or l2 or l4 or l6 or l8 or l10 or l12 or l14 or l16 or l18 or l20 or l22 or l24 or
l26 or l28 or l30)
            even = l0 ^ l2 ^ l4 ^ l6 ^ l8 ^ l10 ^ l12 ^ l14 ^ l16 ^ l18 ^ l20 ^ l22 ^ l24 ^ l26 ^
l28 ^ l30;

    always @ (l1 or l3 or l5 or l7 or l9 or l11 or l13 or l15 or l17 or l19 or l21 or l23 or l25 or
l27 or l29 or l31)
```

```
			odd = l1 ^ l3 ^ l5 ^ l7 ^ l9 ^ l11 ^ l13 ^ l15 ^ l17 ^ l19 ^ l21 ^ l23 ^ l25 ^ l27 ^ l29
^ l31;

		always @ (o0 or o1 or o2 or o3 or o4 or o5 or o6 or o7 or o8 or o9 or o10 or o11 or o12
or o13 or o14 or o15 or o16 or o17 or o18 or o19 or o20 or o21 or o22 or o23 or o24 or o25 or
o26 or o27 or o28 or o29 or o30 or o31)
			numerator = o0 ^ o1 ^ o2 ^ o3 ^ o4 ^ o5 ^ o6 ^ o7 ^ o8 ^ o9 ^ o10 ^ o11 ^ o12 ^
o13 ^ o14 ^ o15 ^ o16 ^ o17 ^ o18 ^ o19 ^ o20 ^ o21 ^ o22 ^ o23 ^ o24 ^ o25 ^ o26 ^ o27 ^ o28
^ o29 ^ o30 ^ o31;

		multiply m0 (tmp, numerator, D);

		always @ (even or odd or tmp)
			if (even == odd) error = tmp;
			else error = 0;

		always @ (a31) alpha = a31;

endmodule
```

inverse.v

```
module inverse(y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;

	always @ (x)
	case (x) // synopsys full_case parallel_case
		1: y = 1; // 0 -> 255
		2: y = 195; // 1 -> 254
		4: y = 162; // 2 -> 253
		8: y = 81; // 3 -> 252
		16: y = 235; // 4 -> 251
		32: y = 182; // 5 -> 250
		64: y = 91; // 6 -> 249
		128: y = 238; // 7 -> 248
		135: y = 119; // 8 -> 247
		137: y = 248; // 9 -> 246
		149: y = 124; // 10 -> 245
		173: y = 62; // 11 -> 244
		221: y = 31; // 12 -> 243
		61: y = 204; // 13 -> 242
		122: y = 102; // 14 -> 241
		244: y = 51; // 15 -> 240
		111: y = 218; // 16 -> 239
		222: y = 109; // 17 -> 238
		59: y = 245; // 18 -> 237
		118: y = 185; // 19 -> 236
		236: y = 159; // 20 -> 235
		95: y = 140; // 21 -> 234
		190: y = 70; // 22 -> 233
		251: y = 35; // 23 -> 232
		113: y = 210; // 24 -> 231
		226: y = 105; // 25 -> 230
		67: y = 247; // 26 -> 229
		134: y = 184; // 27 -> 228
		139: y = 92; // 28 -> 227
		145: y = 46; // 29 -> 226
		165: y = 23; // 30 -> 225
		205: y = 200; // 31 -> 224
		29: y = 100; // 32 -> 223
		58: y = 50; // 33 -> 222
		116: y = 25; // 34 -> 221
		232: y = 207; // 35 -> 220
		87: y = 164; // 36 -> 219
```

```
174: y = 82; // 37 -> 218
219: y = 41; // 38 -> 217
49: y = 215; // 39 -> 216
98: y = 168; // 40 -> 215
196: y = 84; // 41 -> 214
15: y = 42; // 42 -> 213
30: y = 21; // 43 -> 212
60: y = 201; // 44 -> 211
120: y = 167; // 45 -> 210
240: y = 144; // 46 -> 209
103: y = 72; // 47 -> 208
206: y = 36; // 48 -> 207
27: y = 18; // 49 -> 206
54: y = 9; // 50 -> 205
108: y = 199; // 51 -> 204
216: y = 160; // 52 -> 203
55: y = 80; // 53 -> 202
110: y = 40; // 54 -> 201
220: y = 20; // 55 -> 200
63: y = 10; // 56 -> 199
126: y = 5; // 57 -> 198
252: y = 193; // 58 -> 197
127: y = 163; // 59 -> 196
254: y = 146; // 60 -> 195
123: y = 73; // 61 -> 194
246: y = 231; // 62 -> 193
107: y = 176; // 63 -> 192
214: y = 88; // 64 -> 191
43: y = 44; // 65 -> 190
86: y = 22; // 66 -> 189
172: y = 11; // 67 -> 188
223: y = 198; // 68 -> 187
57: y = 99; // 69 -> 186
114: y = 242; // 70 -> 185
228: y = 121; // 71 -> 184
79: y = 255; // 72 -> 183
158: y = 188; // 73 -> 182
187: y = 94; // 74 -> 181
241: y = 47; // 75 -> 180
101: y = 212; // 76 -> 179
202: y = 106; // 77 -> 178
19: y = 53; // 78 -> 177
38: y = 217; // 79 -> 176
76: y = 175; // 80 -> 175
152: y = 148; // 81 -> 174
183: y = 74; // 82 -> 173
```

```
233: y = 37; // 83 -> 172
85: y = 209; // 84 -> 171
170: y = 171; // 85 -> 170
211: y = 150; // 86 -> 169
33: y = 75; // 87 -> 168
66: y = 230; // 88 -> 167
132: y = 115; // 89 -> 166
143: y = 250; // 90 -> 165
153: y = 125; // 91 -> 164
181: y = 253; // 92 -> 163
237: y = 189; // 93 -> 162
93: y = 157; // 94 -> 161
186: y = 141; // 95 -> 160
243: y = 133; // 96 -> 159
97: y = 129; // 97 -> 158
194: y = 131; // 98 -> 157
3: y = 130; // 99 -> 156
6: y = 65; // 100 -> 155
12: y = 227; // 101 -> 154
24: y = 178; // 102 -> 153
48: y = 89; // 103 -> 152
96: y = 239; // 104 -> 151
192: y = 180; // 105 -> 150
7: y = 90; // 106 -> 149
14: y = 45; // 107 -> 148
28: y = 213; // 108 -> 147
56: y = 169; // 109 -> 146
112: y = 151; // 110 -> 145
224: y = 136; // 111 -> 144
71: y = 68; // 112 -> 143
142: y = 34; // 113 -> 142
155: y = 17; // 114 -> 141
177: y = 203; // 115 -> 140
229: y = 166; // 116 -> 139
77: y = 83; // 117 -> 138
154: y = 234; // 118 -> 137
179: y = 117; // 119 -> 136
225: y = 249; // 120 -> 135
69: y = 191; // 121 -> 134
138: y = 156; // 122 -> 133
147: y = 78; // 123 -> 132
161: y = 39; // 124 -> 131
197: y = 208; // 125 -> 130
13: y = 104; // 126 -> 129
26: y = 52; // 127 -> 128
52: y = 26; // 128 -> 127
```

```
104: y = 13; // 129 -> 126
208: y = 197; // 130 -> 125
39: y = 161; // 131 -> 124
78: y = 147; // 132 -> 123
156: y = 138; // 133 -> 122
191: y = 69; // 134 -> 121
249: y = 225; // 135 -> 120
117: y = 179; // 136 -> 119
234: y = 154; // 137 -> 118
83: y = 77; // 138 -> 117
166: y = 229; // 139 -> 116
203: y = 177; // 140 -> 115
17: y = 155; // 141 -> 114
34: y = 142; // 142 -> 113
68: y = 71; // 143 -> 112
136: y = 224; // 144 -> 111
151: y = 112; // 145 -> 110
169: y = 56; // 146 -> 109
213: y = 28; // 147 -> 108
45: y = 14; // 148 -> 107
90: y = 7; // 149 -> 106
180: y = 192; // 150 -> 105
239: y = 96; // 151 -> 104
89: y = 48; // 152 -> 103
178: y = 24; // 153 -> 102
227: y = 12; // 154 -> 101
65: y = 6; // 155 -> 100
130: y = 3; // 156 -> 99
131: y = 194; // 157 -> 98
129: y = 97; // 158 -> 97
133: y = 243; // 159 -> 96
141: y = 186; // 160 -> 95
157: y = 93; // 161 -> 94
189: y = 237; // 162 -> 93
253: y = 181; // 163 -> 92
125: y = 153; // 164 -> 91
250: y = 143; // 165 -> 90
115: y = 132; // 166 -> 89
230: y = 66; // 167 -> 88
75: y = 33; // 168 -> 87
150: y = 211; // 169 -> 86
171: y = 170; // 170 -> 85
209: y = 85; // 171 -> 84
37: y = 233; // 172 -> 83
74: y = 183; // 173 -> 82
148: y = 152; // 174 -> 81
```

```
175: y = 76; // 175 -> 80
217: y = 38; // 176 -> 79
53: y = 19; // 177 -> 78
106: y = 202; // 178 -> 77
212: y = 101; // 179 -> 76
47: y = 241; // 180 -> 75
94: y = 187; // 181 -> 74
188: y = 158; // 182 -> 73
255: y = 79; // 183 -> 72
121: y = 228; // 184 -> 71
242: y = 114; // 185 -> 70
99: y = 57; // 186 -> 69
198: y = 223; // 187 -> 68
11: y = 172; // 188 -> 67
22: y = 86; // 189 -> 66
44: y = 43; // 190 -> 65
88: y = 214; // 191 -> 64
176: y = 107; // 192 -> 63
231: y = 246; // 193 -> 62
73: y = 123; // 194 -> 61
146: y = 254; // 195 -> 60
163: y = 127; // 196 -> 59
193: y = 252; // 197 -> 58
5: y = 126; // 198 -> 57
10: y = 63; // 199 -> 56
20: y = 220; // 200 -> 55
40: y = 110; // 201 -> 54
80: y = 55; // 202 -> 53
160: y = 216; // 203 -> 52
199: y = 108; // 204 -> 51
9: y = 54; // 205 -> 50
18: y = 27; // 206 -> 49
36: y = 206; // 207 -> 48
72: y = 103; // 208 -> 47
144: y = 240; // 209 -> 46
167: y = 120; // 210 -> 45
201: y = 60; // 211 -> 44
21: y = 30; // 212 -> 43
42: y = 15; // 213 -> 42
84: y = 196; // 214 -> 41
168: y = 98; // 215 -> 40
215: y = 49; // 216 -> 39
41: y = 219; // 217 -> 38
82: y = 174; // 218 -> 37
164: y = 87; // 219 -> 36
207: y = 232; // 220 -> 35
```

```
            25: y = 116; // 221 -> 34
            50: y = 58; // 222 -> 33
            100: y = 29; // 223 -> 32
            200: y = 205; // 224 -> 31
            23: y = 165; // 225 -> 30
            46: y = 145; // 226 -> 29
            92: y = 139; // 227 -> 28
            184: y = 134; // 228 -> 27
            247: y = 67; // 229 -> 26
            105: y = 226; // 230 -> 25
            210: y = 113; // 231 -> 24
            35: y = 251; // 232 -> 23
            70: y = 190; // 233 -> 22
            140: y = 95; // 234 -> 21
            159: y = 236; // 235 -> 20
            185: y = 118; // 236 -> 19
            245: y = 59; // 237 -> 18
            109: y = 222; // 238 -> 17
            218: y = 111; // 239 -> 16
            51: y = 244; // 240 -> 15
            102: y = 122; // 241 -> 14
            204: y = 61; // 242 -> 13
            31: y = 221; // 243 -> 12
            62: y = 173; // 244 -> 11
            124: y = 149; // 245 -> 10
            248: y = 137; // 246 -> 9
            119: y = 135; // 247 -> 8
            238: y = 128; // 248 -> 7
            91: y = 64; // 249 -> 6
            182: y = 32; // 250 -> 5
            235: y = 16; // 251 -> 4
            81: y = 8; // 252 -> 3
            162: y = 4; // 253 -> 2
            195: y = 2; // 254 -> 1
      endcase
endmodule
```

sigin.v

```
module sigin(data, address);
	input [7:0] address;
	output [7:0] data;
	reg [7:0] data;

	// an arbitrary data source here
	always @ (address) data = address + 1;
endmodule
```

syndrome.v

```
module rsdec_syn_m0 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[7];
		y[1] = x[0] ^ x[7];
		y[2] = x[1] ^ x[7];
		y[3] = x[2];
		y[4] = x[3];
		y[5] = x[4];
		y[6] = x[5];
		y[7] = x[6] ^ x[7];
	end
endmodule

module rsdec_syn_m1 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[6] ^ x[7];
		y[1] = x[6];
		y[2] = x[0] ^ x[6];
		y[3] = x[1] ^ x[7];
		y[4] = x[2];
		y[5] = x[3];
		y[6] = x[4];
		y[7] = x[5] ^ x[6] ^ x[7];
	end
endmodule

module rsdec_syn_m2 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[5] ^ x[6] ^ x[7];
		y[1] = x[5];
		y[2] = x[5] ^ x[7];
		y[3] = x[0] ^ x[6];
```

```
        y[4] = x[1] ^ x[7];
        y[5] = x[2];
        y[6] = x[3];
        y[7] = x[4] ^ x[5] ^ x[6] ^ x[7];
    end
endmodule

module rsdec_syn_m3 (y, x);
    input [7:0] x;
    output [7:0] y;
    reg [7:0] y;
    always @ (x)
    begin
        y[0] = x[4] ^ x[5] ^ x[6] ^ x[7];
        y[1] = x[4];
        y[2] = x[4] ^ x[6] ^ x[7];
        y[3] = x[5] ^ x[7];
        y[4] = x[0] ^ x[6];
        y[5] = x[1] ^ x[7];
        y[6] = x[2];
        y[7] = x[3] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
    end
endmodule

module rsdec_syn_m4 (y, x);
    input [7:0] x;
    output [7:0] y;
    reg [7:0] y;
    always @ (x)
    begin
        y[0] = x[3] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
        y[1] = x[3];
        y[2] = x[3] ^ x[5] ^ x[6] ^ x[7];
        y[3] = x[4] ^ x[6] ^ x[7];
        y[4] = x[5] ^ x[7];
        y[5] = x[0] ^ x[6];
        y[6] = x[1] ^ x[7];
        y[7] = x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
    end
endmodule

module rsdec_syn_m5 (y, x);
    input [7:0] x;
    output [7:0] y;
    reg [7:0] y;
    always @ (x)
```

```
	begin
		y[0] = x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[1] = x[2];
		y[2] = x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[3] = x[3] ^ x[5] ^ x[6] ^ x[7];
		y[4] = x[4] ^ x[6] ^ x[7];
		y[5] = x[5] ^ x[7];
		y[6] = x[0] ^ x[6];
		y[7] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
	end
endmodule

module rsdec_syn_m6 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[1] = x[1] ^ x[7];
		y[2] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[3] = x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[4] = x[3] ^ x[5] ^ x[6] ^ x[7];
		y[5] = x[4] ^ x[6] ^ x[7];
		y[6] = x[5] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
	end
endmodule

module rsdec_syn_m7 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[1] = x[0] ^ x[6];
		y[2] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[3] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[4] = x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[5] = x[3] ^ x[5] ^ x[6] ^ x[7];
		y[6] = x[4] ^ x[6] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[7];
	end
endmodule
```

```
module rsdec_syn_m8 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[7];
		y[1] = x[5] ^ x[7];
		y[2] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[3] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[4] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[5] = x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[6] = x[3] ^ x[5] ^ x[6] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[6];
	end
endmodule

module rsdec_syn_m9 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[6];
		y[1] = x[4] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[3] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[4] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[5] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[6] = x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[5] ^ x[7];
	end
endmodule

module rsdec_syn_m10 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[5] ^ x[7];
		y[1] = x[3] ^ x[5] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6];
		y[3] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[4] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[5] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
```

```
        y[6] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
        y[7] = x[0] ^ x[1] ^ x[4] ^ x[6];
    end
endmodule

module rsdec_syn_m11 (y, x);
    input [7:0] x;
    output [7:0] y;
    reg [7:0] y;
    always @ (x)
    begin
        y[0] = x[0] ^ x[1] ^ x[4] ^ x[6];
        y[1] = x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
        y[2] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
        y[3] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6];
        y[4] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
        y[5] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
        y[6] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
        y[7] = x[0] ^ x[3] ^ x[5];
    end
endmodule

module rsdec_syn_m12 (y, x);
    input [7:0] x;
    output [7:0] y;
    reg [7:0] y;
    always @ (x)
    begin
        y[0] = x[0] ^ x[3] ^ x[5];
        y[1] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
        y[2] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
        y[3] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
        y[4] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6];
        y[5] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
        y[6] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
        y[7] = x[2] ^ x[4] ^ x[7];
    end
endmodule

module rsdec_syn_m13 (y, x);
    input [7:0] x;
    output [7:0] y;
    reg [7:0] y;
    always @ (x)
    begin
        y[0] = x[2] ^ x[4] ^ x[7];
```

```
		y[1] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[2] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[3] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[4] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[5] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6];
		y[6] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[7] = x[1] ^ x[3] ^ x[6];
	end
endmodule

module rsdec_syn_m14 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[1] ^ x[3] ^ x[6];
		y[1] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[3] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[4] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[5] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[6] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6];
		y[7] = x[0] ^ x[2] ^ x[5] ^ x[7];
	end
endmodule

module rsdec_syn_m15 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[0] ^ x[2] ^ x[5] ^ x[7];
		y[1] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[3] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[4] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[5] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[6] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[7] = x[1] ^ x[4] ^ x[6] ^ x[7];
	end
endmodule

module rsdec_syn_m16 (y, x);
	input [7:0] x;
```

```
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[1] ^ x[4] ^ x[6] ^ x[7];
		y[1] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6];
		y[2] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[3] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[4] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[5] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[6] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[7] = x[0] ^ x[3] ^ x[5] ^ x[6];
	end
endmodule

module rsdec_syn_m17 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[0] ^ x[3] ^ x[5] ^ x[6];
		y[1] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[2] = x[1] ^ x[2] ^ x[3] ^ x[4];
		y[3] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[4] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[5] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[6] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[7] = x[2] ^ x[4] ^ x[5] ^ x[7];
	end
endmodule

module rsdec_syn_m18 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[2] ^ x[4] ^ x[5] ^ x[7];
		y[1] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[2] ^ x[3];
		y[3] = x[1] ^ x[2] ^ x[3] ^ x[4];
		y[4] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[5] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[6] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[7] = x[1] ^ x[3] ^ x[4] ^ x[6];
```

```
	end
endmodule

module rsdec_syn_m19 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[1] ^ x[3] ^ x[4] ^ x[6];
		y[1] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[2] ^ x[7];
		y[3] = x[0] ^ x[1] ^ x[2] ^ x[3];
		y[4] = x[1] ^ x[2] ^ x[3] ^ x[4];
		y[5] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[6] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[7] = x[0] ^ x[2] ^ x[3] ^ x[5] ^ x[7];
	end
endmodule

module rsdec_syn_m20 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[0] ^ x[2] ^ x[3] ^ x[5] ^ x[7];
		y[1] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[6];
		y[3] = x[0] ^ x[1] ^ x[2] ^ x[7];
		y[4] = x[0] ^ x[1] ^ x[2] ^ x[3];
		y[5] = x[1] ^ x[2] ^ x[3] ^ x[4];
		y[6] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[7] = x[1] ^ x[2] ^ x[4] ^ x[6] ^ x[7];
	end
endmodule

module rsdec_syn_m21 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[1] ^ x[2] ^ x[4] ^ x[6] ^ x[7];
		y[1] = x[0] ^ x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[2] = x[0] ^ x[5];
```

```
		y[3] = x[0] ^ x[1] ^ x[6];
		y[4] = x[0] ^ x[1] ^ x[2] ^ x[7];
		y[5] = x[0] ^ x[1] ^ x[2] ^ x[3];
		y[6] = x[1] ^ x[2] ^ x[3] ^ x[4];
		y[7] = x[0] ^ x[1] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
	end
endmodule

module rsdec_syn_m22 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[3] ^ x[5] ^ x[6] ^ x[7];
		y[1] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[2] = x[4] ^ x[7];
		y[3] = x[0] ^ x[5];
		y[4] = x[0] ^ x[1] ^ x[6];
		y[5] = x[0] ^ x[1] ^ x[2] ^ x[7];
		y[6] = x[0] ^ x[1] ^ x[2] ^ x[3];
		y[7] = x[0] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
	end
endmodule

module rsdec_syn_m23 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[0] ^ x[2] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[1] = x[1] ^ x[2] ^ x[3] ^ x[4];
		y[2] = x[3] ^ x[6] ^ x[7];
		y[3] = x[4] ^ x[7];
		y[4] = x[0] ^ x[5];
		y[5] = x[0] ^ x[1] ^ x[6];
		y[6] = x[0] ^ x[1] ^ x[2] ^ x[7];
		y[7] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
	end
endmodule

module rsdec_syn_m24 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
```

```
	always @ (x)
	begin
		y[0] = x[1] ^ x[3] ^ x[4] ^ x[5] ^ x[6] ^ x[7];
		y[1] = x[0] ^ x[1] ^ x[2] ^ x[3];
		y[2] = x[2] ^ x[5] ^ x[6] ^ x[7];
		y[3] = x[3] ^ x[6] ^ x[7];
		y[4] = x[4] ^ x[7];
		y[5] = x[0] ^ x[5];
		y[6] = x[0] ^ x[1] ^ x[6];
		y[7] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
	end
endmodule

module rsdec_syn_m25 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[0] ^ x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[6];
		y[1] = x[0] ^ x[1] ^ x[2] ^ x[7];
		y[2] = x[1] ^ x[4] ^ x[5] ^ x[6];
		y[3] = x[2] ^ x[5] ^ x[6] ^ x[7];
		y[4] = x[3] ^ x[6] ^ x[7];
		y[5] = x[4] ^ x[7];
		y[6] = x[0] ^ x[5];
		y[7] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
	end
endmodule

module rsdec_syn_m26 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[5];
		y[1] = x[0] ^ x[1] ^ x[6];
		y[2] = x[0] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[3] = x[1] ^ x[4] ^ x[5] ^ x[6];
		y[4] = x[2] ^ x[5] ^ x[6] ^ x[7];
		y[5] = x[3] ^ x[6] ^ x[7];
		y[6] = x[4] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[4];
	end
endmodule
```

```
module rsdec_syn_m27 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[4];
		y[1] = x[0] ^ x[5];
		y[2] = x[2] ^ x[3] ^ x[4] ^ x[6];
		y[3] = x[0] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[4] = x[1] ^ x[4] ^ x[5] ^ x[6];
		y[5] = x[2] ^ x[5] ^ x[6] ^ x[7];
		y[6] = x[3] ^ x[6] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[7];
	end
endmodule

module rsdec_syn_m28 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[7];
		y[1] = x[4] ^ x[7];
		y[2] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[7];
		y[3] = x[2] ^ x[3] ^ x[4] ^ x[6];
		y[4] = x[0] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
		y[5] = x[1] ^ x[4] ^ x[5] ^ x[6];
		y[6] = x[2] ^ x[5] ^ x[6] ^ x[7];
		y[7] = x[0] ^ x[1] ^ x[2] ^ x[6];
	end
endmodule

module rsdec_syn_m29 (y, x);
	input [7:0] x;
	output [7:0] y;
	reg [7:0] y;
	always @ (x)
	begin
		y[0] = x[0] ^ x[1] ^ x[2] ^ x[6];
		y[1] = x[3] ^ x[6] ^ x[7];
		y[2] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[6] ^ x[7];
		y[3] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[7];
		y[4] = x[2] ^ x[3] ^ x[4] ^ x[6];
```

```
        y[5] = x[0] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
        y[6] = x[1] ^ x[4] ^ x[5] ^ x[6];
        y[7] = x[0] ^ x[1] ^ x[5] ^ x[7];
    end
endmodule

module rsdec_syn_m30 (y, x);
    input [7:0] x;
    output [7:0] y;
    reg [7:0] y;
    always @ (x)
    begin
        y[0] = x[0] ^ x[1] ^ x[5] ^ x[7];
        y[1] = x[2] ^ x[5] ^ x[6] ^ x[7];
        y[2] = x[0] ^ x[1] ^ x[3] ^ x[5] ^ x[6];
        y[3] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[6] ^ x[7];
        y[4] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[7];
        y[5] = x[2] ^ x[3] ^ x[4] ^ x[6];
        y[6] = x[0] ^ x[3] ^ x[4] ^ x[5] ^ x[7];
        y[7] = x[0] ^ x[4] ^ x[6] ^ x[7];
    end
endmodule

module rsdec_syn_m31 (y, x);
    input [7:0] x;
    output [7:0] y;
    reg [7:0] y;
    always @ (x)
    begin
        y[0] = x[0] ^ x[4] ^ x[6] ^ x[7];
        y[1] = x[1] ^ x[4] ^ x[5] ^ x[6];
        y[2] = x[0] ^ x[2] ^ x[4] ^ x[5];
        y[3] = x[0] ^ x[1] ^ x[3] ^ x[5] ^ x[6];
        y[4] = x[0] ^ x[1] ^ x[2] ^ x[4] ^ x[6] ^ x[7];
        y[5] = x[1] ^ x[2] ^ x[3] ^ x[5] ^ x[7];
        y[6] = x[2] ^ x[3] ^ x[4] ^ x[6];
        y[7] = x[3] ^ x[5] ^ x[6];
    end
endmodule

module rsdec_syn (y0, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14, y15, y16, y17,
y18, y19, y20, y21, y22, y23, y24, y25, y26, y27, y28, y29, y30, y31, u, enable, shift, init, clk,
rst_n);
    input [7:0] u;
    input clk, rst_n, shift, init, enable;
    output [7:0] y0;
```

```
output [7:0] y1;
output [7:0] y2;
output [7:0] y3;
output [7:0] y4;
output [7:0] y5;
output [7:0] y6;
output [7:0] y7;
output [7:0] y8;
output [7:0] y9;
output [7:0] y10;
output [7:0] y11;
output [7:0] y12;
output [7:0] y13;
output [7:0] y14;
output [7:0] y15;
output [7:0] y16;
output [7:0] y17;
output [7:0] y18;
output [7:0] y19;
output [7:0] y20;
output [7:0] y21;
output [7:0] y22;
output [7:0] y23;
output [7:0] y24;
output [7:0] y25;
output [7:0] y26;
output [7:0] y27;
output [7:0] y28;
output [7:0] y29;
output [7:0] y30;
output [7:0] y31;
reg [7:0] y0;
reg [7:0] y1;
reg [7:0] y2;
reg [7:0] y3;
reg [7:0] y4;
reg [7:0] y5;
reg [7:0] y6;
reg [7:0] y7;
reg [7:0] y8;
reg [7:0] y9;
reg [7:0] y10;
reg [7:0] y11;
reg [7:0] y12;
reg [7:0] y13;
reg [7:0] y14;
```

```
reg [7:0] y15;
reg [7:0] y16;
reg [7:0] y17;
reg [7:0] y18;
reg [7:0] y19;
reg [7:0] y20;
reg [7:0] y21;
reg [7:0] y22;
reg [7:0] y23;
reg [7:0] y24;
reg [7:0] y25;
reg [7:0] y26;
reg [7:0] y27;
reg [7:0] y28;
reg [7:0] y29;
reg [7:0] y30;
reg [7:0] y31;

wire [7:0] scale0;
wire [7:0] scale1;
wire [7:0] scale2;
wire [7:0] scale3;
wire [7:0] scale4;
wire [7:0] scale5;
wire [7:0] scale6;
wire [7:0] scale7;
wire [7:0] scale8;
wire [7:0] scale9;
wire [7:0] scale10;
wire [7:0] scale11;
wire [7:0] scale12;
wire [7:0] scale13;
wire [7:0] scale14;
wire [7:0] scale15;
wire [7:0] scale16;
wire [7:0] scale17;
wire [7:0] scale18;
wire [7:0] scale19;
wire [7:0] scale20;
wire [7:0] scale21;
wire [7:0] scale22;
wire [7:0] scale23;
wire [7:0] scale24;
wire [7:0] scale25;
wire [7:0] scale26;
wire [7:0] scale27;
```

```
wire [7:0] scale28;
wire [7:0] scale29;
wire [7:0] scale30;
wire [7:0] scale31;

rsdec_syn_m0   rsdec_syn_m0_inst (scale0,   y0);
rsdec_syn_m1   rsdec_syn_m1_inst (scale1,   y1);
rsdec_syn_m2   rsdec_syn_m2_inst (scale2,   y2);
rsdec_syn_m3   rsdec_syn_m3_inst (scale3,   y3);
rsdec_syn_m4   rsdec_syn_m4_inst (scale4,   y4);
rsdec_syn_m5   rsdec_syn_m5_inst (scale5,   y5);
rsdec_syn_m6   rsdec_syn_m6_inst (scale6,   y6);
rsdec_syn_m7   rsdec_syn_m7_inst (scale7,   y7);
rsdec_syn_m8   rsdec_syn_m8_inst (scale8,   y8);
rsdec_syn_m9   rsdec_syn_m9_inst (scale9,   y9);
rsdec_syn_m10 rsdec_syn_m10_inst (scale10, y10);
rsdec_syn_m11 rsdec_syn_m11_inst (scale11, y11);
rsdec_syn_m12 rsdec_syn_m12_inst (scale12, y12);
rsdec_syn_m13 rsdec_syn_m13_inst (scale13, y13);
rsdec_syn_m14 rsdec_syn_m14_inst (scale14, y14);
rsdec_syn_m15 rsdec_syn_m15_inst (scale15, y15);
rsdec_syn_m16 rsdec_syn_m16_inst (scale16, y16);
rsdec_syn_m17 rsdec_syn_m17_inst (scale17, y17);
rsdec_syn_m18 rsdec_syn_m18_inst (scale18, y18);
rsdec_syn_m19 rsdec_syn_m19_inst (scale19, y19);
rsdec_syn_m20 rsdec_syn_m20_inst (scale20, y20);
rsdec_syn_m21 rsdec_syn_m21_inst (scale21, y21);
rsdec_syn_m22 rsdec_syn_m22_inst (scale22, y22);
rsdec_syn_m23 rsdec_syn_m23_inst (scale23, y23);
rsdec_syn_m24 rsdec_syn_m24_inst (scale24, y24);
rsdec_syn_m25 rsdec_syn_m25_inst (scale25, y25);
rsdec_syn_m26 rsdec_syn_m26_inst (scale26, y26);
rsdec_syn_m27 rsdec_syn_m27_inst (scale27, y27);
rsdec_syn_m28 rsdec_syn_m28_inst (scale28, y28);
rsdec_syn_m29 rsdec_syn_m29_inst (scale29, y29);
rsdec_syn_m30 rsdec_syn_m30_inst (scale30, y30);
rsdec_syn_m31 rsdec_syn_m31_inst (scale31, y31);

always @ (posedge clk or negedge rst_n)
begin
	if (~rst_n)
	begin
		y0 <= 0;
		y1 <= 0;
		y2 <= 0;
		y3 <= 0;
```

```
		y4 <= 0;
		y5 <= 0;
		y6 <= 0;
		y7 <= 0;
		y8 <= 0;
		y9 <= 0;
		y10 <= 0;
		y11 <= 0;
		y12 <= 0;
		y13 <= 0;
		y14 <= 0;
		y15 <= 0;
		y16 <= 0;
		y17 <= 0;
		y18 <= 0;
		y19 <= 0;
		y20 <= 0;
		y21 <= 0;
		y22 <= 0;
		y23 <= 0;
		y24 <= 0;
		y25 <= 0;
		y26 <= 0;
		y27 <= 0;
		y28 <= 0;
		y29 <= 0;
		y30 <= 0;
		y31 <= 0;
	end
	else if (init)
	begin
		y0 <= u;
		y1 <= u;
		y2 <= u;
		y3 <= u;
		y4 <= u;
		y5 <= u;
		y6 <= u;
		y7 <= u;
		y8 <= u;
		y9 <= u;
		y10 <= u;
		y11 <= u;
		y12 <= u;
		y13 <= u;
		y14 <= u;
```

```
		y15 <= u;
		y16 <= u;
		y17 <= u;
		y18 <= u;
		y19 <= u;
		y20 <= u;
		y21 <= u;
		y22 <= u;
		y23 <= u;
		y24 <= u;
		y25 <= u;
		y26 <= u;
		y27 <= u;
		y28 <= u;
		y29 <= u;
		y30 <= u;
		y31 <= u;
	end
	else if (enable)
	begin
		y0 <= scale0 ^ u;
		y1 <= scale1 ^ u;
		y2 <= scale2 ^ u;
		y3 <= scale3 ^ u;
		y4 <= scale4 ^ u;
		y5 <= scale5 ^ u;
		y6 <= scale6 ^ u;
		y7 <= scale7 ^ u;
		y8 <= scale8 ^ u;
		y9 <= scale9 ^ u;
		y10 <= scale10 ^ u;
		y11 <= scale11 ^ u;
		y12 <= scale12 ^ u;
		y13 <= scale13 ^ u;
		y14 <= scale14 ^ u;
		y15 <= scale15 ^ u;
		y16 <= scale16 ^ u;
		y17 <= scale17 ^ u;
		y18 <= scale18 ^ u;
		y19 <= scale19 ^ u;
		y20 <= scale20 ^ u;
		y21 <= scale21 ^ u;
		y22 <= scale22 ^ u;
		y23 <= scale23 ^ u;
		y24 <= scale24 ^ u;
		y25 <= scale25 ^ u;
```

```
                y26 <= scale26 ^ u;
                y27 <= scale27 ^ u;
                y28 <= scale28 ^ u;
                y29 <= scale29 ^ u;
                y30 <= scale30 ^ u;
                y31 <= scale31 ^ u;
            end
            else if (shift)
            begin
                y0 <= y1;
                y1 <= y2;
                y2 <= y3;
                y3 <= y4;
                y4 <= y5;
                y5 <= y6;
                y6 <= y7;
                y7 <= y8;
                y8 <= y9;
                y9 <= y10;
                y10 <= y11;
                y11 <= y12;
                y12 <= y13;
                y13 <= y14;
                y14 <= y15;
                y15 <= y16;
                y16 <= y17;
                y17 <= y18;
                y18 <= y19;
                y19 <= y20;
                y20 <= y21;
                y21 <= y22;
                y22 <= y23;
                y23 <= y24;
                y24 <= y25;
                y25 <= y26;
                y26 <= y27;
                y27 <= y28;
                y28 <= y29;
                y29 <= y30;
                y30 <= y31;
                y31 <= y0;
            end
        end

endmodule
```

test_bench.v

```
module rsdec(x, error, with_error, enable, valid, k, clk, rst_n);
	input enable, clk, rst_n;
	input [7:0] k, x;
	output [7:0] error;
	wire [7:0] error;
	output with_error, valid;
	reg with_error, valid;

	wire [7:0] s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, s14, s15, s16, s17, s18,
	s19, s20, s21, s22, s23, s24, s25, s26, s27, s28, s29, s30, s31;
	wire [7:0] lambda, omega, alpha;
	reg [5:0] count;
	reg [32:0] phase;
	wire [7:0] D0, D1, DI;
	reg [7:0] D, D2;
	reg [7:0] u, length0, length1, length2, length3;
	reg syn_enable, syn_init, syn_shift, berl_enable;
	reg chien_search, chien_load, shorten;

	always @ (chien_search or shorten)
	valid = chien_search & ~shorten;

	rsdec_syn rsdec_syn_inst (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, s14, s15,
	s16, s17, s18, s19, s20, s21, s22, s23, s24, s25, s26, s27, s28, s29, s30, s31,
	u, syn_enable, syn_shift&phase[0], syn_init, clk, rst_n);
	rsdec_berl rsdec_berl_inst (lambda, omega,
	s0, s31, s30, s29, s28, s27, s26, s25, s24, s23, s22, s21, s20, s19, s18, s17, s16, s15, s14,
	s13, s12, s11, s10, s9, s8, s7, s6, s5, s4, s3, s2, s1,
	D0, D2, count, phase[0], phase[32], berl_enable, clk, rst_n);
	rsdec_chien rsdec_chien_inst (error, alpha, lambda, omega,
	D1, DI, chien_search, chien_load, shorten, clk, rst_n);
	inverse inverse_inst (DI, D);

	always @ (posedge clk or negedge rst_n)
	begin
		if (~rst_n)
		begin
			syn_enable <= 0;
			syn_shift <= 0;
			berl_enable <= 0;
			chien_search <= 1;
			chien_load <= 0;
			length0 <= 0;
			length2 <= 255 - k;
```

```
		count <= -1;
		phase <= 1;
		u <= 0;
		shorten <= 1;
		syn_init <= 0;
	end
	else
	begin
		if (enable & ~syn_enable & ~syn_shift)
		begin
			syn_enable <= 1;
			syn_init <= 1;
		end
		if (syn_enable)
		begin
			length0 <= length1;
			syn_init <= 0;
			if (length1 == k)
			begin
				syn_enable <= 0;
				syn_shift <= 1;
				berl_enable <= 1;
			end
		end
		if (berl_enable & with_error)
		begin
			if (phase[0])
			begin
				count <= count + 1;
				if (count == 31)
				begin
					syn_shift <= 0;
					length0 <= 0;
					chien_load <= 1;
					length2 <= length0;
				end
			end
			phase <= {phase[31:0], phase[32]};
		end
		if (berl_enable & ~with_error)
			if (&count)
			begin
				syn_shift <= 0;
				length0 <= 0;
				berl_enable <= 0;
			end
```

```
                    else
                        phase <= {phase[31:0], phase[32]};
                if (chien_load & phase[32])
                begin
                    berl_enable <= 0;
                    chien_load <= 0;
                    chien_search <= 1;
                    count <= -1;
                    phase <= 1;
                end
                if (chien_search)
                begin
                    length2 <= length3;
                    if (length3 == 0)
                        chien_search <= 0;
                end
            if (enable) u <= x;
            if (shorten == 1 && length2 == 0)
                shorten <= 0;
            end

        end

        always @ (chien_search or D0 or D1)
            if (chien_search) D = D1;
            else D = D0;

        always @ (DI or alpha or chien_load)
            if (chien_load) D2 = alpha;
            else D2 = DI;

        always @ (length0) length1 = length0 + 1;
        always @ (length2) length3 = length2 - 1;
always @ (syn_shift or s0 or s1 or s2 or s3 or s4 or s5 or s6 or s7 or s8 or s9 or s10 or s11 or s12
or s13 or s14 or s15 or s16 or s17 or s18 or s19 or s20 or s21 or s22 or s23 or s24 or s25 or s26 or
s27 or s28 or s29 or s30 or s31)
if (syn_shift && (s0 | s1 | s2 | s3 | s4 | s5 | s6 | s7 | s8 | s9 | s10 | s11 | s12 | s13 | s14 | s15 | s16 |
s17 | s18 | s19 | s20 | s21 | s22 | s23 | s24 | s25 | s26 | s27 | s28 | s29 | s30 | s31)!= 0)
                with_error = 1;
            else with_error = 0;

endmodule
```