PENERAPAN GENERATOR POLINOMIAL UNTUK OPTIMASI DETEKSI DAN KOREKSI KESALAHAN BIT PADA BOSE-CHAUDHURI-HOCQUENGHEM *CODE*

SKRIPSI

OLEH
DEWI SUSILOWATI
NIM. 210601110066

PROGRAM STUDI MATEMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG
2025

PENERAPAN GENERATOR POLINOMIAL UNTUK OPTIMASI DETEKSI DAN KOREKSI KESALAHAN BIT PADA BOSE-CHAUDHURI-HOCQUENGHEM *CODE*

SKRIPSI

Diajukan Kepada
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang
untuk Memenuhi Salah Satu Persyaratan dalam
Memperoleh Gelar Sarjana Matematika (S.Mat)

Oleh
Dewi Susilowati
NIM. 210601110066

PROGRAM STUDI MATEMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG
2025

PENERAPAN GENERATOR POLINOMIAL UNTUK OPTIMASI DETEKSI DAN KOREKSI KESALAHAN BIT PADA BOSE-CHAUDHURI-HOCQUENGHEM *CODE*

SKRIPSI

Oleh
Dewi Susilowati
NIM. 210601110066

Telah Disetujui untuk Diuji
Malang, 28 Mei 2025

Dosen Pembimbing I	Dosen Pembimbing II
Prof. Dr. H. Turmudi, M.Si., Ph.D	Mohammad Nafie Jauhari, M.Si
NIP. 19571005 198203 1 006	NIPPPK. 19870218 202321 1 018

Mengetahui,
Ketua Program Studi Matematika

Dr. Elly Susanti, M.Sc
NIP. 19741129 200012 2 005

PENERAPAN GENERATOR POLINOMIAL UNTUK OPTIMASI DETEKSI DAN KOREKSI KESALAHAN BIT PADA BOSE-CHAUDHURI-HOCQUENGHEM *CODE*

SKRIPSI

Oleh
Dewi Susilowati
NIM. 210601110066

Telah Dipertahankan di Depan Dewan Penguji Skripsi
dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
untuk Memperoleh Gelar Sarjana Matematika (S.Mat)

Tanggal 18 Juni 2025

Ketua Penguji	: Hisyam Fahmi, M.Kom	.....
Anggota Penguji 1	: Muhammad Khudzaifah, M.Si	.....
Anggota Penguji 2	: Prof. Dr. H. Turmudi, M.Si., Ph.D	.....
Anggota Penguji 3	: Mohammad Nafie Jauhari, M.Si	.....

Mengetahui,
Ketua Program Studi Matematika

Dr. Elly Susanti, M.Sc
NIP. 19741129 200012 2 005

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan di bawah ini:

Nama	: Dewi Susilowati
NIM	: 210601110066
Program Studi	: Matematika
Fakultas	: Sains dan Teknologi
Judul Skripsi	: Penerapan Generator Polinomial untuk Optimasi Deteksi dan Koreksi Kesalahan Bit pada Bose-Chaudhuri-Hocquenghem *Code*.

Menyatakan dengan sebenarnya bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya sendiri, bukan merupakan pengambilan data, tulisan, atau pikiran orang lain yang saya akui sebagai hasil tulisan dan pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka. Apabila di kemudian hari terbukti atau dapat dibuktikan skripsi ini hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 18 Juni 2025
Yang membuat pernyataan,

Dewi Susilowati
NIM. 210601110066

MOTO

“Segala kemungkinan terbuka bagi mereka yang berani mencoba dan tidak takut menghadapi tantangan”

PERSEMBAHAN

Bismillahirrahmanirrahim

Segala puji bagi Allah SWT yang senantiasa memberikan pertolongan dan kemudahan kepada penulis dalam melewati segala proses penyelesaian skripsi ini. Skripsi ini penulis persembahkan kepada:

Ayah dan ibu yang telah berkorban memberikan doa, dukungan, dan nasihat terbaik untuk kesuksesan penulis. Kepada saudara-saudari penulis yang selalu memberikan doa dan semangat kepada penulis. Sahabat-sahabat penulis yang dengan ikhlas memberikan bantuan dan semangat dalam menyelesaikan skripsi ini.

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Allah SWT atas segala rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan skripsi yang berjudul "Penerapan Generator Polinomial untuk Optimasi Deteksi dan Koreksi Kesalahan Bit pada *Bose-Chaudhuri-Hocquenghem Code*". Skripsi ini disusun sebagai bagian dari pemenuhan beban studi untuk mata kuliah Skripsi Matematika di Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Penulis menyadari bahwa penyelesaian skripsi ini tidak terlepas dari bantuan, bimbingan, dan dukungan dari berbagai pihak. Oleh karena itu, penulis ingin menyampaikan ucapan terima kasih yang sebesar-besarnya kepada:

1. Prof. Dr. H. M. Zainuddin, M.A., selaku rektor Universitas Islam Negeri Maulana Malik Ibrahim.
2. Prof. Dr. Hj. Sri Harini, M.Si, selaku dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim.
3. Dr. Elly Susanti, S.Pd, M.Sc, selaku ketua Program Studi Matematika, Universitas Islam Negeri Maulana Malik Ibrahim.
4. Prof. Dr. H. Turmudi, M.Si., Ph. D, selaku dosen pembimbing I yang telah memberikan bimbingan dan arahan dengan penuh kesabaran dan ketelitian.
5. Mohammad Nafie Jauhari, M, Si, selaku dosen pembimbing II yang telah memberikan bimbingan dan arahan dengan penuh kesabaran dan ketelitian.
6. Hisyam Fahmi, M.Kom., selaku ketua penguji dalam ujian skirpsi yang telah memberikan arahan, nasihat, serta saran yang bermanfaat kepada penulis.
7. Muhammad Khudzaifah, M.Si., selaku anggota penguji I dalam ujian skirpsi yang telah memberikan arahan, nasihat, serta saran yang bermanfaat kepada penulis.
8. Seluruh dosen Program Studi Matematika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim.
9. Ayah Hadi Bing Slamet, Ibu Sunami, Siti Aisya Nafeeza dan Muhammad Yusuf Alviano Putra, serta keluarga besar atas doa, dukungan, dan cinta kasih yang selalu menyertai. Segala bentuk dukungan menjadi kekuatan

dalam menyelesaikan skripsi ini. Semoga segala kebaikan yang telah diberikan senantiasa mendapatkan balasan dari Allah SWT.

10. Sahabat penulis, Indah Mariyati, Robi'atul Adawiyyah, penghuni grup "Alip gamau diajak", "spal spil", dan "OTDR" yang senantiasa memberikan semangat, dukungan serta waktu yang berharga dalam masa-masa proses penyusunan skripsi hingga selesai.
11. Seluruh teman-teman TEOREMA Program Studi Matematika Angkatan 2021 yang senantiasa memberikan semangat kepada penulis.

Semoga Allah SWT. Memberikan balasan atas segala bantuan dan kebaikan yang telah diberikan kepada penulis. Penulis menyadari masih terdapat kekurangan dalam penyusunan laporan penelitian ini. Penulis berharap adanya skripsi ini dapat memberikan manfaat dan wawasan bagi penulis maupun pembaca.

Malang, 18 Juni 2025

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR SIMBOL

C : Himpunan kode valid

$c(x)$: Polinomial kode valid

$g(x)$: Polinomial generator

$r(x)$: Bit paritas

t : jumlah maksimum kesalahan bit yang dapat dikoreksi

n : panjang total bit

k : panjang informasi

DAFTAR LAMPIRAN

ABSTRAK

Susilowati, Dewi. 2025. **Penerapan Generator Polinomial untuk Optimasi Deteksi dan Koreksi Kesalahan Bit pada Bose-Chaudhuri-Hocquenghem *Code.*** Skripsi. Program Studi Matematika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: (I) Prof. Dr. H. Turmudi, M.Si., Ph.D. (II) Mohammad Nafie Jauhari, M.Si.

Kata Kunci: Efisiensi Transmisi, Generator Polinomial, $GF(2^4)$, Kode BCH, Koreksi Kesalahan, Lapangan.

Perkembangan teknologi komunikasi digital mendorong kebutuhan akan sistem transmisi data yang efisien, andal dan mampu mengatasi gangguan seperti *noise* yang berpotensi menimbulkan kesalahan bit. Salah satu metode yang efektif dalam mendeteksi dan memperbaiki kesalahan tersebut adalah kode *Bose-Chaudhuri-Hocquenghem* (BCH), yang dikembangkan berdasarkan teori polinomial dalam lapangan (*Galois Field*). Penelitian ini bertujuan untuk mengkaji penerapan generator polinomial pada kode BCH dalam optimalisasi deteksi dan koreksi kesalahan bit. Simulasi dilakukan pada lapangan $GF(2^4)$ dengan menggunakan tiga konfigurasi parameter kode BCH, yaitu BCH(15,11), BCH(15,7), dan BCH(15,5), masing-masing dengan tingkat kemampuan koreksi kesalahan $t = 1, t = 2,$ dan $t = 3$. Polinomial *irreducible* yang digunakan adalah $p(x) = x^4 + x + 1$ untuk membentuk generator polinomial $g(x)$. Proses *encoding* dilakukan dengan mengalikan data biner dengan generator polinomial, sedangkan *decoding* melibatkan perhitungan sindrom untuk mendeteksi dan memperbaiki kesalahan. Hasil penelitian menunjukkan bahwa semakin kecil nilai k, maka kemampuan koreksi kesalahan meningkat, tetapi efisiensi transmisi menurun. Parameter BCH (15,11) memberikan efisiensi transmisi tertinggi, sedangkan BCH(15,5) menunjukkan performa terbaik dalam memperbaiki kesalahan. Dengan demikian, pemilihan parameter kode BCH harus disesuaikan dengan kebutuhan sistem untuk mencapai keseimbangan antara efisiensi dan keandalan transmisi data.

ABSTRACT

Susilowati, Dewi. 2025. **Application of Generator Polynomials for Optimizing Bit Error Detection and Correction in Bose-Chaudhuri-Hocquenghem *Code.*** Undergraduate Thesis. Mathematics Departement, Faculty Science and Technology, Universitas Islam Negeri Maulana Malik Ibrahim Malang. Advisors: (I) Prof. Dr. H. Turmudi, M.Si., Ph.D. (II) Mohammad Nafie Jauhari, M.Si.

Keywords: BCH Code, Error correction, Finite Field, Generator Polynomial, $GF(2^4)$, Transmission Efficiency.

The advancement of digital communication technology demands data transmission systems that are efficient, reliable, and capable of overcoming disturbances such as noise, which can cause bit errors. One effective method for detecting and correcting such errors is the Bose–Chaudhuri–Hocquenghem (BCH) code, developed based on polynomial theory in finite fields (Galois Field). This study aims to examine the application of generator polynomials in BCH codes for optimizing bit error detection and correction. Simulations were carried out using the finite field $GF(2^4)$ with three BCH code parameter configurations: BCH(15,11), BCH(15,7), and BCH(15,5), each with error correction capabilities of $t = 1, t = 2$ and $t = 3$, respectively. The irreducible polynomial used was $p(x) = x^4 + x + 1$ to construct the generator polynomial $g(x)$. The encoding process multiplies binary data by the generator polynomial, while decoding involves syndrome calculation to detect and correct errors. The results show that as the value of k decreases, the error correction capability increases, but transmission efficiency decreases. BCH(15,11) provides the highest transmission efficiency, whereas BCH(15,5) offers the strongest error correction performance. Therefore, selecting the appropriate BCH parameters is crucial to balancing data transmission efficiency and reliability.

مستلص البحث

سوسيلاواتي، دوي. ٢٠٢٥. **تطبيق كثيرات الحدود المولدة لتحسين كشف وتصحيح أخطاء البت في شيفرة بوز–شودوري–هوكوينجهيم**. البحث العلمي، قسم الرياضيات، كلية العلوم والتكنولوجيا، جامعة مولانا مالك إبراهيم الإسلامية الحكومية مالانج. المشرفون: (١) الأستاذ الدكتور . تورمودي، ماجستير في العلوم.(٢) محمد نافي جوهري، ماجستير في العلوم.

الكلمات الأماسة :شيفرة BCH، تصحيح الأخطاء، الحقل المنتهي، $GF(2^4)$، كثير الحدود المولد، كفاءة النقل، المتلازمة .

تتطلب تطورات تقنيات الاتصالات الرقمية أنظمة نقل بيانات تكون فعالة وموثوقة وقادرة على التغلب على التشويش مثل الضوضاء، والتي قد تؤدي إلى أخطاء في البتّات. تُعدّ شيفرة *Bose–Chaudhuri–Hocquenghem* (BCH) من الأساليب الفعالة في كشف وتصحيح هذه الأخطاء، حيث تم تطويرها بالاعتماد على نظرية كثيرات الحدود في الحقول المنتهية .(*Galois Field*) هدف هذا البحث إلى دراسة تطبيق كثيرات الحدود المولدة في شيفرات BCH من أجل تحسين عملية اكتشاف وتصحيح أخطاء البت. تم إجراء المحاكاة باستخدام الحقل المنتهي $GF(2^4)$ مع ثلاث مجموعات من معلمات شيفرة BCH: $BCH(15,11)$, $BCH(15,7)$,$BCH(15,5)$، بقدرات تصحيح أخطاء قدرها $t = 1, t = 3, 2$ على التوالي. تم استخدام كثير الحدود غير القابل للاختزال $p(x) = x^4 + x + 1$ لتكوين كثير الحدود المولد $g(x)$.يتضمن الترميز ضرب البيانات الثنائية في كثير الحدود المولد، بينما تتضمن عملية فك الترميز حساب المتلازمة لاكتشاف وتصحيح الأخطاء. أظهرت النتائج أنه كلما انخفضت قيمة k ، زادت قدرة تصحيح الأخطاء، لكن انخفضت كفاءة النقل. توفّر $BCH(15,11)$ أعلى كفاءة في النقل، في حين تُظهر $BCH(15,5)$ أفضل أداء في تصحيح الأخطاء. وبالتالي، فإن اختيار المعلمات المناسبة لشيفرة BCH ضروري لتحقيق التوازن بين كفاءة النقل وموثوقيته.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Pada era perkembangan teknologi komunikasi dan informasi saat ini, kebutuhan akan sistem transmisi data yang cepat dan andal semakin meningkat. Perkembangan teknologi komunikasi dan informasi telah memungkinkan berbagai perangkat elektronik, seperti komputer, *smartphone*, dan jaringan internet bergantung pada proses pengiriman data yang efisien. Namun, dalam prosesnya sering terjadi gangguan yang dapat menyebabkan kesalahan dalam pengiriman informasi. Salah satu gangguan utama dalam transmisi data adalah *noise*.

Noise merupakan sinyal yang tidak diinginkan dalam proses transmisi data yang dapat mengganggu atau merusak struktur sinyal asli. Gangguan ini dapat berasal dari berbagai sumber, seperti interferensi elektromagnetik, fluktuasi daya sinyal, hingga gangguan cuaca dalam komunikasi nirkabel. *Noise* dapat menyebabkan distorsi pada sinyal yang dikirim sehingga informasi yang diterima menjadi tidak akurat atau bahkan tidak dapat dibaca. Akibat dari gangguan ini, kualitas komunikasi data dapat menurun, menyebabkan hilangnya informasi atau terjadinya kesalahan dalam proses penerimaan (Almazrouei, 2024).

Kesalahan bit (*bit error*) juga menjadi dampak yang mengakibatkan perbedaan antara informasi yang dikirim dan diterima. Hal tersebut dapat merubah nilai bit selama transmisi data. Selain itu, kesalahan bit dapat menyebabkan berbagai gangguan dalam komunikasi, seperti distorsi suara dalam panggilan telepon, tampilan gambar yang rusak pada video, atau bahkan kegagalan dalam

proses pengiriman *file*. Jika tidak diperbaiki, kesalahan ini dapat menyebabkan misinterpretasi data dan menurunkan keandalan sistem komunikasi (Sachenko, 2009).

Berbagai teknik deteksi dan koreksi kesalahan diterapkan untuk memastikan keandalan sistem komunikasi serta menjaga integritas data selama proses transmisi. Salah satu teknik yang efektif dalam mengatasi gangguan akibat *noise* adalah *Bose-Chaudhuri-Hocquenghem* (BCH) *Code*, yang termasuk dalam kategori *Error Correcting Code* (ECC) dan didasarkan pada teori polinomial dalam *Galois Field* (GF). Kode *Bose Chaudhuri Hocquenghem* (BCH) mampu mendeteksi dan memperbaiki beberapa bit kesalahan dalam satu blok data, sehingga sangat cocok digunakan dalam sistem komunikasi digital, penyimpanan data, dan aplikasi lain yang memerlukan tingkat keandalan tinggi.

Generator polinomial dalam kode BCH merupakan elemen fundamental dalam proses deteksi dan koreksi kesalahan. Generator ini digunakan untuk menghasilkan kode kata dengan menambahkan bit redundan ke dalam data asli, sehingga memastikan bahwa data yang dikirim memiliki struktur yang memungkinkan pendeteksian kesalahan di sisi penerima. Pemilihan generator polinomial yang tepat sangat berpengaruh terhadap efisiensi dan kemampuan koreksi kesalahan dari kode BCH, terutama dalam sistem komunikasi yang rentan terhadap gangguan.

Pada perancangan kode BCH, pembentukan generator polinomial dilakukan dengan memanfaatkan struktur aljabar pada lapangan atau *Galois Field* $GF(2^m)$. Medan ini terdiri atas 2^m elemen dan bersifat tertutup terhadap operasi

penjumlahan dan perkalian. Setip elemen dalam $GF(2^m)$ dapat direpresentasikan sebagai polinomial biner berderajat maksimum $m-1$.

Lapangan $GF(2^m)$ memiliki sejumlah keunggulan yang menjadikannya sangat penting dalam pengembangan sistem pengkodean digital khususnya dalam penerapan kode koreksi kesalahan. Salah satu keunggulan utamanya adalah kemampuannya dalam mendukung operasi-operasi aljabar seperti penjumlahan, perkalian, dan pembagian polinomial secara efisien dan konsisten dalam sistem biner. Sifat tertutup dari $GF(2^m)$ terhadap operasi-operasi tersebut memastikan kestabilan dan konsistensi hasil komputasi, sehingga memungkinkan pembentukan struktur kode yang kuat dan andal.

Berbagai penelitian sebelumnya menunjukkan bahwa implementasi kode BCH banyak menggunakan *Galois Field* berorde rendah seperti $GF(2^3), GF(2^6),$ dan $GF(2^7)$. Pada penelitian Gravano (1990) mengembangkan algoritma *decoding* untuk kode BCH menggunakan parameter $n = 15$ dan $k = 5$ dengan kemampuan koreksi tiga bit kesalahan menggunakan $GF(2^3)$. Pendekatan ini memiliki keunggulan dalam kesederhanaan struktur implementasi serta efisiensi konsumsi daya, sehingga sangat sesuai untuk sistem dengan keterbatasan sumber daya.

Salah satu pendekatan penting dalam pembentukan generator polinomial adalah penggunakan *Galois Field* $GF(2^4)$. $GF(2^4)$ merupakan lapangan yang terdiri dari 16 elemen dan memungkinkan representasi data dalam empat bit biner. Dengan menggunakan $GF(2^4)$, sistem dapat membentuk generator polinomial yang lebih kompleks dan fleksibel, sehingga mendukung peningkatan panjang kode dan kapasitas koreksi kesalahan sesuai kebutuhan sistem. Pendekatan ini

memberikan keluasan dalam desain kode, tanpa menambah kompleksitas implementasi secara signifikan.

Pada proses *decoding*, algortima *Berlekamp-Massey* atau *Euclidean Algorithm* digunakan untuk menentukan lokasi kesalahan dan memperbaikinya tanpa perlu mengirim ulang data (Basu, 2014). Implementasi ini banyak diterapkan dalam berbagai teknologi, termasuk komunikasi satelit, penyimpanan data digital, serta sistem jaringan nirkabel yang membutuhkan keandalan tinggi dalam transmisi informasi.

Optimasi dalam penerapan kode BCH merupakan aspek penting dalam sistem komunikasi modern. Pada penelitian ini, optimasi difokuskan pada peningkatan kemampuan deteksi dan koreksi kesalahan dengan cara menerapkan generator polinomial yang sesuai dan menentukan parameter yang tepat, seperti panjang kode, panjang pesan, serta kapasitas koreksi kesalahan. Pemilihan generator polinomial yang optimal memungkinkan kode BCH untuk membentuk struktur kode yang mampu mengatasi gangguan secara efketif, sementara penentuan parameter yang sesuai memastikan bahwa sistem tetap efisien dalam penggunaan sumber daya. Dengan strategi optimasi yang ini, kode BCH mampu memaksimalkan kemampuan koreksi kesalahan dengan tetap menjaga efisiensi sistem, terutama dalam aplikasi yang membutuhkan performa tinggi seperti komunikasi satelit, jaringan 5G, sistem penyimpanan digital, serta perangkat *Internet of Things* (IoT) yang memiliki keterbatasan daya (Nabipour, 2024).

Keunggulan utama kode BCH adalah fleksibiltas dalam menyesuaikan tingkat koreksi kesalahan sesuai dengan kebutuhan sistem. Dengan penerapan yang tepat, kode BCH mampu meningkatkan keandalan komunikasi data, mengurangi

dampak *noise*, serta memastikan transmisi informasi yang lebih akurat dalam berbagai kondisi lingkungan. Selain itu, kode BCH juga memiliki efisiensi yang tinggi dalam penggunaan sumber daya, karena dapat mengoptimalkan jumlah bit redundan yang ditambahkan tanpa mengorbankan kapasitas data secara signifikan. Penerapan kode BCH juga mendukung efisiensi energi dalam perangkat komunikasi karena mengurangi kebutuhan pengulangan transmisi akibat kesalahan, yang pada akhirnya berkontribusi pada penghematan daya dalam sistem berbasis nirkabel dan kabel.

Deteksi dan koreksi kesalahan menjadi aspek penting dalam menjaga keakuratan informasi. Tanpa mekanisme ini, data yang disampaikan dapat terdistorsi, mengakibatkan keputusan yang keliru dan ketidakadilan. Oleh karena itu, seperti halnya islam mengajarkan pentingnya kehati-hatian dalam mencatat transaksi, sistem informasi juga harus menerapkan metode verifikasi koreksi yang ketat demi menjaga kebenaran dan keandalan data. Pentingnya memastikan integrasi dan akurasi data selama transmisi sejalan dengan prinsip kejujuran dan ketelitian yang diajarkan dalam Al-Qur'an. Allah SWT berfirman dalam surah Al-Hujurat ayat 6 yang artinya (Kementerian Agama, 2022):

"Wahai orang-orang yang beriman! Jika seseorang fasik datang kepadamu membawa suatu berita, maka telitilah kebenarannya agar kamu tidak mencelakakan suatu kaum karena ketidaktahuan yang akhirnya membuatmu menyesali perbuatanmu." (Q.S. Al-Hujurat:6).

Ayat ini menekankan pentingnya verifikasi informasi sebelum menyebarkannya agar tidak menimbulkan kesalahan atau keraguan. Hal ini mencerminkan perlunya sistem deteksi dan koreksi kesalahan seperti kode BCH untuk memastikan bahwa data yang dikirim tetap utuh dan dapat dipercaya. Dengan

menerapkan teknologi yang menjaga integritas informasi, manusia dapat menjalakan amanah dalam menyampaikan dan menerima data dengan benar, sebagaimana islam mengajarkan pentingnya keadilan dan ketelitian dalam setiap transaksi dan komunikasi.

Algoritma kode BCH telah banyak dibahas dalam penelitian sebelumnya sebagai salah satu teknik yang efektif dalam mendeteksi dan mengoreksi kesalahan bit. Namun, masih jarang penelitian yang menjelaskan secara rinci mengenai proses penggunaan generator polinomial dalam algoritma kode BCH. Oleh karena itu, penelitian ini akan mengkaji lebih dalam mengenai peran dan implementasi generator polinomial dalam kode BCH untuk deteksi dan koreksi kesalahan bit dan mengangkat penelitian ini dengan judul "Penerapan Generator Polinomial untuk Optimasi Deteksi dan Koreksi Kesalahan Bit pada Bos-Chaudhuri-Hocquenghem *Code*".

1.2 Rumusan Masalah

Berdasarkan latar belakang, rumusan permasalahan yang akan dibahas adalah bagaimana penerapan generator polinomial untuk optimasi deteksi dan koreksi kesalahan bit pada kode Bose-Chaudhuri-Hocquenghem?

1.3 Tujuan

Berdasarkan latar belakang dan rumusan masalah yang sudah ditentukan, penelitian ini bertujuan untuk mengevaluasi mengenai penerapan generator polinomial untuk optimasi deteksi dan koreksi kesalahan bit pada kode Bose-Chaudhuri-Hocquenghem.

1.4 Manfaat

Penelitian ini terdapat beberapa manfaat, beberapa diantaranya adalah sebagai berikut:

1. Manfaat Teoritis

   Penelitian ini memberikan manfaat teoritis untuk memberikan pemahaman mendalam tentang penerapan generator polinomial dalam deteksi dan koreksi kesalahan bit menggunakan kode BCH.

2. Manfaat Praktis

   a. Bagi Peneliti

      1) Menambah wawasan dan pemahaman peneliti mengenai teori kode.
      2) Menjadi pengalaman praktis dalam menerapkan metode BCH untuk meningkatkan keandalan pengiriman data.

   b. Bagi Pembaca

      Pembaca akan memperoleh pemahaman mengenai cara kerja kode BCH dalam mendeteksi dan mengoreksi kesalahan bit.

1.5 Batasan Masalah

Batasan masalah dalam penelitian ini terfokus pada penerapan generator polinomial dalam kontruksi kode BCH untuk deteksi dan koreksi kesalahan bit dengan menggunakan lapangan $GF(2^4)$, sehingga panjang kode maksimum yang dikaji adalah 15 bit.

1.6 Definisi Istilah

1. Bit

   Bit atau *Binary digit* adalah unit terkecil dalam sistem data digital yang hanya memiliki dua nilai, yaitu 0 atau 1. Bit merupakan dasar dari semua informasi yang diproses dan disimpan oleh perangkat komputer dan sistem digital. Nilai 0 dan 1 dalam bit merepresentasikan dua keadaan, seperti mati dan hidup, rendah dan tinggi, atau logika negatif dan positif.

2. *Encoding*

   Proses mengubah informasi dari suatu bentuk ke bentuk lain yang sesuai dengan sistem tertentu, biasanya dalam bentuk simbol, kode, atau format digital, agar dapat disimpan, diproses, atau dikirim secara lebih efisien.

3. *Decoding*

   Proses menerjemahkan atau mengubah kembali informasi yang telak dikodekan ke dalam bentuk aslinya agar dapat dipahami atau digunakan sebagaimana mestinya.

4. Redundan

   Suatu keadaan di mana sesuatu menjadi berlebihan, tidak diperlukan, atau memiliki pengulangan yang tidak perlu.

5. Generator Polinomial

   Polinomial yang digunakan untuk membentuk himpunan kode pada suatu kode linier tertentu, terutama pada kode siklik, di mana setiap kode merupakan kelipatan dari polinomial tersebut. Polinomial ini menentukan struktur dan karakteristik kode, seperti panjang serta kemampuan koreksi kesalahannya, dan beroperasi dalam lapangan.

6. Kode *Bose-Chaudhuri-Hocquenghem*

   Kode linier yang bersifat siklik yang dikontruksi menggunakan teori lapangan dan dirancang untuk mampu mengoreksi sejumlah kesalahan bit tertentu dalam suatu blok data.

BAB II

KAJIAN TEORI

2.1 Lapangan

Lapangan adalah himpunan bilangan yang dilengkapi dengan operasi penjumlahan dan perkalian, di mana kedua operasi tersebut bersifat komutatif. Lapangan memiliki elemen satuan dan semua unsur di R mempunyai *invers* terhadap operasi kedua kecuali elemen nol (identitas pada operasi pertama) (Raisinghania dan Anggarwal, 1980). Dengan kata lain, untuk setiap elemen bukan nol $p \in R$ terdapat $p^{-1} \in R$ sedemikian sehingga $p \cdot p^{-1} = 1$.

Suatu lapangan F dengan operasi (+) dan (·) harus memenuhi aksioma-aksioma berikut (Ling & Xing, 2004):

1. Tertutup

   Untuk semua $a, b \in F$ hasil penjumlahan $a + b$ dan perkalian $a \cdot b$ tetap berada dalam F.

2. Asosiatif

   Untuk semua $a, b, c \in F$ berlaku :

$$a + (b + c) = (a + b) + c \tag{2.1}$$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \tag{2.2}$$

3. Komutatif

   Untuk semua $a, b, c \in F$ berlaku:

$$a + b = b + a \tag{2.3}$$

$$a \cdot b = b \cdot a \tag{2.4}$$

4. Distributif

   Untuk semua $a, b, c \in F$ berlaku:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c) \quad (2.5)$$

$$(a + b) \cdot c = (a \cdot c) + (b \cdot c) \quad (2.6)$$

5. Memiliki elemen identitas

   a. Pada operasi penjumlahan $(+)$, terdapat elemen identitas 0 sehingga untuk semua $a \in F$:

$$a + 0 = 0 + a = a \quad (2.7)$$

   b. Pada operasi perkalian $(\cdot)$, terdapat elemen identitas 1 sehingga untuk semua $a \in F$:

$$a \cdot 1 = 1 \cdot a = a \quad (2.8)$$

6. Mempunyai elemen *invers*

   a. Untuk setiap $a \in F$ terdapat $x \in F$ yang disebut *invers* penjumlahan, yang memenuhi:

$$a + x = x + a = 0 \quad (2.9)$$

   Invers dari a dalam penjumlahan dinotasikan sebagai $-a$.

   b. Untuk setiap $a \neq 0$ terdapat $x \in F$ yang disebut *invers* perkalian, yang memenuhi:

$$a \cdot x = x \cdot a = 1 \quad (2.10)$$

   Invers perkalian dari a dinotasikan sebagi a^{-1}.

2.1.1 *Galois Field*

Galois field adalah suatu *field* yang memiliki jumlah elemen berhingga. *Galois Field* $GF(2^m)$ memiliki 2^m elemen, dimana p adalah bilangan prima, dan m merupakan derajat *ekstensi*. *Field* $GF(2^m)$ memiliki elemen yang dapat

direpresentasikan sebagai polinomial dengan koefisien dalam $\{0,1\}$, dengan setiap elemen memiliki derajat hingga $m-1$ (Rai, 2024). Setiap elemen dalam $GF(2^m)$ dapat ditulis dalam bentuk polinomial sebagai:

$$a(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0 \qquad (2.11)$$

di mana setiap koefisien $a_i \in \{0,1\}$. semua operasi dalam medan ini dilakukan pada modulo 2 (Schulz, 2024).

$GF(2^m)$ dibangun dengan memilih polinomial *irreducible* $p(x)$ berderajat m yang tidak dapat difaktorkan menjadi dua polinomial lain dengan koefisien di $GF(2)$. Setiap elemen pada $GF(2^m)$ merupakan sisa pembagian dari polinomial modulo $p(x)$ (Kyureghyan, 2020). Operasi dalam $GF(2^m)$ mencakup:

1. Penjumlahan: dilakukan dengan penjumlahan koefisien polinomial secara *bitwise* atau modulo 2.
2. Perkalian: dilakukan dengan mengalikan dua polinomial, lalu diambil hasilnya modulo $p(x)$.
3. *Invers* perkalian: dihitung dengan algoritma seperti *extended euclidean Algorithm*.

Pada pembentukan lapangan (*finite field*), pemilihan polinomial dengan sifat tertentu sangat penting. Salah satu jenis polinomial yang berperan dalam struktur dan operasi $GF(2^m)$ adalah polinomial *irreducible*, yang digunakan sebagai dasar dalam membangun lapangan. Selain itu, terdapat pula polinomial generator yang berperan dalam berbagai aplikasi, seperti kode koreksi kesalahan dan kriptografi. Kedua jenis polinomial ini memiliki peranan yang sangat penting dalam memastikan bahwa operasi dalam lapangan dapat berjalan secara efisien dan aman.

Polinomial *irreducible* adalah polinomial yang tidak dapat difaktorkan lebih jauh dalam $GF(2)$. Polinomial ini digunakan untuk membangun *finite field* $GF(2^m)$. Pada pembentukan $GF(2^m)$, polinomial *irreducible* dengan derajat m digunakan untuk menentukan representasi elemen medan dalam bentuk polinomial modulo polinomial *irreducible* yang dipilih.

Defnisi 2.1

Misalkan F adalah suatu *field*. Polinomial $f(x) \in F[x]$ disebut *irreducible* di F jika $f(x)$ tidak dapat dinyatakan sebagai hasil kali dua polinomial *non-konstan* dengan koefisien di F. Dengan kata lain, jika $f(x) = g(x).h(x)$ dengan $g(x), h(x) \in F[x]$, maka salah satu dari $g(x)$ atau $h(x)$ harus merupakan polinomial konstan (Chandoul, 2021).

Sebagai analogi, bilangan prima dalam bilangan bulat adalah bilangan yang hanya dapat dibagi oleh 1 dan dirinya sendiri. Demikian pula, dalam himpunan polinomial, polinomial *irreducible* tidak dapat dibagi menjadi faktor-faktor polinomial yang lebih kecil kecuali dengan polinomial konstanta.

Polinomial *irreducible* memiliki beberapa sifat penting yakni:

1. Tidak memiliki faktor *non-trivial*

   Polinomial *irreducible* tidak dapat dinyatakan sebagai hasil kali dua polinomial *non-konstanta* dalam satu medan tertentu. Jika $f(x)$ *irreducible* dalam $F[x]$, maka $f(x) = g(x).h(x)$ dimana salah satu dari $g(x)$ atau $h(x)$ adalah polinomial konstanta.
2. Polinomial berderajat satu selalu *irreducible* karena tidak mungkin difaktorkan lebih lanjut.

3. Polinomial berderajat lebih tinggi dapat diperiksa dengan algoritma faktorisasi.
4. Polinomial *Irreducible* memiliki akar di ekstensi medan

   Jika $f(x)$ *irreducible* dalam $F[x]$ dan memiliki derajat m, maka akar-akar dari $f(x)$ tidak terdapat F tetapi ada di medan ekstensinya F_{ext}.
5. Polinomial *irreducible* digunakan untuk membentuk lapangan

   Polinomial *irreducible* digunakan untuk membentuk medan $GF(2^m)$. Jika $f(x)$ *irreducible* memiliki derajat m, maka medan yang dihasilkan memiliki 2^m elemen.
6. Memiliki karakter yang berbeda

   Sebuah polinomial dapat *irreducible* dalam satu medan tetapi *irreducible* dalam medan lain. Misal, dalam $GF(3)$, polinomial $x^3 + 1$ *irreducible* karena tidak memiliki akar dalam $GF(3)$. Sedangkan pada $GF(5)$, polinomial yang sama dapat difaktorkan sebagai $(x + 2)(x + 3)$, sehingga tidak *irreducible*.

Pada proses pembentukan lapangan $GF(2^m)$, selain menggunakan polinomial *irreducible*, juga diperlukan polinomial yang memiliki sifat khusus yaitu bersifat primitif. Polinomial primitif merupakan polinomial *irreducible*, namun tidak semua polinomial *irreducible* bersifat primitif. Polinomial primitif adalah polinomial *irreducible* yang memiliki akar α, di mana α merupakan elemen primitif dari $GF(2^m)$. Suatu elemen disebut primitif jika elemen tersebut dapat menghasilkan semua elemen tak nol pada $GF(2^m)$ melalui pemangkatan, mulai dari α^0 hingga α^{2^m-1}. Maka, polinomial primitif akan menghasilkan

elemen medan yang membentuk grup siklik di bawah operasi perkalian (Andriani et. al, 2007).

Pada kasus $GF(2^4)$, medan ini merupakan medan dengan $2^4 = 16$ elemen yang terdiri dari 15 elemen tak nol dan satu elemen nol. Untuk membentuk $GF(2^4)$, digunakan polinomial *irreducible* berderajat 4 sebagai dasar pembentukan medan. Dalam hal ini, polinomial *irreducible* pada $GF(2^4)$ tidak dapat difaktorkan menjadi dua polinomial berderajat lebih rendah dengan koefisien 0 dan 1. Akar α dari polinomial tersebut harus mampu menghasilkan semua elemen tak nol pada $GF(2^4)$ melalui pemangkatan, hingga kembali ke 1 pada pangkat ke-15. Oleh karena itu, polinomial primitif $p(x) = x^4 + x + 1$ dapat digunakan sebagai dasar pembentukan $GF(2^4)$. Berdasarkan polinomial primitif $p(x) = x^4 + x + 1$, semua elemen $GF(2^4)$ dibangkitkan melalui operasi modulo $p(x)$. Setiap elemen α^i dinyatakan dalam sebagai polinomial koefisien biner, menghasilkan total 15 elemen tak nol dari α^1 hingga α^{14}, yaitu:

Tabel 2.1 Tabel Polinomial Primitif pada $GF(2^4)$

$\boldsymbol{\alpha^i}$	**Polinomial**	**Biner**
α^0	1	[0001]
α^1	α	[0010]
α^2	α^2	[0100]
α^3	α^3	[1000]
α^4	$\alpha + 1$	[0011]
α^5	$\alpha^2 + \alpha$	[0110]
α^6	$\alpha^3 + \alpha^2$	[1100]
α^7	$\alpha^3 + \alpha + 1$	[1011]
α^8	$\alpha^2 + 1$	[0101]
α^9	$\alpha^3 + \alpha$	[1010]
α^{10}	$\alpha^2 + \alpha + 1$	[0111]
α^{11}	$\alpha^3 + \alpha^2 + \alpha$	[1110]
α^{12}	$\alpha^3 + \alpha^2 + \alpha + 1$	[1111]
α^{13}	$\alpha^3 + \alpha^2 + 1$	[1101]
α^{14}	$\alpha^3 + 1$	[1001]

2.1.2 Penjumlahan dan Perkalian dalam $GF(2^m)$

Polinomial dalam $GF(2^m)$ merupakan himpunan semua polinomial dengan derajat kurang dari n dan koefisien yang berasal dari himpunan bilangan biner $\{0, 1\}$. Pada polinomial $GF(2^m)$, terdapat dua operasi utama yang digunakan, yaitu operasi penjumlahan dan perkalian. Operasi penjumlahan pada $GF(2^m)$ serupa dengan penjumlahan polinomial pada umumnya. Perbedaan terletak pada koefisien yang digunakan, yakni koefisien biner. Dengan menggunakan operasi XOR dapat dilakukan suatu penjumlahan pada $GF(2^m)$ (Sadikin, 2012). Karena koefisien berada pada $GF(2)$, maka hasil penjumlahan setiap koefisien dilakukan pada modulo 2, yang berlaku:

Tabel 2.2 Operasi Penjumlahan pada $GF(2^m)$

+	0	1
0	0	1
1	1	0

Sementara itu, operasi perkalian dalam $GF(2^m)$ juga serupa dengan perkalian polinomial biasa, namun dengan perbedaan pada aturan koefisiennya. Untuk mengalikan dua polinomial $f(x)$ dan $g(x)$, setiap suku dalam $f(x)$ dikalikan dengan seluruh suku dalam $g(x)$. Hasil perkalian antara x^i dan x^j menghasilkan x^{i+j}. Karena hasil perkalian dalam $GF(2^m)$ dapat menghasilkan polinomial dengan derajat lebih tinggi dari $n - 1$, maka perlu dilakukan proses reduksi menggunakan polinomial *irreducible* $m(x)$ sebagai modulus (Sadikin, 2012).

Pada kasus khusus dengan $m = 4$, medan yang digunakan adalah $GF(2^4)$ yaitu himpunan berhingga yang terdiri atas $2^4 = 16$ elemen. Setiap elemen dalam

medan $GF(2^4)$ di representasikan sebagai polinomial berderajat kurang dari 4 dengan koefisien berasal dari himpunan $\{0, 1\}$. Bentuk umum dari setiap elemen ditulis sebagai:

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \quad (2.12)$$

dengan $a_i \in \{0, 1\}$ yang terdapat 16 kemungkinan kombinasi dari keempat koefisien biner tersebut.

Operasi penjumlahan dilakukan dengan menjumlahkan koefisien-koefisien dari dua polinomial menggunakan operiasi XOR. Sebagai ilustrasi, penjumalahan dua elemen berikut:

$$(x^3 + x) + (x^3 + x^2 + 1)$$

akan menghasilkan:

$$x^2 + x + 1$$

suku x^3 menjadi 0 karena $1 + 1 = 0$.

Operasi perkalian dilakukan sebagaimana pada perkalian polinomial biasa. Namun, ketika hasil perkalian memiliki derajat ≥ 4, maka hasil tersebut harus direduksi terhadap polinomial *irreducible* derajat 4. Salah satu polinomial yang umum digunakan dalam $GF(2^4)$ adalah:

$$m(x) = x^4 + x + 1$$

jika hasil perkalian masih berdejarat kurang dari 4, reduksi tidak diperlukan. Akan tetapi, jika hasilnya berderajat lebih tinggi maka perlu dilakukan reduksi supaya hasil akhir tetap berada dalam medan $GF(2^4)$.

Sebagai contoh:

$$(x^3 + x^2 + 1)(x^2 + x) = x^5 + x^4 + x^3 + x^2$$

dengan substitusi berdasarkan polinomial *irreducible*, $x^4 = x + 1$ dan $x^5 = x(x^4) = x(x + 1) = x^2 + x$. Maka hasilnya menjadi

$$(x^2 + x) + (x + 1) + x^3 + x^2 = x^3 + 1$$

hasil tersebut sudah berderajat kurang dari 4 dan sesuai dalam medan $GF(2^4)$. Proses ini menjamin bahwa hasil dari setiap operasi penjumlahan maupun perkalian tetap berada dalam medan $GF(2^4)$, sehingga struktur bidang hingga tetap konsisten dan tertutup.

2.2 Kode Biner

Definisi 2.2

Diberikan kode alfabet $F_2 = \{0, 1\}$. Kode atas F_2 dinamakan kode biner. Simbol kode yang digunakan untuk suatu kode biner adalah 0 dan 1 (Ling & Xing, 2004).

Sebagai ilustrasi, himpunan $C_1 = \{00, 01, 10, 11\}$ merupakan contoh kode biner dengan panjang $n = 2$ dan jumlah kata $M = 4$, sehingga disebut sebagai kode (2, 4). Contoh lainnya adalah $C_2 = \{000, 001, 010, 101\}$, yang merupakan kode dengan panjang kata $n = 3$ dan jumlah kata $M = 4$, atau kode (3, 4). Sementara itu, $C_3 = \{0001, 0010, 0011, 0101, 1100, 0110\}$ adalah himpunan kode biner dengan panjang $n = 4$ dan jumlah kata $M = 6$, sehingga dapat disebut sebagai kode (4, 6).

2.3 Kode ASCII

ASCII (*American Standard Code for Information Interchange*) adalah sistem pengkodean karakter yang digunakan dalam komputer dan sistem komunikasi untuk merepresentasikan huruf, angka, simbol, dan karakter kontrol dalam bentuk biner. ASCII menggunakan 7-bit untuk merepresentasikan 128 karakter pertama,

yang mencakup huruf A-Z (hurus besar dan kecil), angka 0-9, tanda baca, serta karakter kontrol seperti enter (*Carriage Return*-CR), Tab, dan *Backspace*. Versi yang lebih luas, *Extended* ASCII (8-bit), menambahkan 128 karakter tambahan untuk mendukung berbagai simbol dan karakter internasional (Kurnia, 2013).

2.4 Transmisi Data

Transmisi data merupakan proses pengiriman informasi dari satu perangkat ke perangkat lainnya melalui suatu saluran komunikasi, seperti kabel atau jaringan nirkabel. Proses ini sangat penting terutama dalam komunikasi elektronik, karena keberhasilan komunikasi tergantung pada seberapa akurat data yang diterima dibandingkan dengan data yang dikirimkan.

Namun, pada proses transmisi data sering menghadapi berbagai gangguan yang dapat menyebabkan kesalahan pada data yang dikirimkan. Kesalahan dalam ini bisa disebabkan oleh beberapa faktor yang mempengaruhi kualitas sinyal selama pengiriman data. Berikut adalah faktor yang dapat menyebabkan kesalahan dalam transmisi data:

1. Radiasi Elektromagnetik

   Radiasi elektromagnetik merupakan gelombang energi yang dipancarkan oleh perangkat elektronik, seperti radio, *microwave*, atau perangkat komunikasi nirkabel lainnya. Radiasi ini dapat mengganggu sinyal yang sedang ditransmisikan melalui media komunikasi. Dampaknya, sinyal dapat mengalami distorsi, sehingga data yang diterima berbeda dari data yang dikirimkan (Muslim, 2021).

2. *Crosstalk* (sinyal bocor)

   Crosstalk terjadi ketika sinyal dari satu saluran komunikasi "bocor" ke saluran komunikasi lain yang berdekatan. Misalnya, pada kabel yang terhubung dengan perangkat yang saling berdekatan, atau dalam jaringan yang menggunakan saluran fisik yang sama. Ketika ini terjadi, sinyal yang seharusnya tetap utuh bisa terganggu oleh sinyal lain yang menyebabkan data menjadi rusak atau tidak akurat.

3. Gangguan Akibat Petir

   Petir merupakan fenomena alam yang menghasilkan lonjakan tegangan listrik dengan jumlah besar. Gangguan yang disebabkan oleh petir ini dapat merusak perangkat komunikasi dan menyebabkan transmisi data menjadi terganggu. Dampaknya, penurunan kualitas sinyal selama transmisi, terutama pada jaringan kabel tembaga (Forouzan, 2013).

4. *Noise* (gangguan sinyal)

   Noise adalah gangguan yang berupa sinyal tidak diinginkan yang bercampur dengan sinyal asli selama proses transmisi data. *Noise* bisa datang dari berbagai sumber, seperti peralatan elektronik yang ada di sekitar, atau kondisi lingkungan. Noise ini dapat merusak kualitas data yang diterima, mengubah bit-bit informasi, dan menyebabkan data yang diterima menjadi salah atau tidak akurat.

Dampak dari gangguan-gangguan tersebut adalah munculnya kesalahan bit. Kesalahan bit terjadi ketika data yang diterima tidak sesuai dengan data yang dikirimkan. Bit adalah unit terkecil dari data digital yang hanya memiliki dua kemungkinan nilai, yaitu 0 atau 1. Ketika terjadi kesalahan bit, nilai "0" bisa

berubah menjadi 1, atau sebaliknya, yang menyebabkan data yang diterima tidak sesuai dengan data yang seharusnya.

Adapun macam-macam bit *error* sebagai berikut:

1. *Single-bit error*

   Single-bit error terjadi ketika hanya satu bit dalam unit data seperti *byte*, karakter, atau paket, berubah dari 1 ke 0 atau sebaliknya. Kesalahan jenis ini memiliki kemungkinan sangat kecil terjadi dalam transmisi data serial. Misalnya, jika data ditransmisikan pada kecepatan 1 Mbps, maka setiap bit hanya berlangsung selama 1 mikrodetik ($1\mu s$). Untuk menyebabkan *single-bit error*, gangguan (*noise*) harus memiliki durasi yang sangat singkat, yaitu sekitar $1\mu s$ yang jarang terjadi. Biasanya, *noise* berlangsung lebih lama dari itu, sehingga lebih mungkin menyebabkan kesalahan yang memengaruhi lebih dari satu bit (Forouzan, 2013).

2. *Burst error*

   Burst error terjadi ketika terdapat dua atau lebih bit pada unit data telah berubah dari 1 ke 0 atau dari 0 ke 1. Pada kasus 0100010001000011 dikirim dan 0101110101100011 diterima, dapat diperhatikan bahwa beberapa bit mengalami perubahan tetapi tidak harus berurutan. Panjang *brust error* dihitung dari bit pertama yang rusak hingga bit terakhir yang rusak, meskipun ada bit diantaranya yang tetap benar.

Brust error lebih sering terjadi dibandingkan *single-bit error* karena *noise* biasanya memiliki durasi lebih panjang dari 1 bit, yang berarti bahwa jika *noise* mempengaruhi data, maka akan mempengaruhi satu set bit atau lebih dari satu bit. Jumlah bit yang terkena efeknya tergantung pada data *rate* dan durasi *noise*

tersebut. Untuk mengatasi masalah ini, digunakan metode untuk mendeteksi dan memperbaiki kesalahan yang terjadi pada data. Tujuannya adalah agar data yang diterima tetap akurat dan sesuai dengan yang dikirimkan, meskipun terdapat gangguan-gangguan yang terjadi selama transmisi. Dengan cara ini, dapat dipastikan bahwa informasi yang diterima oleh perangkat penerima tetap benar dan dapat digunakan dengan baik. Deteksi dan koreksi kesalahan bit adalah proses untuk mengidentifikasi dan memperbaiki kesalahan yang terjadi pada bit data. Deteksi dan koreksi kesalahan merupakan aspek penting dalam sistem komunikasi untuk memastikan keandalan dan integritas data yang ditransmisikan.

2.4.1 Deteksi Kesalahan Bit

Deteksi kesalahan bit adalah proses untuk memastikan bahwa data yang diterima sesuai dengan data yang dikirim tanpa perubahan akibat gangguan selama transmisi, teknik ini sangat penting dalam sistem komunikasi dan penyimpanan data digital, karena transmisi data rentan terhadap gangguan seperti *noise*, interferensi atau kerusakan perangkat keras. Proses deteksi dilakukan dengan menambahkan bit kontrol atau bit redundansi ke dalam bit asli.

Teknik deteksi kesalahan umumnya menggunakan kode kesalahan, dimana kode siklik dengan polinomial generator yang umum digunakan. Polinomial generator berfungsi sebagai kunci dalam pembangkitan kode kesalahan. Proses pengkodean melibatkan pembagian polinomial data dengan polinomial generator untuk menghasilkan sisa atau *remainder* yang menjadi bagian dari kode kesalahan. Pada penerimaan data, polinomial yang sana digunakan kembali untuk memeriksa kesalahan.

Kode kesalahan yang sering digunakan untuk deteksi adalah kode siklik, yang menawarkan kemampuan deteksi yang jauh lebih andal dibandingkan metode sederhana seperti bit paritas. Pada kode siklik, data dan bit kontrol direpresentasikan dalam bentuk polinomial biner. Proses pengkodean dilakukan dengan membagi polinomial data oleh polinomial generator yang telah dipilih. Sisa hasil pembagian tersebut, yang dikenal sebagai *remainder*, ditambahkan ke data sebagai kode kesalahan. Polinomial generator berfungsi sebagai kunci dan sangat menentukan kemampuan deteki sistem, karena dapat di rancang untuk mendeteksi berbagai pola kesalahan termasuk kesalahan tunggal, kesalahan ganda, dan kesalahan *burst*.

Pada sisi penerima, proses serupa dilakukan. Data yang diterima kembali dibagi dengan polinomial generator yang sama. Jika hasil pembagian berupa sisa nol, maka data diasumsikan tidak mengalami kesalahan. Sebaliknya, jika sisa tidak nol, maka penerima menyimpulkan bahwa terjadi kesalahan dalam transmisi. Metode ini sangat efisien dan cepat dalam mendeteksi kesalahan, serta banyak digunakan dalam berbagai aplikasi seperti komunikasi jaringan, penyimpanan data digital, dan protokol komunikasi modern seperti ethernet.

2.4.2 Koreksi Kesalahan Bit

Koreksi kesalahan bit adalah proses yang tidak hanya mendeteksi adanya kesalahan dalam data yang diterima, tetapi juga memperbaiki kesalahan tersebut secara otomatis tanpa perlu mengirim ulang data. Teknik ini sangat penting dalam sistem komunikasi dan penyimpanan data digital, di mana keandalan transmisi dan integritas informasi harus tetap terjaga. Untuk melakukan koreksi, digunakan

kode khusus yang mampu menentukan lokasi kesalahan dan memulihkan nilai bit yang benar.

Teknik koreksi kesalahan bekerja dengan menambahkan informasi tambahan atau kode khusus ke dalam data asli saat proses pengiriman. Infomrasi ini dirancang sedemikian rupa sehingga memungkinkan sistem penerima untuk memverifikasi integritas data yang diterima. Apabila terjadi kesalahan, sistem dapat menentukan lokasi bit yang mengalami perubahan dan mengembalikannya ke nilai yang seharusnya. Salah satu langkah paling utama dalam proses koreksi adalah perhitungan nilai sindrom, yang diperoleh melalui operasi terhadap data yang diterima. Nilai sindrom akan menunjukkan ada atau tidaknya kesalahan, serta membantu dalam menentukan posisi.

Pada proses *decoding*, penerima akan menghitung nilai sindrom dari data yang diterima. Nilai sindrom ini diperoleh dari operasi pembagian polinomial data yang diterima dengan polinomial generator, sama seperti saat *encoding*. Jika sindrom berniali nol, maka tidak ada kesalahan yang terdeteksi. Namun, jika sindrom tidak nol, sistem akan menggunakan teknik *decoding* seperti algortima *Berlekamp-Messey* atau algoritma C*hien Search* untuk menentukan lokasi bit-bit yang salah. Setelah lokasi kesalahan ditemukan, sistem dapat membalik bit tersebut ke nilai yang benar, sehingga pesan asli berhasil direkontruksi.

Koreksi kesalahan bit memberikan lapisan perlindungan tambahan terhadap kerusakan data, terutama dalam aplikasi di mana transmisi ulang tidak memungkinkan atau terlalu mahal secara waktu dan sumber daya.

2.5 Kode Siklik

Kode siklik adalah salah satu tipe kode linier yang memiliki sifat *invarian* terhadap pergeseran (*shift-invariant*). Artinya, jika sebuah kode valid dirotasi melingkar (*cyclic shift*), hasil rotasinya tetap merupakan kode valid.

Kode siklik bekerja pada representasi polinomial dalam bidang berhingga $GF(q)$, dimana data biner (0 dan 1) direpresentasikan sebagai koefisien polinomial. Sebuah kode blok linier C disebut siklik jika berlaku

$$Jika\ c(x) \in C, maka\ xc(x) mod\ (x^n - 1) \in C$$

di mana:

C : Himpunan kode valid,

$c(x)$: Polinomial kode yang valid,

$x^n - 1$: Polinomial pembagi utama.

Kode siklik bekerja dalam kerangka representasi polinomial pada lapangan atau *Galois Field* $GF(q)$, di mana q adalah pangkat dari bilangan prima. Kode siklik menyandikan blok data dengan panjang n. Blok ini direpresentasikan sebagai polinomial pada persamaan (2.1) dengan koefisien $c_i \in GF(q)$. Polinomial ini membentuk dasar dalam penyandian data. Proses encoding dilakukan dengan mengalikan polinomial pesan dengan suatu polinomial generator $g(x)$, yang dirancang agar menjadi pembagi dari $x^n - 1$. Dengan demikian, setiap kode valid adalah kelipatan dari $g(x)$ dan secara otomatis memiliki sifat siklik.

2.5.1 Generator Polinomial

Generator polinomial merupakan komponen utama dalam pembentukan kode *Bose Chaudhuri Hocquenghem* (BCH) yang berperan penting dalam proses deteksi dan koreksi kesalahan pada sistem komunikasi digital. Generator

polinomial digunakan dalam proses *encoding* dan *decoding*, sehigga menjadi elemen sentral dalam penjaminan keandalan transmisi data. Pada kode BCH, generator polinomial $g(x)$ dibentuk sebagai pembagi dari polinomial $x^n - 1$, sehingga dapat dituliskan bahwa:

$$g(x)|(x^n - 1)$$

di mana $g(x)$ merupakan generator polinomial yang akan digunakan dalam proses pembentukan kode siklik (Bossert, 2021).

Pembentukan generator polinomial dilakukan dengan memanfaatkan lapangan $GF(2^m)$ sebagai dasar operasi aljabar. Medan ini dikontruksi dari medan biner $\mathbb{F}_2$ menggunakan polinomial *irreducible* berderajat m. Sebagai contoh, pada lapangan $GF(2^4)$, maka $m = 4$ dan polinomial irreducible yang umum dipakai adalah $x^4 + x + 1$. Medan ini terdiri atas $2^4 = 16$, yang mencakup nol dan 15 elemen bukan nol, yaitu

$$\{0, 1, \alpha, \alpha^2, \dots, \alpha^{14}\}$$

dengan α adalah akar primitif dari polinomial irreducible tersebut. Setiap elemen di lapangan dapat di representasikan sebagai kombinasi linier dari basis

$$\{1, \alpha, \alpha^2, \alpha^3\}$$

menjadikan struktur medan ini cocok untuk perancangan kode koreksi kesalahan.

Parameter pertama yang ditentukan dari struktur lapangan $GF(2^4)$ adalah panjang kode n, yang dihitung berdasarkan rumus

$$n = 2^m - 1.$$

Panjang kode n menunjukkan jumlah total bit dalam satu codeword hasil pengkodean, yang terdiri atas bit data dan bi redundansi. Misalnya jika $m = 4$, maka diperoleh $n = 15$, yang mengartikan bahwa setiap kode hasil pengkodean

memiliki panjang 15 bit. Nilai n juga menentukan orde dari polinomial siklik $x^n - 1$ yang akan dibagi oleh generator polinomial $g(x)$.

Parameter berikutnya adalah panjang bit informasi atau bit data, yang dilambangkan dengan k. Nilai k dihitung dari selisih antara panjag kode n dan derajat dari generator polinomial $g(x)$ yaitu $k = n - \deg(g(x))$. Derajat dari $g(x)$, yang biasa disingkat sebagai r menunjukkan jumlah bit redundansi yang ditambahkan untuk memungkinkan proses koreksi kesalahan. Dengan demikian, parameter k merepresentasikan panjang data asli sebelum diberi tambahan bit kontrol untuk mendeteksi dan memperbaiki kesalahan.

Kemampuan koreksi kesalahan dalam kode BCH ditentukan oleh parameter t, yaitu jumlah maksimum bit kesalahan yang dapat dikoreksi dalam satu codeword. Nilai t ini dipilih pada saat perancangan sistem dan sangat mempengaruhi kompleksitas pembentukan generator polinomial. Untuk mengoreksi hingga t kesalahan, digunakan himpunan $2t$ akar dari medan $GF(2^m)$ dalam bentuk:

$$\{\alpha^b, \alpha^{b+1}, \dots, \alpha^{b+2t-1}\}$$

di mana b adalah bilangan bulat positif. Generator polinomial $g(x)$ kemudian dibentuk sebagai kelipatan persekutuan terkecil dari polinomial-polinomial minimal terhadap akar-akar tersebut, yang dituliskan sebagai:

$$g(x) = lcm\left(M_{\alpha^b}(x), M_{\alpha^{b+1}}(x), \dots, M_{\alpha^{b+2t-1}}(x)\right) \tag{2.13}$$

dengan $M_{\alpha^i}(x)$ menyatakan polinomial minimal dari α^i atas $\mathbb{F}_2$. Polinomial minimal merupakan polinomial *irreducible* berderajat paling rendah yang memiliki α^i sebagai akar. Sebagai contoh, untuk $t = 1$ dan $b = 1$, maka $g(x)$ diperoleh dari:

$$lcm\left(M_{\alpha(x)}, M_{\alpha^2}(x)\right).$$

Parameter penting lainnya adalah jarak minimum kode, yang dinotasikan sebagai d_{min}. Pada kode BCH, jarak minimum ini memiliki batas bawah, yaitu

$$d_{min} \geq 2t + 1.$$

Nilai ini menunjukkan hamming terkecil antara dua codeword valid, yang menjadi indikator sejauh mana kesalahan dapat dideteksi atau dikoreksi oleh sistem. Dengan nilai d_{min} yang smeakin besar, kemampuan kode dalam menghadapi gangguan selama transmisi data juga akan semakin baik.

Generator polinomial yang telah diperoleh digunakan dalam tahap *encoding* dengan cara mengalikan data biner $d(x)$ dengan $g(x)$, sehingga kode hasilnya adalah:

$$c(x) = d(x) \cdot g(x) \tag{2.14}$$

Pada tahap *decoding*, generator polinomial dimanfaatkan untuk membentuk sindrom.

2.5.2 Kode *Bose-Chaudhuri-Hocquenghem*

Kode BCH adalah jenis kode koreksi kesalahan yang termasuk dalam kategori kode siklik. Kode ini dirancang untuk mendeteksi dan memperbaiki kesalahan selama proses transmisi data, khususnya dalam sistem komunikasi digital. Kode BCH di kembangkan atas dasar teori polinomial dalam bidang hingga khususnya $GF\ (2^m)$ untuk kasus kode biner. Penggunaan *Galois Field* memungkinkan dilakukannya operasi aljabar secara efisien dan konsisten, sehingga kode BCH sangat cocok diterapkan dalam berbagai sistem digital modern. (Lin & Li, 2021)

Ide utama dari kode BCH adalah membangun sebuah polinomial generator $g(x)$ yang memiliki sejumlah akar tertentu dari elemen primitif pada *Galois Field*. Akar-akar tersebut dipilih sedemikian sehingga supaya kode yang dihasilkan memiliki jarak minimum d yang memenuhi syarat

$$d \geq 2t + 1$$

di mana t merupakan jumlah maksimum kesalahan bit yang dapat dikoreksi. Dengan memenuhi syarat ini, kode BCH mampu memperbaiki hingga t bit kesalahan dalam satu blok data. Oleh karena itu, efisiensi dan keandalan kode sangat bergantung pada pemilihan akar dan pembentukan $g(x)$ secara tepat (Morelos, 2006).

Pada tahap *encoding*, pesan asli direpresentasikan sebagai suatu polinomial $b(x)$ yang memiliki derajat kurang dari k. Polinomial ini kemudian dikalikan dengan x^{n-k}, di mana n adalah panjang total kode, dan hasilnya dibagi dengan $g(x)$ untuk memperoleh sisa pembagian $r(x)$. Polinomial kode $C_{ij}''(x)$ yang siap ditransmisikan dibentuk dari penjumlahan:

$$C_{ij}''(x) = b(x) \cdot x^{n-k} + r(x) \tag{2.15}$$

dan secara otomatis menjadi kelipatan dari $g(x)$. Karena merupakan kelipatan dari polinomial generator, $C_{ij}''(x)$ termasuk dalam himpunan kode BCH yang valid (Cho & Sung, 2009).

Jika selama proses transmisi terjadi gangguan, maka penerima akan menerima polinomial $r(x)$ yang mungkin mengandung kesalahan. Untuk mendeteksi dan memperbaiki kesalahan tersebut, dilakukan proses *decoding* yang dimulai dengan perhitungan sindrom. Nilai sindrom diperoleh dengan mengevaluasi $r(x)$ pada akar-akar dari $g(x)$ yakni:

$$S_i = r(\alpha^i) \tag{2.16}$$

di mana α adalah elemen primitif dalam *Galois Field*. Jika seluruh sindrom bernilai nol, maka tidak ada kesalahan yang terjadi, sebaliknya, jika terdapat nilai sindrom yang tidak nol, maka proses koreksi dilanjutkan (Reed & Chen, 1999).

Langkah berikutnya dalam proses *decoding* adalah membentuk polinomial lokasi kesalahan $\sigma(x)$ berdasarkan nilai-nilai sindrom yang telah dihitung sebelumnya. Polinomial ini dibentuk menggunakan algortima efisien seperti algoritma *Berlekamp-Messey*. Algortima *Berlekamp-Messey* bekerja dengan mencari polinomial derajat terkecil yang dapat menghasilkan urutan sindrom yang diamati, yaitu polinomial lokasi kesalahan. Tahapan pertama dalam algoritma ini adalah inisialisasi parameter: $\sigma^{(0)}(x) = 1, B(x) = 1,$ derajat $L = 0$, perhitungan iterasi $m = 1,$ dan disrepansi awal $b = 1$. Kemudian dilakukan iterasi untuk setiap k dari 1 hingga $2t$, dengan t adalah kemampuan koreksi kesalahan maksimum. Pada setiap iterasi, dihitung disrepansi d_k menggunakan rumus:

$$d_k = S_k + \sum_{i=1}^{L} \sigma_i \cdot S_{k-i} \tag{2.17}$$

Jika $d_k = 0$, maka tidak ada perubahan pada $\sigma(x)$ dan m ditingkatkan. Namun, jika $d_k \neq 0$ maka dilakukan pembaruan polinomial yaitu dengan menghitung:

$$\sigma^{(k)}(x) = \sigma^{(k-1)}(x) - \frac{d_k}{b} \cdot x^m \cdot B(x) \tag{2.18}$$

Jika kondisi $2L \leq k - 1$ terpenuhi, maka dilakukan pembaruan yaitu $L + k = L, B(x) = \sigma^{(k-1)}(x), b = d_k$ dan $m = 1$. Jika tidak ada pembaruan, maka m ditambah satu untuk iterasi berikutnya. Proses ini berlangsung hingga seluruh nilai sindrom selesai diproses dan diperoleh polinomial $\sigma(x)$ dalam bentuk:

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x + \cdots + \sigma_t x^t \quad (2.19)$$

di mana akar-akar dari polinomial ini menunjukkan posisi bit yang salah (Freudenberger, 2019).

Langkah berikutnya adalah menentukan posisi kesalahan menggunakan algoritma *Chien Search.* Algortima *Chien* mengevaluasi $\sigma(x)$ pada semua nilai α^{-i} untuk $i = 0,1, \ldots n-1$ di mana n adalah panjang *codeword.* Untuk setiap nilai i, jika hasil evaluasi $\sigma(\alpha^{-i}) = 0$, maka posisi ke$-i$ dianggap sebagai posisi bit yang salah. Posisi-posisi yang memenuhi kondisi ini kemudian dicatat, dan bit pada posisi tersebut di balik (*bit-flip*) untuk mengoreksi *codeword*.

Keunggulan kode BCH sendiri terletak pada fleksibilitasnya dalam menyesuaikan parameter n, k, dan t untuk berbagai kebutuhan sistem, serta kemampuannya memperbaiki *multiple-bit error* dengan kompleksitas komputasi yang masih dapat ditangani oleh perangkat keras digital modern.

Selain kemampuan koreksi kesalahan, efisiensi transmisi juga merupakan aspek penting yang perlu diperhatikan dalam penilaian performa kode BCH. Efisiensi transmisi menggambarkan seberapa besar proporsi bit data asli yang dapat dikirim dibandingkan dengan total bit dalam satu *codeword*, termasuk bit redundansi yang ditambahkan untuk keperluan deteksi dan korkesi kesalahan. Efisiensi transmisi dirumuskan sebagai:

$$\eta = \frac{k}{n} \times 100\% \quad (2.20)$$

dengan k menyatakan jumlah bit data dan n merupakan panjang total *codeword*. Semakin besar nilai efisiensi, maka semakin banyak informasi aktual yang dapat dikirim tanpa tambahan *overhead* dari bit kontrol. Hasil efisiensi yang tinggi menunjukkan bahwa sebagian besar bit dalam *codeword* merupakan data asli,

sehingga sistem memiliki kapasitas transmisi informasi yang besar dan minim penggunaan sumber daya, namun dengan kemampuan koreksi kesalahan yang lebih rendah karena sedikit bit redundan yang digunakan. Sebaliknya, efisiensi yang rendah mengartikan banyaknya bit dalam *codeword* dialokasikan untuk koreksi kesalahan, yang meningkatkan ketahanan terhadap gangguan, namun mengurangi proporsi data asli yang bisa dikirim, sehingga memperbesar *overhead.* Oleh karena itu, dalam perancangan sistem komunikasi digital, efisiensi perlu diseimbangkan dengan tingkat ketahanan terhadap gangguan agar sistem tetap andal namun tetap hemat dalam penggunaan sumber daya.

2.6 Kajian Integrasi Topik dengan Al-Qur'an

Al-Qur'an sebagai kitab suci memiliki prinsip ketelitian dan akurasi dalam penyampaian wahyu, sehingga setiap ayatnya dijaga dari perubahan atau distorsi. Sejak diturunkan kepada Nabi Muhammad SAW, Al-Qur'an telah melalui proses penjagaan yang ketat melalui hafalan mutawatir, kodifikasi tertulis, serta aturan ketat dalam pelafalan dan penulisannya. Prinsip ini bertujuan memastikan bahwa wahyu Allah SWT tetap autentik, tidak mengalami perubahan sedikit pun, dan tetap menjadi pedoman bagi umat manusia dalam menjalani hidup. Dalam hal ini, Al-Qur'an mengajarkan pentingnya kehati-hatian dalam menerima dan menyampaikan informasi, sebagaimana disebutkan dalam Surah Al-Isra' ayat 36 yang artinya (Kementerian Agama, 2022):

"Dan janganlah kamu mengikuti sesuatu yang tidak kamu ketahui. Sesungguhnya pendengaran, penglihatan, dan hati, semuanya itu akan dimintai pertanggungjawaban.." (QS. Al-Isra': 36)

Ayat ini menegaskan bahwa manusia bertanggung jawab atas informasi yang mereka terima, analisis yang mereka lakukan, serta keputusan yang diambil

berdasarkan data yang ada. Prinsip ini memiliki relevansi yang kuat dengan sistem komunikasi digital, khususnya dalam implementasi Kode BCH yang digunakan untuk mendeteksi dan mengoreksi kesalahan dalam transmisi data.

Pada komunikasi digital, data dikirim dalam bentuk bit yang rentan mengalami kesalahan akibat gangguan seperti *noise* atau interferensi. Jika kesalahan ini tidak terdeteksi, maka informasi yang diterima dapat berubah, mengakibatkan distorsi atau kesalahan dalam pemrosesan data. Oleh karena itu, sistem seperti kode BCH digunakan untuk menverifikasi keakuratan data, mendeteksi kesalahan, dan mengoreksi bit yang keliru agar informasi tetap sesuai dengan aslinya.

Konsep ini memiliki keterkaitan dengan prinsip Al-Qur'an yang menekankan verifikasi informasi sebelum diikuti atau disebarkan. Sebagaimana Al-Qur'an dijaga keasliannya melalui metode yang ketat, sistem kode BCH juga memastikan bahwa data tetap akurat dan bebas dari kesalahan. Ayat ini juga mengajarkan manusia untuk selalu bersikap kritis dan tidak menerima informasi tanpa dasar yang kuat, sebagaimana dalam sistem komunikasi digital di mana setiap data yang dikirim harus melalui proses validasi sebelum dianggap benar. Kajian ini menunjukkan bahwa prinsip kehati-hatian dan verifikasi dalam kode BCH memiliki makna filosofis yang sejalan dengan ajaran islam. Baik dalam ilmu pengetahuan maupun kehidupan sehari-hari, ketelitian dalam menerima informasi, analisis yang cermat, serta tanggung jawab atas keputusan yang diambil merupakan hal yang sangat penting agar kebenaran tetap terjaga.

BAB III

METODE PENELITIAN

3.1 Jenis Penelitian

Penelitian pada penelitian ini menggunakan kualitatif dengan metode simulasi. Tujuan dari pendekatan ini adalah untuk menganalisis efektivitas berbagai bentuk generator polinomial dalam meningkatkan performa deteksi dan koreksi kesalahan pada kode BCH melalui eksplorasi berbasis simulasi.

3.2 Pra Penelitian

Pada tahap awal, dilakukan studi literatur untuk memahami teori dasar mengenai kode BCH san struktur generator polinomial. Peneliti mempelajari karakteristik generator polinomial, kontruksi Galois field $GF(2^m)$, serta prinsip kerja *encoding* dan *decoding* pada kode BCH. Selanjutnya, dilakukan simulasi untuk mengeksplorasi dan menganalisis efektivitas penerapan berbagai bentuk generator polinomial dalam meningkatkan performa deteksi dan koreksi kesalahan bit pada kode *Bose-Chaudhuri-Hocquenghem.*

3.3 Tahapan Penelitian

Tahapan-tahapan penelitian dilakukan dalam urutan sebagai berikut:

1. Simulasi Kode BCH dalam Deteksi dan Koreksi Kesalahan Bit

   Langkah awal penelitian adalah menentukan konfigurasi parameter kode BCH yang akan digunakan dalam simulasi. Pada penelitian ini digunakan tiga parameter yaitu BCH(15,11), BCH(15,7), dan BCH(15,5). Seluruh parameter memiliki panjang kode $n = 15$, namun berbeda dalam jumlah bit

informasi k dan bit paritas. Masing-masing parameter dirancang untuk mengoreksi kesalahan dengan jumlah yang berbeda.

2. Pembentukan Generator Polinomial

   Tahap ini mencakup pembentukan generator polinomial $g(x)$ untuk setiap konfigurasi BCH berdasarkan nilai t yang ditentukan. Proses $GF(2^4)$ menggunakan polinomial *irreducible* $x^4 + x + 1$. Selanjutnya, dilakukan identifikasi grup konjugasi dan pembentukan polinomial minimal untuk setiap akar dari lapangan. Polinomial-polinomial ini kemudian dikombinasikan menggunakan operasi kelipatan persekutuan terkecil untuk membentuk $g(x)$.

3. Proses *Encoding*

   Setelah polinomial generator diperoleh, dilakukan proses encoding terhadap data input. Data input yang digunakan berupa kata "*Deserve*" yang dikonversikan ke dalam representasi biner ASCII 8-bit. Data biner tersebut dibagi menjadi blok-blok sesuai dengan panjang informasi dari masing-masing parameter BCH. Setiap blok direpresentasikan sebagai polinomial, dikalikan dengan x^r, dan dibagi dengan $g(x)$ untuk memperoleh sisa pembagian sebagai bit paritas. Hasil akhir *encoding* berupa *codeword* sepanjang 15 bit.

4. Penyisipan *Error*

   Tahap ini merupakan simulasi gangguan yang mungkin terjadi pada sistem transmisi data. Setiap *codeword* yang dihasilkan dari proses *encoding* diberi kesalahan secara acak pada satu atau beberapa bit sesuai dengan kemampuan koreksi masing-masing parameter. Misalnya, satu bit kesalahan untuk

BCH(15,11), dua bit kesalahan untuk BCH(15,7), dan tiga bit kesalahan untuk BCH(15,5). Kesalahan ditambahkan menggunakan operasi XOR antara *codeword* asli dan vektor kesalahan.

5. Proses *Decoding* dan Koreksi Kesalahan

   Codeword yang telah mengalami kesalahan dianalisis menggunakan proses decoding BCH. Langkah pertama adalah menghitung sindrom dari *codeword* untuk mendeteksi ada atau tidaknya kesalahan. Jika sindrom bernilai nol, dilakukan identifikasi posisi kesalahan dan koreksi menggunakan metode berbasis sindrom. Setelah proses koreksi selesai, data dikembalikan ke bentuk biner awal dan dikonversi kemabali menjadi karakter ASCII untuk validasi hasil.

6. Analisis Hasil

   Tahap terakhir adalah analisi hasil dari setiap parameter kode BCH yang diuji. Evaluasi dilakukan dengan membandingkan kemampuan masing-masing parameter dalam mengoreksi kesalahan dan mengembalikan data asli. Aspek efisiensi juga diperhatikan, yaitu perbandingan antara panjang data informasi dan panjang total *codeword*. Melalui evaluasi ini dapat diketahui sejauh mana parameter kode BCH dapat menjaga integritas data dalam sistem komunikasi digital.

BAB IV

HASIL DAN PEMBAHASAN

4.1 Simulasi Kode BCH dalam Deteksi dan Koreksi Kesalahan Bit

Simulasi yang dilakukan pada penelitian ini bertujuan untuk menguji dan membandingkan kemampuan kode BCH dalam mendeteksi dan mengoreksi kesalahan bit pada data digital. Tiga konfigurasi parameter kode BCH yang digunakan dalam simulasi ini adalah BCH(15,11), BCH(15,7), dan BCH (15,5). Ketiganya memiliki panjang kode (n) yang sama yaitu 15 bit, namun berbeda pada panjang informasi (k) dan jumlah bit paritas $(n-k)$, sehingga menghasilkan perbedaan dalam kemampuan koreksi kesalahan dan efisiensi transmisi.

Berdasarkan struktur BCH, kemampuan koreksi kesalahan (t) ditentukan oleh jumlah bit paritas, di mana semakin banyak bit paritas yang digunakan semakin tinggi kemampuan koreksi kesalahannya. Pada konfigurasi BCH(15,11) dengan 4 bit paritas, kode mampu mengoreksi hingga 1 bit kesalahan. Konfigurasi BCH(15,7) dengan 8 bit paritas, mampu mengoreksi hingga 2 bit kesalahan. Sementara itu, BCH(15,5) memiliki 10 bit paritas dan dapat mengoreksi hingga 3 bit kesalahan.

Sebagai objek simulasi, digunakan pesan teks sederhana berupa kata "*Deserve*". Kata ini terdiri dari tujuh karakter dan setiap karakter dikonversi ke dalam bentuk representasi biner 8-bit ASCII. Hasil konversi tersebut kemudian dibagi dalam blok-blok sesuai panjang informasi (k) dari masing-masing parameter BCH. Proses ini dilakukan untuk menyesuaikan panjang blok input dengan struktur yang digunakan dalam pengkodean BCH.

Simulasi dilakukan dalam empat tahapan utama untuk setiap parameter kode BCH, yaitu: pembentukan polinomial generator, proses *encoding*, penambahan kesalahan, serta proses *decoding* dan koreksi kesalahan. Dengan pendekatan ini, dapat dianalisis bagaimana perbedaan konfigurasi kode BCH memengaruhi efektivitas koreksi kesalahan serta efisiensi transmisi.

4.2 Analisis Berdasarkan Parameter Kode BCH

4.2.1 Kode BCH (15,11)

1. Pembentukan Generator Polinomial

Pada tahap ini dilakukan pembentukan generator polinomial sebagai dasar dalam membangun kode BCH yang akan digunakan dalam upaya mengoptimalkan kemampuan deteksi dan koreksi kesalah bit. Proses pembentukan polinomial generator diawali dengan penetapan parameter dasar kode BCH yaitu panjang kode (n), panjang informasi (k), dan tingkat kemampuan koreksi kesalahan (t). Pada penelitian ini digunakan lapangan $GF(2^4)$, sehingga panjang kode ditentukan berdasarkan rumus:

$$n = 2^m - 1$$

dengan $m = 4$, yang menghasilkan nilai $n = 15$. Parameter kemampuan koreksi kesalahan ditetapkan sebesar $t = 1$, yang berarti bahwa kode hanya mampu mengoreksi maksimum satu kesalahan bit dalam setiap blok. Panjang informasi dihitung menggunakan ketentuan:

$$t \geq \frac{n-k}{m}$$

sehingga diperoleh nilai $k = 11$. Jarak minimum (d_{min}) dari kode diperoleh dari:

$$d_{min} \geq 2t + 1$$

sehingga dalam hal ini jarak minimum adalah 3.

Langkah berikutnya adalah membangun lapangan $GF(2^4)$ dengan memilih polinomial *irreducible* berderajat empat sebagai dasar pembentukannya. polinomial *irreducible* pada $GF(2^4)$ tidak dapat difaktorkan menjadi dua polinomial berderajat lebih rendah dengan koefisien 0 dan 1. Akar α dari polinomial tersebut harus mampu menghasilkan semua elemen tak nol pada $GF(2^4)$ melalui pemangkatan, hingga kembali ke 1 pada pangkat ke-15. Oleh karena itu, polinomial primitif $p(x) = x^4 + x + 1$, dapat digunakan sebagai dasar pembentukan $GF(2^4)$. Berdasarkan polinomial primitif tersebut, seluruh elemen dalam medan ini dapat dibangkitkan melalui pemangkatan terhadap α, yang kemudian direduksi dengan operasi modulo terhadap $p(x)$. Setiap hasil pemangkatan α^i dapat direpresentasikan dalam bentuk polinomial dengan koefisien biner, yang selanjutnya dapat dinyatakan dalam bentuk representasi biner 4-bit. Melalui proses tersebut, menghasilkan total 15 elemen tak nol dari α^1 hingga α^{14} sebagaimana ditunjukkan pada Tabel 2.1.

Setelah menentukan polinomial primitif, tahap selanjutnya adalah mengidentifikasi elemen-elemen $GF(2^4)$ yang termasuk dalam grup konjugasi yang sama untuk digunakan dalam pembentukan polinomial minimal. Grup konjugasi dibentuk dari lapangan $GF(2^4)$ yang saling terkait melalui operasi pemangkatan basis dua. Pada $GF(2^4)$, grup konjugasi suatu elemen α^i adalah $\{\alpha^i, \alpha^{2i}, \alpha^{4i}, \alpha^{8i}\}\ mod\ (2^4 - 1)$ diperoleh:

Tabel 4.1 Grup Konjugasi $GF(2^4)$ dibangkitkan oleh $p(x) = x^4 + x + 1$

Elemen Awal α^i	Grup Konjugasi	Polinomial Minimal $m_i(x)$
α^1	$\{\alpha^1, \alpha^2, \alpha^4, \alpha^8\}$	$m_1(x) = x^4 + x + 1$
α^2	$\{\alpha^3, \alpha^6, \alpha^9, \alpha^{12}\}$	$m_2(x) = x^4 + x^3 + 1$
α^4	$\{\alpha^5, \alpha^{10}\}$	$m_3(x) = x^2 + x + 1$
α^8	$\{\alpha^7, \alpha^{11}, \alpha^{13}, \alpha^{14}\}$	$m_4(x) = x^4 + x^3 + x^2 + x + 1$

Berdasarkan Tabel 4.1 polinomial minimal dapat ditentukan dari setiap grup konjugasi yang dibentuk oleh elemen-elemen pada lapangan $GF(2^4)$. Setiap grup konjugasi menghasilkan satu polinomial minimal yang merupakan faktor *irreducible* dari polinomial karakteristik medan $GF(2^4)$. Pada pembentukan kode BCH (15,11) dengan kemampuan koreksi kesalahan $t = 1$, digunakan akar α^1 untuk membentuk polinomial minimal. Akar ini memiliki grup konjugasi $\{\alpha^1, \alpha^2, \alpha^4, \alpha^8\}$ dengan polinomial minimal $m_1(x) = x^4 + x + 1$. Karena $\deg(m_1(x)) = 4$, sehingga diperoleh:

$$k = n - \deg(m_1(x)) = 15 - 4 = 11$$

Oleh karena itu, polinomial generator yang digunakan adalah $g_1(x) = x^4 + x + 1$.

2. Proses *Encoding*

Proses *encoding* dimulai dengan mengonversi pesan teks ke dalam representasi biner dengan panjang 8-bit sesuai dengan tabel ASCII. Pada bagian simulasi ini, digunakan parameter BCH (15,11) yaitu dengan panjang kode $n = 15$, panjang data $k = 11$, dan panjang paritas $n - k = 4$. Dengan panjang paritas 4 bit, kode ini mampu mengoreksi hingga satu bit kesalahan. Sebagai contoh, apabila pengirim ingin mengirimkan pesan "*Deserve*", maka pesan tersebut terlebih dahulu diubah ke dalam representasi biner 8-bit sebagai berikut:

"D" menjadi [0 1 0 0 0 1 0 0]

"e" menjadi [0 1 1 0 0 1 0 1]

"s" menjadi [0 1 1 1 0 0 1 1]

"e" menjadi [0 1 1 0 0 1 0 1]

"r" menjadi [0 1 1 1 0 0 1 0]

"v" menjadi [0 1 1 1 0 1 1 0]

"e" menjadi [0 1 1 0 0 1 0 1]

Selanjutnya, pesan biner yang telah terbentuk dibagi menjadi beberapa blok $b_{ij}(x)$ dengan panjang 11 bit, sesuai dengan parameter $k = 11$ dari kode BCH (15,11). Setiap blok 11-bit kemudian direpresentasikan dalam bentuk polinomial biner, di mana setiap bit yang bernilai q menunjukkan adanya suku pada pangkat x yang bersesuaian. Representasi biner dan polinomial dari masing-masing blok ditunjukkan pada Tabel 4.2 berikut:

Tabel 4.2 Representasi Pesan Biner dan Polinomial pada Kode BCH(15,11)

Biner	**Polinomial $b'_{ij}(x)$**
$b_{11} = [01000100011]$	$b'_{11}(x) = x^9 + x^5 + x + 1$
$b_{12} = [00101011100]$	$b'_{12}(x) = x^8 + x^6 + x^4 + x^3 + x^2$
$b_{13} = [11011001010]$	$b'_{13}(x) = x^{10} + x^9 + x^7 + x^6 + x^3 + x$
$b_{14} = [11100100111]$	$b'_{14}(x) = x^{10} + x^9 + x^8 + x^5 + x^2 + x + 1$
$b_{15} = [01100110010]$	$b'_{15}(x) = x^9 + x^8 + x^5 + x^4 + x$
$b_{16} = [10000000000]$	$b'_{16}(x) = x^{10}$

Bedasarkan Tabel 4.2 representasi polinomial diperoleh dengan mengonversi setiap blok data biner 11-bit ke dalam bentuk polinomial biner dengan derajat paling tinggi sesuai dengan posisi bit tertinggi bernilai satu. Setiap bit dalam blok biner mewakili koefisien dari suku-suku dalam polinomial, dimulai dari bit paling kiri sebagai koefisien x^{10} dan seterusnya hingga bit paling kanan sebagai konstanta atau x^0.

Setelah setiap blok data $b_{ij}(x)$ direpresentasikan dalam bentuk polinomial, langkah selanjutnya adalah melakukan proses *encoding* BCH. Tahap pertama dalam proses ini adalah mengalikan polinomial $b'_{ij}(x)$ dengan $x^{n-k} = x^{15-11} = x^4$ untuk menyisipkan ruang bit paritas. Selanjutnya, hasil perkalian tersebut dibagi dengan polinomial generator $g_1(x) = x^4 + x + 1$ menggunakan metode pembagian polinomial biner. Sisa dari pembagian ini disebut residu $r_{ij}(x)$, yang akan menjadi bit paritas dalam *codeword* akhir.

Tabel 4.*3* Hasil Perkalian dan Residu pada Kode BCH(15,11)

$\boldsymbol{b'_{ij}(x).x^4}$	**Residu** $\boldsymbol{r_{ij}(x)}$
$x^{13} + x^9 + x^5 + x^4$	$r_{11}(x) = x$
$x^{12} + x^{10} + x^8 + x^7 + x^6$	$r_{12}(x) = x^3 + x$
$x^{14} + x^{13} + x^{11} + x^{10} + x^7 + x^5$	$r_{13}(x) = 0$
$x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4$	$r_{14}(x) = x^3$
$x^{13} + x^{12} + x^9 + x^8 + x^5$	$r_{15}(x) = x^3 + x + 1$
x^{14}	$r_{16}(x) = x^3 + 1$

Setelah memperoleh residu $r_{ij}(x)$ dari hasil pembagian polinomial $b'_{ij}(x).x^4$ oleh polinomial generator $g_1(x) = x^4 + x + 1$, langkah selanjutnya adalah membentuk *codeword* $C_{ij}(x)$. *Codeword* ini diperoleh dengan menjumlahkan hasil perkalian $b'_{ij}(x).x^4$ dengan redisu $r_{ij}(x)$, yaitu $C_{ij}(x) = b'_{ij}(x).x^4 + r_{ij}(x)$. Setelah *codeword* $C_{ij}(x)$ diperoleh dalam bentuk polinomial, proses selanjutnya adalah mengonversi polinomial tersebut ke dalam representasi biner 15-bit. Representasi biner ini menunjukkan posisi suku-suku pangakat x^n yang memiliki koefisien 1 dalam polinomial. Hasil konversi ditampilkan pada Tabel 4.4 berikut:

Tabel 4.4 *Codeword* $C_i(x)$ pada Kode BCH(15,11)

Codeword $\boldsymbol{C_{ij}(x)}$	**Biner** $\boldsymbol{C'_{ij}(x)}$
$C_{11}(x) = x^{13} + x^9 + x^5 + x^4 + x$	[010001000110010]
$C_{12}(x) = x^{12} + x^{10} + x^8 + x^7 + x^6 + x^3 + x$	[001010111001010]
$C_{13}(x) = x^{14} + x^{13} + x^{11} + x^{10} + x^7 + x^5$	[110110010100000]
$C_{14}(x) = x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3$	[111001001111000]
$C_{15}(x) = x^{13} + x^{12} + x^9 + x^8 + x^5 + x^3 + x + 1$	[011001100101011]
$C_{16}(x) = x^{14} + x^3 + 1$	[100000000001001]

3. Penambahan *Error* dan Hasil *Codeword*

Setelah memperoleh setiap *codeword* $C_{ij}(x)$ dan merepresentasikannya dalam bentuk biner 15-bit, langkah selanjutnya dalam simulasi sistem deteksi kesalahan adalah menambahkan *error* $e_i(x)$secara acak ke dalam bit-bit biner. Kesalahan bit ditambahkan secara acak pada posisi berbeda di setiap blok. Setiap $e_i(x)$ hanya memiliki satu bit bernilai 1, yang artinya hanya ada satu *error* per blok. Berikut ditunjukkan Tabel 4.5 yang merupakan posisi kesalahan yang disisipkan:

Tabel 4.5 Contoh Posisi Bit *Error* pada Kode BCH(15,11)

Error	$\boldsymbol{e_i(x)}$	**Posisi *Error* (dari kiri)**
$e_{11}(x)$	[010000000000000]	Bit ke-2
$e_{12}(x)$	[001000000000000]	Bit ke-3
$e_{13}(x)$	[000100000000000]	Bit ke-4
$e_{14}(x)$	[000010000000000]	Bit ke-5
$e_{15}(x)$	[000001000000000]	Bit ke-6
$e_{16}(x)$	[000000100000000]	Bit ke-7

Setelah diketahui bahwa setiap *codeword* mengalami satu kesalahan bit pada posisi tertentu sebagaimana ditunjukkan pada Tabel 4.5, maka diperoleh enam buah *codeword* yang telah mengalami perubahan. Proses penambahan ini dilakukan menggunakan operasi logika XOR antara *codeword* asli $C'_{ij}(x)$ dan vektor kesalahan $e_{ij}(x)$ yang menghasilkan *codeword* baru $C''_{ij}(x)$ yang telah di

tambahkan kesalahan bit. Berikut adalah hasil perubahan masing-masing *codeword* akibat penambahan vektor kesalahan:

$$C''_{11}(x) = C'_{11}(x) \oplus e_{11}(x)$$
$$= [010001000110010] \oplus [010000000000000]$$
$$= [000001000110010]$$

$$C''_{12}(x) = C'_{12}(x) \oplus e_{12}(x)$$
$$= [001010111001010] \oplus [001000000000000]$$
$$= [000010111001010]$$

$$C''_{13}(x) = C'_{13}(x) \oplus e_{13}(x)$$
$$= [110110010100000] \oplus [000100000000000]$$
$$= [110010010100000]$$

$$C''_{14}(x) = C'_{14}(x) \oplus e_{14}(x)$$
$$= [111001001111000] \oplus [000010000000000]$$
$$= [111011001111000]$$

$$C''_{15}(x) = C'_{15}(x) \oplus e_{15}(x)$$
$$= [011001100101011] \oplus [000001000000000]$$
$$= [011000100101011]$$

$$C''_{16}(x) = C'_{15}(x) \oplus e_{16}(x)$$
$$= [100000000001001] \oplus [000000100000000]$$
$$= [100000100001001]$$

Codeword hasil penjumlahan antara masing-masing *codeword* asli dengan vektor kesalahan merupakan data yang akan dikirimkan atau ditransmisikan ke pihak penerima. Data ini telah mengalami satu kesalahan bit pada setiap blok, sebagaimana telah disimulasikan sebelumnya. Keenam blok *codeword* tersebut

kini merepresentasikan pesan “Deserve” dalam bentuk biner yang telah mengalami gangguan. *Codeword* yang diperoleh dari representasi biner pesan “*Deserve*” adalah sebagai berikut:

[000001000110010000010111001010110010010100000111011001111000011000100101011100000100001001].

4. Proses *Decoding* dan Koreksi Kesalahan

Setelah proses transmisi, penerima menerima *codeword* $C_{ij}''(x)$. Proses *decoding* dilakukan untuk memeriksa dan memperbaiki kemungkinan kesalahan yang terjadi selama pengiriman. Tujuan utama proses *decoding* adalah untuk mengembalikan data asli dari *codeword* yang mungkin telah mengalami gangguan. Proses *decoding* pada kode BCH dilakukan menggunakan pendekatan berbasis polinomial yang bekerja dalam lapangan $GF(2)$. Setiap *codeword* $(C_i''(x))$ yang diterima di analisis untuk mengetahui apakah terdapat kesalahan selama proses pengiriman.

Sebelum melakukan perhitungan sindrom, *codeword* yang diterima dalam bentuk biner harus terlebih dahulu dikonversi ke dalam bentuk polinomial.

Tabel 4.6 Konverensi *Codeword* ke dalam Bentuk Polinomial pada Kode BCH(15,11)

Biner ($C_{ij}''(x)$)	**Polinomial**
[000001000110010]	$x^9 + x^5 + x^4 + x$
[000010111001010]	$x^{10} + x^8 + x^7 + x^6 + x^3 + x$
[110010010100000]	$x^{14} + x^{13} + x^{10} + x^7 + x^5$
[111011001111000]	$x^{14} + x^{13} + x^{12} + x^{10} + x^9 + x^6 + x^5 + x^4 + x^3$
[011000100101011]	$x^{13} + x^{12} + x^8 + x^5 + x^3 + x + 1$
[100000100001001]	$x^{14} + x^8 + x^3 + 1$

Langkah pertama dari proses *decoding* adalah menghitung sindrom dari setiap *codeword* yang diterima. Sindrom ini berfungsi sebagai indikator untuk

mengetahui apakah terdapat kesalahan dalam data. Sindrom ke$-ij$ dihitung sebagai evaluasi dari polinomial *codeword* yang rusak, dengan rumus:

$$S_{ij}(x) = r(\alpha)$$

Jika hasil sindrom $S_{ij}(x) = 0$, maka dapat disimpulkan bahwa tidak ada kesalahan dalam *codeword* yang diterima. Sebaliknya, jika $S_{ij}(x) \neq 0$ maka terdapat kesalahan pada bit-bit *codeword*, dan proses koreksi kesalahan perlu dilakukan untuk mengembalikan data dan bentuk aslinya. Namun, pada kasus dengan kapasitas koreksi $t = 1$, sindrom dapat dihitung menggunakan operasi modulo terhadap polinomial generator $g_i(x)$. Hal ini karena nilai sindrom hanya membutuhkan satu untuk mendeteksi dan mengidentifikasi posisi satu kesalahan. Berikut adalah hasil perhitungan sindrom dari setiap blok *codeword* yang diterima:

$$S_{11}(x) = (x^9 + x^5 + x^4 + x)\ mod\ (x^4 + x + 1)$$
$$= x^3 + x^2 + 1$$

$$S_{12}(x) = (x^{10} + x^8 + x^7 + x^6 + x^3 + x)\ mod\ (x^4 + x + 1)$$
$$= x^3 + x^2 + x + 1$$

$$S_{13}(x) = (x^{14} + x^{13} + x^{10} + x^7 + x^5)\ mod\ (x^4 + x + 1)$$
$$= x^3 + x^2 + x$$

$$S_{14}(x) = (x^{14} + x^{13} + x^{12} + x^{10} + x^9 + x^6 + x^5 + x^4 + x^3) mod\ (x^4 + x + 1)$$
$$= x^2 + x + 1$$

$$S_{15}(x) = (x^{13} + x^{12} + x^8 + x^5 + x^3 + x + 1)\ mod\ (x^4 + x + 1)$$
$$= x^3 + x$$

$$S_{16}(x) = (x^{14} + x^8 + x^3 + 1)\ \ mod\ (x^4 + x + 1)$$

$= x^2 + 1$

Setelah menghitung sindrom $S_{ij}(x)$, tahap selanjutnya adalah melakukanidentifikasi posisi *error* berdasarkan nilai sindrom. Setiap nilai sindrom yang tidak sama dengan nol menunjukkan adanya satu bit kesalahan pada posisi tertentu. Karena kode BCH(15,11) mampu mengoreksi satu bit kesalahan, maka proses identifikasi dapat dilakukan secara langsung dari hasil sindrom. Pada $S_{11}(x)$ menunjukkan hasil $x^3 + x^2 + 1$ atau dalam biner $[1101]$, yang dikonversi ke desimal menghasilkan nilai 13. Pada sistem BCH, posisi kesalahan dihitung dari kiri (bit ke-15 sebagai x^{14}). Maka, posisi kesalahan terdapat pada bit ke $15 - 13 = 2$, yaitu pada pangkat x^{13} atau posisi ke-2 dari kiri.

Setelah posisi kesalahan diketahui, maka dilakukan koreksi dengan menambahkan vektor *error* koreksi $e'_{ij}(x)$ yang memiliki bobot 1 pada posisi kesalahan tersebut. Penambahan ini dilakukan menggunakan operasi XOR. Berikut adalah proses koreksi kesalahan:

$P_{11}(x) = C''_{11}(x) \oplus e'_{11}(x)$

$= [000001000110010] \oplus [010000000000000]$

$= [010001000110010]$

$P_{12}(x) = C''_{12}(x) \oplus e'_{12}(x)$

$= [000010111001010] \oplus [001000000000000]$

$= [001010111001010]$

$P_{13}(x) = C''_{13}(x) \oplus e'_{13}(x)$

$= [110010010100000] \oplus [000100000000000]$

$= [110110010100000]$

$$P_{14}(x) = C''_{14}(x) \oplus e'_{14}(x)$$
$$= [111011001111000] \oplus [000010000000000]$$
$$= [111001001111000]$$
$$P_{15}(x) = C''_{15}(x) \oplus e'_{15}(x)$$
$$= [011000100101011] \oplus [000001000000000]$$
$$= [011001100101011]$$
$$P_{16}(x) = C''_{16}(x) \oplus e'_{16}(x)$$
$$= [100000100001001] \oplus [000000100000000]$$
$$= [100000000001001]$$

Setelah seluruh blok *codeword* yang diterima mengalami proses deteksi dan koreksi kesalahan, diperoleh kembali deretan bit hasil *decoding* yang telah bebas dari gangguan. Bit-bit tersebut merupakan gabungan dari enam blok hasil koreksi dengan panjang 15-bit, sehingga membentuk satu rangkaian bit sepanjang 90-bit sebagai berikut:

[010001000110010001010111001010110110010100000111001001
111000011001100101011100000000001001].

Tahap terakhir dalam proses ini adalah membagi deretan bit tersebut menjadi beberapa blok dengan panjang masing-masing 8 bit. Proses ini bertujuan untuk mengkonversi setiap blok biner menjadi karakter yang sesuai berdasarkan tabel ASCII. Setelah dilakukan pembagian dan konversi, diperoleh karakter karakter sebagai berikut:

[0 1 0 0 0 1 0 0] = “D”

[0 1 1 0 0 1 0 1] = ”e”

[0 1 1 1 0 0 1 1] = “s”

[0 1 1 0 0 1 0 1] = “e”

[0 1 1 1 0 0 1 0] = “r”

[0 1 1 1 0 1 1 0] = “v”

[0 1 1 0 0 1 0 1] = “e”

Dengan demikian, hasil akhir dari proses decoding berhasil mengembalikan pesan asli yang dikirimkan yaitu kata “*Deserve*”.

4.2.2 Kode BCH (15,7)

1. Pembentukan Generator Polinomial

Pada parameter BCH(15,7), digunakan lapangan $GF(2^4)$ dengan panjang kode $n = 15$. parameter koreksi kesalahan ditetapkan sebesar $t = 2$, artinya kode ini mampu mengoreksi maksimum dua kesalahan bit per blok. Panjang informasi k dihitung menggunakan ketentuan

$$t \geq \frac{n-k}{m}$$

sehingga diperoleh $k = 7$. Jarak minimum d_{min} memenuhi ketentuan

$$d_{min} \geq 2t + 1$$

sehingga dalam hal ini jarak minimum adalah 5.

Langkah pertama dalam proses ini adalah membangun lapangan $GF(2^4)$ dengan memilih polinomial *irreducible* berderajat empat sebagai dasar pembentukannya. Polinomial *irreducible* pada $GF(2^4)$ tidak dapat difaktorkan menjadi dua polinomial berderajat lebih rendah dengan koefisien 0 dan 1. Akar α dari polinomial tersebut harus mampu menghasilkan semua elemen tak nol pada $GF(2^4)$ melalui pemangkatan, hingga kembali ke 1 pada pangkat ke-15. Oleh karena itu, polinomial primitif

$$p(x) = x^4 + x + 1$$

dapat digunakan sebagai dasar pembentukan $GF(2^4)$. Berdasarkan polinomial primitif tersebut, seluruh elemen dalam medan ini dapat dibangkitkan melalui pemangkatan terhadap α, yang kemudian direduksi dengan operasi modulo terhadap $p(x)$. Setiap hasil pemangkatan α^i dapat direpresentasikan dalam bentuk polinomial dengan koefisien biner, yang selanjutnya dapat dinyatakan dalam bentuk representasi biner 4-bit. Melalui proses tersebut, menghasilkan total 15 elemen tak nol dari α^1 hingga α^{14} sebagaimana ditunjukkan pada Tabel 2.1.

Setelah menentukan polinomial primitif, tahap selanjutnya adalah mengidentifikasi elemen-elemen $GF(2^4)$ yang termasuk dalam grup konjugasi yang sama untuk digunakan dalam pembentukan polinomial minimal. Grup konjugasi dibentuk dari lapangan $GF(2^4)$ yang saling terkait melalui operasi pemangkatan basis dua. Pada $GF(2^4)$, grup konjugasi suatu elemen α^i adalah $\{\alpha^i, \alpha^{2i}, \alpha^{4i}, \alpha^{8i}\}\ mod\ (2^4 - 1)$ diperoleh yang telah ditunjukkan pada Tabel 4.1.

Berdasarkan Tabel 4.1 polinomial minimal dapat ditentukan dari setiap grup konjugasi yang dibentuk oleh elemen-elemen pada lapangan $GF(2^4)$. Setiap grup konjugasi menghasilkan satu polinomial minimal yang merupakan faktor *irreducible* dari polinomial karakteristik medan $GF(2^4)$. Pada pembentukan kode BCH (15,7) dengan kemampuan koreksi kesalahan $t = 2$, digunakan akar $\alpha^1, \alpha^2, \alpha^3,$ dan α^4 untuk membentuk polinomial minimal. Akar ini memiliki grup konjugasi $\{\alpha^1, \alpha^2, \alpha^4, \alpha^8\}$ dengan polinomial minimal

$$m_1(x) = x^4 + x + 1,\ m_2(x) = x^4 + x^3 + 1$$

Karena $\deg((m_1(x)) \cdot (m_2(x))) = 8$, maka diperoleh

$$k = n - \deg\left((m_1(x)) \cdot \left(m_2(x)\right)\right) = 15 - 8 = 7$$

Maka polinomial generator yang digunakan adalah

$$g_2(x) = x^8 + x^7 + x^5 + x^4 + x^3 + x + 1.$$

2. Proses *Encoding*

Pada tahap ini, proses *encoding* dilakukan berdasarkan representasi biner pesan yang telah dikonversi ke dalam format $8-$bit pada parameter BCH(15,11). Pada bagian simulasi ini, digunakan parameter BCH (15,7) yaitu dengan panjang kode $n = 15$, panjang data $k = 7$, dan panjang paritas $n - k = 8$. Dengan panjang paritas 8 bit, kode ini mampu mengoreksi hingga dua bit kesalahan.

Selanjutnya, pesan biner yang telah terbentuk dibagi menjadi beberapa blok $b_{ij}(x)$ dengan panjang 7 bit, sesuai dengan parameter $k = 7$ dari kode BCH (15,7). Setiap blok $7-$ bit kemudian direpresentasikan dalam bentuk polinomial biner, di mana setiap bit yang bernilai 1 menunjukkan adanya suku pada pangkat x yang bersesuaian. Representasi biner dan polinomial dari masing-masing blok ditunjukkan pada Tabel 4.7 berikut:

Tabel 4.7 Representasi Pesan Biner dan Polinomial pada Kode BCH(15,7)

Biner	**Polinomial $b'_{ij}(x)$**
$b_{21} = [0100010]$	$b'_{21}(x) = x^5 + x$
$b_{22} = [0011001]$	$b'_{22}(x) = x^4 + x^3 + 1$
$b_{23} = [0101110]$	$b'_{23}(x) = x^5 + x^3 + x^2 + x$
$b_{24} = [0110110]$	$b'_{24}(x) = x^5 + x^4 + x^2 + x$
$b_{25} = [0101011]$	$b'_{25}(x) = x^5 + x^3 + x + 1$
$b_{26} = [1001001]$	$b'_{26}(x) = x^6 + x^3 + 1$
$b_{27} = [1101100]$	$b'_{27}(x) = x^6 + x^5 + x^3 + x^2$
$b_{28} = [1100101]$	$b'_{28}(x) = x^6 + x^5 + x^2 + 1$

Bedasarkan Tabel 4.7 representasi polinomial diperoleh dengan mengonversi setiap blok data biner 7-bit ke dalam bentuk polinomial biner dengan derajat paling tinggi sesuai dengan posisi bit tertinggi bernilai satu. Setiap bit dalam blok biner mewakili koefisien dari suku-suku dalam polinomial, dimulai dari bit paling kiri sebagai koefisien x^6 dan seterusnya hingga bit paling kanan sebagai konstanta atau x^0.

Setelah setiap blok data $b_{ij}(x)$ direpresentasikan dalam bentuk polinomial, langkah selanjutnya adalah melakukan proses *encoding* BCH. Tahap pertama dalam proses ini adalah mengalikan polinomial $b'_{ij}(x)$ dengan $x^{n-k} = x^{15-7} = x^8$ untuk menyisipkan ruang bit paritas. Selanjutnya, hasil perkalian tersebut dibagi dengan polinomial generator $g_2(x) = x^8 + x^7 + x^5 + x^4 + x^3 + x + 1$ menggunakan metode pembagian polinomial biner. Sisa dari pembagian ini disebut residu $r_{ij}(x)$, yang akan menjadi bit paritas dalam *codeword* akhir.

Tabel 4.8 Hasil Perkalian dan Residu pada Kode BCH(15,7)

$\boldsymbol{b'_{ij}(x).x^8}$	**Residu** $\boldsymbol{r_{ij}(x)}$
$x^{13} + x^9$	$r_{21}(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x$
$x^{12} + x^{11} + x^8$	$r_{22}(x) = x^6 + x^5 + x^4 + x^3 + x^2 + 1$
$x^{13} + x^{11} + x^{10} + x^9$	$r_{23}(x) = x^4 + x^3 + x^2 + 1$
$x^{13} + x^{12} + x^{10} + x^9$	$r_{24}(x) = x^7 + x^6 + x^4 + x^3 + x + 1$
$x^{13} + x^{11} + x^9 + x^8$	$r_{25}(x) = x^7 + x^2 + x + 1$
$x^{14} + x^{11} + x^8$	$r_{26}(x) = x^5 + x^2$
$x^{14} + x^{13} + x^{11} + x^{10}$	$r_{27}(x) = x^3 + x^2 + 1$
$x^{14} + x^{13} + x^{10} + x^8$	$r_{28}(x) = x^7 + x^6 + x^5 + x^4 + x^2$

Setelah memperoleh residu $r_{ij}(x)$ dari hasil pembagian polinomial $b'_{ij}(x).x^8$ oleh polinomial generator $g_2(x) = x^8 + x^7 + x^5 + x^4 + x^3 + x + 1$, langkah selanjutnya adalah membentuk *codeword* $C_{ij}(x)$. *Codeword* ini diperoleh dengan menjumlahkan hasil perkalian $b'_{ij}(x).x^8$ dengan redisu $r_{ij}(x)$,

yaitu $C_{ij}(x) = b'_{ij}(x).x^8 + r_{ij}(x)$. Setelah *codeword* $C_{ij}(x)$ diperoleh dalam bentuk polinomial, proses selanjutnya adalah mengonversi polinomial tersebut ke dalam representasi biner 15-bit. Representasi biner ini menunjukkan posisi suku-suku pangakat x^n yang memiliki koefisien 1 dalam polinomial. Hasil konversi ditampilkan pada Tabel 4.9 berikut:

Tabel 4.9 *Codeword* $C_i(x)$ pada Kode BCH(15,7)

Codeword $\boldsymbol{C_{ij}(x)}$	**Biner** $\boldsymbol{C'_{ij}(x)}$
$C_{21}(x) = x^{13} + x^9 + x^6 + x^5 + x^4 + x^3 + x^2 + x$	[0100010011111110]
$C_{22}(x) = x^{12} + x^{11} + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + 1$	[0011001011111101]
$C_{23}(x) = x^{13} + x^{11} + x^{10} + x^9 + x^4 + x^3 + x^2 + 1$	[0101110000111101]
$C_{24}(x) = x^{13} + x^{12} + x^{10} + x^9 + x^7 + x^6 + x^4 + x^3 + x + 1$	[0110110110110111]
$C_{25}(x) = x^{13} + x^{11} + x^9 + x^8 + x^7 + x^2 + x + 1$	[0101011100001111]
$C_{26}(x) = x^{14} + x^{11} + x^8 + x^5 + x^2$	[1001001001001000]
$C_{27}(x) = x^{14} + x^{13} + x^{11} + x^{10} + x^3 + x^2 + 1$	[1101100000011010]
$C_{28}(x) = x^{14} + x^{13} + x^{10} + x^8 + x^7 + x^6 + x^5 + x^4 + x^2$	[1100101111101010]

3. Penambahan *Error* dan Hasil *Codeword*

Setelah memperoleh setiap *codeword* $C_{ij}(x)$ dan merepresentasikannya dalam bentuk biner 15-bit, langkah selanjutnya dalam simulasi sistem deteksi kesalahan adalah menambahkan *error* $e_{ij}(x)$secara acak ke dalam bit-bit biner. Kesalahan bit ditambahkan secara acak pada posisi berbeda di setiap blok. Setiap $e_{ij}(x)$ hanya memiliki satu bit bernilai 1, yang artinya hanya ada satu *error* per blok. Berikut ditunjukkan Tabel 4.10 yang merupakan posisi kesalahan yang disisipkan:

Tabel 4.10 Contoh Posisi Bit *Error* pada Kode BCH(15,7)

Error	$\boldsymbol{e_i(x)}$	**Posisi *Error* (dari kiri)**
$e_1(x)$	[011000000000000]	Bit ke-2 dan ke-3
$e_2(x)$	[001100000000000]	Bit ke-3 dan ke-4
$e_3(x)$	[000110000000000]	Bit ke-4 dan ke-5

$e_4(x)$	[000011000000000]	Bit ke-5 dan ke-6
$e_5(x)$	[000001100000000]	Bit ke-6 dan ke-7
$e_6(x)$	[010000100000000]	Bit ke-2 dan ke-7
$e_7(x)$	[011000000000000]	Bit ke-2 dan ke-3
$e_8(x)$	[001100000000000]	Bit ke-3 dan ke-4

Setelah diketahui bahwa setiap *codeword* mengalami satu kesalahan bit pada posisi tertentu sebagaimana ditunjukkan pada Tabel 4.10, maka diperoleh enam buah *codeword* yang telah mengalami perubahan. Proses penambahan ini dilakukan menggunakan operasi logika XOR antara *codeword* asli $C'_{ij}(x)$ dan vektor kesalahan $e_{ij}(x)$ yang menghasilkan *codeword* baru $C''_{ij}(x)$ yang telah di tambahkan kesalahan bit. Berikut adalah hasil perubahan masing-masing *codeword* akibat penambahan vektor kesalahan:

$$C''_{21}(x) = C'_{21}(x) \oplus e_{21}(x)$$
$$= [010001001111110] \oplus [011000000000000]$$
$$= [001001001111110]$$

$$C''_{22}(x) = C'_{22}(x) \oplus e_{22}(x)$$
$$= [001100101111101] \oplus [001100000000000]$$
$$= [000000101111101]$$

$$C''_{23}(x) = C'_{23}(x) \oplus e_{23}(x)$$
$$= [010111000011101] \oplus [000110000000000]$$
$$= [010001000011101]$$

$$C''_{24}(x) = C'_{24}(x) \oplus e_{24}(x)$$
$$= [011011011011011] \oplus [000011000000000]$$
$$= [011000011011011]$$

$$C''_{25}(x) = C'_{25}(x) \oplus e_{25}(x)$$
$$= [010101110000111] \oplus [000001100000000]$$
$$= [010100010000111]$$
$$C''_{26}(x) = C'_{26}(x) \oplus e_{26}(x)$$
$$= [100100100100100] \oplus [010000100000000]$$
$$= [110100000100100]$$
$$C''_{27}(x) = C'_{27}(x) \oplus e_{27}(x)$$
$$= [110110000001101] \oplus [011000000000000]$$
$$= [101110000001101]$$
$$C''_{28}(x) = C'_{28}(x) \oplus e_{28}(x)$$
$$= [110010111110101] \oplus [001100000000000]$$
$$= [111110111110101]$$

Codeword hasil penjumlahan antara masing-masing *codeword* asli dengan vektor kesalahan merupakan data yang akan dikirimkan atau ditransmisikan ke pihak penerima. Data ini telah mengalami satu kesalahan bit pada setiap blok, sebagaimana telah disimulasikan sebelumnya. Kedelapan blok *codeword* tersebut kini merepresentasikan pesan "*Deserve*" dalam bentuk biner yang telah mengalami gangguan. *Codeword* yang diperoleh dari representasi biner pesan "*Deserve*" adalah sebagai berikut:

[00100100111111000000010111110101000100001110101100001101101010001000011111010000010010010111000000110111111011111010101].

4. Proses *Decoding* dan Koreksi Kesalahan

Setelah proses transmisi, penerima menerima *codeword* $C''_{ij}(x)$. Proses *decoding* dilakukan untuk memeriksa dan memperbaiki kemungkinan kesalahan yang terjadi selama pengiriman. Tujuan utama proses *decoding* adalah untuk mengembalikan data asli dari *codeword* yang mungkin telah mengalami gangguan. Proses *decoding* pada kode BCH dilakukan menggunakan pendekatan berbasis polinomial yang bekerja dalam lapangan $GF(2)$. Setiap *codeword* $(C''_{ij}(x))$ yang diterima di analisis untuk mengetahui apakah terdapat kesalahan selama proses pengiriman.

Sebelum melakukan perhitungan sindrom, *codeword* yang diterima dalam bentuk biner harus terlebih dahulu dikonversi ke dalam bentuk polinomial.

Tabel 4.11 Konverensi *Codeword* ke dalam bentuk Polinomial pada Kode BCH(15,7)

Biner $C''_{ij}(x)$	**Polinomial**
[001001001111110]	$x^{12}+x^9+x^6+x^5+x^4+x^3+x^2+x$
[000000101111101]	$x^8+x^6+x^5+x^4+x^3+x^2+1$
[010001000011101]	$x^{13}+x^9+x^4+x^3+x^2+1$
[011000011011011]	$x^{13}+x^{12}+x^7+x^6+x^4+x^3+x+1$
[010100010000111]	$x^{13}+x^{11}+x^7+x^2+x+1$
[110100000100100]	$x^{14}+x^{13}+x^{11}+x^5+x^2$
[101110000001101]	$x^{14}+x^{12}+x^{11}+x^{10}+x^3+x^2+1$
[111110111110101]	$x^{14}+x^{13}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^6+x^5+x^4+x^2+1$

Langkah pertama dari proses *decoding* adalah menghitung sindrom dari setiap *codeword* yang diterima. Sindrom ini berfungsi sebagai indikator untuk mengetahui apakah terdapat kesalahan dalam data. Sindrom dihitung menggunakan operasi modulo terhadap polinomial generator $g_2(x)$, dengan rumus

$$S_j(x) = r(\alpha^j) = \sum_{i=0}^{n-1} r_i \cdot (\alpha^j)^i$$

Jika hasil sindrom $S_i(x) = 0$, maka dapat disimpulkan bahwa tidak ada kesalahan dalam *codeword* yang diterima. Sebaliknya, jika $S_i(x) \neq 0$ maka terdapat kesalahan pada bit-bit *codeword*, dan proses koreksi kesalahan perlu dilakukan untuk mengembalikan data dan bentuk aslinya. Berikut adalah hasil perhitungan sindrom dari setiap blok *codeword* yang diterima:

Tabel 4.12 Hasil Sindrom pada Kode BCH(15,7)

***Codeword* Biner**	S_1	S_2	S_3	S_4
[001001001111110]	α^1	α^2	α^7	α^4
[000000101111101]	1	1	α^7	1
[010001000011101]	α^{14}	α^{13}	α^{13}	α^{11}
[011000011011011]	α^{13}	α^{11}	α^{11}	α^7
[010100010000111]	α^{12}	α^9	α^7	α^3
[110100000100100]	α^3	α^6	0	α^{12}
[101110000001101]	α^1	α^2	α^{13}	α^4
[111110111110101]	1	1	α^{13}	1

Berdasarkan hasil sindrom yang ditunjukkan pada Tabel 4.12, seluruh *codeword* menunjukkan satu atau lebih sindrom bernilai tak nol. Oleh karena itu, dapat disimpulkan bahwa seluruh *codeword* mengalami kesalahan dalam proses transmisi. Langkah sleanjutnya yang dilakukan adalah koreksi kesalahan. Proses koreksi kesalahan pada kode BCH dilakukan dengan menggunakan algoritma *Berlekamp-Massey*, yang berfungsi untuk menemukan polinomial *error locator* $\Lambda(x)$. Akar-akar dari polinomial ini menunjukkan posisi bit yang mengalami kesalahan dalam *codeword*. Proses ini dimulai dengan inisialisasi nilai awal

$$\Lambda^{(0)}(x) = 1, B(x) = 1, L = 0, m = 1, dan\ b = 1$$

Pada iterasi pertama, *disrepancy* $d_1$diketahui bernilai α^1, sehingga polinomial di *update* menjadi:

$$\Lambda^{(1)}(x) = \Lambda^{(1)}(x) + \frac{d_1}{b} x^m B(x) = 1 + \frac{\alpha^1}{1} x^1 = 1 + \alpha^5 x$$

Iterasi kedua menunjukkan *disrepancy* $d_2 = S_2 + \Lambda_1 \cdot S_1 = \alpha^2 + \alpha^1 \cdot \alpha^1 = \alpha^2 + \alpha^2 = 0$, karena $d_2 = 0$ sehingga tidak ada perubahan pada polinomial dan didapatkan $\Lambda^{(2)}(x) = \Lambda^{(1)}(x) = 1 + \alpha^1 x$. Pada iterasi ketiga, *disrepancy* $d_3 = \alpha^7$, sehingga diperoleh hasil akhir polinomial *error locator* yaitu:

$$\Lambda(\mathrm{x}) = 1 + \alpha^1 x + \alpha^{14} x^2$$

Setelah polinomial *error locator* diperoleh, dilakukan proses *Chien Search* untuk mencari akar dari $\Lambda(\mathrm{x})$ dengan cara mensubstitusikan α^{-i} ke dalam polinomial, untuk $i = 0,1,\ldots,14$. Akr-akar dari $\Lambda(x)$ adalah posisi bit yang salah. Dari hasil evaluasi diperoleh bahwa $\Lambda(\alpha^{-3}) = 0$ dan $\Lambda(\alpha^{-2}) = 0$, sehingga dapat disimpulkan bahwa posisi bit yang salah berada pda bit ke-3 dan ke-2 dari sebelah kiri. Langkah selanjutnya adalah melakukan koreksi dengan membalik bit-bit pada posisi tersebut, menghasilkan *codeword* baru yaitu:

Tabel 4.13 *Codeword* pada Kode BCH(15,7)

Codeword $\boldsymbol{C''_{ij}(x)}$	***Codeword*** $\boldsymbol{P_i(x)}$
[001001001111110]	[010001001111110]
[000000101111101]	[001100101111101]
[010001000011101]	[010111000011101]
[011000011011011]	[011011011011011]
[010100010000111]	[010101110000111]
[110100000100100]	[100100100100100]
[101110000001101]	[110110000001101]
[111110111110101]	[110010111110101]

Setelah seluruh blok *codeword* yang diterima mengalami proses deteksi dan koreksi kesalahan, diperoleh kembali deretan bit hasil *decoding* yang telah bebas dari gangguan. Bit-bit tersebut merupakan gabungan dari enam blok hasil

koreksi dengan panjang 15-bit, sehingga membentuk satu rangkaian bit sepanjang 120-bit sebagai berikut:

[010001001111110001100101111101010111000011101011011011011
011010101110000111100100100100100110110000001101110010111110101].

Tahap terakhir dalam proses ini adalah membagi deretan bit tersebut menjadi beberapa blok dengan panjang masing-masing 8 bit. Proses ini bertujuan untuk mengkonversi setiap blok biner menjadi karakter yang sesuai berdasarkan tabel ASCII. Setelah dilakukan pembagian dan konversi, diperoleh karakter karakter sebagai berikut:

[0 1 0 0 0 1 0 0] = “D”

[0 1 1 0 0 1 0 1] = ”e”

[0 1 1 1 0 0 1 1] = “s”

[0 1 1 0 0 1 0 1] = “e”

[0 1 1 1 0 0 1 0] = “r”

[0 1 1 1 0 1 1 0] = “v”

[0 1 1 0 0 1 0 1] = “e”

Dengan demikian, hasil akhir dari proses decoding berhasil mengembalikan pesan asli yang dikirimkan yaitu kata “*Deserve*”.

4.2.3 Kode BCH (15,5)

1. Pembentukan Generator Polinomial

Pada parameter BCH(15,5), digunakan lapangan $GF(2^4)$ dengan panjang kode $n = 15$. parameter koreksi kesalahan ditetapkan sebesar $t = 3$, artinya

kode ini mampu mengoreksi maksimum tiga kesalahan bit per blok. Panjang informasi k dihitung menggunakan ketentuan

$$t \geq \frac{n-k}{m}$$

sehingga diperoleh $k = 5$. Jarak minimum d_{min} memenuhi ketentuan

$$d_{min} \geq 2t + 1$$

sehingga dalam hal ini jarak minimum adalah 7.

Langkah pertama dalam proses ini adalah membangun lapangan $GF(2^4)$ dengan memilih polinomial *irreducible* berderajat empat sebagai dasar pembentukannya. Polinomial *irreducible* pada $GF(2^4)$ tidak dapat difaktorkan menjadi dua polinomial berderajat lebih rendah dengan koefisien 0 dan 1. Akar α dari polinomial tersebut harus mampu menghasilkan semua elemen tak nol pada $GF(2^4)$ melalui pemangkatan, hingga kembali ke 1 pada pangkat ke-15. Oleh karena itu, polinomial primitif

$$p(x) = x^4 + x + 1$$

dapat digunakan sebagai dasar pembentukan $GF(2^4)$. Berdasarkan polinomial primitif tersebut, seluruh elemen dalam medan ini dapat dibangkitkan melalui pemangkatan terhadap α, yang kemudian direduksi dengan operasi modulo terhadap $p(x)$. Setiap hasil pemangkatan α^i dapat direpresentasikan dalam bentuk polinomial dengan koefisien biner, yang selanjutnya dapat dinyatakan dalam bentuk representasi biner 4-bit. Melalui proses tersebut, menghasilkan total 15 elemen tak nol dari α^1 hingga α^{14} sebagaimana ditunjukkan pada Tabel 2.1.

Setelah menentukan polinomial primitif, tahap selanjutnya adalah mengidentifikasi elemen-elemen $GF(2^4)$ yang termasuk dalam grup konjugasi

yang sama untuk digunakan dalam pembentukan polinomial minimal. Grup konjugasi dibentuk dari lapangan $GF(2^4)$ yang saling terkait melalui operasi pemangkatan basis dua. Pada $GF(2^4)$, grup konjugasi suatu elemen α^i adalah $\{\alpha^i, \alpha^{2i}, \alpha^{4i}, \alpha^{8i}\}\ mod\ (2^4 - 1)$ diperoleh yang telah ditunjukkan pada Tabel 4.1.

Berdasarkan Tabel 4.1 polinomial minimal dapat ditentukan dari setiap grup konjugasi yang dibentuk oleh elemen-elemen pada lapangan $GF(2^4)$. Setiap grup konjugasi menghasilkan satu polinomial minimal yang merupakan faktor *irreducible* dari polinomial karakteristik medan $GF(2^4)$. Pada pembentukan kode BCH (15,5) dengan kemampuan koreksi kesalahan $t = 3$, digunakan akar $\alpha^1, \alpha^2, \alpha^3,\ \alpha^4, \alpha^5$ dan α^6 untuk membentuk polinomial minimal. Akar ini memiliki grup konjugasi $\{\alpha^1, \alpha^2, \alpha^4, \alpha^8\}$ dengan polinomial minimal

$$m_1(x) = x^4 + x + 1,\ m_2(x) = x^4 + x^3 + 1, m_3(x) = (x^2 + x + 1)$$

Karena $\deg\Big(\big(m_1(x)\big) \cdot \big(m_2(x)\big) \cdot \big(m_3(x)\big)\Big) = 10$, maka diperoleh

$$k = n - \deg\Big(\big(m_1(x)\big) \cdot \big(m_2(x)\big) \cdot \big(m_3(x)\big)\Big) = 15 - 10 = 5$$

Maka polinomial generator yang digunakan adalah $g_3(x) = x^{10} + x^5 + 1$.

2. Proses *Encoding*

Pada tahap ini, proses *encoding* dilakukan berdasarkan representasi biner pesan yang telah dikonversi ke dalam format $8-$bit pada parameter BCH(15,11). Pada bagian simulasi ini, digunakan parameter BCH (15,5) yaitu dengan panjang kode $n = 15$, panjang data $k = 5$, dan panjang paritas $n - k = 10$. Dengan panjang paritas 10 bit, kode ini mampu mengoreksi hingga tiga bit kesalahan.

Selanjutnya, pesan biner yang telah terbentuk dibagi menjadi beberapa blok $b_{ij}(x)$ dengan panjang 5 bit, sesuai dengan parameter $k = 5$ dari kode BCH (15,5). Setiap blok $5-$ bit kemudian direpresentasikan dalam bentuk polinomial biner, di mana setiap bit yang bernilai 1 menunjukkan adanya suku pada pangkat x yang bersesuaian. Representasi biner dan polinomial dari masing-masing blok ditunjukkan pada Tabel 4.14 berikut:

Tabel 4.14 Representasi Pesan Biner dan Polinomial pada BCH(15,5)

Biner	**Polinomial $c_i'(x)$**
$b_{31} = [01000]$	$b_{31}'(x) = x^3$
$b_{32} = [10001]$	$b_{32}'(x) = x^4 + 1$
$b_{33} = [10010]$	$b_{33}'(x) = x^4 + x$
$b_{34} = [10111]$	$b_{34}'(x) = x^4 + x^2 + x$
$b_{35} = [00110]$	$b_{35}'(x) = x^2 + x$
$b_{36} = [11001]$	$b_{36}'(x) = x^4 + x^3 + 1$
$b_{37} = [01011]$	$b_{37}'(x) = x^3 + x + 1$
$b_{38} = [10010]$	$b_{38}'(x) = x^4 + x$
$b_{39} = [01110]$	$b_{39}'(x) = x^3 + x^2 + x$
$b_{310} = [11001]$	$b_{310}'(x) = x^4 + x^3 + 1$
$b_{311} = [10010]$	$b_{311}'(x) = x^4 + x$
$b_{312} = [10000]$	$b_{312}'(x) = x^4$

Bedasarkan Tabel 4.14 representasi polinomial diperoleh dengan mengonversi setiap blok data biner 5-bit ke dalam bentuk polinomial biner dengan derajat paling tinggi sesuai dengan posisi bit tertinggi bernilai satu. Setiap bit dalam blok biner mewakili koefisien dari suku-suku dalam polinomial, dimulai dari bit paling kiri sebagai koefisien x^4 dan seterusnya hingga bit paling kanan sebagai konstanta atau x^0.

Setelah setiap blok data $c_i(x)$ direpresentasikan dalam bentuk polinomial, langkah selanjutnya adalah melakukan proses *encoding* BCH. Tahap pertama dalam proses ini adalah mengalikan polinomial $b_{ij}'(x)$ dengan $x^{n-k} = x^{15-5} = x^{10}$ untuk menyisipkan ruang bit paritas. Selanjutnya, hasil perkalian tersebut

dibagi dengan polinomial generator $g_3(x) = x^{10} + x^5 + 1$ menggunakan metode pembagian polinomial biner. Sisa dari pembagian ini disebut residu $r_{ij}(x)$, yang akan menjadi bit paritas dalam *codeword* akhir.

Tabel 4.15 Hasil Perkalian dan Residu pada Kode BCH(15,5)

$\boldsymbol{b'_{ij}(x).x^{10}}$	**Residu** $\boldsymbol{r_{ij}(x)}$
x^{13}	$r_{31}(x) = x^8 + x^3$
$x^{14} + x^{10}$	$r_{32}(x) = x^9 + x^5 + x^4 + 1$
$x^{14} + x^{11}$	$r_{33}(x) = x^9 + x^6 + x^4 + x$
$x^{14} + x^{12} + x^{11} + x^{10}$	$r_{34}(x) = x^9 + x^7 + x^6 + x^5 + x^4 + x^2 + x + 1$
$x^{12} + x^{11}$	$r_{35}(x) = x^7 + x^6 + x^2 + x$v
$x^{14} + x^{13} + x^{10}$	$r_{36}(x) = x^9 + x^8 + x^5 + x^4 + x^3 + 1$
$x^{13} + x^{11} + x^{10}$	$r_{37}(x) = x^8 + x^6 + x^5 + x^3 + x + 1$
$x^{14} + x^{11}$	$r_{38}(x) = x^9 + x^6 + x^4 + x$
$x^{13} + x^{12} + x^{11}$	$r_{39}(x) = x^8 + x^7 + x^6 + x^3 + x^2 + x$
$x^{14} + x^{13} + x^{10}$	$r_{310}(x) = x^9 + x^8 + x^5 + x^4 + x^3 + 1$
$x^{14} + x^{11}$	$r_{311}(x) = x^9 + x^6 + x^4 + x$
x^{14}	$r_{312}(x) = x^9 + x^4$

Setelah memperoleh residu $r_{ij}(x)$ dari hasil pembagian polinomial $b'_{ij}(x).x^{10}$ oleh polinomial generator $g_3(x) = x^{10} + x^5 + 1$, langkah selanjutnya adalah membentuk *codeword* $C_{ij}(x)$. *Codeword* ini diperoleh dengan menjumlahkan hasil perkalian $b'_{ij}(x).x^{10}$ dengan redisu $r_{ij}(x)$, yaitu $C_{ij}(x) = b'_{ij}(x).x^{10} + r_{ij}(x)$. Setelah *codeword* $C_{ij}(x)$ diperoleh dalam bentuk polinomial, proses selanjutnya adalah mengonversi polinomial tersebut ke dalam representasi biner 15-bit. Representasi biner ini menunjukkan posisi suku-suku pangakat x^n yang memiliki koefisien 1 dalam polinomial. Hasil konversi ditampilkan pada Tabel 4.16 berikut:

Tabel 4.16 *Codeword* $C_i(x)$ pada BCH(15,5)

$\boldsymbol{Codeword\ C_{ij}(x)}$	**Biner** $\boldsymbol{C''_{ij}(x)}$
$C_{31}(x) = x^{13} + x^8 + x^3$	[010000100001000]
$C_{32}(x) = x^{14} + x^{10} + x^9 + x^5 + x^4 + 1$	[100011000110001]
$C_{33}(x) = x^{14} + x^{11} + x^9 + x^6 + x^4 + x$	[100101001010010]

$C_{34}(x) = x^{14} + x^{12} + x^{11} + x^{10} + x^9 + x^7 + x^6 + x^5 + x^4 + x^2 + x + 1$	[101111011110111]
$C_{35}(x) = x^{12} + x^{11} + x^7 + x^6 + x^2 + x$	[001100011000110]
$C_{36}(x) = x^{14} + x^{13} + x^{10} + x^9 + x^8 + x^5 + x^4 + x^3 + 1$	[110011100111001]
$C_{37}(x) = x^{13} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^3 + x + 1$	[010110101101011]
$C_{38}(x) = x^{14} + x^{11} + x^9 + x^6 + x^4 + x$	[100101001010010]
$C_{39}(x) = x^{13} + x^{12} + x^{11} + x^8 + x^7 + x^6 + x^3 + x^2 + x$	[011100111001110]
$C_{310}(x) = x^{14} + x^{13} + x^{10} + x^9 + x^8 + x^5 + x^4 + x^3 + 1$	[110011100111001]
$C_{311}(x) = x^{14} + x^9 + x^6 + x^4 + x$	[100101001010010]
$C_{312}(x) = x^9 + x^4$	[100001000010000]

3. Penambahan *Error* dan Hasil *Codeword*

Setelah memperoleh setiap *codeword* $C_{ij}(x)$ dan merepresentasikannya dalam bentuk biner 15-bit, langkah selanjutnya dalam simulasi sistem deteksi kesalahan adalah menambahkan *error* $e_{ij}(x)$secara acak ke dalam bit-bit biner. Kesalahan bit ditambahkan secara acak pada posisi berbeda di setiap blok. Setiap $e_{ij}(x)$ hanya memiliki satu bit bernilai 1, yang artinya hanya ada satu *error* per blok. Berikut ditunjukkan Tabel 4.17 yang merupakan posisi kesalahan yang disisipkan:

Tabel 4.17 Contoh Posisi Bit *Error* pada BCH(15,5)

Error	$\boldsymbol{e_i(x)}$	**Posisi *Error* (dari kiri)**
$e_1(x)$	[011100000000000]	Bit ke-13, ke-12 dan ke-11
$e_2(x)$	[001110000000000]	Bit ke-12, ke-11 dan ke-10
$e_3(x)$	[000111000000000]	Bit ke-11, ke-10 dan ke-9
$e_4(x)$	[000011100000000]	Bit ke-10, ke-9 dan ke-8
$e_5(x)$	[010001100000000]	Bit ke-9, ke-8 dan ke-13
$e_6(x)$	[011000100000000]	Bit ke-8, ke-13 dan ke-12
$e_7(x)$	[011100000000000]	Bit ke-13, ke-12 dan ke-11
$e_8(x)$	[001110000000000]	Bit ke-12, ke-11 dan ke-10
$e_9(x)$	[000111000000000]	Bit ke-11, ke-10 dan ke-9
$e_{10}(x)$	[000011100000000]	Bit ke-10, ke-9 dan ke-8
$e_{11}(x)$	[010001100000000]	Bit ke-9, ke-8 dan ke-13
$e_{12}(x)$	[011000100000000]	Bit ke-8, ke-13 dan ke-12

Setelah diketahui bahwa setiap *codeword* mengalami satu kesalahan bit pada posisi tertentu sebagaimana ditunjukkan pada Tabel 4.17, maka diperoleh enam buah *codeword* yang telah mengalami perubahan. Proses penambahan ini dilakukan menggunakan operasi logika XOR antara *codeword* asli $C'_{ij}(x)$ dan vektor kesalahan $e_{ij}(x)$ yang menghasilkan *codeword* baru $C''_{ij}(x)$ yang telah di tambahkan kesalahan bit. Berikut adalah hasil perubahan masing-masing *codeword* akibat penambahan vektor kesalahan:

$$C''_{31}(x) = \mathrm{C}'_{31}(x) \oplus e_{31}(x)$$
$$= [010000100001000] \oplus [011100000000000]$$
$$= [001100100001000]$$

$$C''_{32}(x) = \mathrm{C}'_{32}(x) \oplus e_{32}(x)$$
$$= [100011000110001] \oplus [001110000000000]$$
$$= [101101000110001]$$

$$C''_{33}(x) = \mathrm{C}'_{33}(x) \oplus e_{33}(x)$$
$$= [100101001010010] \oplus [000111000000000]$$
$$= [100010001010010]$$

$$C''_{34}(x) = \mathrm{C}'_{34}(x) \oplus e_{34}(x)$$
$$= [101111011110111] \oplus [000011100000000]$$
$$= [101100111110111]$$

$$C''_{35}(x) = \mathrm{C}'_{35}(x) \oplus e_{35}(x)$$
$$= [001100011000110] \oplus [010001100000000]$$
$$= [011101111000110]$$

$$C''_{36}(x) = \mathrm{C}'_{36}(x) \oplus e_{36}(x)$$
$$= [110011100111001] \oplus [011000100000000]$$
$$= [101011000111001]$$
$$C''_{37}(x) = \mathrm{C}'_{37}(x) \oplus e_{37}(x)$$
$$= [010110101101011] \oplus [011100000000000]$$
$$= [001010101101011]$$
$$C''_{38}(x) = \mathrm{C}'_{38}(x) \oplus e_{38}(x)$$
$$= [100101001010010] \oplus [001110000000000]$$
$$= [101011001010010]$$
$$C''_{39}(x) = \mathrm{C}'_{39}(x) \oplus e_{39}(x)$$
$$= [011100111001110] \oplus [000111000000000]$$
$$= [011011111001110]$$
$$C''_{310}(x) = \mathrm{C}'_{310}(x) \oplus e_{310}(x)$$
$$= [110011100111001] \oplus [000011100000000]$$
$$= [110000000111001]$$
$$C''_{311}(x) = \mathrm{C}'_{311}(x) \oplus e_{311}(x)$$
$$= [100101001010010] \oplus [010001100000000]$$
$$= [110100101010010]$$
$$C''_{312}(x) = \mathrm{C}'_{312}(x) \oplus e_{312}(x)$$
$$= [100001000010000] \oplus [011000100000000]$$
$$= [111001100010000]$$

Codeword hasil penjumlahan antara masing-masing *codeword* asli dengan vektor kesalahan merupakan data yang akan dikirimkan atau ditransmisikan ke

pihak penerima. Data ini telah mengalami satu kesalahan bit pada setiap blok, sebagaimana telah disimulasikan sebelumnya. Keenam blok *codeword* tersebut kini merepresentasikan pesan "*Deserve*" dalam bentuk biner yang telah mengalami gangguan. *Codeword* yang diperoleh dari representasi biner pesan "*Deserve*" adalah sebagai berikut:

[001100100001000101101000110001100010001010010101100111110111011101111000110101011000111001001010101101011101011001010010011011111001110110000000111001110100101010010111001100010000].

4. Proses *Decoding* dan Koreksi Kesalahan

Setelah proses transmisi, penerima menerima *codeword* $C''_{ij}(x)$. Proses *decoding* dilakukan untuk memeriksa dan memperbaiki kemungkinan kesalahan yang terjadi selama pengiriman. Tujuan utama proses *decoding* adalah untuk mengembalikan data asli dari *codeword* yang mungkin telah mengalami gangguan. Proses *decoding* pada kode BCH dilakukan menggunakan pendekatan berbasis polinomial yang bekerja dalam lapangan $GF(2)$. Setiap *codeword* $(C''_{ij}(x))$ yang diterima di analisis untuk mengetahui apakah terdapat kesalahan selama proses pengiriman.

Sebelum melakukan perhitungan sindrom, *codeword* yang diterima dalam bentuk biner harus terlebih dahulu dikonversi ke dalam bentuk polinomial.

Tabel 4.18 Konverensi *Codeword* ke dalam bentuk Polinomial pada Kode BCH(15,5)

Biner $C''_{ij}(x)$	**Polinomial**
[001100100001000]	$x^{13} + x^{11} + x^8 + x^3$
[101101000110001]	$x^{14} + x^{12} + x^{11} + x^9 + x^5 + x^4 + 1$
[100010001010010]	$x^{14} + x^{10} + x^6 + x^4 + x$

[101100111110111]	$x^{14}+x^{12}+x^{11}+x^{8}+x^{7}+x^{6}+x^{5}+x^{4}+x^{2}+x+1$
[011101111000110]	$x^{13}+x^{12}+x^{11}+x^{9}+x^{8}+x^{7}+x^{6}+x^{2}+x$
[101011000111001]	$x^{14}+x^{12}+x^{10}+x^{9}+x^{5}+x^{4}+x^{3}+1$
[001010101101011]	$x^{12}+x^{10}+x^{8}+x^{6}+x^{5}+x^{3}+x+1$
[101011001010010]	$x^{14}+x^{12}+x^{10}+x^{9}+x^{6}+x^{4}+x$
[011011111001110]	$x^{13}+x^{12}+x^{10}+x^{9}+x^{8}+x^{7}+x^{6}+x^{3}+x^{2}+x$
[110000000111001]	$x^{14}+x^{13}+x^{5}+x^{4}+x^{3}+1$
[110100101010010]	$x^{14}+x^{13}+x^{11}+x^{8}+x^{6}+x^{4}+x$
[111001100010000]	$x^{14}+x^{13}+x^{12}+x^{9}+x^{8}+x^{4}$

Langkah pertama dari proses *decoding* adalah menghitung sindrom dari setiap *codeword* yang diterima. Sindrom ini berfungsi sebagai indikator untuk mengetahui apakah terdapat kesalahan dalam data. Sindrom dihitung menggunakan operasi modulo terhadap polinomial generator $g_2(x)$, dengan rumus:

$$S_j(x) = r\left(\alpha^j\right) = \sum_{i=0}^{n-1} r_i \cdot \left(\alpha^j\right)^i$$

Jika hasil sindrom $S_i(x) = 0$, maka dapat disimpulkan bahwa tidak ada kesalahan dalam *codeword* yang diterima. Sebaliknya, jika $S_i(x) \neq 0$ maka terdapat kesalahan pada bit-bit *codeword*, dan proses koreksi kesalahan perlu dilakukan untuk mengembalikan data dan bentuk aslinya. Berikut adalah hasil perhitungan sindrom dari setiap blok *codeword* yang diterima:

Tabel 4.19 Hasil Sindrom pada Kode BCH(15,5)

***Codeword* Biner**	S_1	S_2	S_3	S_4	S_5	S_6
[001100100001000]	α^6	α^{12}	α^2	α^9	0	α^4
[101101000110001]	α^5	α^{10}	α^7	α^5	0	α^{14}
[100010001010010]	α^4	α^8	1	α^4	0	1
[101100111110111]	α^3	α^6	α^{11}	α^{12}	0	α^7
[011101111000110]	α^4	α^2	α^7	α^4	0	α^{14}
[101011000111001]	α^{10}	α^5	α^3	α^{10}	0	α^6
[001010101101011]	α^6	α^{12}	α^{13}	α^9	0	α^{11}
[101011001010010]	α^5	α^{10}	α^4	α^5	0	α^2
[011011111001110]	α^4	α^8	α^3	α^1	0	α^6

[110000000111001]	α^3	α^6	0	α^{12}	0	0
[110100101010010]	α^1	α^2	α^3	α^4	0	α^6
[111001100010000]	α^{10}	α^5	α^4	α^{10}	0	α^8

Berdasarkan hasil sindrom yang ditunjukkan pada Tabel 4.19, seluruh *codeword* menunjukkan satu atau lebih sindrom bernilai tak nol. Oleh karena itu, dapat disimpulkan bahwa seluruh *codeword* mengalami kesalahan dalam proses transmisi. Langkah sleanjutnya yang dilakukan adalah koreksi kesalahan. Proses koreksi kesalahan pada kode BCH dilakukan dengan menggunakan algoritma *Berlekamp-Massey*, yang berfungsi untuk menemukan polinomial *error locator* $\Lambda(x)$. Akar-akar dari polinomial ini menunjukkan posisi bit yang mengalami kesalahan dalam *codeword*. Proses ini dimulai dengan inisialisasi nilai awal

$$\Lambda^{(0)}(x) = 1, B(x) = 1, L = 0, m = 1, dan\ b = 1$$

Pada iterasi pertama, *disrepancy* $d_1$diketahui bernilai α^6, sehingga polinomial diupdate menjadi

$$\Lambda^{(1)}(x) = 1 + \alpha^6 x$$

Iterasi kedua menunjukkan *disrepancy* $d_2 = S_2 + \Lambda_1 \cdot S_1 = \alpha^{12} + \alpha^6 \cdot \alpha^6 = \alpha^{12} + \alpha^{12} = 0$ sehingga tidak ada perubahan. Pada iterasi ketiga, *disrepancy* $d_3 = \alpha^0$, sehingga diperoleh hasil akhir polinomial *error locator* yaitu:

$$\Lambda(x) = 1 + \alpha^2 x + \alpha^3 x^2$$

Setelah polinomial *error locator* diperoleh, dilakukan proses *Chien Search* untuk mencari akar dari $\Lambda(x)$ dengan cara mensubstitusikan α^{-i} ke dalam polinomial, untuk $i = 0$ hingga 14. Dari hasil evaluasi diperoleh bahwa $\Lambda(\alpha^{-11}) = 0$, $\Lambda(\alpha^{-12}) = 0$ dan $\Lambda(\alpha^{-13}) = 0$, sehingga dapat disimpulkan bahwa posisi bit yang salah berada pda bit ke-13, ke-12 dan ke-11 dari sebelah

kiri. Langkah selanjutnya adalah melakukan koreksi dengan membalik bit-bit pada posisi tersebut, menghasilkan *codeword* baru yaitu:

Tabel 4.20 *Codeword* pada Kode BCH(15,5)

$\boldsymbol{Codeword\ C''_{ij}(x)}$	$\boldsymbol{Codeword\ P_i(x)}$
[001100100001000]	[010000100001000]
[101101000110001]	[100011000110001]
[100010001010010]	[100101001010010]
[101100111110111]	[101111011110111]
[011101111000110]	[001100011000110]
[101011000111001]	[110011100111001]
[001010101101011]	[010110101101011]
[101011001010010]	[100101001010010]
[011011111001110]	[011100111001110]
[110000000111001]	[110011100111001]
[110100101010010]	[100101001010010]
[111001100010000]	[100001000010000]

Setelah seluruh blok *codeword* yang diterima mengalami proses deteksi dan koreksi kesalahan, diperoleh kembali deretan bit hasil *decoding* yang telah bebas dari gangguan. Bit-bit tersebut merupakan gabungan dari enam blok hasil koreksi dengan panjang 15-bit, sehingga membentuk satu rangkaian bit sepanjang 180-bit sebagai berikut:

[010000100001000100011000110001100101001010010101111011110111001100011000110110011100111001110010101101011010111001010010100100111001110011101100111001110011001010010100101001000010000].

Tahap terakhir dalam proses ini adalah membagi deretan bit tersebut menjadi beberapa blok dengan panjang masing-masing 8 bit. Proses ini bertujuan untuk mengkonversi setiap blok biner menjadi karakter yang sesuai berdasarkan tabel ASCII. Setelah dilakukan pembagian dan konversi, diperoleh karakter karakter sebagai berikut:

[0 1 0 0 0 1 0 0] = “D”

[0 1 1 0 0 1 0 1] = ”e”

[0 1 1 1 0 0 1 1] = “s”

[0 1 1 0 0 1 0 1] = “e”

[0 1 1 1 0 0 1 0] = “r”

[0 1 1 1 0 1 1 0] = “v”

[0 1 1 0 0 1 0 1] = “e”

Dengan demikian, hasil akhir dari proses decoding berhasil mengembalikan pesan asli yang dikirimkan yaitu kata “*Deserve*”.

4.3 Analisis Hasil

Penerapan generator polinomial untuk optimasi deteksi dan koreksi kesalahan bit pada kode BCH menggunakan beberapa parameter yaitu BCH(15,11), BCH(15,7) dan BCH(15,5). Setiap parameter dipilih untuk mewakili tingkat efisiensi dan kemampuan koreksi kesalahan yang berbeda dengan seluruh operasi dilakukan dalam lapangan $GF(2^4)$. Analisis dilakukan pada tiga parameter kode BCH yang berbeda untuk mengamati efisiensi berdasarkan persamaan (2.20) dan tingkat keandalan koreksi kesalahan dibantu dengan bantuan pemrograman *Sagemath*.

Tabel 4.21 Parameter Kode BCH pada $GF(2^4)$

Kode BCH	$\boldsymbol{n}$	$\boldsymbol{k}$	$\boldsymbol{t}$	$\boldsymbol{n-k}$	**Efisiensi%**
BCH(15,11)	15	11	1	4	73,3%
BCH(15,7)	15	7	2	8	46,7%
BCH(15,5)	15	5	3	10	33,3%

Berdasarkan Tabel 4.21 kode BCH(15,11) merupakan parameter dengan efisiensi transmisi paling tinggi di antara ketiga parameter yang telah dianalisis,

yakni sebesar 73,3%. Generator polinomial yang digunakan adalah $g(x) = x^4 + x + 1$ yang dibentuk dari satu polinomial minimal berderajat empat. Kode BCH(15,11) terdiri atas 15 bit panjang total yang mencakup 11 bit data dan 4 bit redundansi, kompleksitas pada sistem ini tergolong rendah. Proses *encoding* dan *decoding* sederhana karena hanya melibatkan satu akar dan satu polinomial minimal. Berdasarkan hasil simulasi, kode BCH(15,11) mampu menghasilkan *codeword* yang valid dan berhasil mengoreksi satu bit kesalahan per blok dengan akurat. Oleh karena itu, parameter inioptimal untuk sistem komunikasi yang membutuhkan efisiensi tinggi dan kecepatan tinggi seperti *real-time* atau komunikasi jarak pendek. Namun, karena hanya mampu mengoreksi satu bit kesalahan, optimasi parameter ini lebih menekankan efisiensi dibandingkan ketahanan terhadap gangguan.

Kode BCH(15,7) memiliki panjang kode 15 bit, terdiri atas 7 bit data dan 8 bit redundansi, kode BCH(15,7) mampu mengoreksi dua kesalahan bit. Efisiensi kode BCH(15,7) sebesar 46,7%, yang menempatkannya sebagai parameter menengah antara efisiensi dan ketahanan. Generator polinomial yang digunakan adalah hasil kali dua polinomial minimal: $g(x) = (x^4 + x + 1)(x^4 + x^3 + 1) = x^8 + x^7 + x^5 + x^4 + x + 1$. Derajat generator adalah delapan, yang menghasilkan delapan bit redundansi. Kompleksitas implementasi meningkat karena sistem harus menghitung empat sindrom dan membentuk polinomial lokasi kesalahan menggunakan algoritma *Berlekamp-Massey*, serta menentukan lokasi bit kesalahan melalui *metode Chien search*.

Berdasarkan hasil simulasi, BCH(15,7) berhasil mengoreksi dua kesalahan bit secara akurat dalam setiap blok. Dengan efisiensi yang masih cukup baik,

parameter ini merupakan pilihan optimal untuk sistem komunikasi yang membutuhkan keseimbangan antara efisiensi dan kemampuan koreksi, seperti pada jaringan industri, komunikasi perangkat bergerak, atau sistem monitoring yang bekerja di lingkungan dengan gangguan sedang. Kode BCH(15,7) memberikan performa seimbang, kecepatan transmisi menurun dibanding BCH(15,11) karena lebih banyak bit redundansi harus dikirim, namun sebagai gantinya daya tahan sistem meningkat terhadap gangguan atau *noise*.

Kode BCH(15,5) dirancang untuk memberikan ketahanan tinggi terhadap kesalahan dengan kemampuan koreksi hingga tiga bit. Kode BCH(15,5) memiliki efisiensi transmisi paling rendah yaitu 33,3% karena hanya 5 bit dari total 15 bit digunakan untuk data, sedangkan 10 bit sisanya merupakan bit redundansi. Generator polinomial dibentuk dari tiga polinomial minimal yaitu $g(x) = (x^4 + x + 1)(x^4 + x^3 + 1)(x^2 + x + 1)$ yang menghasilkan polinomial berderajat sepuluh. Dengan bit redundansi sebanyak 10 bit setiap blok, kompleksitas implementasi menjadi tinggi. Proses koreksi kesalahan memerlukan perhitungan enam sindrom dan pembentukan polinomial lokasi kesalahan yang kompleks, serta evaluasi akar melalui *Chien search*. Meskipun efisiensi rendah, hasil simulasi menunjukkan bahwa parameter ini sangat andal dalam mengoreksi tiga bit kesalahan sekaligus, menjadikannya ideal untuk sistem komunikasi yang beroperasi dalam lingkungan dengan tingkat gangguan tinggi. Kode BCH(15,5) menempatkan prioritas utama pada keandalan dan ketahanan terhadap kesalahan, dengan mengorbankan efisiensi transmisi dan kesederhanaan proses.

Berdasarkan analisis terhadap ketiga parameter kode BCH yaitu BCH(15,11) BCH(15,7), dan BCH(15,5) dapat disimpulkan bahwa penerapan generator

polinomial berpengaruh langsung terhadap efisiensi, kemampuan koreksi kesalahan, dan kompleksitas sistem. Semakin tinggi kemampuan koreksi kesalahan, maka semakin banyak polinomial minimal yang harus digunakan sebagai faktor pembentuk generator polinomial, sehingga menghasilkan generator polinomial $g(x)$ dengan derajat yang lebih tinggi.

Pada parameter BCH(15,11), digunakan satu polinomial minimal $m(x) = x^4 + x + 1$ yang memberikan efisiensi tinggi dan kompleksitas rendah karena hanya mengoreksi satu kesalahan. Pada BCH(15,7), dua polinomial minimal dikalikan untuk membentuk generator $g(x)$ berderajat delapan yang menambahkan kemampuan koreksi menjadi dua bit namun menurunkan efisiensi. Sementara itu, pada BCH(15,5) menggabungkan tiga polinomial menjadi generator berderajat sepuluh, sehingga mampu mengoreksi tiga kesalahan tetapi dengan efisiensi paling rendah dan kompleksitas paling tinggi.

Penjelasan ini juga memperkuat pentingnya penyesuaian nilai k terhadap parameter BCH yang digunakan. Jika nilai k yang digunakan dalam simulasi tidak sesuai dengan parameter yang telah ditentukan, maka akan terjadi ketidaksesuaian struktur blok data yang menyebabkan proses *encoding* dan *decoding* menjadi tidak valid. Selain itu, sistem juga akan kehilangan kemampuan koreksi kesalahan yang seharusnya dimiliki. Nilai k yang terlalu besar akan mengurangi jumlah bit redundansi yang tersedia, sehingga menurunkan kemampuan koresi kesalahan, sedangkan nilai k yang terlalu kecil akan menyebabkan efisiensi transmisi turun drastis tanpa peningkatan signifikan dalam keandalan.

Kompleksitas dan performa sistem kode BCH sangat ditentukan oleh bentuk dan derajat dari generator polinomial yang digunakan. Oleh karena itu, dalam

perancangan sistem komunikasi berbasis kode BCH, pemilihan generator polinomial harus dilakukan secara cermat dengan mempertimbangkan kebutuhan akan efisiensi, tingkat kesalahan pada saluran, dan kapasitas pemrosesan sistem. Generator polinomial yang lebih sederhana dan berderajat rendah lebih sesuai untuk sistem cepat dan ringan, sedangkan generator yang lebih kompleks dibutuhkan pada sistem menuntut ketahanan tinggi terhadap kesalahan.

4.4 Kajian Penelitian dalam Persepektif Islam

Penerapan prinsip optimasi dalam sistem koreksi kesalahan digital dapat dikaitkan secara langsung dengan nilai-nilai ajaran islam sebagaimana tercantum dalam QS. Al-A'raf ayat 31 yang artinya (Kementerian Agama, 2022):

"Wahai anak Adam! Pakailah pakaianmu yang indah di setiap (memasuki) masjid, makan dan minumlah, tetapi jangan berlebih-lebihan. Sungguh, Allah tidak menyukai orang-orang yang berlebihan." (QS. Al-A'raf: 31)

Ayat ini menegaskan larangan untuk melakukan tindakan yang berlebihan atau pemborosan, baik dalam bentuk konsumsi materi maupun dalam penggunaan sumber daya secara umum. Pada teknologi informasi, pemborosan dapat terjadi apabila suatu sistem dibangun dengan struktur atau algoritma yang tidak efisien, yang menyebabkan penggunaan sumber daya komputasi secara berlebihan tanpa memberikan peningkatan performa yang signifikan.

Penerapan prinsip optimasi dalam penelitin ini bertujuan untuk menghasilkan sistem koreksi kesalahan digital yang tidak hanya akurat, tetapi juga efisien dan proporsional. Penggunaan generator polinomial dipilih secara selektif berdasarkan kemampuannya untuk memperbaiki kesalahan secara optimal dengan beban komputasi minimal. Strategi ini mencerminkan nilai ihsan, yakni melakukan pekerjaan dengan sebaik-baiknya, dan tawazun, yaitu menciptakan keseimbangan

antara kinerja sistem pemanfaatan sumber daya. Oleh karena itu, proses optimasi yang dilakukan tidak hanya menjadi kegiatan teknis, melainkan juga bentuk pengalaman nilai-nilai islam dalam ranah pengembangan teknologi.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan hasil analisis dan simulasi yang telah dilakukan, dapat disimpulkan bahwa penerapan generator polinomial berperan penting dalam mengoptimalkan proses deteksi dan koreksi kesalahan pada kode BCH. Generator polinomial dibentuk dari hasil perkalian polinomial-polinomial minimal yang dipilih berdasarkan kemampuan koreksi kesalahan (t) yang diinginkan. Semakin besar nilai t, semakin kompleks bentuk generator polinomial yang digunakan, dan hal ini secara langsung mempengaruhi efisiensi transmisi serta kompleksitas sitem. Melalui pengujian terhadap tiga parameter kode BCH, yaitu BCH(15,11), BCH(15,7), dan BCH(15,5), diperoleh gambaran bahwa konfigurasi BCH(15,11) dengan generator polinomial $g(x) = x^4 + x + 1$ memberikan efisiensi transmisi tertinggi namun hanya mampu mengoreksi satu kesalahan bit. Sebaliknya, BCH(15,5) yang menggunakan generator polinomial berderajat sepuluh mampu mengoreksi hingga tiga kesalahan bit, namun dengan efisiensi paling rendah. Hal ini menunjukkan *trade-off* antara tingkat koreksi kesalahan dan efisiensi yang sangat bergantung pada bentuk dan derajat generator polinomial yang diterapkan. Seluruh proses operasi, mulai dari pembentukan generator polinomial hingga simulasi deteksi dan koreksi kesalahan, dilakukan dalam lapangan $GF(2^4)$ dengan bantuan perangkat lunak *SageMath*. Hasil simulasi menunjukkan bahwa pemilihan generator polinomial yang sesuai tepat berperan penting dalam mengoptimalkan kinerja sistem pengkodean BCH, baik dalam hal efisiensi data maupun kemampuan deteksi dan koreksi kesalahan secara andal.

5.2 Saran

Penelitian saat ini hanya terbatas pada kode BCH dengan menggunakan panjang kode tetap yaitu 15 bit dan dilakukan pada lapangan $GF(2^4)$. Saran untuk penelitian selanjutnya, diharapkan untuk mengeksplorasi terhadap lapangan dengan derajat yang lebih tinggi untuk dapat digunakan pada kode BCH yang mampu mengoreksi lebih banyak kesalahan dengan efisiensi yang masih dapat dikendalikan.

DAFTAR PUSTAKA

Almazrouei, M. (2024). *Advancements in Digital Communication and Error Correction*. IEEE Transactions on Communications.

Andriani, R. (2007). Menentukan polinomial minimal atas GF(p) yang membangun GF(p^n). *Jurnal Matematika UNNES*.

Basu, S. (2014). Error Corecction Codes and Their Applications in Modern Communication Systems. *International Journal of Computer Science & Network Security*, 14(8), 15-22.

Chandoul, A. (2021). Note on Irreducible Polynomials over Finite Field.

Gravano, S. (1990). Decoding the triple-error-correcting (15,5) binary BCH code by the analytic solution of the cubic error-locator polynomial over GF(2^4). *International Journal of Electronics*, 68(1), 107-116.

Grillet, P. A. (2007). *Abstract Algebra*. Springer Science & Business Media.

Handayani, R. (2020). Teori Bilangan. *In Paper Knowledge: Toward a media history of documents*.

Kementeriam Agama Republik Indonesia. (2022). *Al-Qur'an dan Terjemahannya.* Lajnah Penthasihan Mushaf Al-Qur'an, Kementerian Agama RI.

Kurnia, D. (2013). *Optimasi Konversi String Hasil Least Significant Bit Steganography.*

Kurnia, R. (2013). *Pengantar Sistem Digital.* Andi Publisher.

Kyureghyan, G. M. (2020). A recurrent construction of irreducible polynomials of fixed degree over finite fields. *Applicable Algebra in Engineering, Communication and Computing*, 33, 163-171.

Ling, S. & Xing, C. (2004). *Coding Theory*. Cambridge University Press.

Munir, R. (2004). *Matematika Diskrit.* Informatika.

Muslim, M. (2021). Pengaruh gangguan elektromagnetik terhadap transmisi sinyal data dlam sistem komunikasi. *Jurnal Teknik Elektro dan Komputer*.

Nabipour, S., Javidan, J., & Drechsler, R. (2024). Trends and challenges in design of embedded BCH error correction codes in multi-levels NAND flash memory devices. *Memories - Materials, Devices, Circuits and Systems*.

Rai, S. S. (2024). Pseudo-Deterministic Construction of Irreducible Polynomials over Finite Fields.

Raisinghania, M. D. (1980). *Modern Algebra.* S. Chand & Company Ltd.

Sachenko, A. e. (2009). Bit Error Rate and Signal Processing in Digital Transmission Systems. *Journal of Telecommunications and Information Technology*, 1, 23-30.

Sadikin, R. (2012). *Kriptografi untuk Keamanan Jaringan.* Yogyakarta: C. V. ANDI OFFSET.

Sansani, S. (2008). *Penggunaan Aritmatika Modulo dan Balikan Modulo pada Modifikasi Algoritma Knapsack.* Bandung: Jurusan Teknologi Informatika ITB.

Schulz, M. (2024). *On the Recursive Behavior of the Number of Irreducible Polynomials with Certain Properties over Finite Fields*.

LAMPIRAN

Lampiran 1. *Script Code Sagemath:* Proses *Encoding* dan *Decoding*

```
F2 = GF(2)
R.<x> = PolynomialRing(F2)

message = "Deserve"
binary_string = ''.join(format(ord(c), '08b') for c
in message)

params_list = [
    {"n": 15, "k": 11, "t": 1, "g": x^4 + x + 1},
    {"n": 15, "k": 7, "t": 2, "g": (x^4 + x +
1)*(x^4 + x^3 + 1)},
    {"n": 15, "k": 5, "t": 3, "g": (x^4 + x +
1)*(x^4 + x^3 + 1)*(x^2 + x + 1)}
]

def correct_known_error(corrupted_poly,
error_positions):
    correction = sum([x^i for i in
error_positions])
    return corrupted_poly + correction

def compute_syndromes(corrupted_codeword, t):
    syndromes = []
    alpha = x.mod(nth_root=1)  # primitive element
    for i in range(1, 2 * t + 1):
        s = corrupted_codeword(x=F2.fetch_int(2)^i)
# evaluate at α^i
        syndromes.append(s)
    return syndromes

for params in params_list:
    n = params["n"]
    k = params["k"]
    t = params["t"]
    g = params["g"]
    print("\n===== BCH({},{}), t={}
=====".format(n, k, t))
    print("Generator Polynomial g(x):
{}\n".format(g))

    blocks = [binary_string[i:i+k].ljust(k, '0')
for i in range(0, len(binary_string), k)]
    total_bits = len(blocks) * n
    data_bits = len(blocks) * k
```

```
    for idx, b in enumerate(blocks):
        m = sum([x^(k - i - 1) for i in range(k) if
b[i] == '1'])
        shifted = m * x^(n - k)
        _, r = shifted.quo_rem(g)
        c = shifted + r

        print("Block {}:".format(idx + 1))
        print("Message bits:       {}".format(b))
        print("m(x):               {}".format(m))
        print("x^(n-k)*m(x):
{}".format(shifted))
        print("Residu r(x):        {}".format(r))
        print("Codeword C(x):      {}\n".format(c))

        available_error_positions = [n - 1 - i for
i in range(1, 7)]
        rotated_positions =
available_error_positions[idx %
len(available_error_positions):] +
available_error_positions[:idx %
len(available_error_positions)]
        error_positions = rotated_positions[:t]

        error_poly = sum([x^i for i in
error_positions])
        corrupted = c + error_poly

        print("Error introduced at positions (from
MSB=left): {}".format(error_positions))
        print("Corrupted Codeword:
{}\n".format(corrupted))

        S = []
        for i in range(1, 2*t + 1):
            alpha =
GF(2**4).multiplicative_generator()
            # Extend polynomial to proper field if
needed
            P = PolynomialRing(GF(2**4), 'y').gen()
            f = corrupted.change_ring(GF(2**4))
            s_i = f(alpha^i)
            S.append(s_i)
        print("Syndromes S1 to S{}:
{}\n".format(2*t, S))

        corrected = correct_known_error(corrupted,
error_positions)
```

```
        print("Corrected Codeword:
{}".format(corrected))

        if corrected == c:
            print("☑ Correction successful.\n")
        else:
            print("❌ Correction failed.\n")

    efficiency = float(data_bits) / total_bits
    print("Efficiency: {:.4f} ({} / {}
bits)".format(efficiency, data_bits, total_bits))
```

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

RIWAYAT HIDUP

Dewi Susilowati, lahir di Pasuruan pada tanggal 2 April 2003. Penulis merupakan anak pertama dari tiga bersaudara. Anak dari Bapak Hadi Bing Slamet dan Ibu Sunami. Penulis telah menempuh pendidikan mulai dari TK Sholihiyah yang lulus pada tahun 2009, dilanjutkan menempuh pendidikan sekolah dasar di SD Negeri Wonosari dan lulus pada tahun 2015. Kemudian penulis melanjutkan pendidikan sekolah menengan pertama di SMP Negeri 2 Gempol dan lulus pada tahun 2018. Setelah itu, melanjutkan pendidikan sekolah menengah kejuruan di SMK Negeri 1 Gempol dan lulus pada tahun 2021. Pada tahun yang sama, penulis melanjutkan pendidikan di Universitas Islam Negeri Maulana Malik Ibrahim Malang pada Program Studi Matematika, Fakultas Sains dan Teknologi. Selama menempuh pendidikan tinggi, penulis turut berkontribusi aktif dalam berbagai kegiatan baik internal maupun eksternal kampus. Penulis juga aktif dalam kepanitiaan internal maupun eksternal kampus serta mengikuti kegiatan di luar kampus seperti pelatihan dan seminar.

KEMENTERIAN AGAMA RI
UNIVERSITAS ISLAM NEGERI
MAULANA MALIK IBRAHIM MALANG
FAKULTAS SAINS DAN TEKNOLOGI
Jl. Gajayana No.50 Dinoyo Malang Telp. / Fax. (0341)558933

BUKTI KONSULTASI SKRIPSI

Nama : Dewi Susilowati
NIM : 210601110066
Fakultas / Jurusan : Sains dan Teknologi / Matematika
Judul Skripsi : Penerapan Generator Polinomial untuk Optimasi Deteksi dan Koreksi Kesalahan Bit pada Bose-Chaudhuri-Hocquenghem *Code*
Pembimbing I : Prof. Dr. H. Turmudi, M.Si., Ph.D
Pembimbing II : Mohammad Nafie Jauhari, M.Si

No	Tanggal	Hal	Tanda Tangan
1.	19 Desember 2024	Konsultasi Topik dan Data	1.
2.	19 Desember 2024	Konsultasi Bab I, II, dan III	2.
3.	26 Desember 2024	Konsultasi Bab I, II, dan III	3.
4.	14 Januari 2025	Konsultasi Bab I, II, dan III	4.
5.	20 Januari 2025	Konsultasi Bab I, II, dan III	5.
6.	11 Februari 2025	Konsultasi Bab I, II, dan III	6.
7.	24 Februari 2025	ACC Bab I, II, dan III	7.
8.	6 Maret 2025	Konsultasi Kajian Agama Bab I dan II	8.
9.	10 Maret 2025	ACC Kajian Agama Bab I dan II	9.
10.	20 Maret 2025	ACC Seminar Proposal	10.
11.	23 April 2025	Konsultasi Revisi Seminar Proposal	11.
12.	28 April 2025	Konsultasi Bab IV dan V	12.
13.	30 April 2025	Konsultasi Bab IV dan V	13.
14.	8 Mei 2025	Konsultasi Bab IV dan V	14.

KEMENTERIAN AGAMA RI
UNIVERSITAS ISLAM NEGERI
MAULANA MALIK IBRAHIM MALANG
FAKULTAS SAINS DAN TEKNOLOGI
Jl. Gajayana No.50 Dinoyo Malang Telp. / Fax. (0341)558933

15.	9 Mei 2025	ACC Bab IV dan V	15.
16.	14 Mei 2025	Konsultasi Kajian Agama Bab IV	16.
17.	15 Mei 2025	ACC Kajian Agama Bab IV	17.
18.	21 Mei 2025	ACC Seminar Hasil	18.
19.	9 Juni 2025	Konsultasi Revisi Seminar Hasil	19.
20.	10 Juni 2025	ACC Sidang Skripsi	20.
21.	18 Juni 2025	ACC Keseluruhan	21.

Malang, 18 Juni 2025
Mengetahui,
Ketua Program Studi Matematika

Dr. Elly Susanti, M.Sc.
NIP. 19741129 200012 2 005