BLM230 Bilgisayar Mimarisi
Proje Ödevi

Yunus Alim
AVŞAR

Tarih: 08.06.2025

Ders: BLM230

NO: 23360859710

YÖNERGELER

- 8, 16 ve 32 bitlik veriler üzerinde Hamming SEC-DED Code fonksiyonu uygulanarak veriler bellekte saklanabilecek.

- Belleğe yazılacak herhangi veri için Hamming code'un ne olacağını hesaplayıp kullanıcıya yansıtacak.

- Bellekten okunan verilerde herhangi bit üzerinde (yapay olarak) hata oluşturmaya izin verecek.

- Yapay olarak oluşturulan bu hatayı (karşılaştırma sonucunda ortaya çıkan) sendrom kelimesinden yorumlayıp başta yapay olarak oluşturulan hatalı biti teyit edecek.

- Görsel öğeler kullanarak olabildiğince kullanıcı dostu bir simülasyon arayüzü tasarlamanız istenmektedir.

- İstediğiniz ve hakim olduğunuz bir görsel programlama dilini kullanabilirsiniz.

Proje Tanıtımı:

Bu proje, Hamming SEC-DED (Single Error Correction, Double Error Detection) yöntemini kullanarak:

- Binary verilerin hamming koda dönüştürülmesi,
- Bellekte saklanması,
- Yapay hatalarla test edilmesi,
- Tek bitlik hataların düzeltilmesi,
- Çift bitlik hataların tespit edilmesi,

işlemlerini gerçekleştiren bir simülatör yazılımıdır.

Proje Python dili ile yazılmış olup ekstra bir kütüphane içermemektedir. Pythonın kendi kütüphanelerinden tkinter, gui oluşturma amaçlı, random ise rastgele veri oluşturma ve rastgele hata oluşturma amaçlı kullanılmıştır.

Proje temel olarak iki bölümden oluşmuş olup bunlar;

1. hamming_core.py
2. hamming_gui.py

Core kısmı projedeki hesapları ve arka plandaki işleyişin kodlarını saklarken

Gui kısmı projedeki kullanıcı arayüzü ekrandaki butonların işleyişi gibi interaktif şeylerin kodlarını barındırmaktadır.

İÇİNDEKİLER:

Bu dokümanda, projenin sadece kullanımı değil teknik kısımlarına yani kod bloklarına da değinilmiştir.

Proje github'da https://github.com/YunusAlim/Hamming_SEC-DED_Sim adresinde detaylıca anlatılmış ve yayınlanmıştır. Bunun yanında https://www.youtube.com/watch?v=-gV-d8igrlE&ab_channel=YunusAlimAvsar adresinde video ile tanıtımı yapılmıştır.

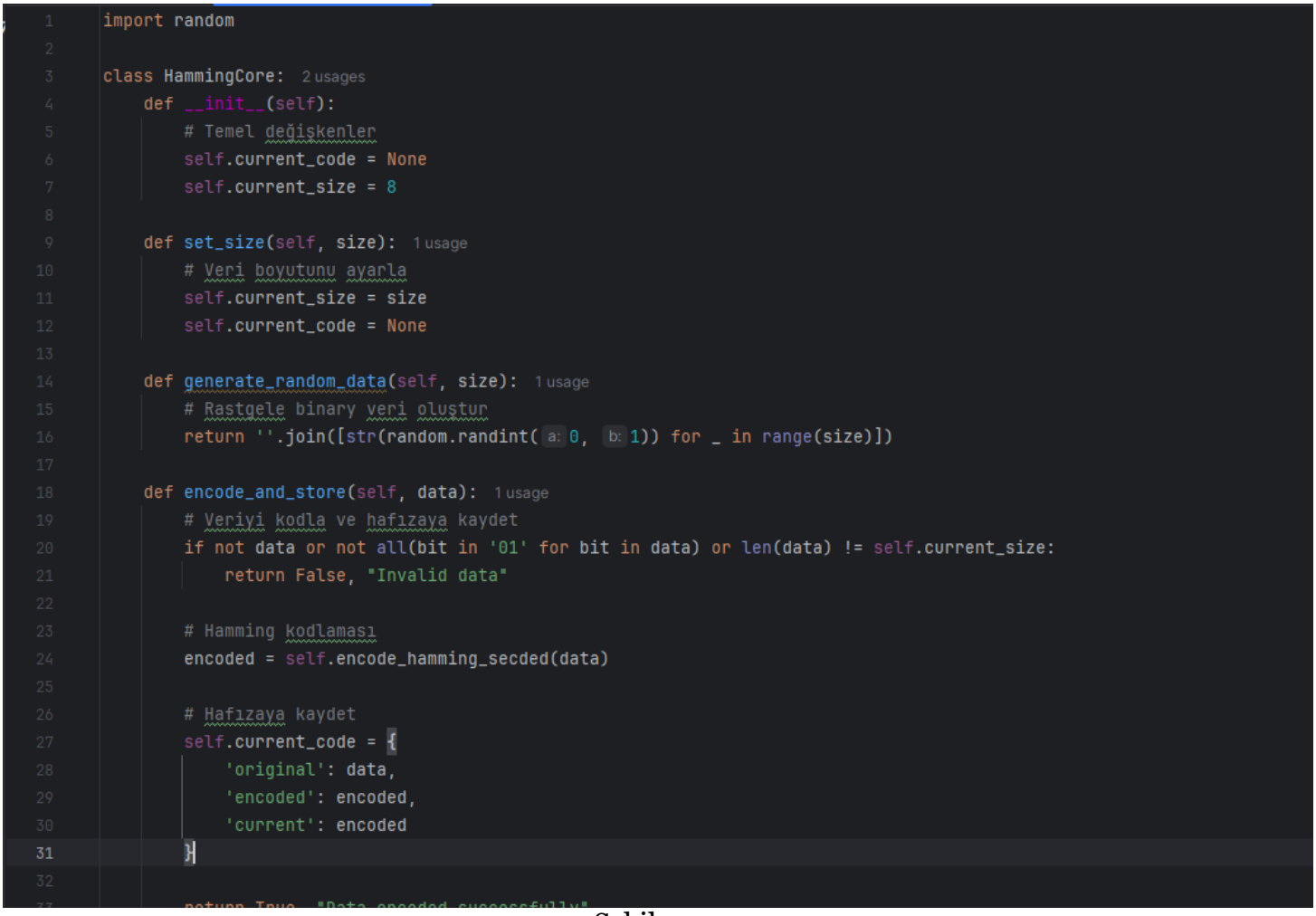
1. Dosya bağlantısı

Kodun Proje Klasör Yapısı:

```
├── hamming_core.py
├── hamming_gui.py
└── README.md
```

1.1. hamming_core.py:

Kodun bu kısmı projenin çekirdeği yani asıl işlemlerin yapıldığı kısımdır. Kodu, adım adım inceleyelim.



```python
import random

class HammingCore:  2 usages
    def __init__(self):
        # Temel değişkenler
        self.current_code = None
        self.current_size = 8

    def set_size(self, size):  1 usage
        # Veri boyutunu ayarla
        self.current_size = size
        self.current_code = None

    def generate_random_data(self, size):  1 usage
        # Rastgele binary veri oluştur
        return ''.join([str(random.randint( a: 0,  b: 1)) for _ in range(size)])

    def encode_and_store(self, data):  1 usage
        # Veriyi kodla ve hafızaya kaydet
        if not data or not all(bit in '01' for bit in data) or len(data) != self.current_size:
            return False, "Invalid data"

        # Hamming kodlaması
        encoded = self.encode_hamming_secded(data)

        # Hafızaya kaydet
        self.current_code = {
            'original': data,
            'encoded': encoded,
            'current': encoded
        }
```

Şekil 1.1.1

Şekil 1.1.1' de gui çalıştırıldıktan sonra açılan ekrandaki ilk işlemlerin yapılmasını sağlayan verinin boyutunu seçmemizi sağlayan kod bloğu yer almaktadır. Tüm kodu fotoğraflar ile açıklamayacağım çünkü kodun içindeki yorum satırları sayesinde tüm kod blokları sınıflar ve fonksiyonların ne işe yaradığı açıklanmıştır.

```python

    def introduce_single_error(self):  1 usage
        # Tek bit hata
        if not self.current_code:
            return False, "No data to modify"

        current_data = list(self.current_code['current'])
        error_pos = random.randint( a: 0, len(current_data) - 1)
        current_data[error_pos] = '1' if current_data[error_pos] == '0' else '0'
        self.current_code['current'] = ''.join(current_data)

        return True, f"Single error introduced at position {error_pos}"

    def introduce_double_error(self):  1 usage
        # Çift bit hata
        if not self.current_code:
            return False, "No data to modify"

        current_data = list(self.current_code['current'])
        error_positions = random.sample(range(len(current_data)),  k: 2)

        for pos in error_positions:
            current_data[pos] = '1' if current_data[pos] == '0' else '0'

        self.current_code['current'] = ''.join(current_data)

        return True, f"Double errors introduced at positions {error_positions}"

    def detect_and_correct(self):  1 usage
        # Hataları tespit et ve düzelt
        if not self.current_code:
            return False, "No data to analyze"

```

Şekil 1.1.2

Şekil 1.1.2’ de ise tek ve çift bitlik hata oluşturma fonksiyonları gösterilmiştir.

```python
    def detect_and_correct(self):  1 usage
        # Hataları tespit et ve düzelt
        if not self.current_code:
            return False, "No data to analyze"

        # Kodanaliz
        decoded, status = self.decode_hamming_secded(self.current_code['current'])

        # Sendrom
        syndrome = 0
        received = self.current_code['current']
        for i in range(len(received)):
            if self.is_power_of_2(i) and i != 0:
                parity = 0
                for j in range(i, len(received), 2 * i):
                    for k in range(j, min(j + i, len(received))):
                        parity ^= int(received[k])
                if parity != 0:
                    syndrome += i

        result = {
            'status': status,
            'syndrome': syndrome,
            'current_data': self.current_code['current'],
            'original_data': self.current_code['encoded']
        }

        if "Single error detected" in status:
            error_pos = syndrome
            corrected = self.correct_single_error(self.current_code['current'], error_pos)
            corrected_decoded, _ = self.decode_hamming_secded(corrected)

            result.update({
                'error_position': error_pos,
                'corrected_data': corrected,
                'corrected_decoded': corrected_decoded
            })
```

Şekil 1.1.3

Şekil 1.1.3’ te Kodun belkide en önemli kısmı olan hatalı biti bulup düzeltme fonksiyonu belirtilmiştir.

```python
def decode_hamming_secded(self, received):  3 usages
    #  SEC-DED kodunu çöz
    if not received:
        return None, "No data to decode"

    total_bits = len(received)
    syndrome = 0

    # Sendrom
    for i in range(total_bits):
        if self.is_power_of_2(i) and i != 0:
            parity = 0
            for j in range(i, total_bits, 2 * i):
                for k in range(j, min(j + i, total_bits)):
                    parity ^= int(received[k])
            if parity != 0:
                syndrome += i

    # Genel parity hesapla
    overall_parity = 0
    for bit in received:
        overall_parity ^= int(bit)

    # Hata durumunu belirle
    if syndrome == 0 and overall_parity == 0:
        return self.extract_data_bits(received), "No error detected"
    elif syndrome != 0 and overall_parity != 0:
        return self.extract_data_bits(received), f"Single error detected at position {syndrome}"
    elif syndrome != 0 and overall_parity == 0:
        return None, "Double error detected (uncorrectable)"
    else:  # syndrome == 0 and overall_parity != 0    ai hata alma sebebi genelparity bitinin kontrolu
        return None, "Error in overall parity bit"
```

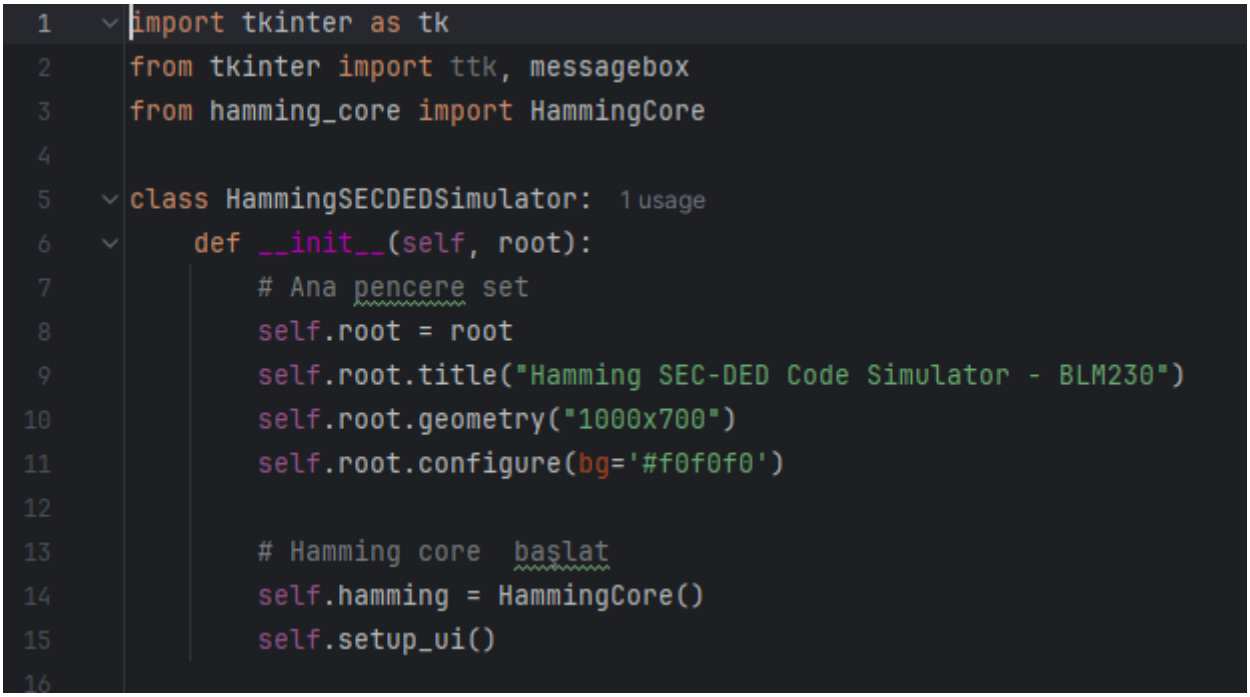
Şekil 1.1.4

Şekil 1.1.4'te şekil 1.1.3' ün olmazsa olmazı parity bitleri ile sendrom hesabı verilmiştir.

Core kısmı ile önemli bazı kod bloklarının ekran görüntüleri ve açıklamaları bunlardır daha detaylı bir açıklama için kodun içindeki yorum satırlarına bakabilirsiniz.

1.2. hamming_gui.py

Kodun bu kısmı UI için yapılan grafiksel iyileştirmeler düzenlemeler ve tabiki ekran komutlarını içeren kısımdır.



```python
import tkinter as tk
from tkinter import ttk, messagebox
from hamming_core import HammingCore

class HammingSECDEDSimulator:  1 usage
    def __init__(self, root):
        # Ana pencere set
        self.root = root
        self.root.title("Hamming SEC-DED Code Simulator - BLM230")
        self.root.geometry("1000x700")
        self.root.configure(bg='#f0f0f0')

        # Hamming core  başlat
        self.hamming = HammingCore()
        self.setup_ui()

```

Şekil 1.2.1

Şekil 1.2.1'de kullanıcı arayüzü ekranının temel kodu ve şekillendirilmesi bunun yanında ise gui için en önemli şey olan tkinter kütüphanesinin dosyaya eklenmesini görüyoruz.

```python
    def detect_and_correct(self):  1 usage
        #  tespit et ve düzelt
        success, result = self.hamming.detect_and_correct()
        if not success:
            messagebox.showwarning( title: "Warning", result)
            return

        # Sonuçları göster
        display_result = "🔍 ERROR DETECTION & CORRECTION\n"
        display_result += "=" * 40 + "\n\n"
        display_result += f"Current Data: {result['current_data']}\n"
        display_result += f"Status: {result['status']}\n\n"
        display_result += f"Syndrome: {result['syndrome']} (binary: {bin(result['syndrome'])[2:]})\n\n"

        if "Single error detected" in result['status']:
            display_result += f"Error Position: {result['error_position']}\n"
            display_result += f"Corrected Data: {result['corrected_data']}\n"
            display_result += f"Corrected Decoded: {result['corrected_decoded']}\n"

            if messagebox.askyesno( title: "Apply Correction",  message: "Apply the correction?"):
                success, message = self.hamming.apply_correction()
                if success:
                    self.update_display()
                    self.update_status( status: "Error corrected successfully",  color: 'green')
                    display_result += "\n✅ Correction applied successfully"
                else:
                    display_result += f"\n❌ {message}"
            else:
                display_result += "\n⏸️ Correction not applied (user choice)"
        elif "Double error detected" in result['status']:
            display_result += "❌ Double error detected - cannot be corrected\n"
            display_result += "💡 Suggestion: Request data retransmission"
            self.update_status( status: "Double error detected - uncorrectable",  color: 'red')
        else:
            display_result += "✅ No errors detected"
            self.update_status( status: "No errors detected",  color: 'green')

        self.results_text.delete( index1: '1.0', tk.END)
        self.results_text.insert( index: '1.0', display_result)
```

Şekil 1.2.2

Şekil 1.2.2’ de düzeltme işlemi sonrasındaki ekrana yazılması gereken mesajlar ve sonuçları gösteren kodlar gözükmektedir.

```python
def main():  1 usage
    # başlat
    root = tk.Tk()
    app = HammingSECDEDSimulator(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

Şekil 1.2.3

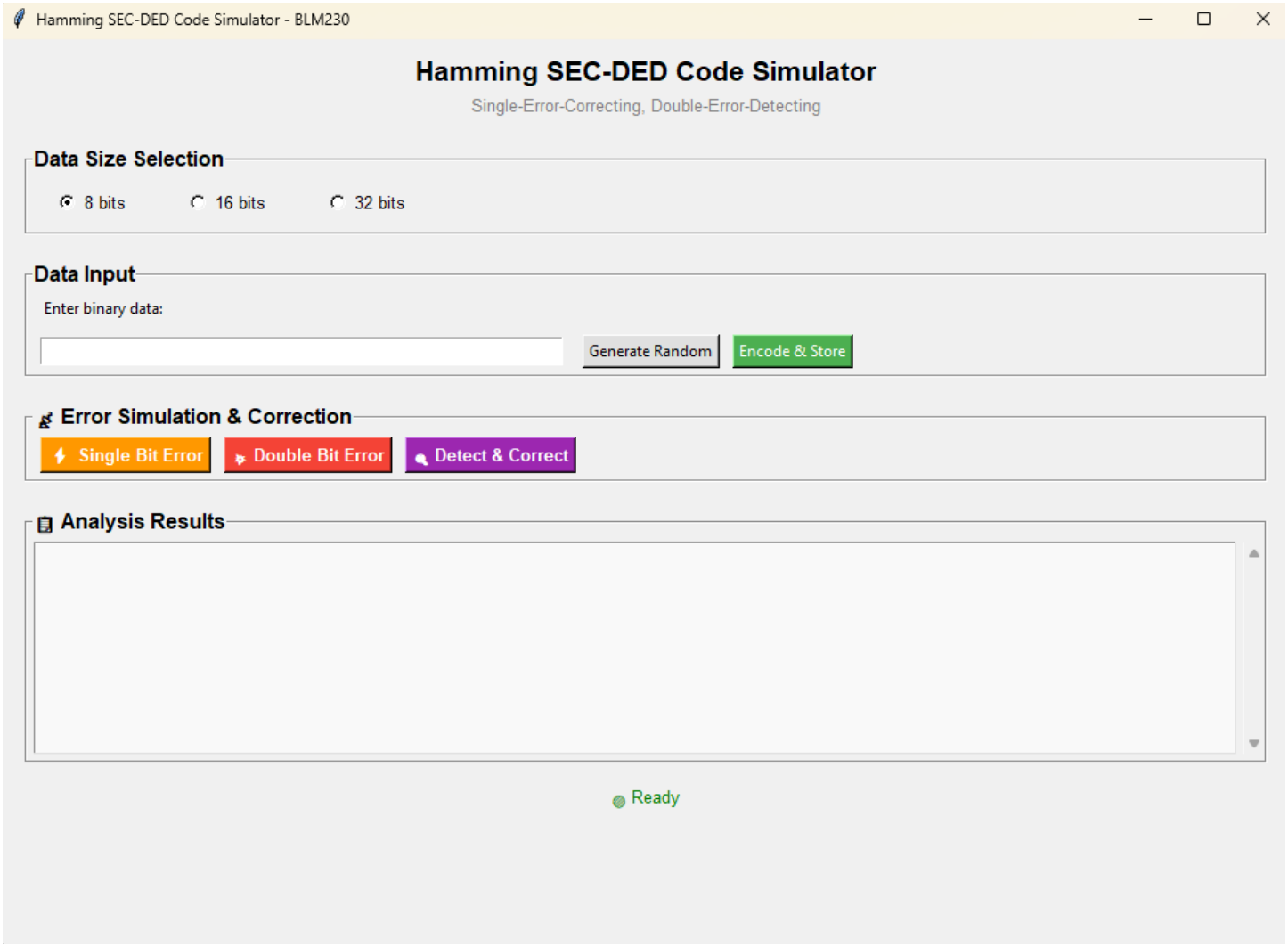
Şekil 1.2.3’ de ise kodun sürekli çalışmasını sağlayan temel kod verilmiştir.

1.3. README.md

Kodun tanıtımını, dosyalamayı, nasıl çalıştırılacağı gibi bazı önemli bilgileri içeren bir not dosyası.

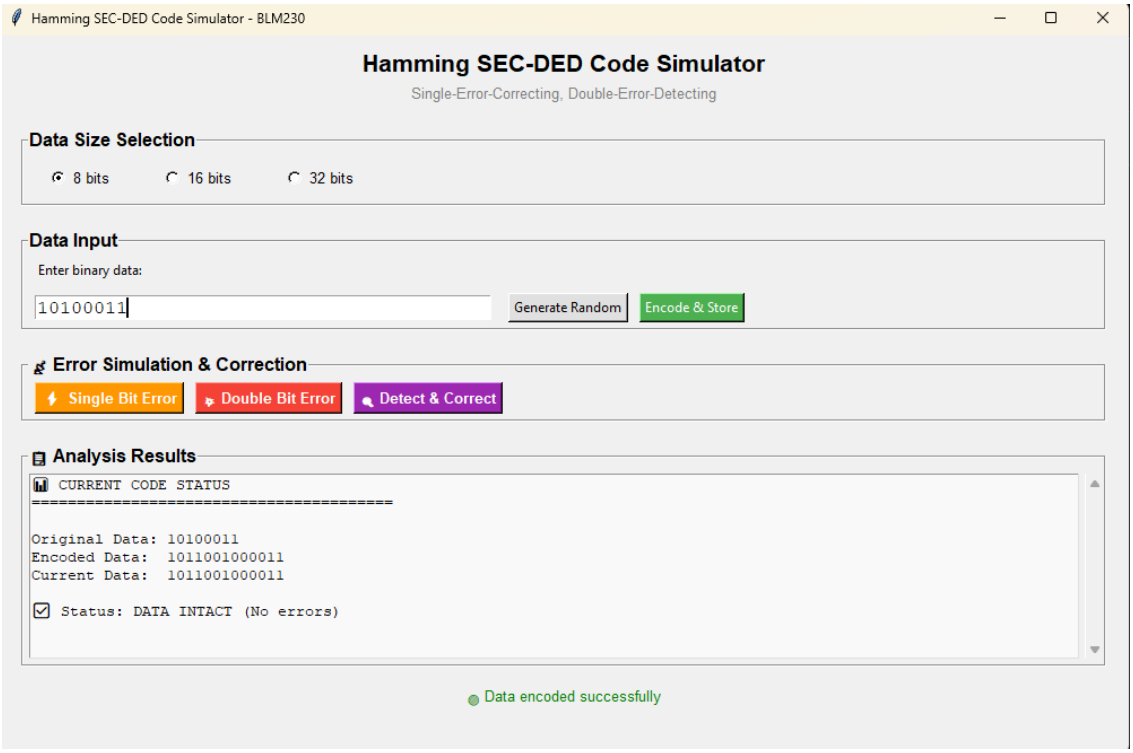
2. Kullanıcı arayüzü

Kod çalıştırıldıktan sonra gelen ekranı ve bu ekrandaki önemli bazı butonların ne işe yaradığını inceleyelim.



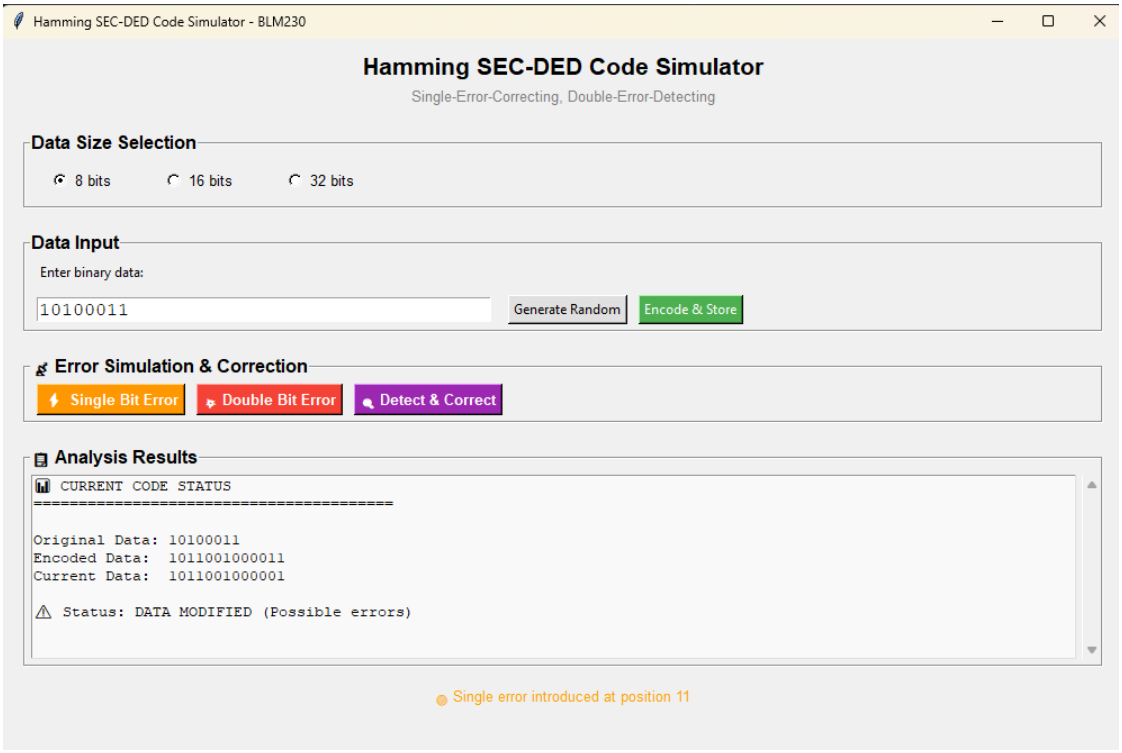
Şekil 2.1.1

Şekil 2.1.1' de uygulama çalıştırıldıktan sonra gelen ilk ekran verilmiştir. Bu ekranda görüldüğü üzere 8, 16 veya 32 bitlik bir veri seçimi bulunmaktadır. Değişiklik yapılmazsa 8 bit seçili gelmektedir.
Generate Random tuşu ile seçilen bit sayısına göre rastgele olarak bir veri oluşturulur Encode tuşuna basılırsa ise bu veri hamming koda dönüştürülür ve hafızaya yazılır.



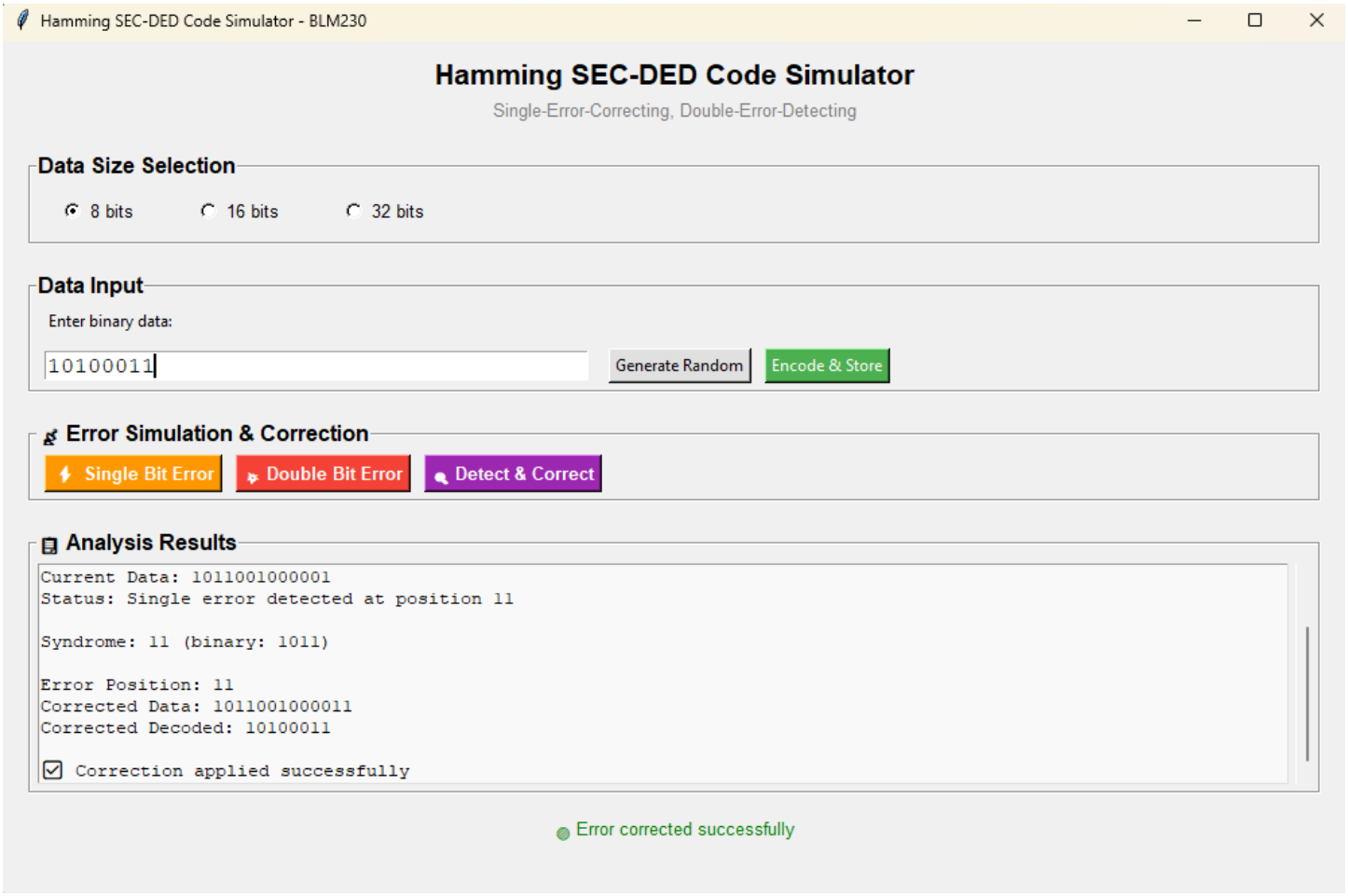
Şekil 2.1.2

Şekil 2.1.2' de girilen kodun encode edildiği yani hamming koda dönüştürüldüğü ekranı görmekteyiz.



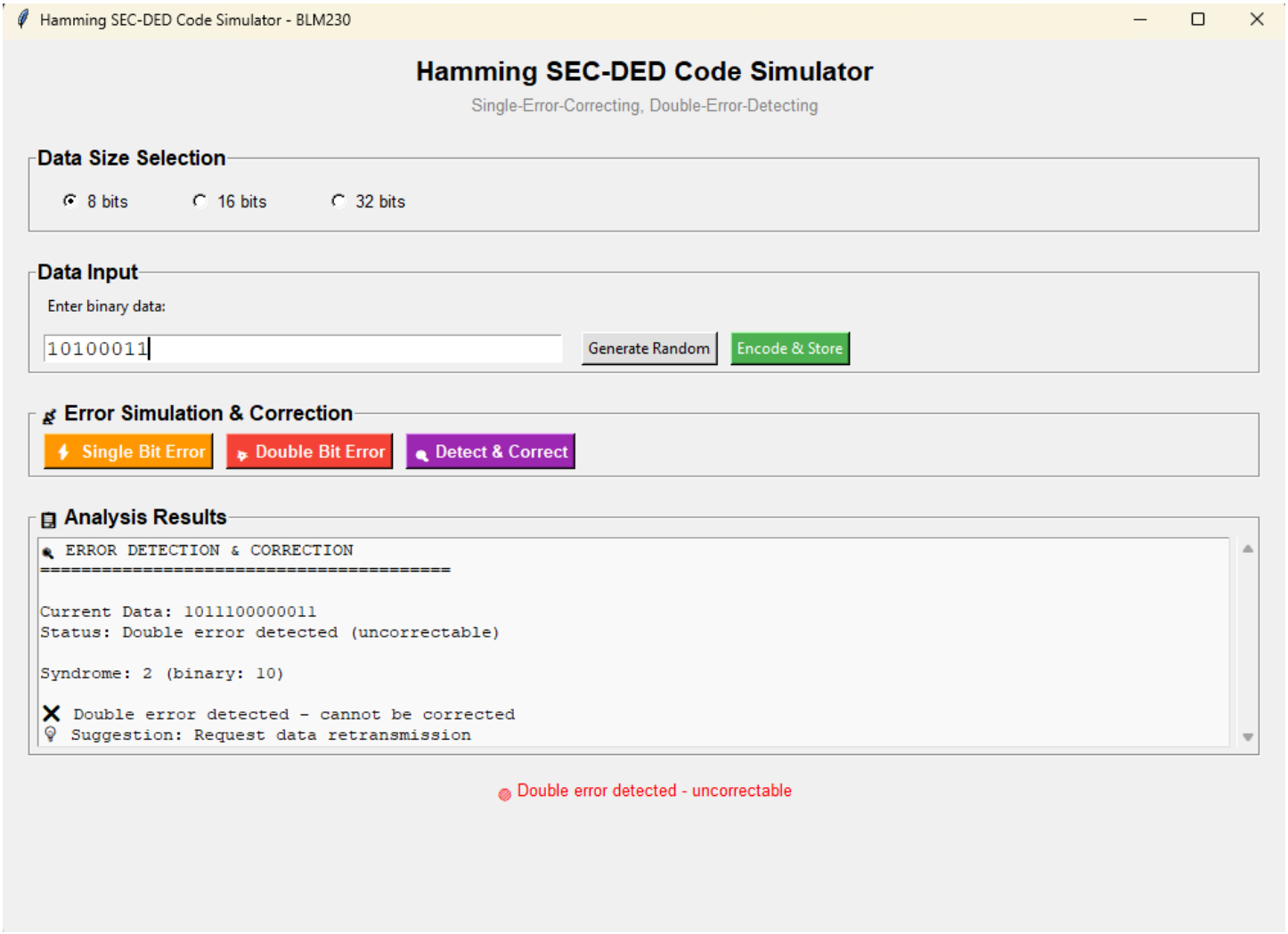
Şekil 2.1.3

Şekil 2.1.3' te aynı verinin bir bitlik hata ile bozulduğu durumdaki ekranı görmekteyiz.



Şekil 2.1.4

Şekil 2.1.4'te bir biti bozulan verinin hangi bitinde hatanın olduğunun tespit edilmesi ve o bitin doğru bit ile değiştirilmesini en sonunda ise tekrardan girdiğimiz veriye dönüştürülmesini yani decode edilme işlemini görmekteyiz.



Şekil 2.1.5

Şekil 2.1.5' te aynı verinin 2 bitinin bozulduğunu ve bunu düzeltmek istediğimizde sadece hatanın tespitinin yapıldığı ama düzeltme işleminin gerçekleştirilemediğini görmekteyiz.