T.C
KOCAELİ SAĞLIK VE TEKNOLOJİ ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR/YAZILIM MÜHENDİSLİĞİ

PROJE KONUSU: SAYISAL TASARIM PROJESİ

ÖĞRENCİ ADI: Aybüke Türidi Elif Beyza BEYAZ
ÖĞRENCİ NUMARASI: 220502005 220502033
Github Linkleri
Aybüke Türidi : https://github.com/220502005
Elif Beyza Beyaz : https://github.com/beyzabeyaz0

DERS SORUMLUSU:
PROF. DR./DR. ÖĞR. ÜYESİ Nevcihan Duru
TARİH: 02.06.2024

1 GİRİŞ

1.1 Projenin amacı

Bu projenin amacı, kullanıcıların basit mantık devrelerini tasarlayıp simüle edebilecekleri bir platform geliştirmektir. Bu platform, temel mantık kapıları kullanılarak devreler oluşturmayı ve çıktıları gözlemlemeyi sağlar. Kullanıcı dostu arayüzü sayesinde sürükle-bırak yöntemiyle devre elemanları kolayca bağlanabilir. Platform, tasarlanan devreleri anında simüle ederek geri bildirim sağlamakta ve hataların tespit edilip düzeltilmesini kolaylaştırmaktadır.

- Basit ve sezgisel bir kullanıcı arayüzü sunarak, kullanıcıların basit mantık devrelerini kolayca oluşturabilmelerini sağlamak.
- Kullanıcıların oluşturdukları devreleri gerçek zamanlı olarak simüle edebilmesi ve giriş değerlerini değiştirerek anında çıkan sonuçları görebilmesi.
- Kullanıcıların giriş ve çıkışları etkileşimli olarak değiştirebileceği ve gözlemleyebileceği bir platform sunmak, böylece teorik bilgileri pratik uygulamalarla pekiştirmek.

2 GEREKSİNİM ANALİZİ

2.1 Arayüz gereksinimleri

- Kullanıcıların kolayca anlayıp kullanabileceği, sade ve anlaşılır bir arayüz tasarımı.
- Temel mantık kapıları ve diğer devre elemanlarının kolayca erişilebileceği bir araç takımı.
- Kullanıcının devreyi kontrol edebilmesi için bir kontrol paneli.
- Devre elemanları arasındaki bağlantıları kolayca çizebilmek için kullanıcı dostu bir bağlantı aracı.
- Kullanıcıların devre tasarımını tamamladıktan sonra devreyi anında çalıştırabilmesi ve sonuçları gerçek zamanlı olarak gözlemleyebilmesi.

2.2 Fonksiyonel gereksinimler

- Kullanıcı, arayüze basit mantık kapılarını sürükleyip bırakarak ekleyebilmeli ve istediği konuma yerleştirebilmelidir.

- Kullanıcı, giriş ve çıkış elemanları ekleyerek devrelerin girişlerini ve çıkışlarını belirleyebilmelidir.

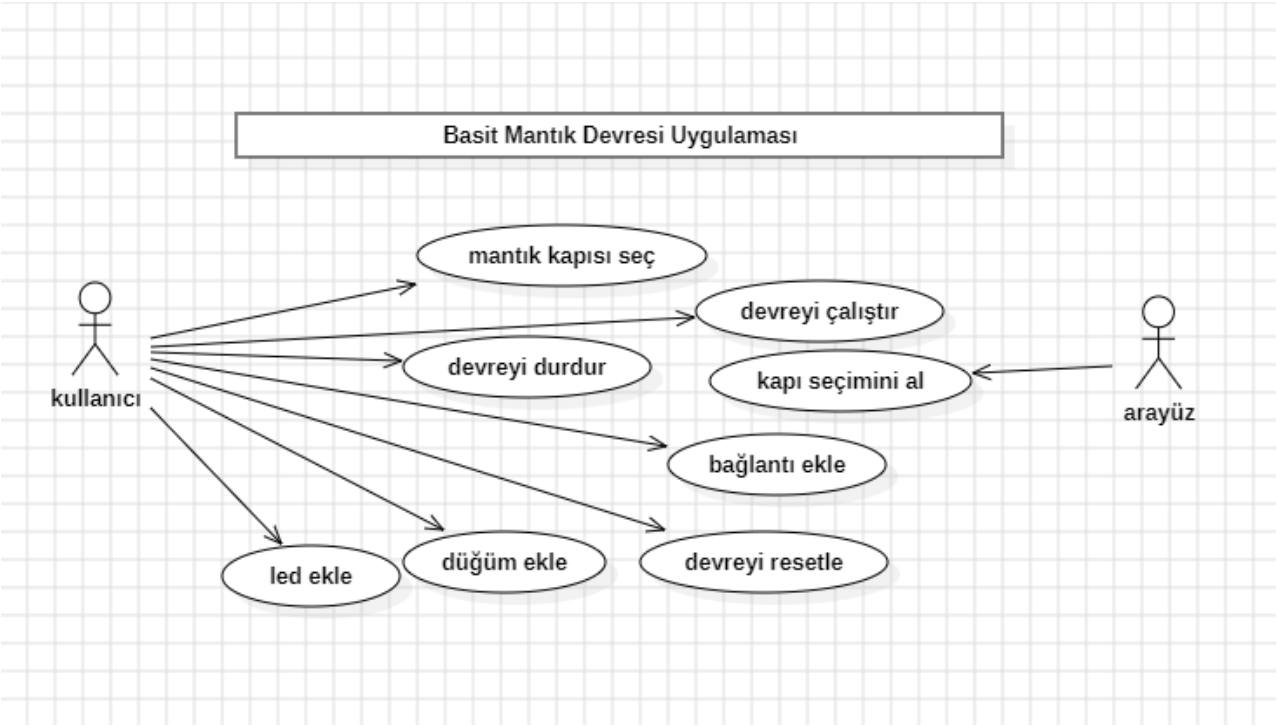
- Kullanıcı, eklenen devreyi çalıştırabilmeli, durdurabilmeli ve sıfırlayabilmelidir. Bu işlem, devrenin işlevini gerçek zamanlı olarak gözlemlemeyi sağlamalıdır.

- Kullanıcı, eklenen kapıları ve bağlantı elemanlarını istediği zaman kaldırabilmeli ve canvas alanı üzerinde devreyi düzenleyebilmelidir.

- Kullanıcı, canvas üzerinde oluşturulan devreyi kaydedebilmeli, yükleyebilmeli ve genişletebilmelidir. Ayrıca, devrenin gerçek zamanlı olarak çalışmasını gözlemleyebilmelidir.

3 TASARIM
3.1 Mimari tasarım

USE CASE DİYAGRAMI:



3.2 Kullanılacak teknolojiler

Bu proje, Python programlama dili ve Tkinter kütüphanesi kullanılarak geliştirilen basit bir mantık kapısı simülatörüdür. Tkinter, Python ‘un standart grafik kullanıcı arayüzü (GUI) kütüphanesi olup, sezgisel bir arayüz ve çeşitli GUI bileşenleri sunar.

Bu projede, Tkinter'ın Canvas bileşeni grafiksel çizimler ve kullanıcı etkileşimi için, PhotoImage bileşeni mantık kapılarının ve bağlantı elemanlarının görsellerini yüklemek için kullanılmıştır.
Ayrıca, Radiobutton ile mantık kapısı seçimi yapılabilir ve Button bileşenleriyle mantık kapıları eklenebilir. Bağlantı düğümleri ve kablolar eklenerek devreler oluşturulabilir, giriş ve çıkış kutuları ile devrelerin çalışması gözlemlenebilir.
Kontrol düğmeleri sayesinde devrelerin çalıştırılması, durdurulması ve sıfırlanması mümkündür.

Diğer Tkinter Bileşenleri ve Fonksiyonları
Canvas: Bu bileşen, grafik çizimler ve kullanıcı etkileşimleri için kullanılan bir tuval sağlar. Mantık kapıları ve bağlantılar gibi grafik öğeleri çizmek için kullanılır.
PhotoImage: Resim dosyalarını yüklemek ve görüntülemek için kullanılır.

3.3 Kullanıcı Arayüzü Tasarımı

“.LogicGateSimulator “Sınıfı, bir Tkinter penceresi içinde mantık kapılarını simüle etmek için kullanılır. LogicGateSimulator sınıfı, kullanıcı ara yüzünü oluşturur ve mantık kapılarını eklemek, taşımak ve bağlamak gibi işlevleri yönetir.

Program, her mantık kapısı için bir görsel kullanır. Görseller, “.png” formatında ve farklı mantık kapılarına karşılık gelen dosyalardır. Örneğin, "and_gate. png" bir AND kapısının görselini temsil eder.

Kullanıcıya mantık kapılarını seçme ve eklemeyi sağlayan bir dizi arayüz elemanı vardır. Bunlar radyo düğmeleri (AND, OR, NOT, vb.), sürükleyip bırakmayı sağlayan bir “canvas” (tuval) ve bağlantı düğmeleri eklemek için düğmeler içerir.

Bağlantı düğümleri ve bağlantı kablosu eklemek, kullanıcıların mantık kapıları arasında bağlantılar kurmasına olanak tanır. Bu, kullanıcıların farklı kapıların çıkışlarını ve girişlerini birbirine bağlamasını sağlar. Mantık kapılarını sürükleyerek bir kapıyı belirli bir konuma taşımak, kullanıcıların devreleri tasarlamasına olanak tanır.

3 UYGULAMA
3.2 Kodlanan bileşenlerin açıklamaları

- Kütüphane olarak Tkinter kullanılmıştır.

```
import tkinter as tk
from tkinter import ttk
```

- Özellik tablosu için sınıf

```
class Ozellik_Tablosu:
    def __init__(self, master, etiket, tur, ozellik, guncelleme):
        self.master = master
        self.item_id = etiket
        self.item_type = tur
        self.properties = ozellik
        self.update_callback = guncelleme

        self.window = tk.Toplevel(master)
        self.window.title("ÖZELLİK TABLOSU")

        self.property_entries = {}

        row = 0
        for prop, value in ozellik.items():
            label = ttk.Label(self.window, text=prop)
            label.grid(row=row, column=0, sticky="e")

            entry = ttk.Entry(self.window)
            entry.insert(0, value)
            entry.grid(row=row, column=1)

            self.property_entries[prop] = entry

            row += 1

    kaydet = ttk.Button(self.window, text="Kaydet", command=self.ozellik_kaydet)
    kaydet.grid(row=row, columnspan=2)

def ozellik_kaydet(self):
    guncellenmis = {}
    for prop, entry in self.property_entries.items():
        guncellenmis[prop] = entry.get()
    self.update_callback(self.item_id, guncellenmis)
    self.window.destroy()
```

Özellik tablosu için bir sınıf oluşturulmuştur. Tablonun nitelikleri belirlenmiştir ve pencere oluşturulması için Tkinter kullanılmıştır.

Bir for döngüsü bulunmaktadır. Bu döngü, verilen özellik sözlüğündeki her bir Anahtar-değer çifti (property-value pair) için çalışır.

Label Oluşturma: ttk. Label, Tkinter' in bir parçası olan ve bir etiket (label) oluşturan bir bileşendir. text=prop parametresi, etiketin içeriğini belirler (özelliğin adı).

Etiketin Yerleşimi: label. grid(row=row, column=0, sticky="e") komutu, etiketi pencere üzerinde belirli bir satır ve sütuna yerleştirir. sticky="e" ifadesi, etiketin hücrenin doğusuna (sağ) hizalanmasını sağlar.

Girdi Alanı (Entry) Oluşturma: ttk. Entry, bir girdi alanı oluşturan bileşendir. entry. İnsert(0, value), girdi alanına varsayılan olarak value değerini ekler.

Girdi Alanının Yerleşimi: entry. grid(row=row, column=1) komutu, girdi alanını pencere üzerinde belirli bir satır ve sütuna yerleştirir.

Girdi Alanının Kaydedilmesi: self. property_entries[prop] = entry ifadesi, bu girdi alanını self. property_entries sözlüğüne kaydeder. Böylece ileride bu alana erişim sağlanabilir.

Satır Numarasının Arttırılması: row += 1 komutu, her iterasyonda satır numarasını bir artırarak, bir sonraki özelliğin bir alt satıra yerleşmesini sağlar.

Özellik kaydetmek için bir fonksiyon bulunur. Bu fonksiyondaki
for prop, entry in self. property_entries. items(): döngüsü, her bir girdi alanındaki güncellenmiş değeri entry. Get() metodu ile alır ve güncellenmiş sözlüğüne ekler.

- Mantık devreleri sistemi için sınıf

```
class Mantik_Devresi_Olusturma:
    def __init__(self, master):
        self.master = master
        self.master.title("MANTIK DEVRELERİ")

        self.canvas = tk.Canvas(master, width=1000, height=700, bg="light grey")
        self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        self.gates = []
        self.connections = {}
        self.input_values = {}
        self.output_boxes = {}
        self.current_gate = None
        self.drag_data = {"x": 0, "y": 0, "item": None}
```

Mantık devleri oluşturulacak pencere için canva oluşturulur.

Self. Canvas = tk. Canvas(master, width=1000, height=700, bg="light grey"): Canvas, üzerinde çizim yapılabilecek bir alan sağlar. Genişliği 1000, yüksekliği 700 piksel olarak ayarlanır ve arka plan rengi açık gridir.

Tuval, ana pencerenin sol tarafına yerleştirilir, mevcut tüm alanı kaplar ve pencere boyutlandırıldığında genişler.

```
# Resimler
self.images = {
    "AND": tk.PhotoImage(file="and_gate.png"),
    "OR": tk.PhotoImage(file="or_gate.png"),
    "NOT": tk.PhotoImage(file="not_gate.png"),
    "NAND": tk.PhotoImage(file="nand_gate.png"),
    "NOR": tk.PhotoImage(file="nor_gate.png"),
    "BUFFER": tk.PhotoImage(file="buffer.png"),
    "XOR": tk.PhotoImage(file="xor_gate.png"),
    "XNOR": tk.PhotoImage(file="xnor_gate.png"),
}
```

Mantık kapılarının sembolleri

```
# Kapı seçim arayüzü
self.gate_selection_frame = ttk.Frame(master)
self.gate_selection_frame.pack(side=tk.RIGHT, fill=tk.Y)

logic_gate_frame = ttk.LabelFrame(self.gate_selection_frame, text="Mantık Kapıları")
logic_gate_frame.pack(fill=tk.BOTH, expand=True)

self.selected_gate = tk.StringVar()
self.selected_gate.set("AND")
```

Bu kod, mantık kapılarını seçmek için bir arayüz oluşturur. Sağ tarafta bir çerçeve oluşturur, bu çerçeve içinde "Mantık Kapıları" başlıklı bir alan yaratır ve varsayılan olarak "AND" kapısını seçer.

- Arayüz için olan tüm düğmeler ve yapılar

```
for gate_type in self.images:
    button = ttk.Radiobutton(logic_gate_frame, text=gate_type, variable=self.selected_gate, value=gate_type)
    button.pack(anchor=tk.W)

self.ekle_butonu = ttk.Button(self.gate_selection_frame, text="Ekle", command=self.mantik_kapisi_ekle)
self.ekle_butonu.pack(side=tk.BOTTOM, fill=tk.X)

self.buton_1 = ttk.Frame(master)
self.buton_1.pack(side=tk.TOP, fill=tk.Y)

self.baglanti_butonu = ttk.Label(self.buton_1, text="Bağlantı Elemanları")
self.baglanti_butonu.pack()

self.cizgi_cizme = ttk.Button(self.buton_1, text="Bağlantı Kablosu", command=self.baglanti_kablosu_ekle)
self.cizgi_cizme.pack()

self.baglantidugum_butonu = ttk.Button(self.buton_1, text="Bağlantı Düğümü", command=self.baglanti_dugum_ekle)
self.baglantidugum_butonu.pack()

self.giris_cikis_butonu = ttk.Label(self.buton_1, text="Giriş/Çıkış Elemanları")
self.giris_cikis_butonu.pack()

self.led_button = ttk.Button(self.buton_1, text="LED", command=self.led_ekleme)
self.led_button.pack()

self.giris_butonu = ttk.Button(self.buton_1, text="Giriş Kutusu", command=self.giris_kutusu_ekleme)
self.giris_butonu.pack()

 self.cikis_butonu = ttk.Button(self.buton_1, text="Çıkış Kutusu", command=self.cikis_kutusu_ekle)
 self.cikis_butonu.pack()

 self.giris_cikis_butonu = ttk.Label(self.buton_1, text="Kontrol Elemanları")
 self.giris_cikis_butonu.pack()

 self.run_button = ttk.Button(self.buton_1, text="ÇALIŞTIR", command=self.cikis_kutusu_hesaplama2)
 self.run_button.pack()

 self.stop_button = ttk.Button(self.buton_1, text="DURDUR", command=self.stop_simulation)
 self.stop_button.pack()

 self.reset_button = ttk.Button(self.buton_1, text="RESETLE", command=self.reset_simulation)
 self.reset_button.pack()

 self.canvas.bind("<ButtonPress-1>", self.on_press)
 self.canvas.bind("<B1-Motion>", self.on_drag)
 self.canvas.bind("<ButtonRelease-1>", self.on_release)

 self.canvas.tag_bind("gate", "<Double-1>", self.ozellik_tablosu_penceresi)
 self.canvas.tag_bind("io", "<Double-1>", self.ozellik_tablosu_penceresi)
 self.canvas.tag_bind("connection", "<Double-1>", self.ozellik_tablosu_penceresi)
```

Düğmeler Tkinter metotlarıyla oluşturulur.

- Düğmelerin durumlarını işlevsel olarak gerçekleştirmek için metotlar

Bu arayüz metodları, mantık kapıları, bağlantı kablosu, bağlantı düğümleri, LED'ler, giriş ve çıkış kutularını eklemeye olanak tanır.

Her bir öğe eklenirken, görüntüleri yerleştirir ve bunların özelliklerini ve bağlantılarını saklamak için gerekli veri yapılarını günceller.

```
def mantik_kapisi_ekle(self):
    kapi_turu = self.selected_gate.get()
    x, y = self.canvas.winfo_width() / 4, self.canvas.winfo_height() / 4
    self.current_gate = self.canvas.create_image(x, y, image=self.images[kapi_turu], anchor=tk.CENTER, tags="gate")
    self.gates.append((self.current_gate, kapi_turu))
    self.connections[self.current_gate] = {"inputs": [], "output": None}
```

```
def baglanti_kablosu_ekle(self):
    self.canvas.bind("<ButtonPress-1>", self.start_wire)
    self.canvas.bind("<B1-Motion>", self.draw_wire)
    self.canvas.bind("<ButtonRelease-1>", self.end_wire)

def start_wire(self, event):
    self.wire_start_x = event.x
    self.wire_start_y = event.y
    self.current_wire = self.canvas.create_line(event.x, event.y, event.x, event.y, fill="black", width=2, tags="conn

def draw_wire(self, event):
    self.canvas.coords(self.current_wire, self.wire_start_x, self.wire_start_y, event.x, event.y)

def end_wire(self, event):
    self.connections[self.current_wire] = {"inputs": [], "output": None}
    self.gates.append((self.current_wire, "WIRE"))
    self.canvas.unbind("<ButtonPress-1>")
    self.canvas.unbind("<B1-Motion>")
    self.canvas.unbind("<ButtonRelease-1>")
```

```
def baglanti_dugum_ekle(self):
    self.canvas.bind("<ButtonPress-1>", self.dugum_yerlestirme)

def dugum_yerlestirme(self, event):
    connection_node = self.canvas.create_oval(event.x - 5, event.y - 5, event.x + 5, event.y + 5, fill="black", tags="connection")
    self.gates.append((connection_node, "NODE"))
    self.canvas.unbind("<ButtonPress-1>")
```

```
def led_ekleme(self):
    x, y = self.canvas.winfo_width() / 1.2, self.canvas.winfo_height() / 1.2
    led = self.canvas.create_oval(x - 30, y - 30, x + 30, y + 30, fill="red", tags="io")
    self.gates.append((led, "LED"))
    self.connections[led] = {"inputs": [], "output": None}

def giris_kutusu_ekleme(self):
    x, y = self.canvas.winfo_width() / 8, self.canvas.winfo_height() / 5
    input_box = self.canvas.create_rectangle(x - 20, y - 20, x + 20, y + 20, fill="blue", tags="io")
    text = self.canvas.create_text(x, y, text="0", font=("Arial", 14, "bold"),tags=("io", "io_text"))
    self.gates.append((input_box, "INPUT"))
    self.connections[input_box] = {"inputs": [], "output": text}
    self.input_values[input_box] = {"value": "0", "text_id": text}

def cikis_kutusu_ekle(self):
    x, y = self.canvas.winfo_width() / 2.6, self.canvas.winfo_height() / 4
    output_box = self.canvas.create_rectangle(x - 30, y - 20, x + 30, y + 20, fill="green", tags="io")
    self.gates.append((output_box, "OUTPUT"))
    self.output_boxes[output_box] = {"input1": "0", "input2": "0", "gate": "AND"}
```

- Ekranda şekillerin hareket halini ayarlama

Bu kod, fareyle bir nesnenin sürüklendiği olayları yönetir. Kullanıcı fareyi hareket ettirdiğinde, nesneyi fareye göre taşır. Bu sayede, kullanıcı grafik arayüzünde nesneleri kolayca yer değiştirebilir.

```
def on_press(self, event):
    item = self.canvas.find_closest(event.x, event.y)
    if item:
        tags = self.canvas.gettags(item)
        if "gate" in tags or "io" in tags or "connection" in tags:
            self.drag_data["item"] = item
            self.drag_data["x"] = event.x
            self.drag_data["y"] = event.y

def on_drag(self, event):
    item = self.drag_data["item"]
    if item:
        dx = event.x - self.drag_data["x"]
        dy = event.y - self.drag_data["y"]
        self.canvas.move(item, dx, dy)
        self.drag_data["x"] = event.x
        self.drag_data["y"] = event.y

def on_release(self, event):
    self.drag_data["item"] = None
    self.drag_data["x"] = 0
    self.drag_data["y"] = 0
```

- Özellik tablosunun ekran için metotlar

Bu metod nesnenin türünü belirler, ardından nesnenin özelliklerini alır ve bir özellik tablosu penceresi açar. Kullanıcı değişiklikleri uyguladığında, nesnenin özelliklerini günceller. Özellikler arasında kapı türleri, giriş/çıkış kutusu renkleri ve bağlantı kablosu rengi bulunur. Bu işlem, kullanıcının nesnelerin özelliklerini kolayca incelemesine ve düzenlemesine olanak tanır.

```
def ozellik_tablosu_penceresi(self, event):
    item = self.canvas.find_closest(event.x, event.y)[0]
    item_type = self.canvas.gettags(item)[0]
    properties = self.ozellikleri_al(item, item_type)
    Ozellik_Tablosu(self.master, item, item_type, properties, self.ozellikleri_guncelle)

def ozellikleri_al(self, item, item_type):
    if item_type == "gate":
        gate_type = [gate[1] for gate in self.gates if gate[0] == item][0]
        return {"Etiket": gate_type, "Giriş Sayısı": "2"}
    elif item_type == "io":
        fill_color = self.canvas.itemcget(item, "fill")
        if fill_color == "blue":
            initial_value = self.input_values[item]["value"]
            return {"Etiket": "Input Box", "Renk": fill_color, "Başlangıç Değeri": initial_value}
        elif fill_color == "green":
            output_box_data = self.output_boxes[item]
            return {"Etiket": "Output Box", "Renk": fill_color, "Giriş 1": output_box_data["input1"], "Giriş 2": output_box_data
        else:
            return {"Etiket": "LED", "Renk": fill_color}
    elif item_type == "connection":
        return {"Etiket": "Connection", "Renk": self.canvas.itemcget(item, "fill")}
    return {}
```

```
def ozellikleri_guncelle(self, item, properties):
    item_type = self.canvas.gettags(item)[0]
    if item_type == "gate":
        for index, gate in enumerate(self.gates):
            if gate[0] == item:
                self.gates[index] = (gate[0], properties["Etiket"])
    elif item_type == "io":
        fill_color = properties["Renk"]
        self.canvas.itemconfig(item, fill=fill_color)
        if fill_color == "blue":
            initial_value = properties["Başlangıç Değeri"]
            self.input_values[item]["value"] = initial_value
            text_id = self.input_values[item]["text_id"]
            self.canvas.itemconfig(text_id, text=initial_value)
        elif fill_color == "green":
            self.output_boxes[item] = {
                "input1": properties["Giriş 1"],
                "input2": properties["Giriş 2"],
                "gate": properties["Mantık Kapısı"]
            }
            self.cikis_kutusu_hesaplama1(item)
```

```
    for index, gate in enumerate(self.gates):
        if gate[0] == item:
            self.gates[index] = (gate[0], properties["Etiket"])
elif item_type == "connection":
    fill_color = properties["Renk"]
    self.canvas.itemconfig(item, fill=fill_color)
    for index, gate in enumerate(self.gates):
        if gate[0] == item:
            self.gates[index] = (gate[0], properties["Etiket"])
```

- Çıkış kutusundaki girişlerle hesaplama yapmayı sağlayan metotlar

Bu metodlar, çıkış kutusu bileşenlerinin değerlerini hesaplamak için kullanılır. “cikis_kutusu_hesaplama1” fonksiyonu, belirli bir çıkış kutusunun değerini hesaplar ve ekrana yerleştirir.” cikis_kutusu_hesaplama2” fonksiyonu ise tüm çıkış kutularının değerlerini hesaplar ve ekrana günceller. “cikis_kutusu_hesaplama3” fonksiyonu, giriş değerlerine ve mantık kapısı türüne dayanarak mantıksal işlemi gerçekleştirir ve sonucu döndürür.

```
def cikis_kutusu_hesaplama1(self, item):
    data = self.output_boxes[item]
    input1 = data["input1"]
    input2 = data["input2"]
    gate = data["gate"]
    result = self.cikis_kutusu_hesaplama3(input1, input2, gate)
    text_id = self.canvas.find_withtag("output_text_" + str(item))
    x1, y1, x2, y2 = self.canvas.coords(item)
    if text_id:
        self.canvas.itemconfig(text_id, text=result,font=("Arial", 14, "bold"))
    else:
        self.canvas.create_text((x1 + x2) / 2, (y1 + y2) / 2, text=result, font=("Arial", 14, "bold"),tags=("outp

def cikis_kutusu_hesaplama2(self):
    for output_box, data in self.output_boxes.items():
        input1 = data["input1"]
        input2 = data["input2"]
        gate = data["gate"]
        result = self.cikis_kutusu_hesaplama3(input1, input2, gate)
        self.canvas.itemconfig(output_box, text=result, fill="white")
```

```
def cikis_kutusu_hesaplama3(self, input1, input2, gate):
    input1 = int(input1)
    input2 = int(input2)
    if gate == "AND":
        return str(input1 & input2)
    elif gate == "OR":
        return str(input1 | input2)
    elif gate == "NOT":
        return str(not input1)
    elif gate == "NAND":
        return str(not (input1 & input2))
    elif gate == "NOR":
        return str(not (input1 | input2))
    elif gate == "XOR":
        return str(input1 ^ input2)
    elif gate == "XNOR":
        return str(not (input1 ^ input2))
    return "0"
```

- Çalıştırma, durdurma, resetleme metotlar ve main modülü

Bu kod, bir simülasyonun durdurulması ve sıfırlanması işlemlerini gerçekleştirir. “stop_simulation” fonksiyonu şu anda herhangi bir işlevsellik sağlamıyor, yalnızca pass ifadesini içeriyor. “reset_simulation” fonksiyonu ise canvas'teki tüm nesneleri ve simülasyon sırasında kullanılan veri yapılarını temizleyerek simülasyonu sıfırlar.

```
    def stop_simulation(self):
        pass

    def reset_simulation(self):
        self.canvas.delete("all")
        self.gates.clear()
        self.connections.clear()
        self.input_values.clear()
        self.output_boxes.clear()

if __name__ == "__main__":
    root = tk.Tk()
    app = Mantik_Devresi_Olusturma(root)
    root.mainloop()
```

3.3 Görev dağılımı

Rapor ve kodlama aşamalarının hepsi ortak gerçekleştirilmiştir.

3.4 Karşılaşılan zorluklar ve çözüm yöntemleri

Zorluklar:

- Düğümleştirme gerçekleşemediği için devre olarak hesaplama yapılamıyor.

Çözüm Yöntemleri:

- Bunun için çıkış kutusundan özellik tablosuna giriş yaparak bu hesaplama yaptırılıyor.

3.5 Proje isterlerine göre eksik yönler

Devre simülasyonun gerçekleşmemesi

4 TEST VE DOĞRULAMA

4.2 Yazılımın test süreci

Yazılım için ayrı bir test kodu yazılmadı fakat konsoldan olabilecek durumlar için girişler yapıldı.

4.3 Yazılımın doğrulanması

4.3.1 Çeşitli girişler uygulanarak temel durumlara uygunluğu test edildi.