1. Başlık Sayfası

- **Proje Adı:** Hamming SEC-DED Simülatörü
- **Ders Adı:** BLM230 - Sayısal Mantık Tasarımı veya ilgili dersin adı
- **Hazırlayan:** [Adınız Soyadınız]
- **Öğrenci Numarası:** [Öğrenci Numaranız]
- **Teslim Tarihi:** [Güncel Tarih - örn: 07 Haziran 2025]

2. Giriş

- **Proje Amacı:** Bu proje, Hamming kodunun (Single-Error Correcting, Double-Error Detecting - SEC-DED) çalışma prensiplerini ve hata tespiti/düzeltmesi yeteneklerini görsel bir arayüz aracılığıyla simüle etmeyi amaçlamaktadır. Kullanıcıların ikilik veri üzerinde Hamming kodlama, belleğe yazma, yapay hata enjekte etme ve bu hataları tespit edip düzeltme süreçlerini deneyimlemesini sağlamak temel hedeftir.
- **Hamming Kodu Hakkında Kısa Bilgi:** Hamming kodu, dijital verilerde oluşan tek bitlik hataları düzeltebilen ve çift bitlik hataları tespit edebilen bir hata düzeltme kodudur. Veri ile birlikte eklenen parite bitleri sayesinde bu yeteneği kazanır. Özellikle bellek sistemleri ve veri iletiminde güvenilirliği artırmak için yaygın olarak kullanılır. SEC-DED özelliği, tek bir bitin yanlış olması durumunda bu biti düzeltebilme, ancak iki bitin yanlış olması durumunda bu durumu fark edebilme kapasitesini ifade eder.

3. Sistem Mimarisi ve Tasarım

- **Genel Mimari:** Geliştirilen simülatör, kullanıcı arayüzü (GUI), bellek yönetimi ve Hamming kodlama/kod çözme algoritmaları olmak üzere üç ana modülden oluşmaktadır. Bu modüller, kullanıcı girişlerini işlemek, veriyi kodlayıp bellekte saklamak, hataları simüle etmek ve düzeltme işlemlerini gerçekleştirmek üzere tasarlanmıştır.
- **Modüllerin Açıklaması:**
  - **`main.py` (Kullanıcı Arayüzü - GUI):**
    - Uygulamanın ana giriş noktasıdır ve `tkinter` ile oluşturulan kullanıcı arayüzünü içerir.
    - Kullanıcıdan ikilik veri, bellek adresi ve hata enjekte edilecek bit pozisyonu gibi girdileri alır.
    - Belleğe yazma, hata enjekte etme ve okuma/düzeltme işlemleri için butonlar sağlar.
    - Tüm işlem çıktılarını ve durum mesajlarını kullanıcıya gösterir.
    - Profesyonel bir dark mode teması ile kullanıcı dostu bir deneyim sunar.
  - **`memory.py` (Bellek Yönetimi):**
    - Bir `Memory` sınıfı tanımlar ve bu sınıf, verilerin adreslere göre saklandığı basit bir bellek simülasyonunu yönetir.
    - `write()` metodu ile verilen ikilik veriyi (Hamming kodu hesaplanmış haliyle) belirtilen adrese kaydeder.
    - `read()` metodu ile belirtilen adresteki kodlanmış veriyi okur.
    - `inject_error()` metodu ile belirli bir adresteki verilere yapay olarak tek bit hatası eklenmesine olanak tanır.

    - **`hamming.py` (Hamming Kodlama ve Kod Çözme Algoritmaları):**
        - `get_parity_bit_count()`: Veri uzunluğuna göre gerekli parite biti sayısını dinamik olarak hesaplar.
        - `calculate_parity_bits()`: 8, 16 veya 32 bitlik verilere dinamik olarak uygun Hamming SEC-DED kodunu hesaplar ve global parite bitini ekler.
        - `inject_error()`: Kodlanmış bir veri üzerinde belirli bir bit pozisyonunda (1-tabanlı) tek bit hatası oluşturur (bit flip).
        - `detect_and_correct()`: Kodlanmış verideki hataları tespit eder, tek bit hatalarını düzeltir ve çift bit hatalarını bildirir. Sendrom kelimesi ve global parite kontrolü ile hatanın tipini ve konumunu belirler.
    - **`utils.py` (Yardımcı Fonksiyonlar):**
        - `is_valid_binary_string()`: Girilen stringin geçerli bir ikilik veri olup olmadığını ve belirli uzunluklara uyup uymadığını kontrol eder.
        - `format_hamming_code()`: Hamming kodunu P (Parite), D (Veri) ve GP (Global Parite) etiketleriyle daha okunabilir bir formatta sunar.
        - `binary_to_hex()` ve `hex_to_binary()`: İkilik ve heksadesimal sayılar arasında dönüşüm yapar (şu anki uygulamada kullanılmasa da potansiyel yardımcı fonksiyonlardır).
- **Şema İle Bağlantı (aa.png):**
    - Ek 1'deki "Error-Correcting Code Function" şeması, simülatörümüzün temel çalışma mantığını yansıtmaktadır.
    - **"f" blokları (Encoder):** Giriş (`Data In`) tarafındaki "f" bloğu, `hamming.py` modülündeki `calculate_parity_bits` fonksiyonuna karşılık gelir. Bu fonksiyon, veri bitlerini alarak Hamming kodunu (veri + parite + global parite) üretir.
    - **"Memory" bloğu:** Uygulamamızdaki `memory.py` modülünde bulunan `Memory` sınıfını temsil eder. Kodlanmış verilerin depolandığı yerdir.
    - **"Compare" bloğu:** Çıkış tarafındaki "Compare" bloğu, `hamming.py` modülündeki `detect_and_correct` fonksiyonunun ilk aşamasını simgeler. Okunan verideki parite bitlerinin kontrol edilerek sendromun hesaplandığı ve hatanın tespit edildiği kısımdır.
    - **"Corrector" bloğu:** "Compare" bloğundan gelen hata bilgisi (sendrom) temelinde, `hamming.py` modülündeki `detect_and_correct` fonksiyonunun düzeltme aşamasını temsil eder. Eğer tek bit hatası tespit edilirse, bu blok hatayı düzelterek doğru veriyi (`Data Out`) sağlar.
    - **"Error Signal":** `detect_and_correct` fonksiyonunun döndürdüğü `status` bilgisine karşılık gelir; hatanın varlığını ve tipini gösterir.

4. Uygulamanın Çalışması ve Fonksiyonları

- **Uygulamanın Başlatılması ve Genel Görünüm:**
    - `main.py` dosyasının çalıştırılmasıyla uygulama açılır.
    - [Uygulamanızın açılış ekran görüntüsü buraya gelecek. Dark mode temasını gösteren bir ekran görüntüsü.]
    - Arayüz, giriş alanları, işlem butonları ve çıktı/durum bilgileri için ayrılmış alanlardan oluşmaktadır.
- **Veri Belleğe Yazma (`write_to_memory`):**
    - "Veri" alanına 8, 16 veya 32 bitlik ikilik bir veri girilir.

    - "Adres" alanına heksadesimal formatta bir bellek adresi girilir.
    - "Belleğe Yaz" butonuna tıklandığında, girilen veri Hamming koduna dönüştürülür ve belirtilen adrese saklanır.
    - Çıktı alanında, yazılan verinin orijinal uzunluğu, oluşan Hamming kodunun uzunluğu ve formatlanmış Hamming kodu gösterilir.
    - [Belleğe yazma işlemi sonrası ekran görüntüsü buraya gelecek. Özellikle "Belleğe yazıldı" çıktısını ve formatlı kodu gösterin.]
- **Yapay Hata Enjekte Etme (`inject_error_gui`):**
    - Öncelikle bellekte bir veri olması gerekmektedir.
    - "Adres" alanına hata enjekte edilecek verinin bulunduğu adres girilir.
    - "Hata Enjekte Edilecek Bit Pozisyonu" alanına 1-tabanlı olarak hatanın oluşacağı bitin konumu girilir.
    - "Hata Enjekte Et" butonuna tıklandığında, belirtilen adresteki kodlanmış verinin ilgili biti ters çevrilir (0 ise 1, 1 ise 0 yapılır).
    - Çıktı alanında, hatanın enjekte edildiği pozisyon ve bozulmuş kodun formatlanmış hali gösterilir.
    - [Hata enjekte etme işlemi sonrası ekran görüntüsü buraya gelecek. Enjekte edilen hatanın olduğu kısmı vurgulayın.]
- **Veriyi Okuma ve Düzeltme (`read_and_correct`):**
    - "Adres" alanına okunacak verinin bulunduğu adres girilir.
    - "Oku ve Düzelt" butonuna tıklandığında, bellekten veri okunur ve `detect_and_correct` fonksiyonu çalıştırılır.
    - Çıktı alanında, okunan orijinal kod, hatanın durumu (hata yok, tek bit hatası, çift bit hatası) ve varsa düzeltilmiş kod gösterilir.
    - **Senaryolar:**
        - **Hata Yok:** Okunan kodda hata bulunmadığında, durum "No Error" olarak belirtilir ve düzeltilmiş kod (aynı kod) gösterilir.
        - **Tek Bit Hatası:** Tek bir bitin hatalı olduğu durumda, sendrom hesaplanır, hata pozisyonu belirlenir, bit düzeltilir ve durum "Single Bit Error at position X" şeklinde bildirilir. Düzeltilmiş kod gösterilir.
        - **Çift Bit Hatası:** İki bitin hatalı olduğu durumda, durum "Double Bit Error Detected" olarak bildirilir. Bu tür hatalar Hamming SEC-DED tarafından düzeltilemez, ancak tespit edilir.
    - [Farklı senaryolar (hata yok, tek bit hatası, çift bit hatası) için ayrı ayrı ekran görüntüleri ekleyin. Sendrom bilgisini ve düzeltme durumunu gösterin.]
- **Kullanıcı Dostu Arayüz ve Dark Mode Tasarımı:**
    - Uygulama, `tkinter.ttk` modülü kullanılarak modern ve estetik bir dark mode teması ile tasarlanmıştır.
    - Koyu arka plan renkleri (`#282c34`, `#333842`), açık yazı renkleri (`#abb2bf`) ve vurgu renkleri (`#61afef`, `#c678dd`) kullanılarak göz yormayan, profesyonel bir görünüm elde edilmiştir.
    - Butonların üzerine gelindiğinde renk değiştirmesi gibi etkileşimli öğeler kullanıcı deneyimini artırmaktadır.
    - Hata veya başarı durumlarında durum çubuğunun renginin değişmesi, anında geri bildirim sağlar.

5. Sonuç

- Geliştirilen Hamming SEC-DED Simülatörü, ikilik veriler üzerinde hata düzeltme kodlarının temel prensiplerini etkin bir şekilde görselleştirmektedir. Kullanıcılar, 8, 16 ve 32 bitlik verilerle deneyler yaparak Hamming kodunun veri bütünlüğünü nasıl sağladığını somut bir şekilde anlayabilirler. Tek bit hatalarının otomatik olarak düzeltilmesi ve çift bit hatalarının tespiti yetenekleri başarıyla simüle edilmiştir.
- **Gelecekteki Geliştirmeler:**
    - Daha büyük veri blokları için destek (örneğin, 64 bit).
    - Birden fazla bellek adresi için görsel bir bellek haritası veya listesi gösterimi.
    - Daha fazla hata türü simülasyonu (örneğin, rasgele hata oluşturma).
    - Performans analizleri ve raporlama özellikleri eklenmesi.

6. Kaynaklar ve Linkler

- **GitHub Proje Deposu:**
    - [GitHub repository URL'nizi buraya yapıştırın. Örn: `https://github.com/KullaniciAdiniz/Hamming-SEC-DED-Simulator`]
- **Simülatör Demo Videosu:**
    - [YouTube (veya başka bir platform) demo video URL'nizi buraya yapıştırın. Örn: `https://www.youtube.com/watch?v=xxxxxxxxxxx`]