Laser Satellite Coding

By

Mr. Porntep Thangletmatha
Mr. Gunn Chinthammit

Final Report of the Three-Credits Course
TS 4999 Senior Project

Assumption University
April, 2000
Laser Satellite coding

Final Report of Senior Project

TS 4999

STUDENT NAMES:

1. Mr. Porntep Thangletmatha ID: 3915211
2. Mr. Gunn Chinthammit ID: 3915605

Submitted in Partial Fulfillment of the Requirements for

The Degree of Bachelor of Science (Telecommunication Science)

Assumption university
April 2000

PROJECT ADVISOR:

Dr. Dobri Atanassov Batovski

PROJECT COMMITTEE:

1. A. Thannachai Thumthawatworn
2. A. Chatmakut Ratanamook
Faculty of Science and Technology  
Assumption University  

Project Title : Laser Satellite Coding  

Student Names : Mr. Porntep Thangletmatha ID: 3915211  
Mr. Gunn Chinthammit ID: 3915605  

Project Advisor : Dr. Dobri Atanassov Batovski  

Academic Year : 1999  

The Department of Telecommunications Science, Faculty of Science and Technology, Assumption University had approved this final report of the Three-credit course, TS 4999 : Senior Project, submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Telecommunications Science.

Approval by :  

(Dr. Dobri Atanassov Batovski)  
Project Advisor  

(A. Thannachai Thumthawatworn)  
Committee Member  

(A. Chatmakut Ratanamook)  
Committee Member
Acknowledgement

It is a great pressure for all of members to acknowledge the help of many people for the completion of this project. First and foremost, we owe a great deal to our advisor, Dr. Dobri Atanassov Batovski for valuable guidance, error control coding source codes, and a lot of ideas for this project and motivation.

Second, we would like to thank our committee members, A.Chatmakut Ratanamook and A.Thannachai Thumthawatworn for their suggestions, some useful error control coding books and beneficial recommendations in our project and final report.

Finally, we also would like to thank all of instructors in Telecommunication Science Department who gave us a lot of knowledge for doing this project. And special thank due to our friends who gave us some ideas.

Mr. Porntep Thangletmatha
Mr. Gunn Chinthammit
Abstract

Laser communication systems offer many advantages over radio frequency (RF) systems. Before proceeding to the task of understanding the technologies and the methodology for designing a laser communication system, it is worthwhile to initially explore these advantages, many of which relate directly to their architecture and applications.

The Laser Satellite system is the transfer of data from one satellite to its companion. In most cases, established classical modulation schemes are used. Transfer is accommodated using a variety of noncoherent and coherent schemes. And in the transmission, we need the security for the reliability in the system. Therefore there need to have the coding technique to support our satisfactory. But in the process of encoding and decoding, there might be occurred the error. That is the major factor to find some technique to correct the error. About our error correction schemes are Reed-Solomon code. So, this project is to make the software tool to simulate the error control coding and decoding for this Laser Satellite system in order to study the efficiency and effectiveness of these error correction codes.

For the studying the efficiency and effectiveness of these error correction code, we can process them by make the software tool to simulate the system for correcting the coding in the Laser Satellite system. Then considering or evaluating the result of the correction whether it can proceed. That will indicate the software tool is efficient or not.
Motivation

Laser communication systems offer many advantages over radio frequency (RF) systems. It is worthwhile to initially explore these advantages, many of which relate directly to the architecture and applications.

Many future applications exist. Relays from low Earth orbit (LEO) to geosynchronous Earth orbit (GEO) satellites, LEO networks with point-to-point connectivity, LEO to ground, GEO to ground, LEO and GEO to aircraft, and aircraft to aircraft links are all possible in the near-Earth realm. Satellite to submarine communications is technically feasible and future operational needs may require its development. Deep space development will be enhanced by the development of large Earth orbiting and Earth-based receivers and will enable high-data-rate transfer from planetary mission spacecraft.

There are the benefit or the good reason to find some techniques to implement the security of the transmission of data in the Laser Satellite Communication system. Especially, the coding system is the major source of error in transmission. Therefore we need the technique to correct the error in the coding or we call it that “error control coding”. We proud to present some technique of the error control coding by using the Reed-Solomon code scheme. We study in the scheme of Reed-Solomon code and then to create the software tool for simulation the process of error control coding.

After we simulate the process of error control coding, we will evaluate the result of controlling for the effectiveness of this error correction code. Finally, we hope that this report will be useful for who want to advance the studying. That we will appropriate very much.
## Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BACKGROUND</strong></td>
<td>1</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2. Why Laser Communication</td>
<td>2</td>
</tr>
<tr>
<td>3. Laser Communication Application Overview</td>
<td>3</td>
</tr>
<tr>
<td>3.1. Satellite Crosslinks</td>
<td>4</td>
</tr>
<tr>
<td>3.2. Satellite to Aircraft</td>
<td>5</td>
</tr>
<tr>
<td>3.3. Satellite to Ground</td>
<td>6</td>
</tr>
<tr>
<td>3.4. Satellite to Submarine</td>
<td>7</td>
</tr>
<tr>
<td>4. Satellite Beam Acquisition, Tracking, and Pointing</td>
<td>8</td>
</tr>
<tr>
<td>5. Communication Techniques</td>
<td>11</td>
</tr>
<tr>
<td>5.1. Direct-Detection Digital Baseband Signaling</td>
<td>12</td>
</tr>
<tr>
<td>5.2. Direct-Detection Analog Signaling</td>
<td>12</td>
</tr>
<tr>
<td>5.3. Coherent-Modulation Signaling</td>
<td>13</td>
</tr>
<tr>
<td>6. Data Transfer</td>
<td>15</td>
</tr>
<tr>
<td>7. Satellite Error Control Coding</td>
<td>17</td>
</tr>
<tr>
<td>7.1. Rationale for Coding and Types of Code</td>
<td>17</td>
</tr>
<tr>
<td>7.1.1. Bose-Chaudhuri-Hocquenghem (BCH) Codes</td>
<td>20</td>
</tr>
<tr>
<td>7.1.2. Reed-Solomon Codes</td>
<td>21</td>
</tr>
<tr>
<td>8. Reed-Solomon Codes</td>
<td>22</td>
</tr>
<tr>
<td>8.1. Properties of Reed-Solomon Codes</td>
<td>23</td>
</tr>
<tr>
<td>8.1.1. Symbol Errors</td>
<td>24</td>
</tr>
<tr>
<td>8.1.2. Decoding</td>
<td>24</td>
</tr>
<tr>
<td>8.1.3. Coding Gain</td>
<td>25</td>
</tr>
<tr>
<td>8.2. Architecture for Encoding &amp; Decoding of Reed-Solomon Codes</td>
<td>25</td>
</tr>
<tr>
<td>8.2.1. Encoder Architecture</td>
<td>25</td>
</tr>
<tr>
<td>8.2.2. Decoder Architecture</td>
<td>27</td>
</tr>
<tr>
<td>8.3 Implementation of Reed-Solomon Encoder &amp; Decoder</td>
<td>28</td>
</tr>
<tr>
<td>8.3.1. Hardware Implementation</td>
<td>28</td>
</tr>
<tr>
<td>8.3.2. Software Implementation</td>
<td>29</td>
</tr>
</tbody>
</table>
OBJECTIVE
DESCRIPTION AND SCOPE OF PROJECT
PROJECT DESIGN
PROJECT IMPLEMENTATION
EXPERIMENT AND ANALYSIS
CONCLUSIONS
RECOMMENDATION
REFERENCES
List of Tables

Table 1: Binary Modulation Techniques 12
Table 2: Rate One Half Reed-Solomon Codes 25
Table 3: List of parameters in Decoder architecture 28
Table 4: Data rate for decoding 30
Table 5: Successive content of register#1 of FIGURE 16 33
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIGURE 1</td>
<td>Example of Laser Satellite Communication</td>
<td>2</td>
</tr>
<tr>
<td>FIGURE 2</td>
<td>Use of a GEO relay to avoid a foreign ground station</td>
<td>5</td>
</tr>
<tr>
<td>FIGURE 3</td>
<td>Aircraft to satellite data link.</td>
<td>5</td>
</tr>
<tr>
<td>FIGURE 4</td>
<td>Satellite to aircraft link.</td>
<td>6</td>
</tr>
<tr>
<td>FIGURE 5</td>
<td>Satellite to submarine application.</td>
<td>7</td>
</tr>
<tr>
<td>FIGURE 6</td>
<td>Point ahead Angle and Pointing Error Representation for Optical Satellite Cross-Link</td>
<td>9</td>
</tr>
<tr>
<td>FIGURE 7</td>
<td>Basic Block Diagram of Optical-Satellite Cross-Link (with the use of beacon)</td>
<td>10</td>
</tr>
<tr>
<td>FIGURE 8</td>
<td>Coherent-detection receiver.</td>
<td>14</td>
</tr>
<tr>
<td>FIGURE 9</td>
<td>Data Transfer link model.</td>
<td>16</td>
</tr>
<tr>
<td>FIGURE 10</td>
<td>Simplified models of digital communication system</td>
<td>18</td>
</tr>
<tr>
<td>FIGURE 11</td>
<td>Reed-Solomon coding process.</td>
<td>22</td>
</tr>
<tr>
<td>FIGURE 12</td>
<td>Reed-Solomon codeword diagram.</td>
<td>23</td>
</tr>
<tr>
<td>FIGURE 13</td>
<td>Encoder architecture.</td>
<td>27</td>
</tr>
<tr>
<td>FIGURE 14</td>
<td>Decoder Architecture.</td>
<td>28</td>
</tr>
<tr>
<td>FIGURE 15</td>
<td>(a) Burst error detection; (b) character error detection.</td>
<td>31</td>
</tr>
<tr>
<td>FIGURE 16</td>
<td>The syndrome generators of a single error correction RS code.</td>
<td>33</td>
</tr>
<tr>
<td>FIGURE 17</td>
<td>A complete single character correction RS decoder.</td>
<td>35</td>
</tr>
<tr>
<td>FIGURE 18</td>
<td>A complete single character correction RS encoder.</td>
<td>37</td>
</tr>
</tbody>
</table>
1. Introduction

Laser satellite communication involves transmission at frequencies in the $10^{14}$ Hz (optical) range which is around seven or eight orders of magnitude higher than the radio frequency (RF) systems. Transmission at such frequencies provides three main advantages, namely the greater bandwidth, smaller beam divergence angles, and smaller antennas. In fact communication with lasers is being used now a days in three modes of communication. These are aerial laser-beam communications, fiber optic communication and optical computers. In aerial laser beam communications, data and images are transmitted through the atmosphere using low power laser beams. It has the advantage that it is almost impossible to jam by known means. Moreover because of the extremely wide bandwidth of laser radiation, up to 10 discrete conversions can be executed simultaneously. Such an aerial laser beam communications is, however, completely weather dependent. On a clear day, a laser beam can be transmitted several miles but when atmospheric conditions involve fog, mist, rain or smog, the transmission is limited to shorter distances. In fiber optic communication, laser transmission is carried out through guided media of optical fibers. This system is capable of transmitting 4 gigabits of information per second over a span of some 120 km. Optical computer solves the problems of slowness and heat build up associated with electronic computers. This is because here light is used instead of electrical current. Further light being at the upper end of the electromagnetic spectrum can be encoded with much more information. Also an optical circuit has a zero resistance to flow and therefore is capable to carry much more information than the equivalent sized electronic circuit. There is no problem in using optical signals in parallel channels.

Being atmospheric dependent, laser communication therefore can not be used for communication between Earth station and a geo-synchronous satellite. However, it is quite suitable for communication between the satellites themselves or deep space communication. A typical example for such a cross satellite laser communication is as
that shown in FIGURE 1. Here GSs are the geo-synchronous satellites whereas ESs are some other satellites (e.g., Earth observation, or special purpose satellites). These ESs communicate to GSs with laser communication and eventually the GSs communicate with the Earth station through RF microwave communication. The optical transmitters and receiver packages are smaller and lighter than the equivalent BF microwave subsystems. This helps in reducing the spacecraft cost and weight. For deep space communication where the planets desired to be communicated with the Earth are quite far, the received signals are very weak and then this inter-satellite laser communication solves the problem. Here the deep spacecraft will have deep space optical link with a geo-synchronous satellite, which provides microwave link to the Earth station. Conclusively, therefore the laser satellite communication serves for intersatellite communications and here the analysis of cross optical-link is necessary.

FIGURE 1: Example of Laser Satellite Communication (GSs = Geo-synchronous Satellites; ESs = Earth Observation Satellites or any other Deep Space Satellite).

2. Why Laser Communications?

Laser communication systems offer many advantages over radio frequency (RF) systems. Before proceeding to the task of understanding the technologies and the methodology for designing a laser communication system, it is worthwhile to initially explore these advantages, many of which relate directly to the architecture and applications.

Most differences between laser communications (sometimes referred to as lasercom) and RF (Radio Frequency) arise from the very large difference in the wavelengths. RF
wavelengths are thousands of times longer than those at optical frequencies. This high ratio of wavelength leads to some interesting differences in the two systems. First, the beamwidth attainable with the laser communication system in narrower than that of the RF system by the same ratio at the same antenna diameters (the telescope of the laser communication system is frequently referred to as an antenna). For the given transmitter power level, the laser beam is brighter at the receiver by the square of this ratio due to the very narrow beam that exits the transmit telescope. Taking advantage of this brighter beam or higher gain, as it may quantitatively be stated, permits the laser communication designer to come up with the system that has a much smaller antenna than the RF system and further, need transmit much less power than the RF system for the same received power. The narrow laser beam is a two-edged sword, however, since it is much harder to point and acquisition of the other satellite terminal is more difficult.

System comparisons reveal these advantages of laser communications over RF:

- Smaller antenna size
- Lower weight, usually significant
- Lower power
- Minimal integration impact on the satellite

Last (but very important), laser communications is capable of much higher data rates than RF, again by virtue of that same wavelength (frequency) ratio.

3. Laser Communication Applications overview

There are a number of applications for which laser communications is well suited. Laser communications will never totally replace RF systems for space to ground communications where the message must be sent from space to ground or ground to space. The reason for this is of course that laser signals do not readily pass through clouds. When we say readily there is one application that exists for laser communication where even a laser’s poor penetration through clouds can be used and that is in submarine laser communications. Suffice it to say that, in general, laser communications form space to earth is subject to the vagaries of cloud cover. There
are mitigating approaches, however, that permit us to get around the problem of cloud cover.

The range of space applications for laser communications was illustrated. These included the following:

- Satellite crosslinks
- Satellite to aircraft
- Satellite to ground
- Satellite to submarine

3.1. **Satellite crosslinks:** are communication links in space and may be from LEO (Low Earth Orbit) to LEO (Low Earth Orbit), LEO (Low Earth Orbit) to GEO (Geostationary Earth Orbit), or from GEO (Geostationary Earth Orbit) to GEO (Geostationary Earth Orbit). Of course, these links are full duplex; that is, data flows both directions simultaneously. For laser links, there is no broadcast capability; links are point to point and cooperative effort is required between the terminals to close the links and transmit data.

LEO to GEO crosslinks are frequently used to transmit data from a data gathering LEO to a GEO where the data in turn will be transmitted to a user on the ground. In this case, the link is asymmetric; that is, high data rate is sent from the LEO to the GEO while low data rate for satellite command and control is passed in the opposite direction from the GEO to the LEO. The alternative to transmitting the data as it is taken is to record the data (store and forward mentioned previously). If the data gatherer is capable of extremely high data rates, the data must be transmitted if the onboard recorder is not capable of recording at the required rate or is unable to store the very large amount of the data required.

GEO to GEO links are useful. For a military satellite system, for example, a GEO relay may be used to avoid the use of a vulnerable ground station located on foreign soil. In times of conflict, the security of the ground station may be in question and if the link is of strategic importance, a GEO relay may be used. This situation is illustrated in the FIGURE 2.
3.2. Satellite to aircraft: link application can involve data being gathered by an aircraft (for example, a reconnaissance aircraft), and sent to a satellite, or the opposite where the aircraft receives the data for end-user use. It is easy to envision the relaying of command data to a satellite and thence to an airborne command post for force direction. These two applications are illustrated in the FIGURE 3 and 4. The reconnaissance aircraft data rate may be much higher than that of the command post.
3.3. **Satellite to ground** link may be an excellent application for laser communication since the data rate is virtually unlimited. RF (Radio Frequency) systems have difficulty transmitting very high data rates to the ground from synchronous orbit. RF must transmit a wavelength, which passes through the atmosphere, as does laser communications. The windows for RF are at the lower frequencies (longer wavelengths), which means that RF antennas tend to be extremely large (to reduce beam divergence) and large power levels are required. In addition, crowded frequency assignments are increasingly causing problems for RF downlinks.

Laser communications, on the other hand, possess wavelengths that pass through the atmosphere readily. This coupled with the need for relatively small antennas offers laser communications a big advantage over RF systems provided the cloud cover problem can be overcome or mitigated. Numerous studies have indicated that cloud cover can be mitigated by the expedient of providing several ground stations at different locations so that the transmission can be sent to the one that is cloud-free. This is termed cloud-cover diversity and four stations can provide an availability of over 97%. If a single aircraft is flown at 35,000-foot altitude, an availability of over
95% is obtained. As well, the aircraft can simply fly around the cloud cover for nearly 100% availability. The data pass through the atmosphere relatively intact even at low elevation angles, although not as low as with RF. This was demonstrated during the Airborne Flight Test System (AFTS) flights where 1-Gbps data rate was transmitted to the ground at an average bit error rate of 10⁻⁷ from ranges of up to 100 km and at elevation angles as low as 7 deg. Privacy/low probability of intercept (LPI) is also an issue since an RF transmission may make a footprint several hundred kilometers in diameter on the ground while a laser communication system would produce a footprint of only several hundred meters in diameter.

3.4. Satellite to submarine: commonly known as submarine laser communications (SLC), is an exceedingly interesting application for laser communication. Seawater transmits bluish-green light well enough to permit communication to a significant depth. The United States government has for a number of years investigated the technology required to transmitted data to a submerged submarine. Tests have been performed where an aircraft, carrying a spaceborne-type laser has successfully transmitted data to a submarine at depth through thick cloud cover. Earlier it was mentioned that clouds could be penetrated. There is a caveat, however, in that only data rates to a few hundred bits per second can be received. This is due to the fact that as the beam passes through the cloud, it scatters and arrives at the bottom of the cloud from a number of different direction and at different times. This multipath effect stretches the communications pulse so much that only a relatively low data rate can be received. This is not much of a problem if low data rate command data are being transmitted. There has even been some investigation and testing of a two-way laser link to subs. The satellite to submarine link application is illustrated in the FIGURE 5.

**FIGURE 5**: Satellite to submarine application.
4. Satellite Beam Acquisition, Tracking, and Pointing

In Laser or optical satellite communication the transmitting beam should be quite narrow because it would then have maximum power spectrum. But this extreme narrowness in the beam creates beam-pointing problems. It is in fact very essential that the beam should be correctly pointed always to the receiving satellite otherwise the communication link will be disturbed. However it is allowed that the beam may be pointing within an pointing error $\pm \theta_e$ radians (normally $\theta_e$ is in micro-radians). This problem is not faced with the RF systems because there beam-widths are much wider. It is therefore clear that to determine pointing error, the transmitting satellite must illuminate the receiving satellite as accurately as possible. It then requires that the transmitting satellite should know the location of receiving satellite as accurately as possible. Also, it has to know its own altitude as accurately as possible so that it may aim its beam at the known direction correctly.

For correct information regarding the receiving satellite's location, an optical beacon (unmodulated light source) transmitted by the receiving satellite back to the transmitting satellite is also being used. Here in fact the transmitting satellite firstly receives the beacon from the receiving satellite and then transmits its modulated laser beam (data etc) back to the receiving satellite. Thus the receiving satellite's location can be obtained and the pointing error can be minimized. However, there might be a situation in which though the transmitting satellite has received the beacon but before it could transmit the laser-modulated beam, the receiving satellite may move out of the transmitter's beam-width (transmitted beam-width is kept nearly equal to twice of pointing error). This situation would be as that shown in the FIGURE 6. In such a case it is necessary to know the angle by which the receiving satellite has moved ahead (or back) of the transmitting satellite. This angle of drifting of the receiving satellite is called the "point ahead angle".
FIGURE 6: Point ahead Angle and Pointing Error Representation for Optical Satellite Cross Link.

Thus if the tangential velocity of the receiving satellite is $V_r$, m/s, then the point-ahead angle is given as

$$\alpha = \frac{V_r}{150} \text{ micro-radians}$$

In case the above point ahead angle exceeds the one half of the laser modulated beam's beam-width then the use of this point ahead angle is being made. It is clear from equation that the point-ahead angle is independent of the distance of optical satellite cross-link. For velocities at Earth orbiting speeds ($V_r \approx 30$ km/s), the point ahead angle is approximately equal to 200 micro-radians. Thus whereas the laser modulated beam's beam-width may be of the order 10 to 100 $\mu$ rad, the point ahead angle is of the order of 200 $\mu$ rad.

In case the optical beacon is to be used both the transmitting as well as the receiving satellites would have optical transmitters and receiver and in such a situation the basic block diagram of the optical-satellite cross link would be as that shown illustration follow the FIGURE 7. Here the receiving optics tracks the arrival beam direction and adjusts the transmitting beam direction. Normally separate wavelengths are used for the optical beams in each direction. When there is no need of point-ahead angle the transmit and receive optics can be gimbaled together and the laser transmits through
the receive optics. But when the point ahead is needed then command control (either stored or received from Earth station) must adjust transmitting direction relative to receive direction. This requires an accurate satellite attitude control.

**FIGURE 7:** Basic Block Diagram of Optical-Satellite Cross-Link (with the use of beacon).

With above discussions it is evident that before optical data transmissions, the transmitting antenna must acquire the beacon from the receiver. This beacon is transmitted from the receiving satellite with a beam-width wide enough to cover the uncertainty angle of the beacon-receiving satellite. Once the beacon has been acquisitioned the satellite continuously tracks the LOS (line of sight vector) of the arriving beacon since the latter may vary due to the relative motion. In case the point ahead is needed, the satellite laser transmitter must point ahead by the proper angle and direction. In some cases commands for the point ahead are given from the Earth station through the RF or microwave links. For that it is essential that the Earth station accurately knows the instantaneous tangential velocity \( \dot{r} \), and also it is accurately transmitted to the satellite. After having LOS tracked and point-ahead angle determined the satellite optics points the return data beam to the proper position.

It would be of importance to mention here, that though the above manoeuvers may be quite accurately carried out, even then some errors might be present there. These might be due to altitude reference errors in the satellite, mechanical and structural
variations, bore-sight errors. However, the contributions of these errors are relatively quite small. But these errors can not be completely overcome, as these errors are open loop errors. The contribution due to vibration and bore-sight errors is generally below 1 micro-radians but the error due to altitude control may be high. It is therefore very essential to have proper altitude control so that overall pointing accuracy be as high as possible.

5. Communication Techniques

Not surprisingly, the modulation formats for optical communication are very similar to conventional RF techniques. In fact, many were adapted directly from the RF techniques. In addition to modulation techniques, a brief overview of coding types applicable to these modems is presented. Coding gain on the communication link is usually beneficial in that it reduces the required signal energy at the receiver for a given data rate. It should be noted that this is not an exhaustive list of modulation and coding techniques. Other modulation techniques and coding types may also provide beneficial means to the communication system. As a designer, you are encouraged not to be limited solely by these approaches.

The discussion begins by assembling the modulation techniques into several general categories. First and foremost, modulation waveforms are separated into two major groupings, namely direct and coherent detection waveforms. Subsets of these are also identified. For direct-detection system designs, digital baseband modulation is the most often used and most straightforward. Analog modulation of the waveforms has also been explored for sonic applications. These all will be discussed below.

Coherent-detection techniques fall into two primary categories. Homodyne and heterodyne modulation techniques for coherent detection systems will be also identified below.

Finally, coding techniques will be discussed and their benefit to the overall link performance will be analyzed. While most coding requires additional bandwidth to accomplish, the signal gain realized has a positive effect on the system design. Care must be taken to not increase the complexity of the laser source or receiver to realize
this gain. The designer must trade this complexity with the overall system design drivers like weight and power. Many simple coding techniques do exist that provide good gain.

5.1. Direct-Detection Digital Baseband Signaling:

The simplest form of modulation used in a direct-detection laser communication linking is binary baseband signaling. Here, one state is identified as a binary "one" and another as a binary "zero." Many different approaches to binary baseband modulation exist. The TABLE 1 identifies some of the more common techniques. Depending upon the type of laser source present (usually limited to pulse and quasi-CW sources), different binary modulation techniques are chosen.

TABLE 1 : Binary Modulation Techniques

<table>
<thead>
<tr>
<th>NRZ</th>
<th>Non-return to zero</th>
</tr>
</thead>
<tbody>
<tr>
<td>RZ</td>
<td>Return to zero (bi-phase)</td>
</tr>
<tr>
<td>PPM</td>
<td>Pulse position modulation</td>
</tr>
<tr>
<td>PBM</td>
<td>Pulse binary modulation</td>
</tr>
<tr>
<td>PDBM</td>
<td>Pulse delay binary modulation</td>
</tr>
<tr>
<td>PPBM</td>
<td>Pulse polarization binary modulation</td>
</tr>
<tr>
<td>PIM</td>
<td>Pulse interval modulation</td>
</tr>
<tr>
<td>PQM</td>
<td>Pulse quaternary modulation</td>
</tr>
</tbody>
</table>

5.2. Direct-Detection Analog Signaling:

In addition to digital signaling, several analog modulation formats can also be postulated for use in a laser communication system. While digital signaling is clearly the most efficient way to transmit binary data, some systems still use analog techniques to transmit information. Analog techniques are important to laser communication systems that link to RF bent-pipe information sources. Recall that many of today's satellites use down-converting and up-converting to relay the information from one source to the destination. No demodulation of the signal takes place onboard the satellite. On these satellites it may be more efficient to transmit the analog intermediate-frequency waveform directly without demodulation and re-
modulation. This is where analog signaling plays an important role in space laser cross-links.

Analog signaling is not adaptable to pulsed lasers, and as such will be discussed for CW-type lasers with external modulators and for directly current-driven laser diode sources. The linearity of the modulator output is directly proportional the efficiency of these analog modulation formats. Care must be taken to use modulators and modulatable sources, which have good linear modulation characteristics.

- **Amplitude Modulation**

  Classical *amplitude modulation* (AM) can be used to modulate the laser. The carrier optical signal is intensity modulated by the information content. At the receiver, the photodiode acts like an envelope detector, stripping the amplitude information off of the optical carrier and passing the information waveform directly to the electronics for further processing. Classical demodulation techniques are then used to detect this AM signal.

- **Phase Modulation**

  Similarly, *phase modulation* (PM) is typically achieved by externally phase modulating the optical waveform using an *electro-optic* (EO) modulator. Recently, PM has also been performed on-chip in laser-diode power amplifier arrangements. The phase of the carrier is continuously varied allowing the information to be transmitted.

- **Frequency Modulation**

  Continuous linear-frequency modulation of a laser source has been very difficult to achieve over large frequency deviation ranges. For narrow bandwidth signals, some form of frequency modulation could be considered; however, other techniques usually are more efficient. Until an effective and efficient *frequency modulation* (FM) technique is developed, FM will usually not be in the trade space.

5.3. Coherent-Modulation Signaling:

In a coherent-detection system the incident optical phase front is summed (or mixed) with a local oscillator at the receiver. In an optical communication system, the mixing
takes place at the detector. FIGURE 8 illustrates a technique used to mix the local-oscillator signal with the received signal. A simple optical beam beam-splitter is used to co-align the two signals and optical elements are used to image the signal onto the detector. The detector in a coherent system is typically a PIN photodiode since front-end receiver gain is provided by the amplitude of the local oscillator waveform.

![Diagram](image)

**FIGURE 8:** Coherent-detection receiver.

Several classical modulation schemes are used with coherent detection. These are discussed below.

- **Heterodyne Signaling**
  Several forms of heterodyne signaling are identified and their bit error probabilities are given. In *amplitude shift keying* (ASK), the amplitude of the signal is varied as a function of time. Each bit of information is represented by either the presence or absence of the optical waveform. Typically, ASK information is imparted on the optical signal using either directly current-driven laser diodes or external modulation using, either acousto-optical or electro-optical modulators. When using heterodyne techniques to demodulate the waveform, the classic bit error probability for ASK results from
  \[
P_{e} = (1/2) \text{erfc}\left((1/2) \left(\frac{z}{2}\right)^{1/2}\right)
\]
  where \( z \) is the SNR given previously and again \( \text{erfc}(x) \) is the complementary error function.

In *binary frequency shift keyed* (BFSK) systems, two distinct frequencies are transmitted and the signal is received by two receivers tuned to two distinct transmitted frequencies. Modulation is accomplished by chirping the direct-drive current of laser diode sources, resulting in two distinct frequencies being transmitted. The bit error probability for FSK can be expressed as
  \[
P_{e} = (1/2) \text{erfc}\left(z^{1/2}/2\right)
\]
In binary phase shift keyed (BPSK) systems, two distinct signals, identified by their phase difference, are transmitted. Modulation is accomplished either by an external electro-optical phase modulator or by a technique reported by Mecherle whereby a signal preconditioning occurs, allowing FSK-type modulation techniques to yield a direct-phase-modulated signal. The receiver identifies the binary state by detecting the phase relationship of the signal to a phase reference. The probability of bit error for PSK is given by

\[ P_e = \frac{1}{2} \text{erfc}[\frac{z}{2 \sqrt{2}}] \]

Similar to PSK is differential PSK (DPSK). In DPSK, the information is transmitted the signal changes phase states. The bit error probability for DPSK is given as

\[ P_e = \frac{1}{2} \exp[-\frac{z^2}{2}] \]

**Homodyne Signaling**

For homodyne detection of ASK and PSK waveforms, the required bandwidth can be reduced to approximately half that of a comparable heterodyne. This results in a 3-dB performance improvement in SNR for homodyne over heterodyne.

The bit error probability for homodyne ASK can be expressed as

\[ P_e = \frac{1}{2} \text{erfc}[\frac{z}{2}] \]

which is equivalent to heterodyne FSK. Similarly, the bit error probability for homodyne PSK can be expressed as

\[ P_e = \frac{1}{2} \text{erfc}[\frac{z}{\sqrt{2}}] \]

These techniques will be used in the process of data transfer that is going to talk about in the next section.

6. Data Transfer

Perhaps the simplest link to understand for the communication engineer is the transfer of data from one satellite to its companion. In most cases, established classical modulation schemes are used. Transfer is accommodated using a variety of noncoherent and coherent schemes. Clock regeneration at the receiver is facilitated by
the modulation schemes and classical synchronization techniques. Optical source and modulation technology must be carefully examined to determine the best modulation approach for each source. Not all modulation schemes are possible with all source technology. Limitations exist due to the physicals of the source technology. Also, receivers must be matched to the type of modulation used. Once again, not all receivers lend themselves well to all modulation schemes.

Contributors to the overall end-to-end data transfer link model are shown in the FIGURE 9. As with the other systems, background energy, in the direct-detection case, is a contributor to the system sensitivity. However, in most cases employing medium to high data rates (megabits to gigabits), except in operation near or through, the solar disk, the signal shot noise and preamplifier noise contributions overwhelm the noise contributions present from background. In the coherent-detection case, the total oscillator shot noise is the major contributor to system noise.

FIGURE 9: Data Transfer link model.
7. Satellite Error-Control Coding

After we have studied about data transfer, we will know the importance of coding and decoding and also the error-control coding. That complete the process of transmission, we must study another important signal-processing operation, namely, channel coding, which is used to provide for the reliable transmission of digital information over the channel. In particular, we present a survey of error-control coding techniques that rely on the systematic addition of redundant symbols to the transmitted information so as to facilitate two basic objectives at the receiver: error detection and error correction. We also consider a type of code called a trellis code that combines coding with modulation.

We begin the lecture with some preliminary considerations that include a brief discussion of the role of coding in the reliable transmission of digital information over a noisy channel.

7.1. Rationale for Coding, and types of codes

The task facing the designer of a digital communication system is that of providing a cost-effective facility for transmitting information from one end of the system (sender) at a rate and a level of reliability and quality that are acceptable to user at the other end (receiver). The two key system parameters available to the designer are transmitted signal power and channel bandwidth. These two parameters, together with the power spectral density of receiver noise, determine the signal energy-per-bit to noise power spectral density ratio (SNR) $E_b/N_0$. This ratio uniquely determines the bit error rate for a particular modulation scheme. Practical considerations usually place a limit on the value that we can assign to $E_b/N_0$. Accordingly, in practice, we often arrive at a modulation scheme and find that it is not possible to provide acceptable data quality (i.e., low enough error performance). For a fixed $E_b/N_0$, the only practical option available for changing data quality from problematic to acceptable is to use error-control coding.
Another practical motivation for the use of coding is to reduce the required SNR $E_b/N_0$ for a fixed bit error rate. This reduction in $E_b/N_0$ may, in turn, be exploited to reduce the required transmitted power or reduce the hardware costs by requiring a smaller antenna size in the case of radio communications.

**Error control** for data integrity may be exercised by means of forward error correction (FEC). FIGURE 10 shows the model of a digital communication system using such an approach.

![Diagram](image)

(a) Coding and modulation performed separately.

![Diagram](image)

(b) Coding and modulation combined.

FIGURE 10: Simplified models of digital communication system.

The discrete source generates information in the form of binary symbols. The *channel encoder* in the transmitter accepts message bits and adds *redundancy* according to a prescribed rule, thereby producing encoded data at a higher bit rate. The *channel decoder* in the receiver exploits the redundancy to decide which message bits were actually transmitted. The combined goal of the channel encoder and decoder is to minimize the effect of channel noise. That is, the number of errors between the channel encoder input (derived from the source) and the channel decoder output (delivered to the user) is minimized.
The addition of redundancy in the coded messages implies the need for increased transmission bandwidth. Moreover, the use of error-control coding adds complexity to the system, especially for the implementation of decoding operations in the receiver. Thus, the design trade-offs in the use of error-control coding to achieve acceptable error performance include considerations of bandwidth and system complexity.

There are many different error-correcting codes (with roots on diverse mathematical disciplines) that we can use. Historically, these codes have been classified into block codes and convolutional codes. The distinguishing feature for the classification is the presence or absence of memory in the encoders for the two codes.

To generate an \((n, k)\) block code, the channel encoder accepts information in successive \(k\)-bit blocks; for each block, it adds \(n-k\) redundant bits that are algebraically related to the \(k\) message bits, thereby producing an overall encoded block of \(n\) bits, where \(n > k\). The \(n\)-bit block is called a code word, and \(n\) is called the block length of the code. The channel encoder produces bits at the rate \(R = (n/k)R_S\), where \(R_S\) is the bit rate of the information source. The dimensionless ratio \(r = k/n\) is called the code rate, where \(0 < r < 1\). The bit rate \(R\), coming out of the encoder, is called the channel data rate. Thus, the code rate is a dimensionless ratio, whereas the data rate produced by the source and the channel data rate are both measured in bits per second.

In a convolutional code, the encoding operation may be viewed as the discrete-time convolution of the input sequence with the impulse response of the encoder. The duration of the impulse response equals the memory of the encoder. Accordingly, the encoder for a convolutional code operates on the incoming message sequence, using a "sliding window" equal in duration to its own memory. This, in turn, meant that in a convolutional code, unlike a block code, the channel encoder accepts message bits as a continuous sequence and thereby generates a continuous sequence of encoded bits at a higher rate.
In the model depicted in FIGURE 10(a), the operations of channel coding and modulation are performed separately. When, however, bandwidth efficiency is of major concern, the most effective method of implementation forward error-control correction coding is to combine it with modulation as a single function, as shown in FIGURE 10(b). In such an approach, coding is redefined as a process of imposing certain patterns on the transmitted signal.

The bulk of the material presented in these lectures relates to channel coding techniques suitable for forward error correction (FEC). There is, however, another major approach known as automatic-repeat-request (ARQ) which is also widely used for solving the error-control problem. The philosophy of ARQ is quite different from that of FEC. Specifically, ARQ utilizes redundancy for the sole purpose of error detection. Upon detection, the receiver requests a repeat transmission, which necessitates the use of a return path (feedback channel).

### 7.1.1. Bose-Chaudhuri-Hocquenqhem (BCH) Codes

One of the most important and powerful classes of linear block codes are BCH codes, which are cyclic codes with a wide variety of parameters. The most common binary BCH codes, known as primitive BCH codes, are characterized for any positive integers \( m \) (equal to or greater than 1) and \( t \) [less than \((2^m - 1)/2\)] by the following parameters:

- **Block length:** \( n = 2^m - 1 \)
- **Number of message bits:** \( k \geq n - mt \)
- **Minimum distance:** \( d_{\text{min}} \geq 2t + 1 \)

Each BCH code is a \( t \)-error correcting code in that it can detect and correct up to \( t \) random errors per code word. The Hamming single-error correcting codes can be described as BCH codes.

The BCH codes offer flexibility in the choice of code parameters, namely, block length and code rate. Furthermore, for block lengths of a few hundred bits or
less, the BCH codes are among the best known codes of the same block length and code rate.

A detailed treatment of the construction of BCH codes and their algebraic development is beyond the scope of our present discussion. To provide a feel for their capability, we present in Table 8.6, the code parameters and generator polynomials for binary block BCH codes of length up to $2^s - 1$.

7.1.2. Reed-Solomon Codes

The Reed-Solomon codes are an important subclass of non-binary BCH codes; they are often abbreviated as RS codes.

The encoder for an RS code differs from a binary encoder in that it operates on multiple bits rather than individual bits. Specifically, the encoder for an RS $(n, k)$ code of $m$-bit symbols groups the incoming binary data stream into blocks each $km$ bits long. Each block is treated as $k$ symbols, with each symbol having $m$ bits. An RS $(n, k)$ code is used to encode $m$-bit symbols into blocks consisting of $n = 2^m - 1$ symbols, that is, $m(2^m - 1)$ bits, where $m \geq 1$. Thus, the encoding algorithm expands a block of $k$ symbols to $n$ symbols by adding $n - k$ redundant symbols. When $m$ is an integer power of two, the $m$-bit symbols are called bytes. A popular value of $m$ is 8; indeed, 8-bit RS codes are extremely powerful.

A $t$-error-correcting RS code has the following parameters:

- Block length: $n = 2^m - 1$ symbols
- Message size: $k$ symbols
- Parity-check size: $n - k = 2t$ symbols
- Minimum distance: $d_{\text{min}} = 2t + 1$ symbols

The block length of the RS code is one less than the size of a code symbol, and the minimum distance is one greater than the number of parity-check symbols. Indeed, the minimum distance is always equal to the design distance of the code. It is easy to show that no $(n, k)$ linear block code can have minimum distance greater than $n - k + 1$. An $(n, k)$ linear block code for which the minimum distance equals $n - k + 1$ is
called a maximum-distance separable code. Accordingly, the RS codes make highly efficient use of redundancy, and block lengths and symbol sizes can be adjusted readily to accommodate a wide range of message sizes. Moreover, the RS codes provide a wide range of code rates that can be chosen to optimize performance. Finally, efficient decoding techniques are available for use with RS codes, which is one more reason for their wide application.

8. Reed-Solomon Codes

After we know some part about the Reed-Solomon. Now we will talk more about this again.

Reed-Solomon codes are block-based error correcting codes with a wide range of applications in digital communications and storage. Reed-Solomon codes are used to correct errors in many systems including:

- Storage devices (including tape, Compact Disk, DVD, barcodes, etc)
- Wireless or mobile communications (including cellular telephones, microwave links, etc)
- Satellite communications
- Digital television/DVB
- High-speed modems such as ADSL, xDSL, etc.

A typical system is shown here:

![Diagram of Reed-Solomon coding process](image-url)

**FIGURE 11**: Reed-Solomon coding process.

The Reed-Solomon encoder takes a block of digital data and adds extra "redundant" bits. Errors occur during transmission or storage for a number of reasons (for example...
noise or interference, scratches on a CD, etc). Reed-Solomon decoder processes each block and attempts to correct errors and recover the original data. The number and type of errors that can be corrected depends on the characteristics of the Reed-Solomon code.

8.1. Properties of Reed-Solomon codes

Reed-Solomon codes are a subset of BCH codes and are linear block codes. A Reed-Solomon code is specified as RS \( (n, k) \) with \( s \)-bit symbols.

This means that the encoder takes \( k \) data symbols of \( s \) bits each and adds parity symbols to make an \( n \) symbol codeword. There are \( n-k \) parity symbols of \( s \) bits each. A Reed-Solomon decoder can correct up to \( t \) symbols that contain errors in a codeword, where \( 2t = n-k \).

The following diagram shows a typical Reed-Solomon codeword (this is known as a Systematic code because the data is left uncharged and the parity symbols are appended):

![Figure 12: Reed-Solomon codeword diagram.](image)

**Example:** A popular Reed-Solomon code is RS \( (255,223) \) with 8-bits symbols. Each codeword contains 255 code word bytes, of which 223 bytes are data and 32 bytes are parity. For this code:

\[
\begin{align*}
n & \triangleq 255, \ k & \triangleq 223, \ s & \triangleq 8 \\
2t & \triangleq 32, \ t & \triangleq 16
\end{align*}
\]

The decoder can correct any 16-symbol errors in the code word: i.e. errors in up to 16 bytes anywhere in the codeword can be automatically corrected.
Given a symbol size \( s \), the maximum codeword length \( (n) \) for a Reed-Solomon code is \( n \leq 2^s - 1 \). For example, the maximum length of a code with 8-bit symbols \( (s = 8) \) is 255 bytes. Reed-Solomon codes may be shortened by (conceptually) making a number of data symbols zero at the encoder, not transmitting them, and then re-inserting them at the decoder.

**Example:** The \((255, 223)\) code described above can be shortened to \((200, 168)\). The encoder takes a block of 168 data bytes, (conceptually) adds 55 zero bytes, creates a \((255, 223)\) codeword and transmits only the 168 data bytes and 32 parity bytes.

The amount of processing "power" required to encode and decode Reed-Solomon codes is related to the number of parity symbols per codeword. A large value of \( t \) means that a large number of errors can be corrected but requires more computational power than a small value of \( t \).

### 8.1.1. Symbol Errors

One symbol error occurs when 1 bit in a symbol is wrong or when all the bits in a symbol are wrong.

**Example:** RS \((255, 223)\) can correct 16 symbol errors. In the worst case, 16 bit errors may occur, each in a separate symbol (byte) so that the decoder corrects 16 bit errors. In the best case, 16 complete byte errors occur so that the decoder corrects 16 ⚫ 8 bit errors.

Reed-Solomon codes are particularly well suited to correcting burst errors (where a series of bits in the codeword are received in error).

### 8.1.2. Decoding

Reed-Solomon algebraic decoding procedures can correct errors and erasures. An erasure occurs when the position of an errored symbol is known. A decoder can correct
8.1.3. Coding Gain

The advantage of using Reed-Solomon codes is that the probability of an error remaining in the decoded data is (usually) much lower than the probability of an error if Reed-Solomon is not used. This is often described as **coding gain**.

**Example:** A digital communication system is designed to operate at a Bit Error Ratio (BER) of $10^{-9}$, i.e. no more than 1 in $10^9$ bits are received in error. This can be achieved by boosting the power of the transmitter or by adding Reed-Solomon (or another type of Forward Error Correction). Reed-Solomon allows the system to achieve this target BER with a lower transmitter output power. The power saving given by Reed-Solomon (in decibels) is the **coding gain**.

8.2. Architectures for encoding and decoding Reed-Solomon codes

Reed-Solomon encoding and decoding can be carried out in software or in special-purpose hardware.

**Finite (Galois) Field Arithmetic**

Reed-Solomon codes are based on a specialist area of mathematics known as Galois fields or finite fields. A finite field has the property that arithmetic operations (+,-, ., etc.) on field elements always have a result in the field. A Reed-Solomon encoder or decoder needs to carry out these arithmetic operations. These operations require special hardware or software functions to implement.
up to \( t \) errors or up to \( 2t \) erasures. Erasure information can often be supplied by the demodulator in a digital communication system, i.e. the demodulator “flags” received symbols that are likely to contain errors.

The **Reed-Solomon** (RS) family of codes are block codes and as such are \( M \)-ary rather than binary-based codes. That is to say they act upon blocks or segments of data as opposed to each data element. The decoded probability of bit error is given by

\[
P_{be} = (n+1)/(2n^2)[(2t+1) \sum_{i=1}^{2t+1} \binom{n}{i} p_i^* q_i^{*r} + \sum_{j=2t+2}^{n} \binom{n}{j} j p_j^* q_j^{*r}] (3.74)
\]

where \( n \) is the code length and \( 2t+1 \) the number of correctable bits. A number of RS \((n, k, t)\) codes, where the code rate is given by \( k/n \), are given in TABLE 2. As can be extracted from the table, the best RS code is the RS(256, 128, 64).

**TABLE 2. Rate One-Half Reed-Solomon Codes.**

<table>
<thead>
<tr>
<th>( n )</th>
<th>( k )</th>
<th>( t )</th>
<th>( P_s (P_{be} = 10^{-5}) )</th>
<th>( P_s (P_{be} = 10^{-5}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>16</td>
<td>8</td>
<td>.007513</td>
<td>.004477</td>
</tr>
<tr>
<td>64</td>
<td>32</td>
<td>16</td>
<td>.01057</td>
<td>.00801</td>
</tr>
<tr>
<td>128</td>
<td>64</td>
<td>32</td>
<td>.01198</td>
<td>.01037</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>64</td>
<td>.01208</td>
<td>.01122</td>
</tr>
<tr>
<td>512</td>
<td>256</td>
<td>128</td>
<td>.01152</td>
<td>.01110</td>
</tr>
<tr>
<td>1024</td>
<td>512</td>
<td>256</td>
<td>.01074</td>
<td>.01054</td>
</tr>
</tbody>
</table>

When a codeword is decoded, there are three possible outcomes:

1. If \( 2s + r < 2t \) (\( s \) errors, \( r \) erasures) then the original transmitted code word will always be recovered.

**OTHERWISE**

2. The decoder will detect that it cannot recover the original code word and indicate this fact.

**OR**

3. The decoder will mis-decode and recover an incorrect code word without any indication. The probability of each of the three possibilities depends on the particular Reed-Solomon code and on the number and distribution of errors.
8.1.3. Coding Gain

The advantage of using Reed-Solomon codes is that the probability of an error remaining in the decoded data is (usually) much lower than the probability of an error if Reed-Solomon is not used. This is often described as coding gain.

Example: A digital communication system is designed to operate at a Bit Error Ratio (BER) of $10^{-4}$, i.e. no more than 1 in $10^4$ bits are received in error. This can be achieved by boosting the power of the transmitter or by adding Reed-Solomon (or another type of Forward Error Correction). Reed-Solomon allows the system to achieve this target BER with a lower transmitter output power. The power saving given by Reed-Solomon (in decibels) is the [insert formula].

8.2. Architectures for encoding and decoding Reed-Solomon codes

Reed-Solomon encoding and decoding can be carried out in software or in special-purpose hardware.

Finite (Galois) Field Arithmetic

Reed-Solomon codes are based on a specialist area of mathematics known as Galois fields or finite fields. A finite field has the property that arithmetic operations (+, -, x, / etc.) on field elements always have a result in the field. A Reed-Solomon encoder or decoder needs to carry out these arithmetic operations. These operations require special hardware or software functions to implement.
Generator Polynomial

A Reed-Solomon codeword is generated using a special polynomial. All valid codewords are exactly divisible by the generator polynomial. The general form of the generator polynomial is:

\[ g(x) = (x - \alpha^i)(x - \alpha^{i+1})\ldots(x - \alpha^{i+2t}) \]

and the codeword is constructed using:

\[ c(x) = g(x) \cdot i(x) \]

where \( g(x) \) is the generator polynomial, \( i(x) \) is the information block, \( c(x) \) is a valid codeword and \( \alpha \) is referred to as a primitive element of the field.

Example: Generator for RS(255,249)

\[ g(x) = (x - \alpha^0)(x - \alpha^1)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4)(x - \alpha^5) \]

\[ g(x) = x^6 + \gamma x^5 + \zeta_1 x^4 + \zeta_2 x^3 + \zeta_3 x^2 + \zeta_4 x + \zeta_5 \]

8.2.1. Encoder architecture

The 2t parity symbols in a systematic Reed-Solomon codeword are given by:

The following diagram shows an architecture for a systematic RS(255,249) encoder:

![Encoder Architecture Diagram](image)

**FIGURE 13**: Encoder architecture.

Each of the 6 registers holds a symbol (8 bits). The arithmetic operators carry out finite field addition or multiplication on a complete symbol.
8.2.2. Decoder architecture

**FIGURE 14:** Decoder Architecture.

A general architecture for decoding Reed-Solomon codes is shown in the following diagram.

**TABLE 3:** List of parameter in Decoder architecture.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r(x)</td>
<td>Received codeword</td>
</tr>
<tr>
<td>Si</td>
<td>Syndromes</td>
</tr>
<tr>
<td>L(x)</td>
<td>Error locator polynomial</td>
</tr>
<tr>
<td>Xi</td>
<td>Error locations</td>
</tr>
<tr>
<td>Yi</td>
<td>Error magnitudes</td>
</tr>
<tr>
<td>c(x)</td>
<td>Recovered code word</td>
</tr>
<tr>
<td>v</td>
<td>Number of errors</td>
</tr>
</tbody>
</table>

The received codeword \( r(x) \) is the original (transmitted) codeword \( c(x) \) plus errors:

\[
r(\square) \oplus c(\square) \oplus e(\square)
\]

A Reed-Solomon decoder attempts to identify the position and magnitude of up to \( t \) errors (or 2\( t \) erasures) and to correct the errors or erasures.

**Syndrome Calculation**

This is a similar calculation to parity calculation. A Reed-Solomon codeword has 2\( t \) *syndromes* that depend only on errors (not on the transmitted code word). The syndromes can be calculated by substituting the 2\( t \) roots of the generator polynomial \( g(x) \) into \( r(x) \).
Finding the Symbol Error Locations

This involves solving simultaneous equations with \( t \) unknowns. Several fast algorithms are available to do this. These algorithms take advantage of the special matrix structure of Reed-Solomon codes and greatly reduce the computational effort required. In general two steps are involved:

Find an error locator polynomial

This can be done using the Berlekamp-Massey algorithm or Euclid's algorithm. Euclid's algorithm tends to be more widely used in practice because it is easier to implement; however, the Berlekamp-Massey algorithm tends to lead to more efficient hardware and software implementations.

Find the roots of this polynomial

This is done using the Chien search algorithm.

Finding the Symbol Error Values

Again, this involves solving simultaneous equations with \( t \) unknowns. A widely-used fast algorithm is the Forney algorithm.

8.3. Implementation of Reed-Solomon encoders and decoders

8.3.1. Hardware Implementation

A number of commercial hardware implementations exist. Many existing systems use "off-the-shelf" integrated circuits that encode and decode Reed-Solomon codes. These ICs tend to support a certain amount of programmability (for example, RS(255, k) where \( t = 1 \) to 16 symbols). A recent trend is towards VHDL or Verilog designs (logic cores or intellectual property cores). These have a number of advantages over standard ICs. A logic core can be integrated with other VHDL or Verilog components and synthesized to an FPGA (Field Programmable Gate Array) or ASIC (Application Specific Integrated Circuit) – this enables so-called "System on Chip" designs where multiple modules can be combined in a single IC. Depending on production volumes, logic cores can often give significantly lower system costs than "standard" ICs. By using logic cores, a designer avoids the potential need to do a "lifetime buy" of a Reed-Solomon IC.

8.3.2. Software Implementation

Until recently, software implementations in "real-time" required too much computational power for all but the simplest of Reed-Solomon codes (i.e. codes with small values of \( t \)). The major difficulty in implementing Reed-Solomon codes in software is that general purpose processors do not support Galois field arithmetic operations. For example, to implement a Galois field multiply in software requires a test for 0, two log table look-ups, modulo add and anti-log table look-up. However, careful design together with increases in processor performance mean that software implementations can operate at relatively high data rates. The following table gives some example benchmark figures on a 166MHz Pentium PC:
<table>
<thead>
<tr>
<th>Code</th>
<th>Data rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS (255, 251)</td>
<td>12 Mbps</td>
</tr>
<tr>
<td>RS (255, 239)</td>
<td>2.7 Mbps</td>
</tr>
<tr>
<td>RS (255, 223)</td>
<td>1.1 Mbps</td>
</tr>
</tbody>
</table>

**TABLE 4**: Data rate for decoding.

These data rates are for decoding only: encoding is considerably faster since it requires less computation.

9. Character Error Correction

9.1. Character Error Detection

Sometime messages are transmitted or stored as blocks of characters. The message $M = (100,001,011,101,110,111,010)$, that will be used, consist of a block of seven characters, each one of length 3. Some codes were especially designed for correcting errors occurring in received characters. These code have the recently become popular, because of their application is compact disk technology. Here the information is stored in the form of block of bytes that are very sensitive to errors.

The specific code selected for use in the technology is the RS (Reed-Solomon) code. That only one character can be erroneous within a block. The RS code is actually a burst error correction code, since we correct the multiple bit error falling within the specified frame of character. The length of the character is the length of the burst that should be corrected.

Definition: Let $C$ is a transmitted character and let $C'$ is its received version. The pattern $E = C + C'$ is called the character error pattern.

Although the errors introduce into $C$ are some forms of burst, we had to introduce the above definition in order to distinguish between a character error pattern and burst error pattern. Whereas the second must start by definition with a 1, this is not necessarily the case with the first.
For example: if \( C = (110) \) and \( C' = (111) \), then the character error pattern is \((001)\).

Examine the message \( M \) given before that consists of 21 bits. Applying burst error correction techniques for correcting character errors here is much easier than applying such techniques for correcting any burst of length 3, since we know that the character error pattern starts with bit \#3p \( \) and ends with bit \#3p+2, for some integer \( p \).

![Diagram](image)

**FIGURE 15:** (a) Burst error detection; (b) character error detection.

In the FIGURE 15(a) depicts a burst error detection circuit (for bursts of length 3). About FIGURE 15(b) depicts a character error detection circuit. A full character of length 3 is fed into this circuit during each shift, where the XOR operation is done between the stored character and the character fed in. the circuit on the right-hand side describes the same behavior and represents the convention.

It can be easily checked that after shifting the message \( M \), listed above, into the circuit of FIGURE 15(b), the contents of the register will be ‘all 0’. Let \( M' \) be the received version of \( M \) from the material presented earlier it follow that if a character is erroneous (i.e., we have an error pattern confined to three successive places, starting with place \#3p), then this contents is the character error pattern.

Example \( M' = (100,001,011,101,111,111,010) \).
M' was obtained by introducing the character error pattern (001) into the fifth character (counting from the left). After shifting M' into either of the circuits depicted in FIGURE 15, the final contents will be (001).

Detecting burst errors of length t in a received message is enabled by introducing t parity bits into the transmitted message. It follows that introducing a parity character into a transmitted block of characters enables the recovery of the character error pattern (in the case of a single erroneous character).

9.2. Character Error Correction

The code word in an RS code is a block consisting of $2^n-1$ characters of length n, there are two parity characters and $2^n-3$ information characters in a code word of a single error correction RS code (that enables the correction of one erroneous character). The code is systematic. For brevity, whenever we mention the RS code from now on, we mean specifically the single error correction code.

The decoder of our RS code is based on the syndrome generators of the picture. The cells depicted in the FIGURE 16 store entire characters. The transmitted block of characters has the property that when it is shifted into the two syndrome generators, both will have a final "all 0" contents.

The functioning of register#1 is explained as follows. There is a maximum periodicity LFSR of length n associated with an RS code. (A reminder: n is also the length of the characters, where the transmitted block consists of $2^n-1$ characters.) We denote this register by $R_i$, where $R_i$ denotes its contents after i shifts, starting with the initial contents $R_0 = (1000...0)$. There is a matrix $Q$ associated with $R_i$ it consists of n rows that are the successive contents of $R_i$ starting with $R_1$ and ending with $R_n$. it was shown that $R_i * Q = R_{i+1}$, for any $0 \leq i \leq 2^n-2$. 


Referring now to register #1, we observe that its contents are multiplied by Q before it is XORed with the incoming character. That is, \((\text{new contents}) = (\text{old content}) \times Q + \text{(incoming character)}\). Since any contents of the register is either 'all 0' or equals some \( R_i \), we find that multiplication by Q yields 0 in the first case and \( R_{i+1} \) in the second. For clarification purposes, TABLE 5 lists the successive contents of register#1 for the cases where the messages \( M \) and \( M' \), given in the preceding examples, are shifted into it. Here \( n = 3 \), and the register \( R \) is the one depicted in the picture. The matrix \( Q \) associated with it is:

\[
\begin{align*}
010 \\
Q = 001 \\
110
\end{align*}
\]

<table>
<thead>
<tr>
<th>input block ( M )</th>
<th>input block ( M' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>010</td>
</tr>
<tr>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>101</td>
<td>100</td>
</tr>
<tr>
<td>001</td>
<td>111</td>
</tr>
<tr>
<td>101</td>
<td>110</td>
</tr>
<tr>
<td>101</td>
<td>010</td>
</tr>
<tr>
<td>000</td>
<td>101</td>
</tr>
</tbody>
</table>

**TABLE 5**: Successive contents of register#1 of FIGURE 16.
When considering register #2 of the FIGURE 16, it is recognized as that of FIGURE 15. It sums that received characters. From the preceding discussions we know that if the sum of the characters of a transmitted block is the character “all 0”, and a single-character error occurred during the transmission, then the sum of the received characters yields the character error pattern. For the specific case of the transmitted message M and the received message M' treated above, the sum of the characters is (000) and (001), respectively.

We now explain the error correction procedure of our RS code. A transmitted message M and its received version M' have 2^n-2 characters in common, and may differ in one character. Let E = M + M'. The error block E consists of 2^n-2 characters that are “all 0” and one character that equals the error pattern. As our code is linear and M is a code word, the contents of the syndrome generators of the FIGURE 16 after shifting M' into them is the same as that obtained by shifting only E.

Let the characters of M, M', and E be indexed, where the leftmost character is #0. Let the erroneous character be #i. The error pattern character equals some R_j, where R is the maximum periodicity LFSR associated with our code (base upon which the matrix Q was formed). We now analyze the effect of shifting E into register #1 of FIGURE 16. We start by shifting in the rightmost character. The contents of the register are 0 until the error pattern is shifted in. At this time the contents of the register is R_j. The register is now shifted a further i times, until the leftmost character is shifted in. The input characters during these i shifts are always 0, and the circuit generates successive contents of R (that is the effect of successive multiplication by Q). The final contents of the register is therefore R_{i+i}. At this time the contents of register #2 is the character error pattern R_j . (This effect has already been explained.) To summarize one register yields R_{i+i}, and the other yields R_j = error pattern.

A complete correction process of a single erroneous character means recovering both the error pattern and the location of the erroneous character. The process described above directly yields the error pattern. Recovering the error location means the recovery of the index i. Since we know R_j (that is the error pattern) and we know R_{i+i}, a possible way of recovering i is to feed R_i into the register R and to shift it until we get the pattern known to us as R_{i+i}. by counting the number of shifts we get i.
Another way of recovering \( i \) is by shifting the register \( R \), starting with the contents \( R_{j+i} \), until we get the contents \( R_j \). The number of shifts in this case is \( 2^n-i \). This is based on the observation that \((j+i) + (2^n-1-i) = j + 2^n-1\). However, due to the periodicity of \( R \), its contents after \( j + 2^n-1 \) shifts is \( R_j \). Note now that \( 2^n-2 \) is the location of the erroneous character in \( E \), counting from the right. (This is clarified by observing that the rightmost bits are bit \#2\(^{n}\)-2 counting from the left, and is bit \#0 counting from the right.) We are now ready to explain the error correction circuitry depicted in the FIGURE 17.

![FIGURE 17: A complete single character correction RS decoder.](image)

The received block that consists of \( 2^n-1 \) characters is shifted into the three registers. The block is stored register \#3, while the two at the bottom are the syndrome generators of the FIGURE 16. During this shift the switch \( S \) closes the feedback line of register \#2. After the received block is completely shifted in, switch \( S \) connects the contents of register \#2 to the input of an AND gate. This contents is ‘all 0’ in the case where there is an error. (Note that a complete character enters the AND gate: i.e., this gate has \( n+1 \) single-bit inputs, \( n \) of which are the character from the register and an additional single bit from the circuit “check for equality”.) Now we continue to shift both register \#3 and \#1. While characters exit the circuit, register \#1 acts as the register \( R \), generating successive contents of \( R \) starting with \( R_{j+i} \). After \( 2^n-1-i \) shifts, the content of register \#1 is \( R_j \). Registers \#2 and \#1 then have the same contents and the output from the circuit “check for equality” is 1. The contents of register \#2 that is the error pattern is shifted into the XOR gate and is XORed with the character now
stored in the buffer. Following the counting clarifies that this character is the erroneous one and the XOR operation corrects it. In the case where there is no error, the output form the AND gate is always 0, and the received message is shifted unaltered out of register #3.

Example take the case of the messages M and M' treated before. After M' shifted into the circuit of the above FIGURE 17, the contents of registers #1 and #2 are (101) and (001), respectively. (See the above table also and an example preceding it.) Shifting on register #1, its successive contents are (100), (010),(001). After three shifts we obtain equality between the content of two registers. At this time the erroneous character is shifted into the buffer. (When introducing M', we noted that the erroneous character is located 3 places from the right.) We now XOR it with the error pattern and correct it.

Note that the number of parity characters in M, needed for performing a single-character error correction, is the sum of the lengths of the syndrome generators. This sum is two. That is, out of the $2^n$ characters in M, $2^n-3$ are information characters and two are parity characters.

We conclude this section by introducing the encoding circuitry of our RS code. Here we have to construct a circuit that satisfies simultaneously the two independent parity constraints imposed by the two syndrome generators of FIGURE 16. We repeat the result stated there: the feedback connections of a LFSR corresponding to two LFSR’s connected in parallel are found by the operation $v \cdot M$, where $v$ represents the feedback connections of one LFSR and the row of $M$ consist of linear shifts of the vector corresponding to the feedback connection of the other LFSR. In the case of the RS code, the LFSR’s handle entire characters, and the feedback connections are not denoted by 0 and 1 but rather by matrices. For example, the feedback connections of register #2 that appear to be (11) are actually (I), where I is a unit matrix of dimension $n \times n$. By the same token, the feedback connections of register #1 are (QI). In order to find the single LFSR corresponding to the parallel operation of these two registers, we then perform the operation.
\((I I) \cdot Q I 0 = (Q P I)\) where \(P\) denotes the matrix \(I+Q\).

The encoder whose feedback connections correspond to \((QPI)\) is depicted in the FIGURE 18. As was indicated above, the number of information characters entering the encoder is \(2^n-3\). The final contents of the following LFSR will form the parity characters attached to the information characters attached to the information characters, forming the transmitted codeblock.

**FIGURE 18:** A complete single character correction RS encoder.
**Objective**

1. To study the Reed-Solomon encoding and decoding for satellite communication.
2. To study and modify Simulation Program and use results to support decision making for designing the laser satellite link.
3. To make comfortable for user to coded/decoded monitoring by provide graphic user interface.
4. To analyze the feasibility of using Reed-Solomon coding over laser satellite communication.
Description and Scope of Project

This project will provide the information of Reed-Solomon such as definition, formula, advantage/disadvantage, and architecture of encoder/decoder of Reed-Solomon coding. We will do simulation for the encoding/decoding of the laser communication signal.

Firstly, we will find all the information from many material, texts and search some information and source code of Reed-Solomon from Internet, the book, and journals. Then analyzing collected information and making conclusion. Second, we will use the C programming to create the simulation for the real work of Reed-Solomon encoding/decoding and analyze the output. Finally, we will make the user interface to give the convenient for user that can make understanding more in using the coding technique.

We study about Reed-Solomon coding from the materials and journals to know how the encoder/decoder of Reed-Solomon and how it operates with the laser satellite link communication. All of the performance about the coding that depend on the any function such as satellite link transmitter and receiver. Because this component is the important function to make laser satellite communication system is order to complete the work. Sometime the error may occur in the system because some incompletely component of device or channel. Therefore we must to have the coding to solve the problem of error.

As we have already collected some encoder/decoder source code in C programming language, we will test these several source code whether it is usable or not. We will cut out some source code that is not usable then starting to analyze the existing source code. We will select the most understandable and suitable to our project. At last this program will be tested with several input with different size of packet and many different factor which we are not consider it yet. The output of the encoding program will be used to test on the decoder. After we get the output of the decoder, we will compare this output to that one of the input of encoder so we can check that our encoder and decoder are correct. Moreover we may put some noise into our system so we can check whether our decoder can correct noise or not.
When we use the program of coding. We need to comply or run the DOS that is complicate and difficult to understand. Therefore we think that there should have the user interface to connect with the program easily.

For the feasibility of using the Reed-Solomon coding over the laser satellite communication that depend on the output of coding in the system that are same or exactly or not with the original signal and depend on the effectiveness of its performance.
**Project Design**

Firstly, I want to tell you that this project is analyzing and simulating. It is not designing or programming the source code for RS coding and RS simulator so all of the source codes are collected from our advisor, my friends, and Internet. As we are doing simulation for Reed-Solomon error control coding. First we collected the source code for encoder, decoder and simulator. Then we get all of the source code, which are necessary for us to do simulation. In this project we will divide the simulation into 2 part.

1. Send signal with error control coding via noisy channel.

2. Simulation for the ability of RS coding to collect the error.

After we finish all of simulation task, we will use those gathered data to comparing the received message with the transmitted message. The result of comparing will be shown in the figure later in this report.
Project Implementation

For the first program, it is the program that test the Reed-Solomon with 8 bits symbols due to the demo version we cannot vary the number of bits per symbol. This program divide into 5 major parts.

1. R-S input  In this version we can vary the number of parity symbols and data symbols.
2. **Encode**

Enter the data file for encoding

---

**Encode Options**

- **File Name**: Enter the file name for encoding.
- **Browse**: Browse for the file.
- **Start**: Begin encoding.
- **Cancel**: Cancel operation.

**RS Encoder**

- **Data Source**: Input data.
- **Number of Code Words**: Specify the number of code words.
- **Code Rate**: Specify the code rate.
- **Frame Size**: Specify the frame size.
- **Data Output**: Displays encoded data.

---

The data input was encoded with the specified parameter in the RS input. It's shown the elapsed time. The input data changes to data on the encoded data.
3. Insert Errors  

We can select the encoded data file for error insertion and enter the number of symbol errors or bit error rate.

![Insert Errors dialog box](image)

**Insert Errors...**

- **Encoded data file:** `data.en`
- **Number of errors to be inserted:** 0
- **Bit Error Rate (BER):** 0

**Errors Inserted**

- Number of symbol errors/code word inserted: 0
- **Bit Error Rate (BER):** 0
- **File created:** `data.err`
4. Decode

Enter/select the encoded data file (with or without errors) for decoding. The data must be encoded with a compatible R-S encoder.
Here is the second program, this source code is listed in C language. It is the program
to calculate throughput and Probability of bit error. It's consisting of Galois field,
Generator polynomial, encoder and decoder. The source code of this program is
listed below

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

int binary[10000000];
int bidata[11500000];
unsigned char data[1500000];
unsigned char ddata[1500000];
unsigned char ori_data[13000000];
unsigned char final_data[13000000];
unsigned char decidata[15000000];

#define MAX_RANDOM 0x7fffffff
#undef DEBUG 1
#undef DEBUG 2

#undef CCSDS 1
#undef CCSDS

#define MM 8 /* Symbol size in bits */
#define KK 223 /* Number of data symbols per block */
#define B0 1 /* First root of generator polynomial, alpha form */
#define PRIM 1 /* power of alpha used to generate roots of generator poly */

#else /* CCSDS */

#define MM 8
#define KK 223
#define B0 112
#define PRIM 11
#endif

#define NN ((1 << MM) - 1)

#ifdefine CCSDS
/* CCSDS field generator polynomial: 1+x+x^2+x^7+x^8 */
int Pp[MM+1] = { 1, 1, 1, 1, 0, 0, 0, 1, 1, 1 };

#else /* not CCSDS */
#endif

#init(MM == 2)/* Admittedly silly */
int Pp[MM+1] = { 1, 1, 1 };

#elif(MM == 3)
/* 1 + x + x^3 */
int Pp[MM+1] = { 1, 1, 0, 1 };

#elif(MM == 4)
/* 1 + x + x^4 */
```

#elif(MM == 5)
/* 1 + x^2 + x^5 */
int Pp[MM+1] = { 1, 1, 0, 0, 0, 1 };

#elif(MM == 6)
/* 1 + x + x^6 */
int Pp[MM+1] = { 1, 1, 0, 0, 0, 0, 1 };

#elif(MM == 7)
/* 1 + x^3 + x^7 */
int Pp[MM+1] = { 1, 0, 0, 1, 0, 0, 0, 1 };

#elif(MM == 8)
/* 1+x^2+x^3+x^4+x^8 */
int Pp[MM+1] = { 1, 0, 1, 1, 1, 0, 0, 1 };

#elif(MM == 9)
/* 1+x^4+x^9 */
int Pp[MM+1] = { 1, 0, 0, 0, 1, 0, 0, 0, 1 };

#elif(MM == 10)
/* 1+x^3+x^10 */
int Pp[MM+1] = { 1, 0, 0, 1, 0, 0, 0, 0, 0, 1 };

#elif(MM == 11)
/* 1+x^2+x^11 */
int Pp[MM+1] = { 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1 };

#elif(MM == 12)
/* 1+x+x^4+x^6+x^12 */
int Pp[MM+1] = { 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1 };

#elif(MM == 13)
/* 1+x+x^3+x^4+x^13 */
int Pp[MM+1] = { 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1 };

#elif(MM == 14)
/* 1+x+x^6+x^10+x^14 */
int Pp[MM+1] = { 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 };

#elif(MM == 15)
/* 1+x+x^15 */
int Pp[MM+1] = { 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 };

#elif(MM == 16)
/* 1+x+x^3+x^12+x^16 */
int Pp[MM+1] = { 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1 };

#else
#error "Either CCSDS must be defined, or MM must be set in range 2-16"
#endif

typedef int gf;

/* index->polynomial form conversion table */
static gf Alpha_to[NN + 1];
/* Polynomial->index form conversion table */
static gf Index_of[NN + 1];

#define A0 (NN)

/* Generator polynomial g(x) in index form */
static gf Gg[NN - KK + 1];

static int RS_init; /* Initialization flag */

/* Compute x % NN, where NN is 2**MM - 1, */
/* without a slow divide */
static gf modnn(int x)
{
    while (x >= NN) {
        x -= NN;
        x = (x >> MM) + (x & NN);
    }
    return x;
}

/*#define min(a,b) ((a) < (b) ? (a) : (b))*/

#define CLEAR(a,n) {
    int ci;
    for(ci=(n)-1;ci>=0;ci--)
        (a)[ci] = 0;
}

#define COPY(a,b,n) {
    int ci;
    for(ci=(n)-1;ci>=0;ci--)
        (a)[ci] = (b)[ci];
}

#define COPYDOWN(a,b,n) {
    int ci;
    for(ci=(n)-1;ci>=0;ci--)
        (a)[ci] = (b)[ci];
}

static void init_rs(void);

#ifndef CCSDS

static unsigned char taltab[NN+1],tal1tab[NN+1];
static unsigned char tal[] = { 0x8d, 0xef, 0xec, 0x86, 0xfa, 0x99, 0xaf, 0x7b }; 

static void
gen_tab(void)
{
    int i,j,k;

    for(i=0;i<256;i++) {/* For each value of input */
        taltab[i] = 0;
    }
    for(j=0;j<8;j++) {/* For each column of matrix */
        for(k=0;k<8;k++) {/* For each row of matrix */
            if(i & (1<<k))
taltab[i] ^= tab[7-k] & (1<<j);
}
tab[taltab[i]] = i;
}
#endif /* CCSDS */

#if PRIM != 1
static int Ldec;/* Decrement for aux location variable in Chien search */

static void
gen_Ldec(void)
{
    for(Ldec=1,(Ldec % PRIM) != 0;Ldec+= NN)
    {
        Ldec /= PRIM;
    }
#else
#define Ldec 1
#endif

static void
generate_gf(void)
{
    register int i, mask;

    mask = 1;
    Alpha_to[MM] = 0;
    for (i = 0; i < MM; i++) {
        Alpha_to[i] = mask;
        Index_of[Alpha_to[i]] = i;
        /* If Pp[i] == 1 then, term @^i occurs in poly-repr of @^MM */
        if(Pp[i] != 0)
            Alpha_to[MM] ^= mask; /* Bit-wise EXOR operation */
        mask <<= 1; /* single left-shift */
    }
    Index_of[Alpha_to[MM]] = MM;
    mask >>= 1;
    for (i = MM + 1; i < NN; i++) {
        if(Alpha_to[i - 1] >= mask)
            Alpha_to[i] = Alpha_to[MM] ^ ((Alpha_to[i - 1] ^ mask) << 1);
        else
            Alpha_to[i] = Alpha_to[i - 1] << 1;
        Index_of[Alpha_to[i]] = i;
    }
    Index_of[0] = A0;
    Alpha_to[NN] = 0;
}

static void
gen_poly(void)
{
    register int i, j;

    Gg[0] = 1;
    for (i = 0; i < NN - KK; i++) {
        Gg[i+1] = 1;
        /*
        * Below multiply (Gg[0]+Gg[1]*x + ... +Gg[i]*x^i) by
        * (@**(B0+i)*PRIM + x)
        */
for (j = i; j > 0; j–)
    if (Gg[j] != 0)
        \[ Gg[j] = Gg[j - 1] \]()^{\text{Alpha_to[modnn(Index_of(Gg[j])] + (B0 + i) \* PRIM)}};
else
    Gg[j] = Gg[j - 1];
/* Gg[0] can never be zero */
Gg[0] = Alpha_to[modnn(Index_of(Gg[0])] + (B0 + i) \* PRIM));
}
/* convert Gg[] to index form for quicker encoding */
for (i = 0; i <= NN - KK; i++)
    Gg[i] = Index_of(Gg[i]);

/* take the string of symbols in data[i], i=0..(k-1) and encode
* systematically to produce NN-KK parity symbols in bb[0..bb[NN-KK-1]] data[]
* is input and bb[] is output in polynomial form. Encoding is done by using
* a feedback shift register with appropriate connections specified by the
* elements of Gg[], which was generated above. Codeword is \( \langle X \rangle = \)
* data(X)*X**(NN-KK)+ b(X)
*/

int encode_RA(unsigned char data[KK], unsigned char bb[NN-KK])
{
    register int i, j;
    gf feedback;
    #ifdef DEBUG >= 1 && MM != 8
/* Check for illegal input values */
    for(i=0;i<KK;i++)
        if(data[i] > NN)
            return -1;
    #endif
    if(!RS_init)
        init_RA();
    CLEAR(bb,NN-KK);

    for(i = KK - 1; i >= 0; i–) {  
        feedback = Index_of(data[i] ^ bb[NN - KK - 1]);
        if (feedback != A0) {  /* feedback term is non-zero */
            for (j = NN - KK - 1; j > 0; j–)
                if (Gg[j] != A0)
                    \[ bb[j] = bb[j - 1] \]()^{\text{Alpha_to[modnn(Gg[j] + feedback)}}; 
                else
                    \[ bb[j] = bb[j - 1]; 
                bb[0] = Alpha_to[modnn(Gg[0] + feedback)];
        } else {  /* feedback term is zero. encoder becomes a
            * single-byte shifter */
            for (j = NN - KK - 1; j > 0; j–)
                \[ bb[j] = bb[j - 1]; 
            \[ bb[0] = 0; 
        }
    }
#ifdef CCSDS

/* Convert to l-basis */
for(i=0;i<NN;i++)
data[t] = ttab[data[i]];
#endif

return 0;
}

/*
 * Performs ERRORS+ERASURES decoding of RS codes. If decoding is successful,
 * writes the codeword into data[] itself. Otherwise data[] is unaltered.
 *
 * Return number of symbols corrected, or -1 if codeword is illegal
 * or uncorrectable. If eras_pos is non-null, the detected error locations
 * are written back. NOTE! This array must be at least NN-KK elements long.
 *
 * First 'no_eras' erasures are declared by the calling program. Then, the
 * maximum # of errors correctable is t_after_eras = floor((NN-KK-no_eras)/2).
 * If the number of channel errors is not greater than "t_after_eras" the
 * transmitted codeword will be recovered. Details of algorithm can be found
 * in R. Blahut's "Theory ... of Error-Correcting Codes".
 *
 * Warning: the eras_pos[] array must not contain duplicate entries; decoder failure
 * will result. The decoder *could* check for this condition, but it would involve
 * extra time on every decoding operation.
 */

int
dec_rs(unsigned char data[NN], int eras_pos[NN-KK], int no_eras)
{
  int deg_lambda, cl, deg_omega;
  int i, j, k;
  gf u, q, tmp, num1, num2, den, descr_r;
  gf lambda[NN-KK + 1], s[NN-KK + 1]; /* Err+Eras Locator poly
      * and syndrome poly */
  gf b[NN-KK + 1], i[NN-KK + 1], omega[NN-KK + 1];
  gf root[NN-KK], reg[NN-KK + 1], loc[NN-KK];
  int syn_error, count;

  if(!RS_init)
    init_rsl();

  #ifdef CCSDS
  /* Convert to conventional basis */
  for(i=0;i<NN;i++)
    data[i] = ttab[data[i]];
  #endif

  #ifdef DEBUG >= 1 && MM != 8
  /* Check for illegal input values */
  for(i=0;i<NN;i++)
    if(data[i] > NN)
      return -1;
  #endif
  /* form the syndromes; i.e., evaluate data(x) at roots of g(x)
   * namely @**(B0+i)*PRIM, i = 0, ..., (NN-KK-1)
   */
  for(i=1;i<=NN-KK;i++)
```c
s[i] = data[0];
}
for(j=1;j<NN;j++)
    if(data[j] == 0)
        continue;
    tmp = Index_off[data[j]];
/*
 *      s[i] ^= Alpha_to[mod( ( data[j] % (B0+i-1) ) )]; */
    for(i=1;i<NN-KK;i++)
        s[i] ^= Alpha_to[mod( (data[j] % (B0+i-1)*PRIM*) )];
}
/* Convert syndromes to index form, checking for nonzero condition */
    syn_error = 0;
for(i=NN-KK;i++)
    syn_error |= s[i];
    s[i] = Index_off[s[i]];
}

if((syn_error) {
    /* if syndrome is zero, data[] is a codeword and there are no
       * errors to correct. So return data[] unmodified
       */
    count = 0;
    goto finish;
}
CLEAR(&lambda[1],NN-KK);
    lambda[0] = 1;

    if(no_eras > 0) {
        /* Init lambda to be the erasure locator polynomial */
    lambda[1] = Alpha_to[mod( (PRIM * eras_pos[0]) )];
    for(i=1;i<no_eras;i++)
        u = mod( (PRIM * eras_pos[i]) );
    for(j=1;j>0;j--)
        
            tmp = Index_off[lambda[j-1]];
            if(tmp != A0)
            lambda[i] ^= Alpha_to[mod( (u + tmp) )];
    }
    if(DEBUG >= 1) {
    /* Test code that verifies the erasure locator polynomial just constructed
       Needed only for decoder debugging.
       */
    for(i=1;i<no_eras;i++)
        reg[i] = Index_off[lambda[i]];
    count = 0;
    for(k=NN-Ldec; i <= NN; i++, k = mod( NN+k-Ldec ))
        q = 1;
    for(j=1;j<no_eras;j++)
        if((reg[j] != A0))
            reg[j] = mod( (reg[j] + j) );
            q ^= Alpha_to[reg[j]];
            if(q == 0)
                continue;
    /* store root and error location number indices */
    root[count] = i;
    loc[count] = j;
    count++;
```
if (count != no_eras) {
    printf("\n lambda(x) is \text{WRONG}\n");
    count = -1;
    goto finish;
}
}
#endif
#endif

for (i = 0; i < NN-KK+1; i++)
    b[i] = Index_of[lambda[i]];

/*
 * Begin Berlekamp-Massey algorithm to determine error+erasure
 * locator polynomial
 */

r = no_eras;
cl = no_eras;
while (++ i <= NN-KK) /* r is the step number */
{ /* Compute discrepancy at the r-th step in poly-form */
    discr_r = 0;
    for (i = 0, i < r, i++)
    { if ((lambda[i] != 0) & (s[r - i] != A0))
        discr_r ^= Alpha_to[modnn(Index_of[lambda[i]] + s[r - i])];
    }
    discr_r = Index_of[discr_r]; /* Index form */
    if (discr_r == A0) {
        /* 2 lines below: B(x) \leftarrow x*B(x) */
        COPYDOWN(&b[1], b, NN-KK);
        b[0] = A0;
    } else {
        /* 7 lines below: T(x) \leftarrow lambda(x) - discr_r*x*b(x) */
        t[0] = lambda[0],
        for (i = 0; i < NN-KK; i++)
        { if (b[i] != A0)
            t[i+1] = lambda[i+1] ^ Alpha_to[modnn(discr_r + b[i])];
        else
            t[i+1] = lambda[i+1];
        }

        if (2 * el <= r + no_eras - 1) {
            el = r + no_eras - el;
            /*
            * 2 lines below: B(x) \leftarrow \text{inv(discr_r) }*
            * lambda(x)
            */
            for (i = 0, i <= NN-KK, i++)
                b[i] = (lambda[i] == 0) ? A0 : modnn(Index_of[lambda[i]] - discr_r + NN);
        } else {
            /* 2 lines below: B(x) \leftarrow x*B(x) */
            COPYDOWN(&b[1], b, NN-KK);
            b[0] = A0;
        }
    }
    COPY(lambda, t, NN-KK+1);
}
/* Convert lambda to index form and compute deg(lambda(x)) */
deg_lambda = 0;
for(i=0;i<NN-KK+1;i++){
    lambda[i] = index_of[lambda[i]];
    if(lambda[i] != A0)
        deg_lambda = i;
}

/*
 * Find roots of the error+erasure locator polynomial by Chien
 * Search
 */
COPY(&reg[1],&lambda[1],NN-KK);

/* Number of roots of lambda(x) */
for (i = 1,k=NN-Ldec; i <= NN; i++,k = modnn(NN+k-Ldec)) {
    q = 1;
    for (j = deg_lambda; j > 0; j--){
        if (reg[j] != A0) {
            reg[j] = modnn(reg[j] + j);
            q ^= Alpha_to[reg[j]];
        }
    }
    if (q != 0)
        continue;

/* store root (index-form) and error location number */
root[count] = i;
loc[count] = k;
/* If we've already found max possible roots,
 * abort the search to save time */
if(i+count == deg_lambda)
    break;
}

/*
 * deg(lambda(x)) unequal to number of roots => uncorrectable
 * error detected
 */
count = -1;
goto finish;
}

/*
 * Compute error+eraser evaluator poly omega(x) = s(x)*lambda(x) (modulo
 * x**(NN-KK)). in index form. Also find deg(omega).
 */
deg_omega = 0;
for (i = 0; i < NN-KK;i++){
    tmp = 0;
    j = (deg_lambda < i) ? deg_lambda : i;
    for (j >= 0; j--){
        if (((i + 1 - j) != A0) && (lambda[j] != A0))
            tmp ^= Alpha_to[modnn((i + 1 - j) + lambda[j])];
    }
    if (tmp != 0)
        deg_omega = i;
    omega[i] = index_of[tmp];
}

omega[NN-KK] = A0;
for (j = count-1; j >= 0; j--) {
    num1 = 0;
    for (i = deg_omega; i >= 0; i--) {
        if (omega[i] != A0)
            num1 ^= Alpha_to[modnn(omega[i] + i * root[j])];
    }
    num2 = Alpha_to[modnn(root[j] * (B0 - 1) + NN)];
    den = 0;
    /* lambda[i+1] for i even is the formal derivative lambda_pr of lambda[i] */
    for (i = min(deg_lambda,NN-KK-1) & ~1; i >= 0; i -= 2) {
        if (lambda[i+1] != A0)
            den ^= Alpha_to[modnn(lambda[i+1] + i * root[j])];
    }
    if (den == 0) {
        if (DEBUG >= 1)
            printf("in ERROR: denominator = 0n");
        if (DEBUG >= 2)
            printf("errno = %d\n",errno);
    } else {
        if (DEBUG >= 1)
            printf("lambda[i+1] not 0\n");
        if (DEBUG >= 2)
            printf("errno = %d\n",errno);
    }
} else {
    printf("in ERROR: denominator = 0n");
    if (DEBUG >= 1)
        printf("errno = %d\n",errno);
    if (DEBUG >= 2)
        printf("errno = %d\n",errno);
}

/* Convert to dual-basis */
count = -1;
goto finish;
} /* Apply error to data */
if (num1 != 0) {
    data[loc[i]] ^= Alpha_to[modnn(Index_of[num1] + Index_of[num2] + NN - Index_of[den])];
} else {
}
finish:
#define CCSDS
/* Convert to dual-basis */
for (i = 0; i < NN; i++)
    data[i] = tbltab[data[i]];
#endif
if (eras_pos != NULL)
    for (i = 0; i < count; i++)
        if (eras_pos[i] != NULL)
            eras_pos[i] = loc[i];
return count;
} /* Encoder/decoder initialization - call this first! */
static void init_rs(void)
{
    generate_gf();
    gen_poly();
#ifdef CCSDS
    gen_ltab();
#endif
#ifdef PRIM /= 1
    gen_ldec();
#endif
    RS_init = 1;
}
/* Generate gaussian random double with specified mean and std_dev */
double normal_rand(double mean, double std_dev)
{
    double fac, rsq, v1, v2;
    static double gset;
    static int iset;

    if (iset){
        /* Already got one */
        iset = 0;
        return mean + std_dev * gset;
    }
    /* Generate two evenly distributed numbers between -1 and +1 */
    /* that are inside the unit circle */
    do {
        v1 = 2.0 * (double)(rand()*65538) / MAX_RANDOM - 1;
        v2 = 2.0 * (double)(rand()*65538) / MAX_RANDOM - 1;
        rsq = v1 * v1 + v2 * v2;
    } while (rsq >= 1.0 || rsq == 0.0);
    fac = sqrt(-2.0 * log(rsq) / rsq);
    gset = v1 * fac;
    iset++;
    return mean + std_dev * v2 * fac;
}

int modnoise(unsigned int *bidata, /* Input and Output bits*/
             unsigned int nsyms, /* Bit count */
             double noise /* Noise amplitude */)
{
    double s;
    int bit_err = 0;

    while (nsyms-- != 0) {
        s = normal_rand(0, noise);
        if (*bidata == 0)
            if (s < 0.5)
                *bidata++;
        if (*bidata == 1)
            { s += 1;
                if (s < 0.5)
                    *bidata++;
            }
        if (s < 0.5)
            s = 0.0;
        else
            s = 1.0;
        *bidata = (unsigned char) s;
        bidata++;
    }
    return bit_err;
}
int stringToBinary(char *s) {
    int index=0,i,integer,count,tempArray[8];
    while (*s != '0') {
        if (*s == 0) {
            binary[index] = 0;
            /* convert decimal number to binary number*/
            for (integer = *s;integer != 0;)
            {
                binary[index] = integer%2;
                integer = integer/2;
                index++;
            }
            for ((index%8) != 0;index++)
                binary[index] = 0;
            /* sort the 8-binary digit*/
            count = index;
            for (i=0;i<8;i++)
                tempArray[i]=binary[~count];
            for (i=0;i<8;i++)
                binary[count++]=tempArray[i];
            s++;
        }
        return index;
    }

    /*generate random bits*/
    void gen_bit(long bit_len) {
        long t,i=0;
        for (t = 0; t < bit_len; t++) {
            i = rand()%2;
            *(binary + t) = i;
        } 
    }

    main() {
        unsigned char data1[NN];
        unsigned char data2[NN];
        double ebno0,noise, sd,throughput,sigma1,sigma2;
        int nerror,total_err,bit_err,header;
        int eras_pos[NN],total_corr;
        char c,string[10000];
        int a,b,i=0,j,k,m,n,base;
        int f,J, integer;
        int index1,input,byte_len,byte_len2,count;
        long bit_len;

        printf("<<<This is a Error Control Coding for DVB-S simulator>>>
        *n*); do

        }
```c
{
    printf("Enter a type of input you want to use:\n");
    printf("1->Random generating input OR 2->self-generating input\n");
    printf("3->Exit\n");
    scanf("%d", &input);
    printf("\n");
    }while ((input != 1) & & (input != 2) & & (input != 3));

    if (input == 1)
    {
        printf("Enter the No. of bits use to simulate:\n");
        scanf("%d", &bit_len);
        printf("Enter the No. of Eb\nN0:\n");
        scanf("%lf", &ebn0);
        if (ebn0 == 1)
            sd = 0.312;
        else
            {
                sigma1 = (double)1/10;
                sigma2 = (double)ebn0/10;
                sd = (double)sqrt(pow(10,sigma1)/pow(10,(sigma2)))*0.312 + 0.002;
            }

        noise = sd * sqrt(1/(double)KK/NN));

        byte_len = 0;
        gen_bit(bit_len);
        for (i=0;i<bit_len;)
        {
            count=0;
            for (j=0;j<8;j++)
                if (binary[i++]) != 0)
                    count += (int)pow(2,j);
            ori_data[byte_len++] = count;
        }

        /*divide the continuous data bytes into 223-byte data*/
        m=0; l=0; a=0;
        count = byte_len;
        while (m < byte_len)
        {
            for (j=0,k=0;j<KK;j++)
            {
                if (j != count)
                {
                    data1[j] = ori_data[l++];
                    m++; k++;
                }
                else
                    break;
            }
        }

        /*to make the data into 233-byte data in case that the input data is not
        * equal to 233 bytes*/
        while (j != KK)
        {
            data1[j++] = 0;
            m++; k++; j++;
        }
```
count -= k;

encode_rs(data1,&data1[KK]);

for (b=0;b<NN;b++)
    data[a++] = data1[b];

memcpy(ddata,data,a);

/*convert decimal digits to binary digits*/
base=0;
J=0;
while (base<a)
{
    if (ddata[base] == 0)
        bidata[J++] = 0;
    for (integer=ddata[base];integer!=0;)
    {
        bidata[J++] = integer%2;
        integer = integer/2;
    }
    while ((J%8)!=0)
        bidata[J++] = 0;
    base++;
}

/*add noise*/
bit_err = modnoise(bidata,J,noise);

/*convert binary digits to decimal digits*/
I=0;
base=0;
while (base<J)
{
    count=0;
    for (j=0;j<8;j++)
    {
        if (bidata[base] != 0)
            count += (int)pow(2,j);
        base++;
    }
    decidata[I++] = count;
}

k = 0;
m = 0;
base = 0;
total_err=0;
total_corr = 0;
while (base<a)
{
    nerror = 0;
    /*find the errored position*/
    for (n=0;n<NN;n++)
    {
data2[n] = decidata[n+base];
data1[n] = data[n+base];
if(data1[n] != data2[n])
{
    eras_pos[j++] = n;
    nerror++;
}
}
total_err += nerror;
if (nerror<(NN-KK))
    i = dec_rs(data2,eras_pos,nerror);
else
    i = 0;
total_corr += i;
for (n=0; n<KK; n++)
    final_data[k++] = data2[n];
base += NN;
m += KK;
}
m = (int)ceil((float)byte_len/223.0);
byte_len2 = m*KK;
header = m*(NN-KK);
throughput = (float)byte_len2/(byte_len2+header);
/* find number of errors after decoding */
for (j=0, count=0; j<byte_len;j++)
    if (final_data[j] != ori_data[j])
        count++;
printf("Before decoding>>\n");
printf("No. of bit errors: %dn",bit_err);
printf("Probability of bit error: %fn", (float)bit_err/(float)bit_len);
printf("No. of byte errors: %dn",total_err);
printf("Probability of byte error: %fn", (float)total_err/(float)((byte_len2)+header));
printf("After decoding>>\n");
printf("Throughput is: %fn",throughput);
if (i != -1)
    printf("Errors + Erasures corrected (bytes): %dn",total_corr);
printf("No. of byte errors: %dn",count);
printf("Probability of byte error: %fn",(float)count/(float)byte_len);
}
if (input == 2) 
{
if (getchar() == \n); /* Because you must press enter (\n) whenever you select
 * the type of input, so the statement above is needed to
 * prevent the string[] array to get nothing from the below
 * statement*/
printf("Enter a line of text: ");
while (( c = getchar() ) != \n)
    string[i++] = c;
string[i] = \0;
index1 = stringToBinary(string);
printf("Enter No. of Eb/N0:");
scanf("%d", &ebn0);

if(ebn0 == 1)
    sd = 0.312;
else
{
    sigma1 = (double)1/10;
    sigma2 = (double)ebn0/10;
    sd = (double)sqrt(pow(10,sigma1/pow(10,(sigma2)))*0.312 + 0.002);
}

noise = sd * sqrt(1/(double)KK/NN));

byte_len = 0;
for (i=0;i<index1.;)
{
    count=0;
    for (j=7;j>=0;j--)
    {
        if (binary[i++]) != 0)
            count += (int)pow(2,j);
        ori_data[byte_len++] = count;
    }
}

/*divide the continuous data bytes into 223-byte data*/
m=0; i=0; a=0;
count = byte_len;
while(m < byte_len)
{
    for(j=0,k=0;j<KK,j++)
    {
        if (j != count)
        {
            data[j] = ori_data[i++];
            m++; k++;
        }
        else
        {break;
    }
}

/*to make the data into 233-byte data in case that the input data is not
equal to 233 bytes*/
while (j != KK)
{
    data[j++] = 0;
    m++; k++;
}

count -= k;
encode_rs(data1,&data1[KK]);

for (b=0;b<NN;b++)
    data[a++] = data1[b];
}

memcpy(ddata,data,a);

/*convert decimal digits to binary digits*/
base=0;
J=0;
while (base<a)
{
    if (ddata[base] == 0)
        bidata[J++] = 0;
    for (integer=ddata[base];integer!=0;)
    {
        bidata[J++] = integer%2;
        integer = integer/2;
    }
    while ((J%8)!=0)
        bidata[J++] = 0;
    base++;
}

/*add noise*/
b_bit_err = modnoise(bidata,1,noise);

/*convert binary digits to decimal digits*/
I=0;
based=0;
while (base<\)
{
    count=0;
    for (j=0;j<8;j++)
    {
        if (bidata[base] != 0)
            count += (int)pow(2,j);
        base++;
    }
    decidata[I++] = count;
}
printf("\nBefore decoding>>\n");
printf("The text is: ");
based = 0;
while (base<\)
{
    for (m=0;m<KK;m++)
        if (m<base_len)
            printf("%c", decidata[m]);
        else
            break;
    base += NN;
}
printf("\n");
k = 0;
m = 0;
based = 0;
total_err = 0;
total_corr = 0;
while (base<a)
{
    nerror = 0;
    /*find the errored position*/
    for (n=0;j=0;n<NN,n++)
    {
        data2[n] = decidata[n+base];
data1[n] = data[n+base];
if (data1[n] != data2[n])
{
    eras_pos[j++] = n;
    nerror++;
}
}
total_err += nerror;

if (nerror<=(NN-KK))
    i = dec_rs(data2,eras_pos,nerror);
else
    i = 0;
total_corr += i;
for (n=0;n<KK;n++)
    final_data[k++] = data2[n];
base += NN;
m += KK;
}

printf("\nAfter decoding>>\n");
printf("The text is: ");
for (m=0,m<byte_len,m++)
    printf("%c", final_data[m]);
printf("\n");
for (j=0,count=0,j<byte_len;j++)
    /* find number of errors after decoding */
    if (final_data[j] != ori_data[j])
        count++;
if (i != -1)
    printf("Errors + Erasures corrected: %d\n",total_corr);
printf("No. of byte errors: %d\n",count);
printf("probability of byte error: %.6f\n",(float)count/(float)byte_len);

if (input == 3)
    exit(0);

return 0;
Experiment and Analysis

For the first experiment we are testing the ability to recover the message with error back to the original message by using Reed-Solomon (255,249) with 8 bits symbol. So in this case we can detect error up to 3 error only. Then first we set up the initial parameter in the R-S input. We set no. of parity symbols and data symbols to 6 and 249 respectively. Then the program prepares R-S (255,249) encoder and decoder with 8-bit symbol (3 error recoverable). We need to generate the input file first. In this test we use data.inp for data to be encode.

In the figure above show the elapsed time that is occur from the encoder. The output data is recorded in the data.en file. This file contains the encoded message. As we are doing the simulation, we need to generate the noise. In this program we use error insert to simulate the noise that occur in the transmission channel. In this experiment, we use 3 errors and 1/1000 Bit error rate (BER). Then at the receiver we need decoder with the RS (255,249) or the same rate as the decoder. Decoder receives the message with error from the file “data.err”. Then the decoder recovers the message back.
The results of the decoder can detect and recover the message back. At last we compare the data input with the data.out.

As the result shown that there are 0 difference per code word so that mean the whole code word can be detect back.

At the same RS encoder and decoder set if we increase the number of error exceed than 3 (t in this experiment is set to be 3)
The result is the decoder cannot recover the message.

From this experiment, we can conclude that the RS error control coding can detect and recover the error message back if the number of error does not exceed the value of $t$.

For the second program, this program is testing for the probability of error. In this program we need to put the parameter the number of bit to simulate and $Eb/N0$. The result of this experiment is listed in the table below.

<table>
<thead>
<tr>
<th>No. of bits to simulate</th>
<th>No. of Eb/N0</th>
<th>Before Decoding</th>
<th>After Decoding</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No. of byte errors</td>
<td>Probability of byte error</td>
</tr>
<tr>
<td>100000</td>
<td>0.05</td>
<td>7854</td>
<td>0.40351</td>
</tr>
<tr>
<td>100000</td>
<td>0.075</td>
<td>7815</td>
<td>0.537689</td>
</tr>
<tr>
<td>100000</td>
<td>0.1</td>
<td>7781</td>
<td>0.535329</td>
</tr>
<tr>
<td>100000</td>
<td>0.125</td>
<td>7743</td>
<td>0.532714</td>
</tr>
<tr>
<td>100000</td>
<td>0.15</td>
<td>7714</td>
<td>0.530719</td>
</tr>
<tr>
<td>100000</td>
<td>0.175</td>
<td>7672</td>
<td>0.527829</td>
</tr>
<tr>
<td>100000</td>
<td>0.2</td>
<td>7640</td>
<td>0.525628</td>
</tr>
<tr>
<td>100000</td>
<td>0.225</td>
<td>7605</td>
<td>0.52322</td>
</tr>
<tr>
<td>100000</td>
<td>0.25</td>
<td>7565</td>
<td>0.520465</td>
</tr>
<tr>
<td>100000</td>
<td>0.275</td>
<td>7527</td>
<td>0.517853</td>
</tr>
<tr>
<td>100000</td>
<td>0.3</td>
<td>7492</td>
<td>0.515445</td>
</tr>
<tr>
<td>100000</td>
<td>0.325</td>
<td>7460</td>
<td>0.513244</td>
</tr>
<tr>
<td>100000</td>
<td>0.35</td>
<td>7418</td>
<td>0.510354</td>
</tr>
<tr>
<td>100000</td>
<td>0.375</td>
<td>7376</td>
<td>0.507465</td>
</tr>
<tr>
<td>100000</td>
<td>0.4</td>
<td>7337</td>
<td>0.504782</td>
</tr>
<tr>
<td>100000</td>
<td>0.425</td>
<td>7296</td>
<td>0.501961</td>
</tr>
<tr>
<td>100000</td>
<td>0.45</td>
<td>7266</td>
<td>0.499897</td>
</tr>
<tr>
<td>100000</td>
<td>0.475</td>
<td>7222</td>
<td>0.49687</td>
</tr>
<tr>
<td>100000</td>
<td>0.5</td>
<td>7187</td>
<td>0.494462</td>
</tr>
</tbody>
</table>
If we look at the table above we find out that the number of byte errors is decreasing. That means the RS error control coding is increasing the performance of the transmission over laser satellite link. Laser satellite link is coded by RS(255,223). This test is using RS(255,223).
Conclusions

Laser Satellite communication have many advantage over the Radio Frequency (RF) such as their architecture or application. The Laser Satellite system is the transfer of data from one satellite to its companion. In most cases, established classical modulation schemes are used. Transfer is variety of using the any technique to implement the transmission of data. And in the transmission, we need the security for the reliability in the system. Therefore, this project is to make the software tool to simulate the error control coding and decoding for this Laser Satellite system.

In this project, we simulate the RS encoder/decoder. First we generate the input file and use RS encoder to encode the input data. Then to test the ability to correct error we need to add the error into the channel. At the decoder the data with error will be decode by the decoder. If the number of error is less than the t parameter (the symbol that we can correct the error), the decoder can recover the error. But if the error is greater than the t parameter, the decoder cannot recover the error.

So we can conclude that the RS error correcting code have the ability to correct the error symbol up to the t parameter.

In the second test, we test for the performance of RS coding that effect on decreasing of no.of byte errors. The result has shown out that the no. of byte error is decreasing if we compare the no. of byte error before decoding and after decoding.
**Recommendation**

1. The program should have the calculation of probability of error for checking the probability of received data, which will error before decoding.

2. The advance version of program should have the ability to do graphing for compare the resulted graph of encoding and decoding.

3. In the advance of the program might have the graphic picture to show the process of transmission.
Reference

Text:

Website:
1. http://www.4i2i.com/reed_solomon_codes.htm, Copyright 1998 by 4i2i Communications Ltd.