BURSA TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ
Bilgisayar Mühendisliği Bölümü

BİLGİSAYAR MİMARİSİ

Hamming Code Projesi

22360859062

Mehmet ÖZAY

1. GİRİŞ:

Bu projede, Hamming SEC-DED (Single Error Correction - Double Error Detection) algoritmasını temel alan bir web uygulaması geliştirilmiştir. Kullanıcılar bu uygulama aracılığıyla, 8-bit veya 32-bitlik ikili verileri girebilir, bu veriler üzerinde Hamming kodlaması gerçekleştirebilir, yapay olarak hata ekleyebilir ve bu hataları algılayarak düzeltebilir. Uygulama hem eğitimsel hem de deneysel amaçlar için faydalı olacak şekilde tasarlanmıştır.

Projenin amacı, Hamming algoritmasının mantığını kavratmak, kullanıcıların bu algoritmayı adım adım deneyimlemesini sağlamak ve dijital sistemlerde hata düzeltme mantığının nasıl çalıştığını göstermek olmuştur.

Projenin Github Linki: https://github.com/ozay-mehmet/Haming-Code

2. PROJE DETAYLARI:

2.1 Projenin Geliştirilme Amacı ve Kapsamı

Projenin temel amacı, Hamming kodlarının temel prensiplerini kullanıcıya interaktif bir şekilde öğretmektir. Bu doğrultuda, hem 8-bit hem 16-bit hem de 32-bitlik veri girişlerini destekleyen, hataların kullanıcı tarafından manuel olarak eklendiği ve otomatik olarak düzeltildiği bir simülatör hazırlanmıştır.

Projede sadece tek bitlik hataların düzeltilmesiyle sınırlı kalınmamış, aynı zamanda çift bitlik hataların tespit edilebilmesi de hedeflenmiştir. Kullanıcıya süreçte neyin nasıl çalıştığını gösterebilmek amacıyla, her bir aşama görsel olarak ekranda yansıtılmıştır.

2.2 Kullanılan Teknolojiler ve Dosya Yapısı

Proje, tamamen istemci tarafında çalışan HTML, CSS ve JavaScript teknolojileriyle geliştirilmiştir. Sunucu tarafı gerektirmediği için herhangi bir kurulum işlemi olmadan sadece bir tarayıcıda çalıştırılabilir.

Dosya yapısı şu şekildedir:

- **index.html:** Uygulamanın kullanıcı arayüzünü sağlar. Giriş alanları, butonlar ve çıktı bölümleri bu dosyada tanımlanmıştır.
- **style.css:** Arayüzün görsel tasarımını belirler.
- **script.js:** Hamming kodlama, hata oluşturma ve düzeltme işlemlerini içerir.

2.3 Arayüz ve Kullanım

Kullanıcı, arayüz üzerinden 8-bit veya 32-bitlik veri girişini yaparak işlemlere başlar. Ardından 'Kodu Üret' butonuna tıklayarak ilgili verinin Hamming kodlaması yapılır. Kodlanmış veri, ekranda kullanıcıya gösterilir. Kullanıcı dilerse belirli bir bit indeksine hata ekleyebilir. Bu hata sonrasında 'Hata Tespit ve Düzelt' butonuna basıldığında sistem hatayı bulur ve düzeltir. Tüm adımlar sonucunda elde edilen veri, kullanıcının kontrolüne sunulur.

2.4 Javascript Fonksiyonları ve Algoritmalar

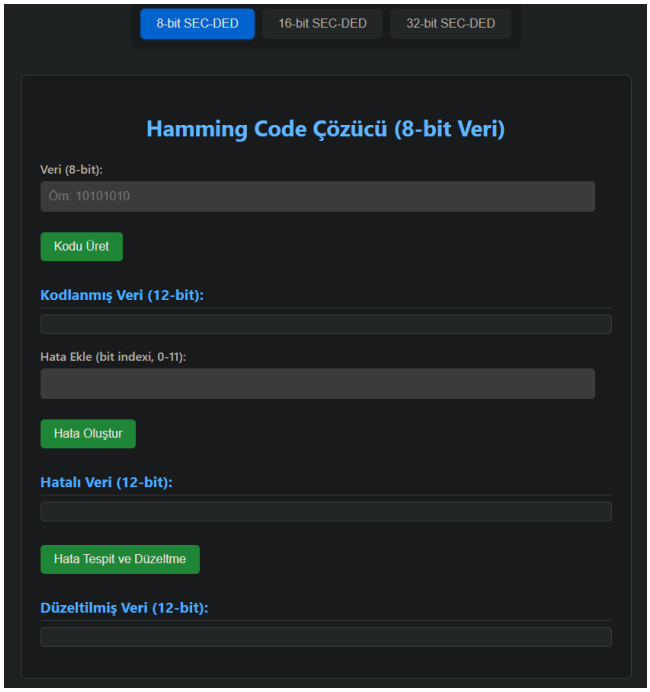
Bu projede, veri iletiminde hata tespiti ve düzeltme işlemlerini gerçekleştirmek amacıyla çeşitli fonksiyonlar geliştirilmiştir. encodeData() fonksiyonu, kullanıcıdan alınan 8-bitlik ham veriyi Hamming (12,8) koduna dönüştürerek 12-bitlik bir çıktı üretir. Bu süreçte, hata tespitini mümkün kılan parity bitleri 1., 2., 4. ve 8. bit konumlarına yerleştirilir. introduceError() fonksiyonu, test ve simülasyon amaçlı olarak kullanıcıdan alınan bit konumuna yapay bir hata ekleyerek veri üzerinde bozulma oluşturur. Ardından detectAndFix() fonksiyonu, bu hatalı 12-bitlik veriyi analiz ederek hata bulunan konumu saptar ve tek bitlik düzeltmeyi otomatik olarak gerçekleştirir.

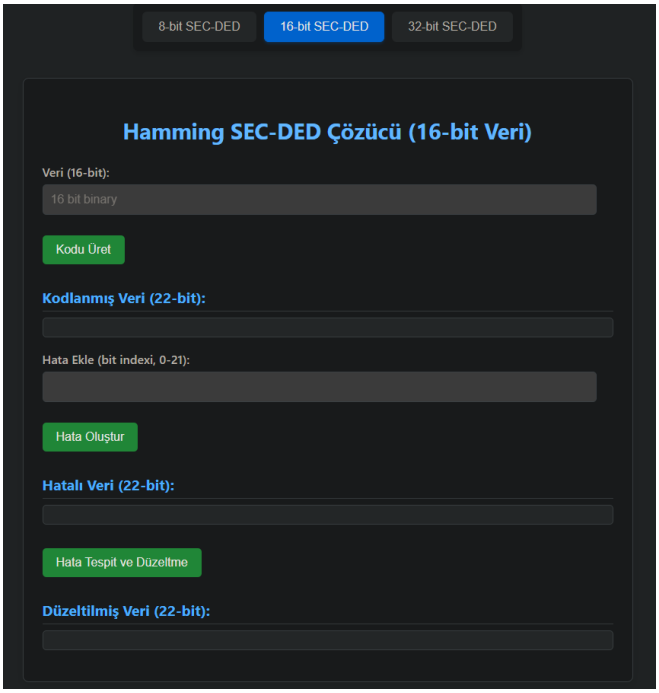
Daha büyük veri boyutları için geliştirilen encodeData32() fonksiyonu, 32-bitlik bir veri girişini alarak SEC-DED (Single Error Correction, Double Error Detection) prensibine dayanan 39-bitlik bir çıktıya dönüştürür. Bu yapı, 32 veri bitine ek olarak 6 parity biti ve 1 genel parity bitinden oluşur; böylece yalnızca tek bitlik hatalar değil, aynı zamanda çift bitlik hatalar da algılanabilir. Bu verinin kontrolü detectAndFix32() fonksiyonu ile sağlanır; fonksiyon, 39-bitlik veri üzerinde bir hata meydana geldiğinde bu hatayı saptayarak doğru konumu belirler ve düzeltme işlemini uygular. Ayrıca genel parity bitini kontrol ederek çift bitlik hataları algılama yeteneğine sahiptir.

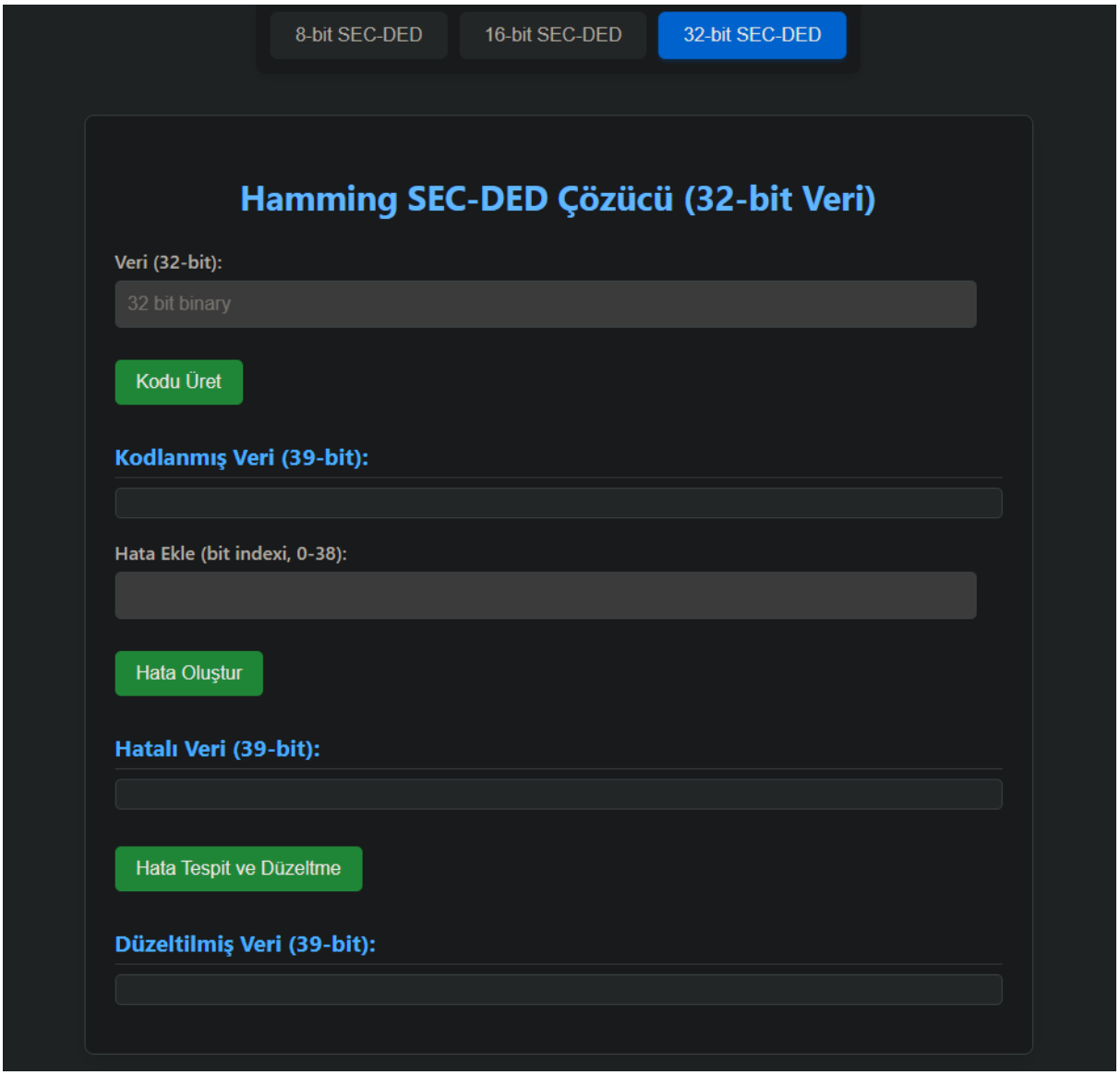
Projenin modüler yapısı sayesinde, kullanıcıdan veri alma, çıktıyı ikili (binary) formatta gösterme, veriyi terminal üzerinde görsel olarak temsil etme gibi yardımcı işlevler de gerçekleştirilmiştir. Ek olarak, kullanıcı arayüzünü kolaylaştırmak amacıyla etkileşimli menü seçenekleri ve kullanıcıdan gelen girişleri doğrulayan yapılar geliştirilmiştir. Bu sayede sistem, hem eğitim amaçlı kullanımlara hem de hata düzeltme algoritmalarının gerçek zamanlı simülasyonlarına uygun hale getirilmiştir.

2.5 Ekran Görüntüleri

2.5.1 - Giriş Ekranları







32 Bitlik Veri

2.5.2 - 8 Bitlik Veri

Projemizde, kullanıcıdan alınan 8 bitlik veri girişi Hamming Code algoritması kullanılarak işlenmektedir. Bu algoritma, veriye hata kontrolü için eklenen parity (eşlik) bitleri sayesinde tek bir bitlik hataların tespit edilip düzeltilmesini mümkün kılar. Veri ile birlikte belirli konumlara yerleştirilen parity bitleri, her biri belirli bir bit grubunu kapsayacak şekilde organize edilir. Bu yapı sayesinde, veri alıcı tarafa ulaştığında parity bitlerinin kontrolü ile veride bir hata olup olmadığı anlaşılır. Eğer veride tek bitlik bir hata varsa, hangi parity bitlerinin beklenen değerden farklı çıktığı analiz edilerek hatanın tam olarak hangi konumda olduğu belirlenir.

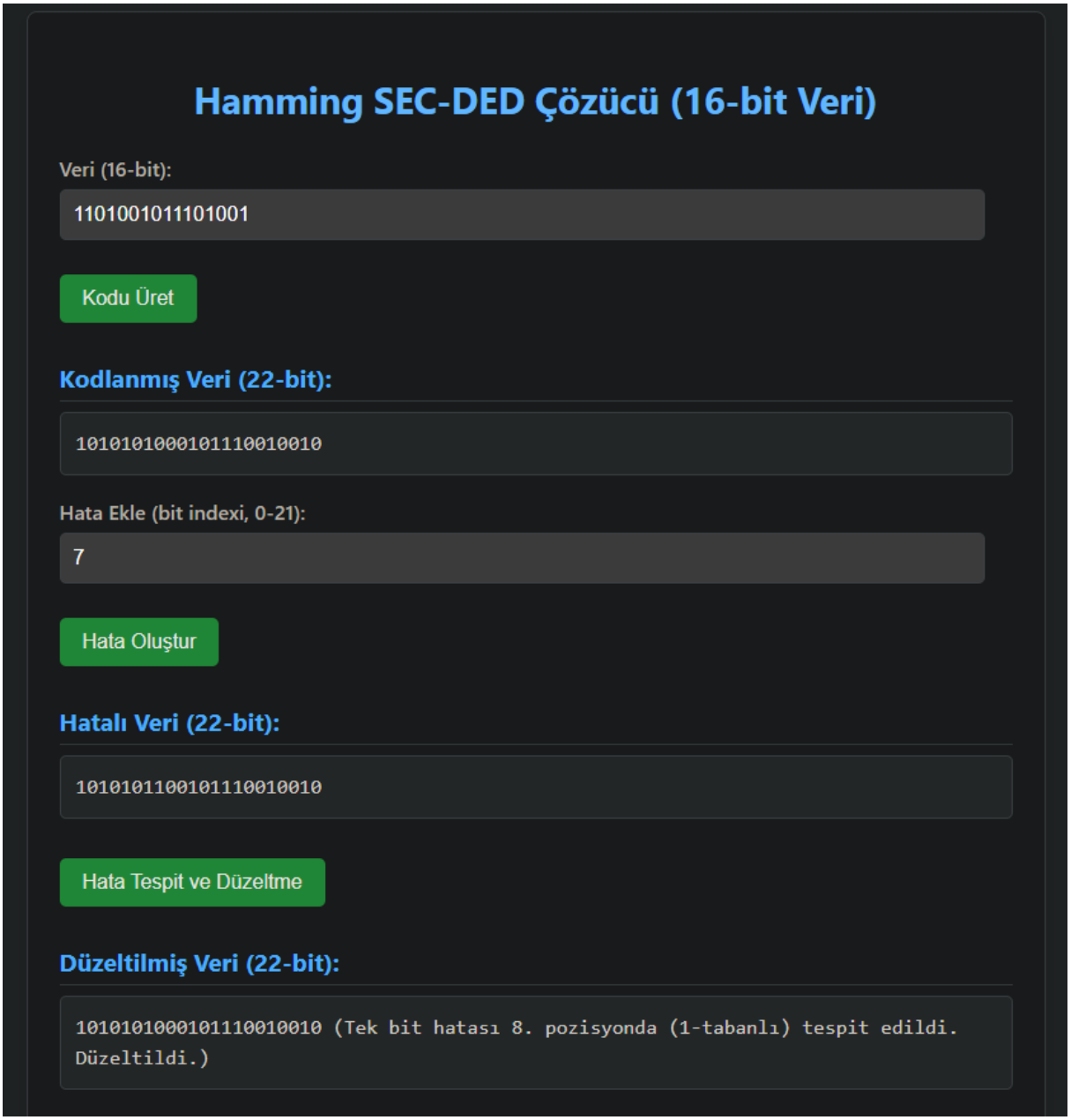
Sistemimiz, hata tespitinin ardından ilgili bitin konumunu otomatik olarak hesaplayarak hatalı biti düzeltir. Bu süreçte, Hamming algoritmasının sağladığı hata lokalizasyon kabiliyeti kullanılır. Tespit edilen hata giderildikten sonra, artık doğruluğu sağlanmış olan düzeltilmiş veri

kullanıcıya sunulur. Bu sayede sistem, veri iletiminde ortaya çıkabilecek tek bitlik hataları hem tespit etme hem de düzeltme yeteneği kazanır.

2.5.3 - 16 Bitlik Veri

Projemizde 16 bitlik veri girişi yapıldığında, Hamming Code algoritması daha geniş bir veri kümesi üzerinde çalışacak şekilde uygulanır. Veri, hata tespiti ve düzeltme işlemleri için gerekli olan parity bitleriyle birlikte yeniden yapılandırılır. Bu durumda, daha fazla veri biti söz konusu olduğundan, doğru parity bitlerini konumlandırmak ve her bir parity bitinin hangi veri bitlerini kontrol edeceğini hesaplamak daha karmaşık bir hale gelir. Ancak algoritmanın temeli değişmez: her parity biti belirli bit gruplarının doğruluğunu kontrol etmekle görevlidir ve hata durumunda hangi parity bitlerinin bozulduğu üzerinden hatalı bitin konumu hesaplanır.

16 bitlik veri iletiminde, özellikle sistemin doğruluk ve güvenilirlik ihtiyacının arttığı durumlarda, bu yöntem oldukça etkilidir. Parity bitlerinden elde edilen hata vektörü yardımıyla, sistem hatalı bitin konumunu tespit eder ve bu bit terslenerek düzeltilmiş veri elde edilir. Genişleyen bit yapısına rağmen, Hamming algoritması hâlâ yalnızca tek bitlik hataları düzeltebilir; çoklu bit hatalarında yalnızca tespit yapılabilir, düzeltme garantilenemez. Bu özellik, 16 bitlik veri işleyen gömülü sistemler, sensör verileri veya düşük bant genişliği gerektiren haberleşme protokollerinde güvenilir bir hata kontrol mekanizması sağlar.

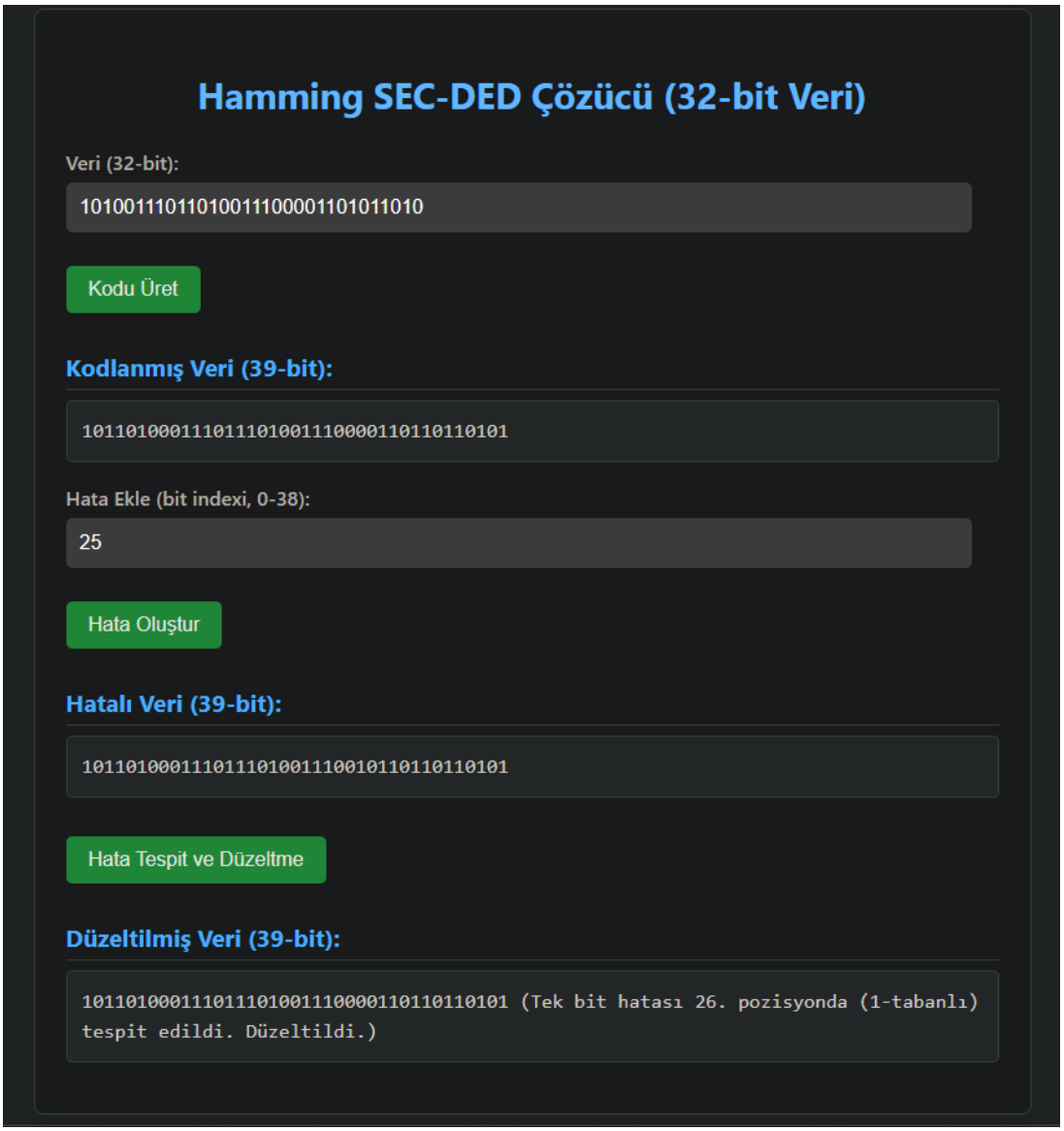


2.5.4 - 32 Bitlik Veri

Projemizin daha büyük veri setleriyle çalıştığı durumlarda, örneğin 32 bitlik veri girişi söz konusu olduğunda, Hamming Code algoritması daha fazla parity bitine ihtiyaç duyar. Çünkü veri uzunluğu arttıkça, parity bitlerinin kapsadığı bit gruplarının sayısı da artar. 32 bitlik veride, hatanın konumunu doğru biçimde tespit edebilmek için en az 6 parity bitine ihtiyaç vardır

(çünkü 26=64>32+62^6 = 64 > 32 + 626=64>32+6). Bu parity bitleri, her biri belirli bir ikili konuma göre veri bitlerini kapsayacak şekilde yerleştirilir. Hamming algoritması, verinin gönderilmesinden önce bu parity bitlerini hesaplar ve iletim sırasında veriye ekler.

Veri hedefe ulaştığında sistem parity bitlerini yeniden kontrol eder ve ortaya çıkan hata sendromu yardımıyla, herhangi bir bitin yanlış iletilip iletilmediği belirlenir. Eğer hata tespit edilirse, algoritma hatalı bitin tam konumunu hesaplayarak düzeltme işlemini otomatik olarak gerçekleştirir.



2.6 Youtube Demo Videosu

https://www.youtube.com/watch?v=72lL3q1FC24

3. KAYNAKÇA:

Stallings, W. (2010). *Computer organization and architecture: Designing for performance* (8th ed.). Upper Saddle River, NJ: Prentice Hall.