Implementation of Reed-Solomon Encoder/Decoder Using Field Programmable Gate Array

Dr. Hikmat N. Abdullah
Electrical Engineering Dept., College of Engineering
Al-Mustansiriya University, Baghdad, Iraq

Abstract

In this paper, (15, 11) and (255, 239) Reed-Solomon codes have been designed and implemented using ALTERA Field Programmable Gate Array (FPGA) device. The design is carried out by writing VHDL modules for different encoder and decoder components. The waveforms are tested using the package MODEL-SIM 5.4a. While synthesis reports and board programming file are obtained using the package QUARTUS II.

ALTERA-FLEX10K10 FPGA board is used as a target device for the designed Reed-Solomon encoder/decoder.

Simulation waveforms show that (15, 11) and (255,239) Reed-Solomon decoders could correct up to 2 and 8 erroneous symbols respectively.

الخلاصة

في هذا البحث، تم تصميم وبناء شفرتي Reed-Solomon بطول (15,11) و (255,239) باستخدام تقنية VHDL مصفوفة البوابات المبرمجة الواسعة (FPGA) نوع ALTERA. وتم إنجاز هذا التصميم بكتابة نماذج بلغة VHDL ل المختلفة مكونات المشفر وحلال الشفرة. ولقد تم استخدام البرنامج MODEL-SIM 5.4a للحصول على الأشكال الموجية للإشارات في التصميم بينما تم استخدام البرنامج QUARTUS II للحصول على تقارير البناء وملفات البرمجة.

وتم اختيار اللوح FLEX10K10 FPGA لغرض البناء الحقيقي لدائرتي المشفر وحلال الشفرة. نتائج المحاكاة بحث أن دائرة حلال الشفرة ذات الطول (15,11) قادرة على تصحيح حد أقصى من الأخطاء مقداره خطاً، في حين أن الدائرة ذات الطول (255,239) قادرة على تصحيح حد أقصى من الأخطاء مقداره ثمانية أخطاء.
1. Introduction

Many digital signaling applications in broadcasting use forward error correction, a technique in which redundant information is added to the signal to allow the receiver to detect and correct errors that may have occurred in transmission. Many different types of code have been devised for this purpose, but Reed-Solomon codes [1] have proved to be a good compromise between efficiency and complexity. A particularly important use of a Reed-Solomon code for television applications is in the DVB-T transmission standard [2].

Hitherto, modulators and demodulators for DVB-T have in general used custom chips to provide the Reed-Solomon encoding and decoding functions. However, there are circumstances (such as for digital radio cameras) where it would be beneficial to include these processes in gate array designs for the transmitter and receiver. This would then provide the flexibility to modify the encoding parameters to suit the particular requirements of the radio camera application without sacrificing the compactness of a single-chip implementation. Although custom core design for gate arrays are available, the charges are significant and can complicate further exploitation of the intellectual property embodied in a design [3].

In this paper, (15,11) and (255,239) Reed-Solomon encoders/decoders have been implemented using ALTERA-FLEX10K10 FPGA. QUARTUS II package is used to perform VHDL models of different units of encoder and decoder and obtain synthesis reports while Model-Sim 5.4a package is used to obtain simulation waveforms.

2. FPGA Implementation of Reed-Solomon Encoder

(15,11) and (255,239) Reed-Solomon codes has been implemented using ALTERA FPGA. These two codes could correct up to 2 and 8 errors respectively. The word length in each symbol is 4 and 8 bits respectively and the codes are based on Galios fields with 16 and 256 elements respectively. The code generator polynomial for the first code is:

\[ g(x) = x^4 + 15x^3 + 3x^2 + x + 12 \] …………………………………………………… (1)

while for the second one is:

\[ g(x) = x^{16} + 59x^{15} + 13x^{14} + 104x^{13} + 189x^{12} + 248x^{11} + 209x^{10} + 30x^9 + 8x^8 + 163x^7 + 65x^6 + 229x^5 + 98x^4 + 98x^3 + 98x^2 + 36x + 59 \] …………………… (2)

The structure of implemented (15,11) Reed-Solomon encoder is shown in Fig.1. All data paths in figure provide for 4 bit values. During the message input period, the selector passes the input values directly to the output and AND gate is enabled. After the eleven calculation steps have been completed the remainder is contained in the D-type registers. The control waveform then changes so that the AND gate prevents further feedback to the multipliers and the four remainder symbol values are clocked out of the registers and routed to
the output by the selector. For (255,239) encoder, the number of stages would be 16 with multipliers coefficients obtained from the generator polynomial corresponding to the code (eq.2) while data paths provide for 8 bit values.

![Diagram of (15,11) Reed-Solomon encoder]

**Figure (1) (15,11) Reed-Solomon encoder**

VHDL modules were written for each element in the encoder for FPGA implementation purpose and a main VHDL program “encoder.vhd” that organize the operations of all these modules according to the hierarchy structure shown below:

```
Encoder.vhd
  ├── Genmult.vhd
  │    ├── Adder.vhd
  │    └── Mux.vhd
  └── D_ff.vhd
```

### 3. FPGA Implementation of Reed Solomon Decoder

**Figure (2)** shows the block diagram of the implemented Reed-Solomon Decoder. In this figure, the first process is to calculate the syndrome values from the incoming codeword. These are then used to find the coefficients of the error locator polynomial $\Lambda_1 \ldots \Lambda_v$ using Berlikamp algorithm. The error locations are identified by the Chien search and the error values are calculated using Forney’s method. As these calculations involve all the symbols of the received codeword, it is necessary to store the message until the results of the calculations are available. Then, to correct the errors, each error values are added (modulo-2) to the appropriate location in the received codeword.
The tree of VHDL programs that is written to implement the decoder is shown below:

![VHDL Tree Diagram]

### 3-1 Syndrome Calculation

The syndrome values are computed as follows: first dividing the received polynomial by each of the factors \((x+\alpha^i)\) of generated polynomial resulting a quotient \(Q_i(x)\) and a remainder that is:

\[
\frac{R(x)}{x+\alpha^i} = Q_i(x) + S_i/(x+\alpha^i) \quad \text{for} \quad 0<i<2^t+1 \quad \text{................................. (3)}
\]

where: \(t\) is the number of error codewords that could be corrected.

Rearranging eq.(3) produces:

\[
S_i = Q_i(x) \times (x+\alpha^i) + R(x) \quad \text{........................................... (4)}
\]

Then substituting \(x=\alpha^i\) in eq.(4), so that:
\( S_i = R(\alpha_i) = R_{n-1}(\alpha_i)^{n-1} + R_{n-2}(\alpha_i)^{n-2} + \ldots + R_1\alpha_i + R_0 \) .................................. (5)

where: the coefficient \( R_{n-1} \ldots R_0 \) are the symbols of the received codeword.

This means that each of the syndrome values can also obtained by substituting \( x = \alpha_i \) in the received polynomial. Figure (3) shows the hardware arrangement used for syndrome calculation.

![Figure (3) Hardware of syndrome calculation](image)

**3-2 Berlikamp Algorithm**

Berlikamp algorithm is one of the more efficient iterative techniques for finding the coefficients of error locator polynomial \( \Lambda_1 \ldots \Lambda_v \) in the following equation:

\[
S_{i+v} + \Lambda_1 S_{i+v-1} + \ldots + \Lambda_v S_i = 0 \quad \text{for } i = 0, \ldots, 2t-v-1 \quad \text{......................... (6)}
\]

and the coefficients of error magnitude polynomial \( \Omega_0 \ldots \Omega_0 \) in the following equation:

\[
\Omega(x) = [S(x)Q(x)] \mod x^{2t} \quad \text{................................................................. (7)}
\]

This is done by forming an approximation to the error locator polynomial, starting with \( \Lambda(x)=1 \), then at each stage, an error value is formed by substituting the approximate coefficients into equations corresponding to the value of \( v \). The error is then used to reduce to refine a correction polynomial, which is then added to improve the approximation \( \Lambda(x) \). The process ends when the approximate error locator polynomial checks consistently with the remaining equations. As soon as the coefficients of error locator polynomial is obtained, the coefficients of the error magnitude polynomial could be computed from eq.(7) in which any terms of degree \( x^{2t} \) or higher in the product are ignored. More details about this algorithm could be found in [4].

**3-3 Calculation of Error Values: Forney Method**

Forney’s algorithm is an efficient method to compute error values \( Z_1 \ldots Z_v \). With this algorithm, the error value is computed using:
\[ Z_j = \frac{Q(x_j-1)}{\Lambda'(x_j-1)} \]  \hspace{1cm} (8)

where: \( \Lambda'(x_j-1) \) is the derivative of \( \Lambda(x) \) for \( x=x_j-1 \) equation only gives valid results for symbol positions containing an error. If the calculation is made at other positions, the result is generally non-zero and invalid.

The Chien search is therefore is needed to identify the error position. Figure (4) shows the hardware of the error value calculation. In this figure, \( \gamma \) is a common constant for all \( \Lambda_i \) and \( \Omega_i \) coefficients.

![Figure (4) Hardware of error value calculation](image)

**3-4 Chien Search for Error Positions**

It is a simple operation at which different element values of Galios Field are substituted successively in \( \Lambda(x) \) equation. The substitution value that produces a value of zero in \( \Lambda(x) \) would identify the error positions. Figure (5) shows the hardware of Chien search.

![Figure (5) Hardware of Chien search](image)
4. Top Level Design

Using QUARTUS II software package provided by ALTERA [5], VHDL design modules are written for each encoder/decoder and error source (channel) components. At the top level of design, a schematic file is created to specify the main entities of the design and to assign input and output pins as it looks like in the ALTERA-FPGA chip. **Figure (6)** shows the top level design file for the (255,239) Reed-Solomon code. In this figure, the input pin clock is used to provide a clock to different components. For this purpose, a special module is used to make necessary scaling. The reset pin is used for resetting purpose. The pin control is used to control the generation of coded message as mentioned in section II. The pins data [7..0] are located to apply real input data from external data source while pins E [7..0] are used to receive error pattern for test purpose. Finally the pins y [7..0] are used to produce the output data from decoder after correcting errors. For actual application, the channel entity is omitted.

For (15,11) Reed-Solomon code, the structure of the top-level design is the same as for (255,239) code shown in **Fig.(6)** with the exception in that the width of data paths everywhere is 4 bits instead of 8 bits.

![Figure (6) Top level design file of reed-solomon encoder/decoder](image)

5. Implementation Results

5-1 Simulation Results

The simulation waveforms for (15,11) and (255,239) Red-Solomon encoder and decoder are carried out using Model-Sim 5.4a package. **Figure (7)** shows waveforms for (15,11) code. The symbols of the coded message are represented as a set of 4 bits displayed in decimal form. In this figure, the signal ‘tx’ is the coded message produced by the encoder while the signal ‘r’ is the received coded message in which two errors in the sixth and thirteenth
symbols are introduced using the error signal ‘e’. The signal ‘y’ denotes the received coded message after correction. As it is clear from this figure, the two errors are corrected successfully by the decoder. A small delay is noticed between the received and corrected message. This is due to the required time to calculate and store syndrome values and other intermediate results until producing the final error correction symbols.

Figure (7) Simulation waveforms for (15,11) reed-solomon code

Figure (8) shows waveforms for (255,239) code. The signal names have the same refer as in the previous figure. Here, the symbols of the coded message are represented as a set of 8 bits and also read in decimal form. The figure shows a segment of the coded message produced by the encoder ‘tx’ (from symbol 233 to symbol 246) since it is not possible to display all symbols of coded message in one screen view. The received coded message ‘r’ is infected by a burst of 8 successive errors (from symbol 234 to symbol 241). These errors are corrected successfully as it is shown by the output signal of the decoder ‘y’.

Figure (8) Simulation waveforms for (255,239) reed-solomon code
5-2 Synthesis Reports

There is a large number of synthesis reports (hardware and software reports) obtained from synthesis operation. They describe all what concern the implementation process like storage resources required, I/O resources required, computation resources required, time delay at different points inside the chip …etc. [6]. Table (1) shows the summary of hardware synthesis reports for (15, 11) Implemented Reed-Solomon encoder/decoder.

Table (1) The summary of hardware synthesis reports for (15, 11) Implemented Reed-Solomon encoder/decoder

<table>
<thead>
<tr>
<th>Resource</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOs</td>
<td>16</td>
<td>160</td>
<td>10%</td>
</tr>
<tr>
<td>FG Function Generators</td>
<td>603</td>
<td>2592</td>
<td>23.5%</td>
</tr>
<tr>
<td>H Function Generators</td>
<td>289</td>
<td>1296</td>
<td>22.3%</td>
</tr>
<tr>
<td>CLB Flip-Flops</td>
<td>708</td>
<td>2592</td>
<td>27.3%</td>
</tr>
</tbody>
</table>

*Maximum period 6.05ns (Maximum frequency: 165.3 MSPS).*
*Maximum path delay from the any node: 6.05ns.*

While Table (2) shows the summary of hardware synthesis reports for (255,239) Implemented Reed-Solomon encoder/decoder.

Table (2) The summary of hardware synthesis reports for (255, 239) Implemented Reed-Solomon encoder/decoder

<table>
<thead>
<tr>
<th>Resource</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOs</td>
<td>28</td>
<td>160</td>
<td>17.5%</td>
</tr>
<tr>
<td>FG Function Generators</td>
<td>2327</td>
<td>2592</td>
<td>89.7%</td>
</tr>
<tr>
<td>H Function Generators</td>
<td>949</td>
<td>1296</td>
<td>73.2%</td>
</tr>
<tr>
<td>CLB Flip-Flops</td>
<td>1971</td>
<td>2592</td>
<td>76.1%</td>
</tr>
</tbody>
</table>

*Maximum period 17.21ns (Maximum frequency: 58.1 MSPS).*
*Maximum path delay from the any node: 17.23ns.*

It is seen from these two reports summary that the FPGA resources requirements increases as the length of Reed-Solomon code increases and as a result the maximum possible operating speed decreases.
5-3 Implementation

After obtaining a correct simulation results for the written VHDL modules, both (15, 11) and (255,239) Reed-Solomon encoder and decoder has been implemented by generating a program files via QUARTUS II package corresponding to each implementation case. These program files are then downloaded to the FPGA ALTERA-FLEX10K10 board whose picture shown in Fig.(9). The data are applied to this development board internally by writing a VHDL module that produces such an input data. The coded messages are recognized by the aid of the seven segment display provided in the board. This operation should be done at a low data rates in order to recognize the values of coded messages symbols that’s displayed in hexadecimal form.

![Figure (9) ALTERA-FLEX device family FLEX10K development board](image)

6. Conclusions

Reed-Solomon codes are block-based error correcting codes with a wide range of applications in digital communications and storage. The usage of FPGA Technology to implement these codes provides many advantages like efficient reconfigurability and universal chip implementation. The design procedure using FPGA Technology is done by writing a hardware description programs (using language like VHDL) to each element in the system and a main program to control the influence of signals in the system. A schematic top-level design is then required to specify pins needed for real hardware interfacing. After correct compilation, obtaining timing analysis, synthesis reports and simulation, a programming file is finally downloaded to the FPGA board.
7. References


