

BLM230 - Bilgisayar Mimarisi

Proje Dokümanı

Proje Adı: Hamming SEC-DED (Single Error
Correcting, Double Error Detecting) Kodu Simülatörü

MUHAMMED FATİH GÖRAL

22360859032

1. Giriş

Bu dokümantasyon, **BLM230 Bilgisayar Mimarisi** dersi kapsamında geliştirilen **Hamming Code Simülörü** adlı projeye ait olup, **SEC-DED (Single-Error-Correcting, Double-Error-Detecting)** Hamming Kodlaması tekniğini temel alarak veri üzerinde hata tespiti ve düzeltme işlemlerini simüle eden bir **Java tabanlı masaüstü uygulama** içermektedir. Hamming Kodu, özellikle bilgisayar sistemlerinde, veri iletişim kanallarında ve hafıza birimlerinde oluşabilecek tekli veya ikili bit hatalarının güvenilir biçimde yönetilmesini sağlayan klasik ve etkili bir hata kontrol yöntemidir.

Geliştirilen bu simülatör, yalnızca teorik bilgiyi aktarmakla kalmaz; aynı zamanda öğrencilerin interaktif yollarla bu süreci deneyimlemelerine olanak tanır. Uygulama, kullanıcı dostu ve sezgisel bir **grafiksel kullanıcı arayüzü (GUI)** sunar. Kullanıcılar, belirli uzunluklarda (8, 16 veya 32 bit) veri girişi yapabilir, bu veriler üzerinden Hamming kodlaması otomatik olarak oluşturulabilir, istenen bitlere hata enjekte edilerek hataların sistem tarafından nasıl tespit edildiği ve düzeltildiği adım adım görselleştirilebilir. Böylece hem **teorik altyapı hem de uygulamalı deneyim** aynı ortamda sunulmuş olur.

Arayüz, **Java Swing** kütüphanesi kullanılarak geliştirilmiş olup, sade ama estetik bir tasarıma sahiptir. Renk kodlamaları sayesinde parity bitleri, hata konumları ve düzeltilmiş bitler farklı renklerle görsel olarak ayırt edilir. Bu da öğrenme sürecini daha anlaşılır ve etkili hale getirir.

Bu dokümantasyon, projenin **teknik mimarisi, kullanım kılavuzu, arayüz açıklamaları, fonksiyonel detayları ve teslim gereksinimlerini** kapsamlı bir şekilde ele almaktır, projeyi geliştiren öğrenciler ve değerlendiren öğretim üyeleri için rehber niteliğindedir.

2. Proje Açıklaması

2.1. Projenin Amacı

Bu projenin temel hedefi, Hamming Kodunun teorik bilgisini pratik bir uygulamaya dökerek öğrenme sürecini desteklemektir. Kullanıcılar, şu işlemleri gerçekleştirebilir:

- Veriyi Hamming Koduna dönüştürme (encoding),
- Kodlanmış veriye hata enjeksiyonu yapma,
- Tek bit hatalarını düzeltme ve çift bit hataları tespit etme.

Proje, SEC-DED Hamming Kodunun tek bit hata düzeltme ve çift bit hata tespit kapasitesini demo ederek, hata düzeltme mekanizmalarının işleyişini görselleştirir. Bu, özellikle bilgisayar mimarisi dersinde hata kontrol kodlarının önemini anlamak için faydalıdır.

2.2. Projenin Özellikleri

- **Kullanıcı Girdisi:**

Kullanıcı, 8, 16 veya 32 bitlik bir veri uzunluğu seçerek her bit için ayrı kutulara 0 veya 1 girişi yapar. Geçersiz karakter (örneğin 2, a) veya eksik giriş durumunda sistem, kullanıcıyı uygun hata mesajlarıyla uyarır.

- **Hamming Kodu Oluşturma:**

Girilen veri, SEC-DED Hamming kodlamasıyla parity bitleri eklenerek genişletilir. Parity bitleri açık kahverengi arka planla, veri bitleri ise yeşil arka planla kutular içinde gösterilir. Bit sıralaması sağdan sola (LSB sağda) olacak şekilde düzenlenir.

- **Hata Enjeksiyonu:**

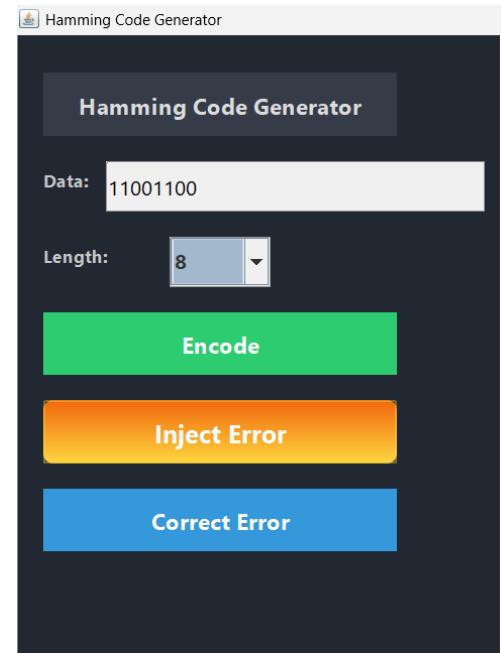
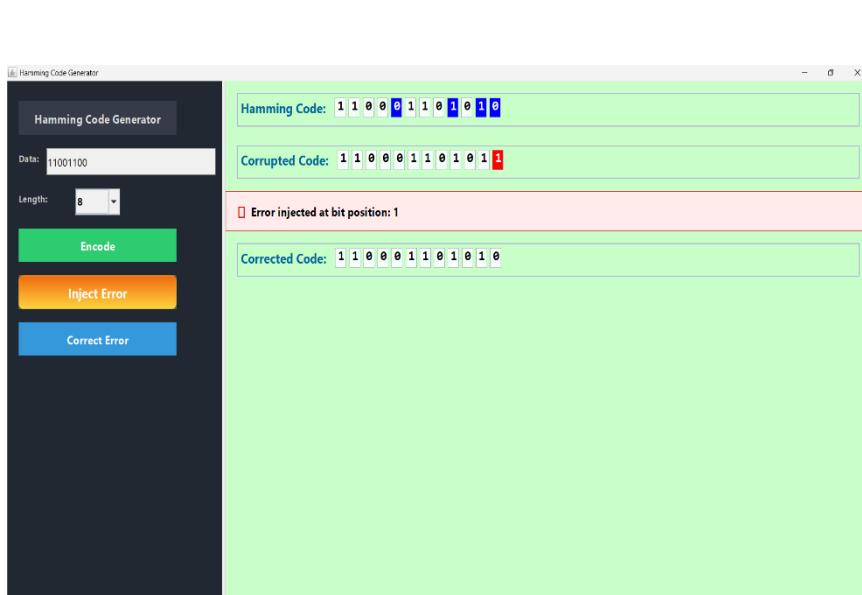
Kullanıcı, en fazla iki bit kutusuna tıklayarak hata enjekte edebilir. Seçilen bitlerin arka planı kırmızıya döner. Aynı bit iki kez seçilirse, hata etkisiz hâle gelir. Hataların etkisi görsel olarak anlık yansıtılır.

- **Hata Düzeltme:**

Kod çözme işlemi sırasında tek bitlik hatalar otomatik olarak tespit edilip düzelttilir. Çift bitlik hatalar algılanır fakat düzeltilemez; bu durumda ekranda açıkça “Double bit error detected: cannot correct” mesajı gösterilir.

- **Görsel Geri Bildirim:**

Tüm işlemler (kodlama, hata ekleme, düzeltme) renklerle desteklenen kutucuk yapısıyla kullanıcıya görsel olarak sunulur. Hangi bitlerin veri, parity, hatalı veya düzeltilmiş olduğunu anlamak kolaylaştırılır. Ayrıca bilgi mesajları alt kısmında net biçimde kullanıcıya gösterilir.



3.1. Çalışma Akışı

1. **Veri Girişi (Data In):** Kullanıcı, bir veri dizisi girer (örneğin, 11001100).
2. **Hamming Kodu Oluşturma (f):** Veri, encodeHamming fonksiyonu ile Hamming Koduna dönüştürülür.
3. **Hata Enjeksiyonu (Memory):** Kullanıcı, belirli bir bit pozisyonunda hata enjekte eder (örneğin, 7. bit).
4. **Hata Tespiti ve Düzeltme (Corrector):** detectError fonksiyonu ile hata pozisyonu belirlenir ve tek bit hataları düzelttilir.
5. **Çıkış (Data Out):** Düzeltilmiş veri kullanıcıya gösterilir.

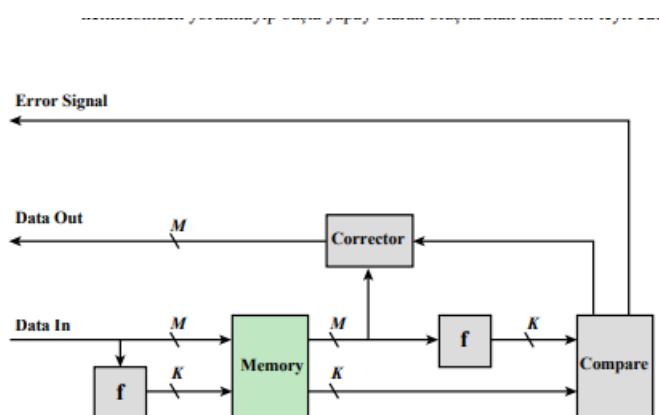


Figure 5.7 Error-Correcting Code Function

4. Kullanım Kılavuzu

4.1. Arayüz Elemanları

Uygulamanın arayüzü, aşağıdaki gibidir.

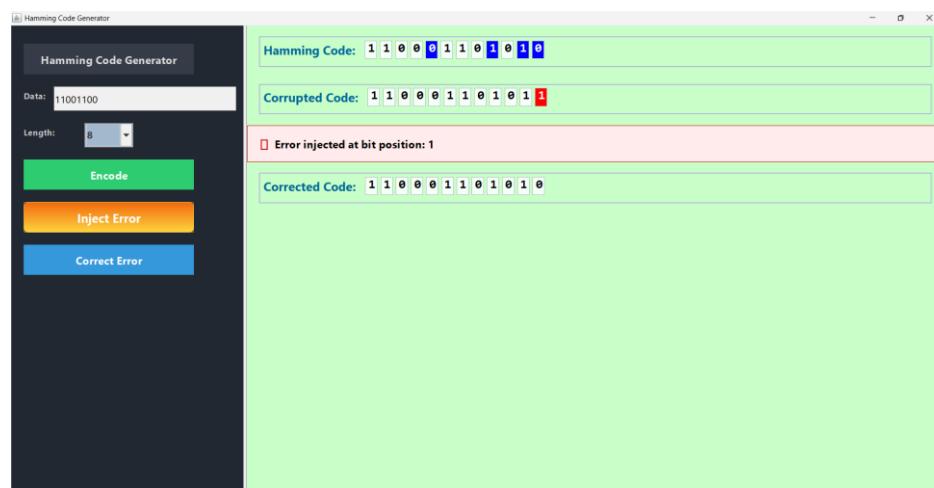
- **Data:** Kullanıcı tarafından girilecek veri dizisi (örneğin, 11001100).
- **Length:** Veri uzunluğu seçimi (8, 16 veya 32 bit).
- **Encode:** Hamming Kodunu oluşturur.
- **Inject Error:** Hata enjeksiyonu için bir pozisyon girer.
- **Correct Error:** Hataları düzeltir veya tespit eder.

4.2. Kullanım Adımları

1. **Veri Girişi:** Data alanına 8, 16 veya 32 bitlik bir veri dizisi girin (örneğin, 11001100).
2. **Hamming Kodu Oluşturma:** Encode butonuna tıklayın. Hamming Kodu, parity bitleri mavi renkte gösterilerek ekranda görüntülenecektir.
3. **Hata Enjeksiyonu:** Inject Error butonuna tıklayın ve bir bit pozisyonu girin (1 ile maksimum bit uzunluğu arasında). Hatalı bit kırmızı renkte vurgulanır.
4. **Hata Düzeltme:** Correct Error butonuna tıklayın. Tek bit hataları düzeltilecek, çift bit hataları ise bir hata mesajıyla bildirilecektir.

4.3. Senaryolar

Senaryo 1: Tek Bit Hatası – Otomatik Düzeltme



Bu senaryoda, kullanıcı 8 bitlik "11001100" verisini girmiş ve Hamming Kodlama (SEC-DED) işlemi gerçekleştirilmişdir. Kodlanan veri "110001101010" olup, mavi renkle işaretlenen bitler parity (eşlik) bitleridir.

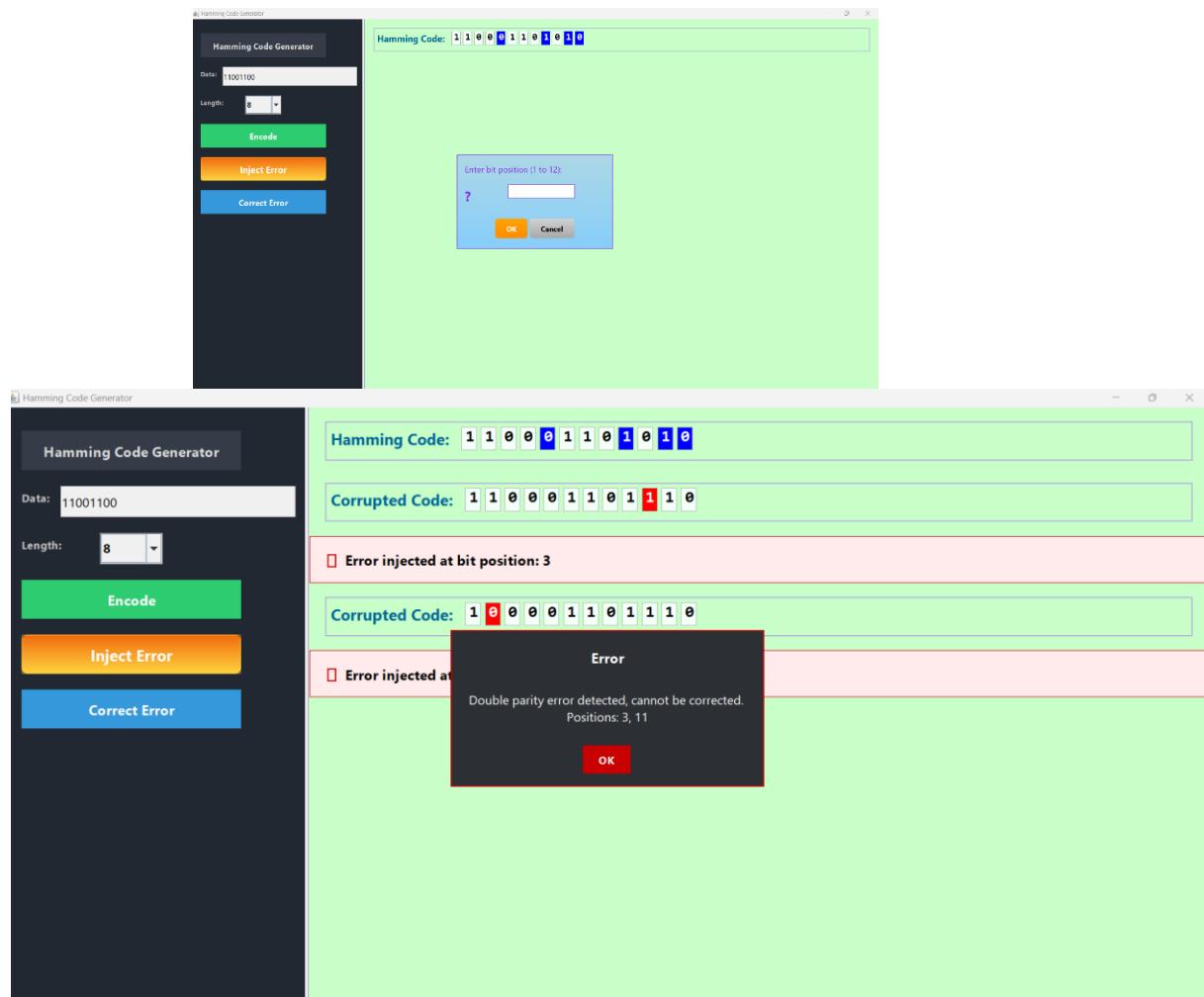
Daha sonra kullanıcı, bir hata pozisyonu seçerek "Inject Error" (Hata Enjekte Et) butonuna basmıştır. En sağdaki (pozisyon 1) bit bozulmuş ve bu bit kırmızı renkle gösterilmiştir. Sistem, bu hatayı başarılı şekilde tespit etmiş ve mesaj olarak:

 Error injected at bit position: 1

şeklinde bilgi vermiştir.

Kullanıcı "Correct Error" butonuna bastığında, sistem hatayı otomatik olarak düzeltmiş ve düzelttilmiş kodu "110001101010" olarak tekrar göstermiştir. Böylece orijinal kod doğru şekilde geri elde edilmiştir.

Senaryo 2: Çift Bit Hatası – Düzeltilemez Durum



Bu senaryoda, kullanıcı yine "11001100" verisini 8 bitlik olarak girmiş ve sistem tarafından "110001101010" Hamming kodu oluşturulmuştur. Parity bitleri mavi renkle gösterilmiştir.

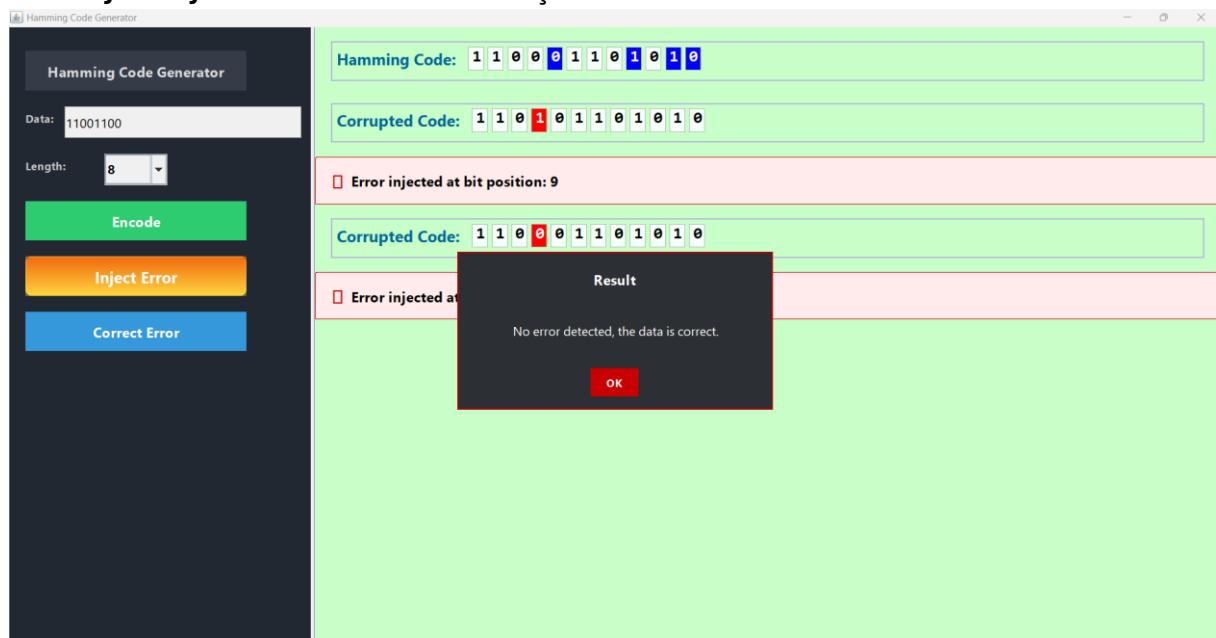
Kullanıcı, iki farklı bit pozisyonunda (3 ve 11) hata enjekte etmiştir. Enjekte edilen bu hatalar sonucunda "100001101110" adlı bozulmuş kod oluşmuştur. Kırmızı ile işaretli bitler hatalı pozisyonları göstermektedir.

Kullanıcı "Correct Error" butonuna bastığında sistem, bu durumda çift hata (double error) olduğunu algılayarak otomatik düzeltme yapamamış ve şu mesajı vermiştir:

! Double parity error detected, cannot be corrected. Positions: 3, 11

SEC-DED (Single-Error-Correcting, Double-Error-Detecting) mantığı gereği, sistem tek hatayı düzeltbilir ancak çift hata durumunda sadece hata olduğunu tespit eder, düzeltme yapamaz.

Senaryo 3: Aynı Bit Tekrar Bozuldu – Geçerli Veri



Bu senaryoda kullanıcı "11001100" verisini girmiştir ve "110001101010" Hamming kodu oluşturulmuştur. Parity bitleri mavi olarak gösterilmiştir.

Kullanıcı ilk olarak **bit pozisyonu 9**'da bir hata enjekte etmiştir. Bu bitin değeri 1'den 0'a dönmüştür ve kırmızıyla işaretlenmiştir.

Daha sonra kullanıcı tekrar aynı pozisyona (9. bit) ikinci kez hata enjekte etmiştir. Bu durumda bitin değeri tekrar 0 → 1 olmuştur, yani eski haline dönmüştür.

Kullanıcı "Correct Error" butonuna bastığında sistem kodu taramış ve herhangi bir hata **algılamamıştır**. Bunun üzerine şu mesaj gösterilmiştir:

✓ No error detected, the data is correct.

Bu senaryo, aynı bitin iki kez değiştirilmesinin aslında **veride hata bırakmadığını**, yani sistemin bunu **doğru veri** olarak algıladığı gösterir. Bu durum, Hamming kodlarının çift hata düzeltmesi yapamama sınırına görsel bir örnektir.

5. Teknik Detaylar

5.1. Kod Yapısı

Senaryo 1: Tek Bit Hatası – Otomatik Düzeltme

```
        }
        int errorPos = detectError(corruptedCode);
        if (errorPos > 0) {
            errorPos = hammingCode.length - errorPos; // Convert to right-based 0-index
            corruptedCode[errorPos] ^= 1; // Correct the error
        }
        drawBitRow( title: "Corrected Code", corruptedCode, highlightIndex: -1 );
        isCorrected = true; // Set correction flag
        errorCount = 0; // Reset error count
        errorPositions[0] = -1;
        errorPositions[1] = -1;
        resultPanel.revalidate();
        resultPanel.repaint();
    });
}
```

⌚ Görsel: Tek bit kırmızı, “Corrected Code” başarılı şekilde gösteriliyor

✳️ Kod Parçası: correctButton action listener → tek hata düzeltme kısmı

Senaryo 2: Çift Bit Hatası – Düzeltilemez Durum

```
130     if (errorCount == 2) {
131         int pos1 = errorPositions[0]; // Kullanıcının girdiği pozisyonu direkt kullan
132         int pos2 = errorPositions[1]; // Kullanıcının girdiği pozisyonu direkt kullan
133         if (pos1 == pos2) {
134             showStyledMessageDialog( title: "Result", message: "No error detected, the data is correct.", JOptionPane.INFORMATION_MESSAGE );
135         } else {
136             showStyledMessageDialog( title: "Error", message: "Double parity error detected, cannot be corrected." );
137         }
138         isCorrected = true; // Set correction flag
139         errorCount = 0; // Reset error count
140         errorPositions[0] = -1;
141         errorPositions[1] = -1;
142         return;
143     }
}
```

⌚ Görsel: 2 kırmızı bit, hata mesajında: "Double parity error detected"

✳️ Kod Parçası: correctButton action listener → çift hata kont

Senaryo 3: Aynı Bit Tekrar Bozuldu – Hata Yok

```
130     if (errorCount == 2) {
131         int pos1 = errorPositions[0]; // Kullanıcının girdiği pozisyonu direkt kullan
132         int pos2 = errorPositions[1]; // Kullanıcının girdiği pozisyonu direkt kullan
133         if (pos1 == pos2) {
134             showStyledMessageDialog( title: "Result", message: "No error detected, the data is correct.", JOptionPane.INFORMATION_MESSAGE );
135         } else {
136             showStyledMessageDialog( title: "Error", message: "Double parity error detected, cannot be corrected. Pos1: " + pos1 + ", Pos2: " + pos2 );
137         }
138         isCorrected = true; // Set correction flag
139         errorCount = 0; // Reset error count
140         errorPositions[0] = -1;
141         errorPositions[1] = -1;
142         return;
143     }
}
```

⌚ Görsel: Aynı bit ikinci kez bozulunca sistem “No error detected” diyor

✳️ Kod Parçası: Aynı bloktaki özel durum kontrolü

Bonus: Hamming Kodu Oluşturma (Parity Bitleriyle Kodlama)

```
private int[] encodeHamming(int[] data) { 1 usage
    int r = 0;
    while ((1 << r) < data.length + r + 1) r++;
    int[] result = new int[data.length + r];
    int j = 0;
    for (int i = 1; i <= result.length; i++) {
        if ((i & (i - 1)) != 0) {
            result[result.length - i] = data[data.length - 1 - j++];
        }
    }
}
```

 **Görsel:** Parity bitler maviyle gösterilmiş “Hamming Code” satırı

 **Kod Parçası:** encodeHamming() metodu

6. Ek Bilgiler

- **Geliştirme Ortamı:** IntelliJ IDEA veya Eclipse gibi bir Java IDE'si kullanılarak geliştirilmiştir.
- **Bağımlılıklar:** Java SE (Standart Edition) gereklidir; ek bir kütüphaneye ihtiyaç duyulmamaktadır.
- **Test Edilen Platformlar:** Windows 10/11 ve macOS üzerinde test edilmiştir.

Bu dokümantasyon, projenin tüm yönlerini kapsayacak şekilde hazırlanmış olup, gerektiğinde güncellenebilir. Daha fazla bilgi için kaynak kodlara veya demo videosuna başvurabilirsiniz.

7. Teslim Bilgileri

Proje, aşağıdaki şekilde teslim edilmiştir:

- **Kaynak Kodlar:** GitHub hesabına yüklenmiştir.
<https://github.com/fatihgoral/HammingCode>
- **Demo Videosu:** YouTube'a yüklenmiştir.
<https://www.youtube.com/watch?v=mcgZDYm0jSY>