1. GMap.NET Kütüphanesi

1.1. GMap.NET Nedir

GMap.NET, .NET için hazırlanmış ücretsiz, çok platformlu ve açık kaynak kodlu harita kontrolüdür. Bünyesinde Google, Yahoo!, Bing, OpenStreetMap, ArcGIS, Pergo, SigPac, Yandex, Mapy.cz, Maps.lt, iKarte.lv, NearMap, HEREMap, CloudMade, WikiMapia ve MapQuest gibi birçok harita sağlayıcısı bulunur. Bu haritalar üzerinde katmanlar (overlay) kullanarak işaret (marker) koyulabilir, alan (polygon) çizilebilir ve rota oluşturulabilir.

Önbellek (cache) destekleyen GMap.NET, internet yokken bile önbellekteki veriyi kullanarak çalışmaya devam eder. Kontrolü güçlü kılan en önemli özelliklerinden biri de harita sağlayıcısının katmanlardan bağımsız olmasıdır. Böylece harita sağlayıcısı kaynaklı problemlerde harita sağlayıcısı değiştirildiğinde katmanlar yeni harita sağlayıcısında da çalışmaya devam eder.

Güncel sürümü v1.8.5 olup NuGet üzerinden başvuru dosyaları indirilebilir.

1.2. GMap.NET Kütüphanesinin Uygulamaya Eklenmesi

Windows Forms projesine GMap.NET.Core.dll ve GMap.NET.WindowsForms.dll başvuru dosyaları eklendikten sonra araç kutusuna da eklenir ve GMapControl aracı forma sürüklenerek projeye eklenmişmiş olur.

GMapControl sınıfının bazı özellikleri şunlardır:

- **CanDragMap <bool>:** Fare tuşuyla harita kaydırmayı aktif eder.
- **MarkersEnabled <bool>:** İşaretleri aktif eder.
- **PolygonsEnabled <bool>:** Alanları aktif eder.
- **RoutesEnabled <bool>:** Rotaları aktif eder.
- **MouseWheelZoomType <enum>:** Harita zoom tipini belirler.

- **`MinZoom <int>`:** Yapılabilecek en az zoom derecesini belirler.
- **`MaxZoom <int>`:** Yapılabilecek en fazla zoom derecesini belirler.
- **`MouseWheelZoomEnabled <bool>`:** Fare tekerleğiyle zoomu aktifleştirir.
- **`Zoom <int>`:** Mevcut zoom derecesini gösterir.
- **`Position <PointLatLng>`:** Harita merkezinin konumunu belirler.

`PointLatLng` sınıfı harita üzerindeki noktaların koordinatlarını tutmayı sağlar. Bu sınıfa ait iki özellik şunlardır:

- **`Lat <double>`:** Noktanın enlemini tutar.
- **`Lng <double>`:** Noktanın boylamını tutar.

1.3. GMap.NET Uygulaması – Haritayı Oluşturmak

Aracın formda çalışması için aşağıdaki özellikler ayarlanmalıdır:

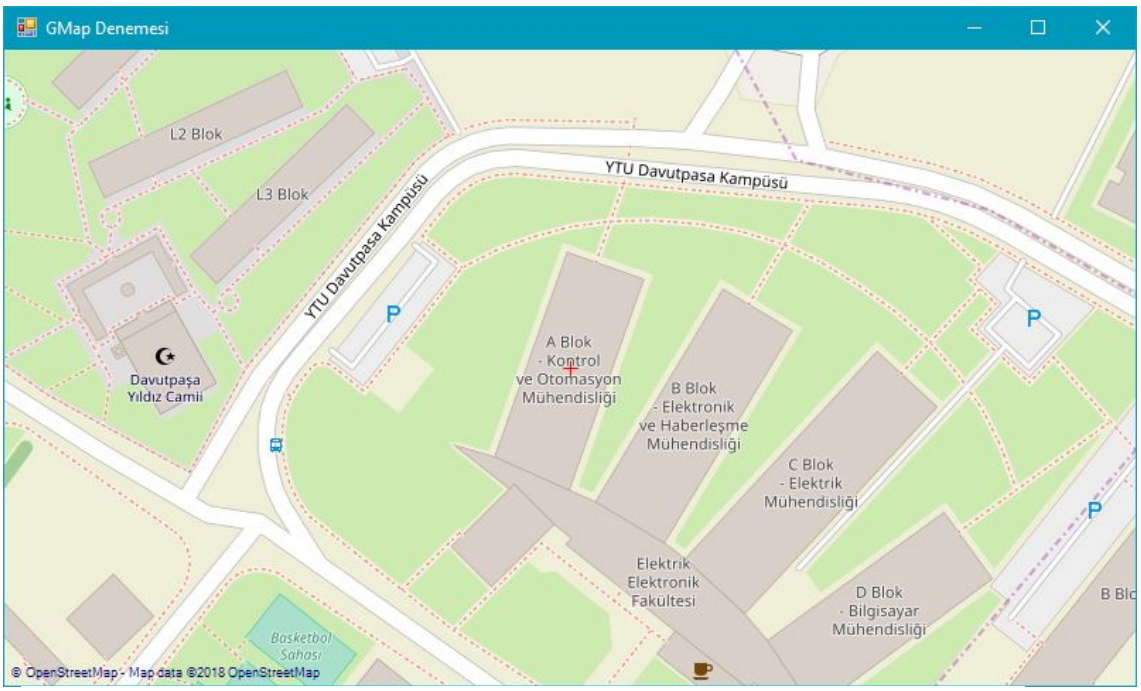
- **`MapProvider`:** Harita sağlayıcısı belirlenir (örnek: `OpenStreetMap`).
- **`Mode`:** Harita çalışma modu belirlenir (örnek: `ServerAndCache`).

Bu ayarlamalardan sonra `SetPositionByKeywords()` methodu kullanarak kelimeyle veya doğrudan koordinat girerek pozisyon ayarlanabilir (Kod Parçası 1.1).

```
GMapControl gmap = new GMapControl();
// Harita kontrolü oluşturuldu.
gmap.MapProvider = GMapProviders.OpenStreetMap;
// Haritanın sağlayıcısı belirlenir.
GMaps.Instance.Mode = AccessMode.ServerAndCache;
// Haritanın çalışma modu belirlenir.
gmap.SetPositionByKeywords("YTÜ KOM");
// Kelimeyle koordinat bulunur.
```

Kod Parçası 1.1. gmap nesnesi kullanarak harita oluşturma

(Kod Parçası 1.1) derlendikten sonra program çalışacak ve program penceresi çizilecektir (Program Çıktısı 1.1).



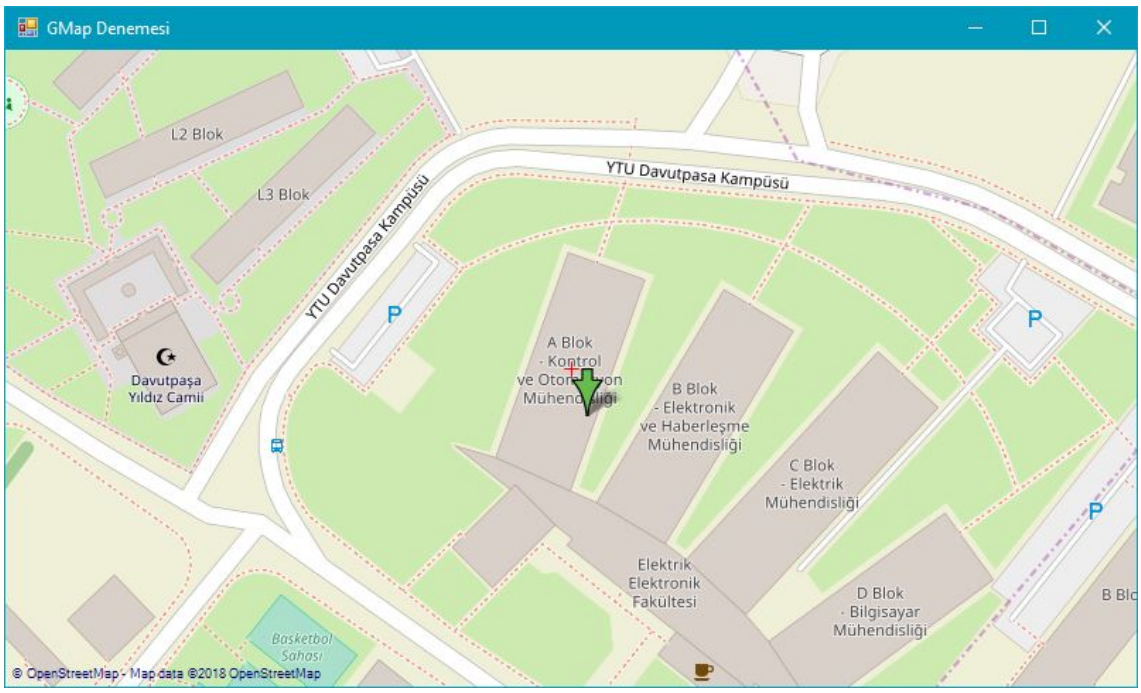
Program Çıktısı 1.1. YTÜ Elektrik – Elektronik Fakültesi'nin kuşbakışı görünüşü

1.4. GMap.NET Uygulaması – Katmanlar, İşaretler ve Araç Kutuları

Haritaya işaret, rota ya da alan eklenecekse bunların katmana eklenmesi gereklidir. `GMapOverlay` sınıfından katman oluşturulabilir. Projeye göre farklı katmanlar oluşturup her katmana farklı işaretlemeler yapmak mümkündür. `GMapMarker` sınıfından işaret oluşturulabilir. Oluşturucu methoduna `PointLatLng` ve `GMarkerGoogleType` parametreler verilerek istenen koordinatta ve işaret görselinde işaret oluşturulur. `GMarkerGoogleType` enum'u işaret görselini belirler.

```
GMapOverlay katman = new GMapOverlay(); // Katman oluşturuldu.
GMapMarker işaret = new GMarkerGoogle(
       new PointLatLng(41.0291982, 28.8899284), // Koordinatlar
       GMarkerGoogleType.arrow); // İşaret tipi
// İşaret oluşturuldu.
katman.Markers.Add(işaret); // İşaret katmana eklendi.
gmap.Overlays.Add(katman);  // Katman haritaya eklendi.
```

Kod Parçası 1.2. Katman nesnesinin türetilip düzenlenmesi



Program Çıktısı 1.2. İşaret KOM bloğu üzerinde

2. ZedGraph

2.1. ZedGraph Nedir

ZedGraph, .NET için hazırlanmış ücretsiz ve açık kaynak kodlu grafik kontrolüdür. 2 boyutlu grafik (çizgi, sütun, pasta grafikleri) çizmek için kullanılır. Grafiği tamamen ve ayrıntılı şekilde düzenlemeyi sağlasa da çoğu ayarlarında varsayılanı olduğu için kullanımı kolaydır.

2.2. ZedGraph Kütüphanesinin Uygulamaya Eklenmesi

Windows Forms projesine `ZedGraph.dll` başvuru dosyaları eklendikten sonra araç kutusuna da eklenir ve `ZedGraphControl` aracı forma sürüklenerek bu kütüphane projeye dahil edilmiş olur.

Grafik alanıyla ilgili işlemler için `ZedGraphControl.GraphPane` alanı kullanılır.

Bu alana ait bazı üye sınıflar şunlardır:

- **`Title`:** Grafik başlığı ve ilgili parametrelerini tutar.
- **`XAxis`:** Grafiğin X ekseniyle ilgili bilgilerini tutar.
- **`YAxis`:** Grafiğin Y ekseniyle ilgili bilgilerini tutar.
- **`IsEnableHZoom <bool>`:** X ekseninde zoom yapmayı aktif eder.
- **`IsEnableHPan <bool>`:** X ekseninde alan seçerek zoom yapmayı aktif eder.
- **`IsShowHScroolBar <bool>`:** X ekseninde hareketi sağlayan barı aktif eder.

Eksen bilgilerini tutan `XAxis` ve `YAxis` sınıflarına ait bazı üyeler şunlardır:

- **`Color`:** Eksen rengini tutan sınıftır.
- **`IsVisible <bool>` :** Eksen görünürlüğünü aktif eder.
- **`Title`:** Eksen başlığı ve ilgili parametrelerini tutar.
- **`Type <AxisType>` :** Eksen tipini belirler.

2.3. ZedGraph Uygulaması – Grafiği Oluşturmak

Aşağıdaki kod parçasıyla projeye sürüklenerek daha önce eklenmiş harita kontrolünü düzenlemek mümkündür:

```
ZedGraphControl zgc_grafik = new ZedGraphControl();
zgc_grafik.GraphPane.Title.Text = “AGNO Değişim Grafiği”;
zgc_grafik.GraphPane.XAxis.Title.Text = “Dönem”;
zgc_grafik.GraphPane.YAxis.Title.Text = “AGNO”;
double[] x = { 1, 2, 3, 4 };
double[] y = { 2.5, 2.78, 3.17, 3.19 };
```

Kod Parçası 2.1. Grafik kontrolünün oluşturulması

(Kod Parçası 2.1) derlendikten sonra grafik, düzenlenmiş başlıklarıyla beraber formda görünecek fakat bu grafiğe herhangi bir veri eklenmediği için grafik ekranı boş olacaktır.

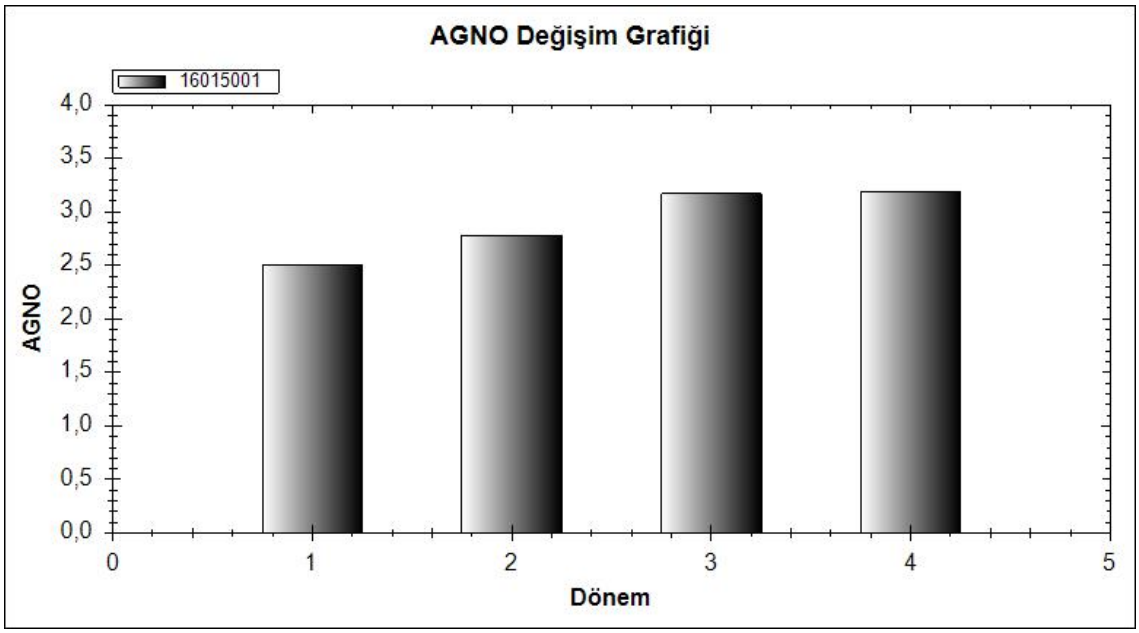
Grafiğe veri eklemek için `GraphPane` sınıfından `AddCurve()` methodu kullanılır. Bu method bu veriyi temsil eden `LineItem` nesnesi döndürür. İsteğe bağlı olarak bu nesnenin referansı saklanabilir. Bu method için aşağıdaki paremetreler kullanılır:

- **`String`:** Verinin adı.
- **`Double[]`:** Verilerin X değerlerinin tutulduğu dizi.
- **`Double[]`:** Verilerin Y değerlerinin tutulduğu dizi.
- **`Color`:** Verilerin grafikte görüneceği renk.

Aşağıdaki kod parçası da yukardakiyle birlikte derlenerek veri gösterilebilir:

```
BarItem v1 = zgc_grafik.AddBar("16015001", x, y, Color.Black);
```

Kod Parçası 2.2. Grafik kontrolüne veri eklenmesi.



Program Çıktısı 2.1. Grafiğin formda çizilmiş hali

`AddBar()` methodunun bu aşırı yüklemesi kullanıldığında dizilerin yapısından dolayı daha fazla nokta eklenemeyecektir. Noktalar üzerinde daha fazla kontrole sahip olmak için `PointPairList` veya `RollingPointPairList` sınıfı kullanılır.

RollingPointPairList sınıfının PointPairList sınıfından farkı RollingPointPairList sınıfın oluşturucu methoduna kapasite parametresi verilerek listenin kaldırabileceği maksimum nokta sayısı belirlenir. Kapasiteye ulaşıldığındaysa FIFO mantığı işler.

Yukarıda verilen kod parçasını kullanmak yerine aşağıda verilen kod parçası kullanıldığında burada bahsettiğimiz gibi noktalar üzerinde daha fazla kontrolümüz olacaktır:

```
PointPairList xy = new PointPairList(x, y);
BarItem v1 = zgc_grafik.AddBar("16015001", xy, Color.Black);
noktalar.Add(5, 3.3);
```

Kod Parçası 2.3. Grafik kontrolüne çalışma zamanında düzenlenebilir veri eklenmesi

Kodda da görüldüğü üzere nesne türetildikten sonra yeni bahsedilen aşırı yükleme kullanılarak veri eklenmiş ve daha sonra da verilere yeni bir örnek eklenmiştir.

Program çalışmadan önce bu ekleme yapıldığı için yeni örnek sorunsuz bir şekilde görülebilmektedir, fakat çalışma zamanında örnek eklendiği takdirde örnek başarıyla eklenebilse de programda görünmeyecektir, bunun sebebi grafik kontrolünün görsel olarak güncellenmemesidir, diğer bir değişle yeniden çizilmemesidir.

Bir program ilgili form penceresinin konumu değiştiğinde yeniden çizilir. Örneğin simge durumuna küçültülünce ya da ekranda yeri değişince yeniden çizilir. Bunları yapmadan yeniden çizmeyi zorlamak için harita kontrolüne ait Refresh() methodu kullanılır. Eğer eklenen yeni örnek grafiğin gösterdiği mevcut alanın dışındaysa yeni noktayı da içine alacak yeni bir alanın çizilmesi için yine GraphPane sınıfından AxisChange() methodu kullanılır.

```
zgc_grafik.Refresh();
zgc_grafik.GraphPane.AxisChange();
```

Kod Parçası 2.4. Grafiğin yeniden çizilmesi

3. PLC

3.1. PLC Nedir?

Programlanabilir lojik kontrolcü, makine kontrolü ve fabrika üretim hatları gibi alanlarda kullanılabilen otomasyon cihazıdır. Bilgisayarlara göre daha fazla giriş ve çıkışı olan bu cihazların en büyük artıları gürültülere, mekanik darbelere ve sıcaklık farklılıklarına dayanıklı olmalarıdır.

PLC'ler, cihazları üreten şirketlere göre farklı işletim sistemlerine sahip olur. Bu işletim sistemlerine kontrol edilecek sisteme uygun programlarlar yüklenebilir. Bu programlar da ms düzeyinde girişleri alarak gerçek zamana yakın çıkışlar üretecek şekilde çalışır.

PLC'ler aşağıdaki 4 ana bölümden oluşur.

- Merkezi işlem birimi:

CPU, yazılan programdaki işlemleri gerçekleştiren birimdir.

- Giriş birimi:

Kontrol edilen sistemle ilgili analog girişleri PLC'nin anlayacağı sayısal seviyeye dönüştüren birimdir. Giriş birimi voltaj değerleri DC veya AC olabilir.

- Hafıza birimi:

PLC içinde farklı görevler yapan hafızalar bulunur. Bir kısmı yazılan programı saklarken bir kısmı veri saklar.

- Çıkış birimi:

PLC'de hesaplanan lojik gerilimi ilgili kumanda elemanlarını sürmek amacıyla dönüştüren birimdir. Çıkış röleli veya transistörlü devreden oluşabilir. Transistörlü çıkış µs düzeyinde çalışırken röleli çıkış ms düzeyinde çalışır.

3.2. Hafıza Birimi

PLC programında kullanılabilecek 3 hafıza alanı bulunuyor. Bunlar aşağıdaki gibidir:

- Giriş Hafızası: Adres atamasında `I` operatörü kullanılır.
- Çıkış Hafızası: Adres atamasında `Q` operatörü kullanılır.
- Genel Hafıza: Adres atamasında `M` operatörü kullanılır.
- Hafızanın en küçük birimi `bit` olarak ifade edilir.
- 8 `bit`lik bir alan `bayt` olarak ifade edilir.
- 2 `bayt`lık bir alan `word` olarak ifade edilir.
- 2 `word`lük bir alan `dword` olarak ifade edilir.

Bazı adres atama örnekleri aşağıdaki gibidir:

- 0. `bayt` 7. `bit` çıkış -> `Q0.7`
- 0. `bayt` çıkış -> `QB0`
- 0. `word` çıkış (`QB0` – `QB1`) -> `QW0`
- 0. `dword` çıkış (`QW0` – `QW1` veya `QB0` – `QB1` – `QB2` – `QB3`) -> `QD0`

3.3. Kontaklar ve Atamalar

PLC programı yazılırken hafızadaki birimleri kullanabilmek için kontaklar kullanılır. Kontakların birbirine bağlanmasıyla kontrol edilecek sisteme göre devre kurulur. Bu kontaklar bool tipindedir. Bir bloğun enerjili olmasına yüksek, olmamasına düşük denir.

Kontrol edilecek sisteme göre kullanılabilecek kontak türleri aşağıdaki gibidir:

- **Normalde açık kontak:** Yüksekken kapalı, düşükken açık kontaktır. (F)
- **Normalde kapalı kontak:** Yüksekken açık, düşükken kapalı kontaktır. (F’)
- **Yükselen kenar algılayan kontak:** Yüksek olduğu an kapalı kontaktır.
- **Alçalan kenar algılayan kontak:** Düşük olduğu an kapalı kontaktır.

Bool bir atama yapılacağı zaman atama bloğu kullanılır. Kullanılabilecek atama blokları aşağıdaki gibidir:

- **Pozitif atama:** Yüksek olduğunda yüksek, düşük olduğunda düşük atar.
- **Negatif atama:** Düşük olduğunda yüksek, yüksek olduğunda düşük atar.
- **Setleme:** Yüksek olduğunda durumu değişse bile yüksek atar.
- **Resetleme:** Yüksek olduğunda durumu değişse bile düşük atar.

3.4. SIEMENS Simatic S7-1200 CPU1214C DC/DC/DC PLC ve TIA PORTAL

Bu PLC, Siemens tarafından üretilen bir PLC modeli olup piyasada 3 farklı modeli bulunur. Bunlar CPU 1211C, CPU 1212C ve CPU1214C olarak adlandırılır. Staj dönemi boyunca CPU1214C DC/DC/DC modeli kullanıldı. DC/DC/DC olduğundan DC beslenir, DC giriş alır ve DC çıkış verir. Kullanılan bu PLC'de ek olarak CSM 1277 haberleşme modülü bulunuyor. Bu modülle Ethernet kablosu kullanarak Profinet üzerinden PC ve HMI ile haberleşme mümkündür.

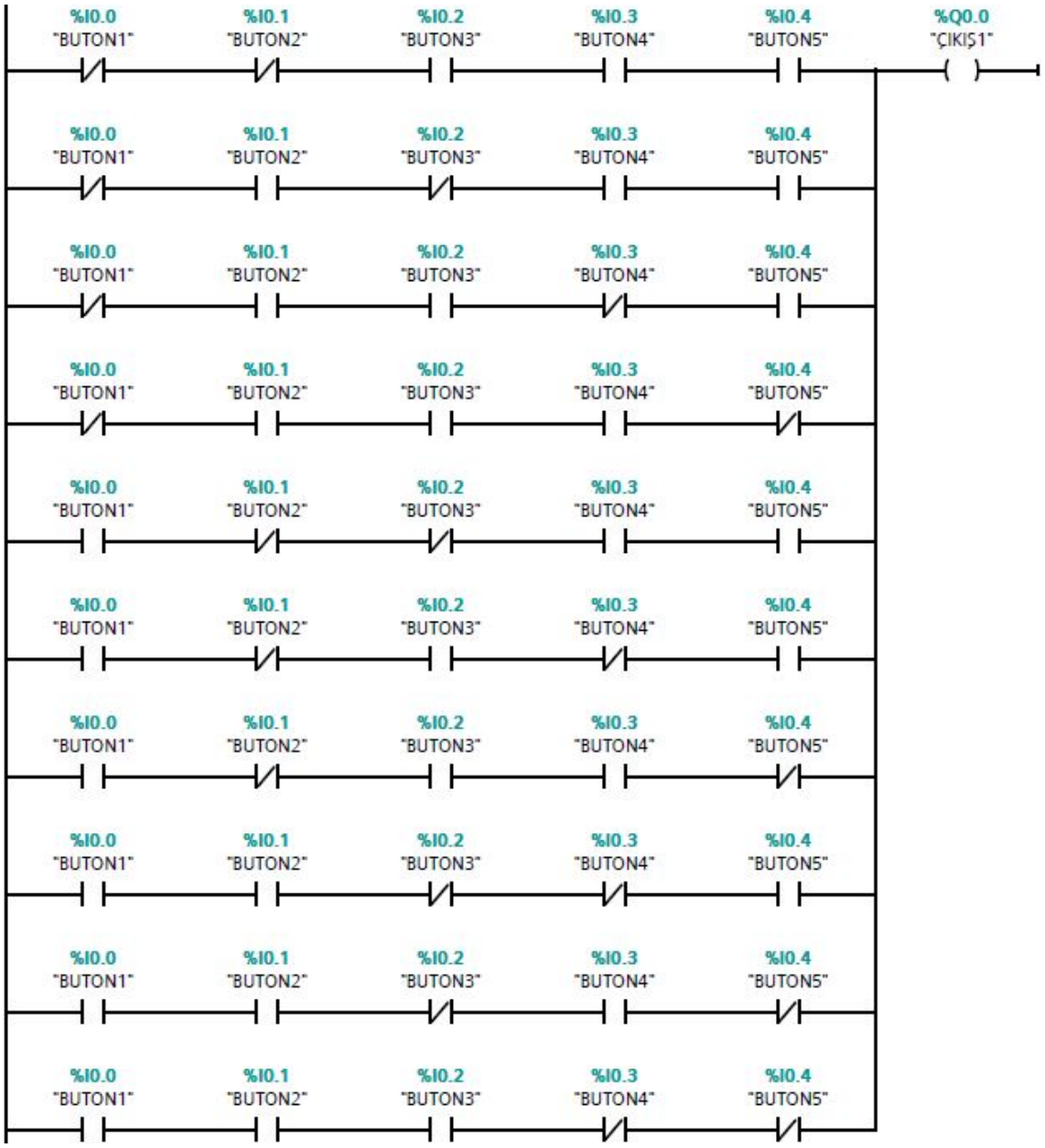
Siemens PLC'lerini programlamak için yine Siemens'in yazdığı Totally Integrated Automation Portal programı kullanılır. Staj dönemi boyunca V13 SP2 sürümü kullanıldı.

TIA PORTAL'da program yazmak için 3 dil bulunur. Bunlar Merdiven Diyagramı (LAD), Fonksiyon Blok Diyagramı (FBD) ve Yapısal Kontrol Dili (SCL) olarak adlandırılır. Staj dönemi boyunca LAD ve SCL tercih edildi.

3.5. LAD'e Giriş

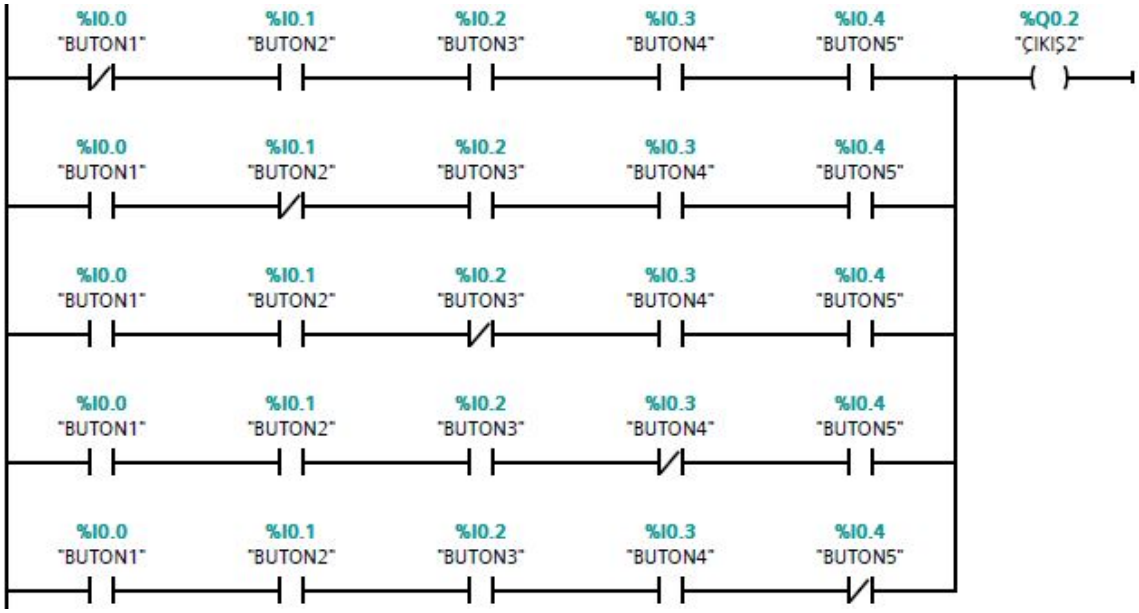
LAD'de soldan sağa ve yukarıdan aşağıya doğru bir akış bulunur. Her bir ağa ilgili bloklar koyularak algoritma oluşturulabilir. PLC, her ağdaki algoritmayı sırasıyla çalıştırır ve tüm ağlar bittikten sonra tekrar başa döner. Buna tarama denir ve bu tarama süresi ağlardaki algoritmalara da bağlı olmak üzere ms seviyesinde olur.

5 girişten oluşan bir sistem olsun. 5 girişten sadece 3 tanesi yüksekse çıkış yüksek olsun. 5 girişten 3 giriş seçilip yüksek, seçilmeyen 2 giriş düşük olacak. Bu cümlenin matematiksel karşılığı C(5,2)'dir. `C(5,2) = 10` olduğundan bu 10 ihtimali LAD ile yazıp ortak çıkışa bağlarsak bu algoritma tasarlanır.



Kod Parçası 3.1. 3 giriş yükseksen çıkışı yüksek olan sistem

Aynı sistemde bu sefer sadece 4 giriş yüksekken çıkış yüksek olsun. Aynı mantıkla C(5,4) = 5 olduğundan bu 5 ihtimali LAD ile yazıp ortak çıkışa bağlarsak bu algoritma da tasarlanır.



Kod Parçası 3.2. 4 giriş yüksekken çıkışı yüksek olan sistem

3.6. Zamanlayıcı Blokları

Zamana göre tasarlayacağımız algoritmalar olduğunda kullanabileceğimiz bloklardır. İlk zamanlar sadece TON bloğundan oluşan zamanlayıcı blokları şuanda 4 tanedir, bunlar aşağıdaki gibidir:

- TON (Timer On-Delay)
- TOF (Timer Off-Delay)
- TP (Timer-Pulse)
- TONR (Timer On-Delay with Reset)

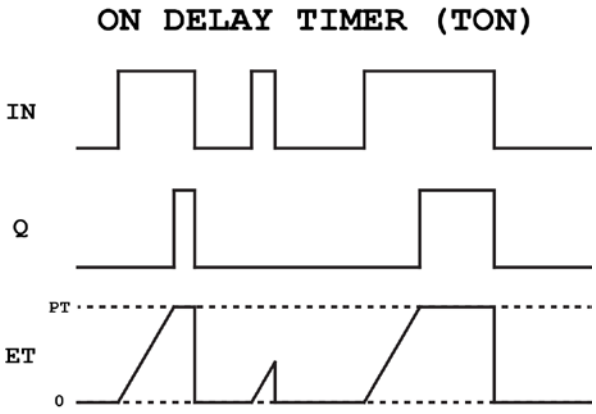
Bu zamanlayıcı bloklarının girişleri şunlardır:

- **IN (Input) <Bool>**: Zamanlayıcının girişi olup zamanlayıcıyı çalıştırır.
- **PT (Preset Time) <Time>:** ET'nin üst sınırıdır.

Bu zamanlayıcı bloklarının çıkışları şunlardır:

- **Q (Output) <Bool>:** Zamanlayıcının çıkışıdır.
- **ET (Elapsed Time) <Time>:** Zamanlayıcı çalışırken geçen süreyi verir.

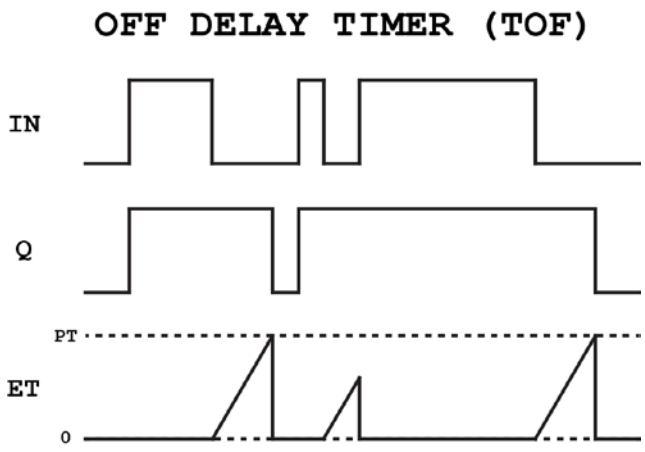
3.6.1. TON (Timer On-Delay)



Bu zamanlayıcı bloğunda IN yüksekken ET artmaya başlar. Artarken PT değerine ulaşırsa Q yüksek olur. IN düşük olursa ET sıfırlanır ve Q da düşük olur. IN yüksekken ET, PT değerine ulaşmadan düşük olursa ET sıfırlanır.

Şekil 3.1. TON işleyişi

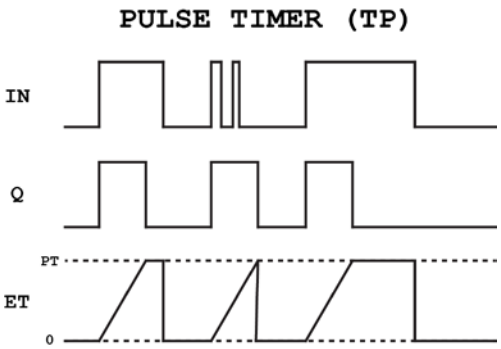
3.6.2. TOF (Timer Off-Delay)



Bu zamanlayıcı bloğunda IN yüksekken Q yüksek olur. IN düşük olduğunda ET artmaya başlar. Artarken PT değerine ulaşırsa Q düşük olur ve ET sıfırlanır. IN düşükken ET, PT değerine ulaşmadan IN yüksek olursa ET sıfırlanır.

Şekil 3.2. TOF işleyişi

3.6.3. TP (Timer-Pulse)



Bu zamanlayıcı bloğunda IN yüksekken Q yüksek olur, ET artmaya başlar ve PT değerine ulaşınca IN değerinden bağımsız olarak Q düşük olur. ET artarken PT değerine ulaşana kadar IN değerinin değişmesi ET ve Q değerlerini etkilemez.

Şekil 3.3. TP işleyişi

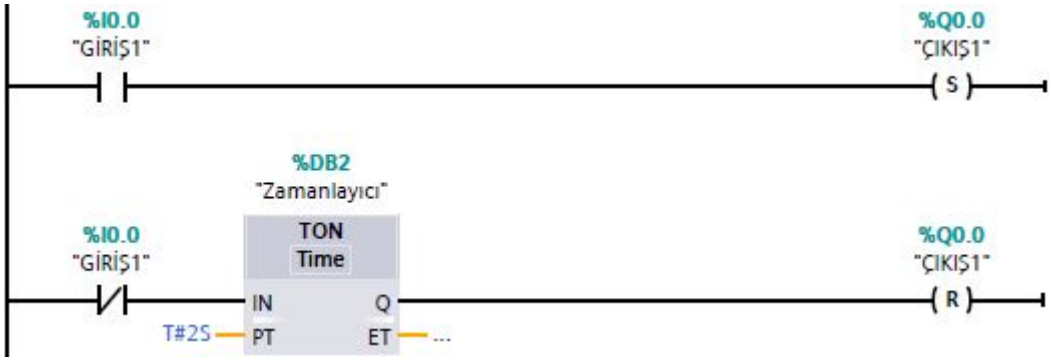
3.6.4. TONR (Timer On-Delay with Reset)

Bu zamanlayıcı bloğunda ek olarak aşağıdaki giriş bulunur:

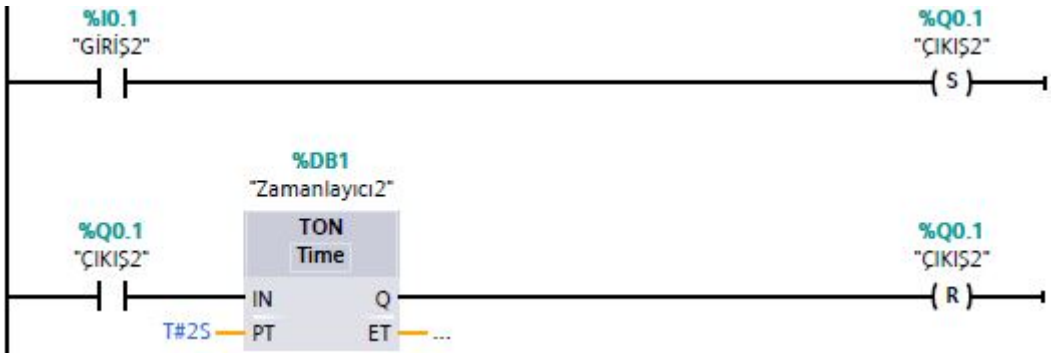
- **R (Reset) <Bool>:** ET değerini sıfırlar.

Bu zamanlayıcı bloğu çalışma prensibi olarak TON bloğuna benzer. İki farkı IN düşükken ET değerinin sıfırlanmaması ve ET değerini sadece R değerinin sıfırlamasıdır. Böylece IN tekrar yüksek olduğunda ET değeri kaldığı yerden artabilecektir.

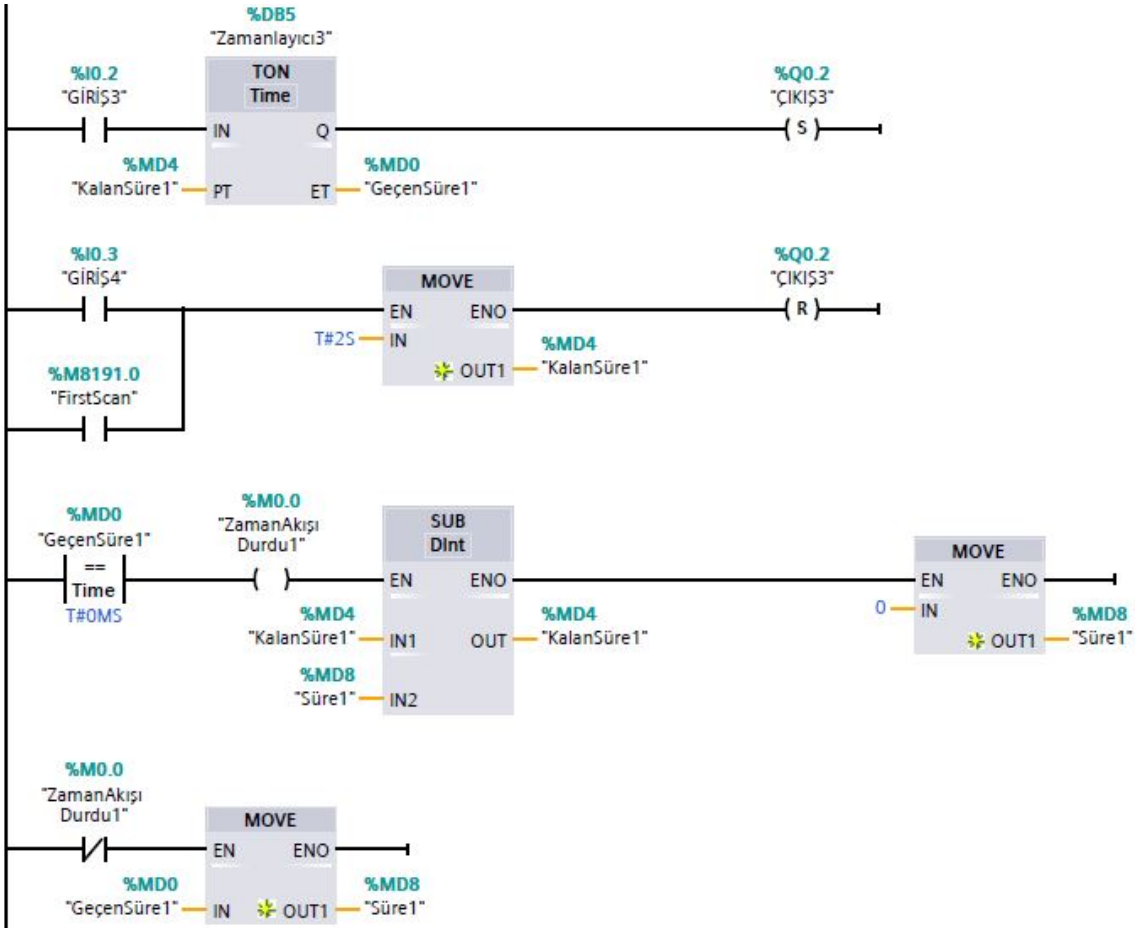
3.6.5. TON Bloğuyla Diğer Zamanlayıcı Bloklarının Tasarlanması



Kod Parçası 3.3. TON ile TOF tasarımı



Kod Parçası 3.4. TON ile TP tasarımı



Kod Parçası 3.5. TON ile TONR tasarımı

`FirstScan` sadece ilk taramada yüksek olup başlangıç değeri atamada kullanılır.

STAJDAN SORUMLU MÜHENDİS	Ünvan Ad ve Soyad	Doktor Öğretim Üyesi Türker TÜRKER	Firma İmza Kaşe	

3.7. Sayıcı Blokları

Sayıcı blokları, tam sayı değerlerini artıran ve/veya azaltan ve istenilen değere ulaştığında çıkış veren bloklardır. Sayısı 3 tane olup bunlar aşağıdaki gibidir.

- CTU (Up Counter)
- CDU (Down Counter)
- CTUD (Up Down Counter)

Bu sayıcılarda ortak olan 2'şer giriş ve çıkış bulunur. Bunlar aşağıdaki gibidir:

- **PV (Preset Value) <Int>**
- **Q (Output) <Bool>:** Sayıcının çıkışıdır.
- **CV (Current Value) <Int>:** Sayıcının mevcut değeridir.

3.7.1. CTU (Up Counter)

Yukarı yönde sayan sayıcıdır. CV >= PV olduğunda çıkış verir. Bu sayıcının girişi aşağıdaki gibidir:

- **CU (Count Up) <Bool>:** Yükselen kenar algılayınca CV değerini artar.
- **R (Reset) <Bool>:** Sayıcının CV değerini 0'lar.

```
IF R THEN
    CV := 0;
ELSIF CU THEN
    CV := CV + 1;
END_IF;
    Q := (CV >= PV);
```

Kod Parçası 3.6. CTU bloğunun SCL ile yazılmış hali

3.7.2. CTD (Down Counter)

Aşağı yönde sayan sayıcıdır. CV == 0 olduğunda çıkış verir. Bu sayıcının girişi aşağıdaki gibidir:

- **CD (Count Down) <Bool>:** Yükselen kenar algılayınca CV değeri azalır.
- **LD (Load) <Bool>:** Sayıcının CV değerini PV değeri yapar.

```
IF LD THEN
    CV := PV;
ELSIF CD THEN
    CV := CV - 1;
END_IF;
    Q := (CV <= 0);
```

Kod Parçası 3.7. CTD bloğunun SCL ile yazılmış hali

3.7.3. CTUD (Up Down Counter)

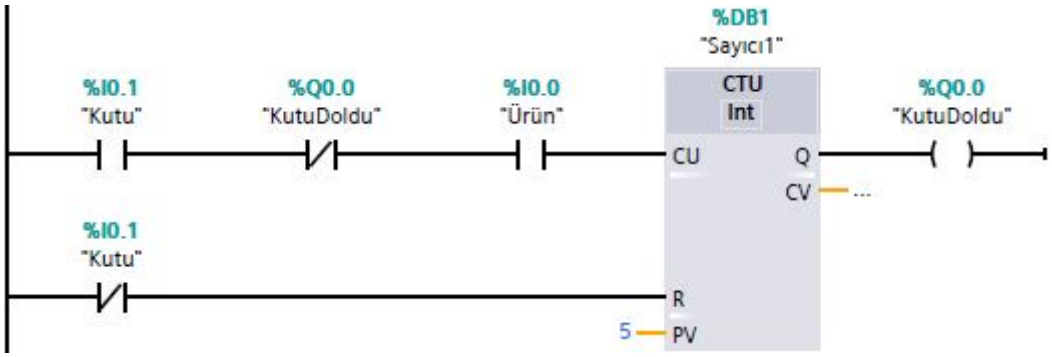
Diğer 2 sayacın birleştiği sayaçtır. Ortak giriş ve çıkışlara ek diğer sayaçlara özel girişler de sayaçta bulunur. Hem yukarı hem de aşağı sayılmak istendiğinde tek blok olarak kullanılabilir. 2 yön için ayrı 2 çıkış bulunur.

```
IF R THEN
    CV := 0;
ELSIF LD THEN
    CV := PV;
ELSE
    IF NOT (CU AND CD) THEN
        IF CU THEN
            CV := CV + 1;
        ELSIF CD THEN
            CV := CV - 1;
        END_IF;
    END_IF;
END_IF;
QU := (CV >= PV);
QD := (CV <= 0);
```

Kod Parçası 3.8. CTUD bloğunun SCL ile yazılmış hali

3.7.4. CTU Bloğuyla Algoritma Örneği

Banttan gelen ürünlerin kutuya düştüğü bir sistemde ürünlerin düştüğü kutunun kapasitesi 5'tir. Kutu dolduğunda bant duracak, yeni kutu koyulduğunda bant harekete başlayacaktır.



Kod Parçası 3.9. Algoritmanın LAD ile yazılmış hali

Eğer kutu varsa ve dolu değilse ürün geldikçe sayıcı sayacaktır. Kutu kapasitesine ulaşıldığında kutu dolmuş olacak ve daha fazla sayım olmayacaktır. Kutu alınıp yenisi koyulduğunda ürün geldikçe sayma işlemi yine kutu kapasitesine kadar devam edecektir.

3.8. Analog Girişler

PLC lojik bir kontrolcü olsa da takılabilecek modüllerle analog sinyallerle de işlem yapabilir hale gelir.

Dijital bir sinyal ya vardır ya yoktur, bu dijital sinyalin varlığı `1 (DOĞRU/YÜKSEK)` ile ifade edilirken yokluğu `0 (YANLIŞ/DÜŞÜK)` ile ifade edilir. Diğer bir ifadeyle alabileceği değer kesiklidir. Örneğin bir mesafe sensörü önündeki cismi ya görür ya da görmez.

Analog bir sinyalse belirli bir aralık arasında değişen değer alır, süreklidir. Örneğin bir sıcaklık sensörü, ölçtüğü sıcaklıkla orantılı bir gerilim çıkışı verir.

3.8.1. ADC ve DAC İşlemleri

PLC ise 0-10V arasındaki gerilimi ADC (Analog to Digital Convertion) işlemiyle 0-27648 aralığındaki tam sayıya dönüştürür. Burada 0V, 0 ile; 10V, 27468 ile temsil edilir. Hesaplanan tam sayıdan voltaj değerine geçiş için aşağıdaki algoritma kullanılır:

$$V = \frac{V_{UstSinir} - V_{AltSinir}}{Z_{UstSinir} - Z_{AltSinir}} * Z_{Hesaplanan}$$

Matematiksel Denklem 3.1. DAC (Digital to Analog Convertion) algoritması

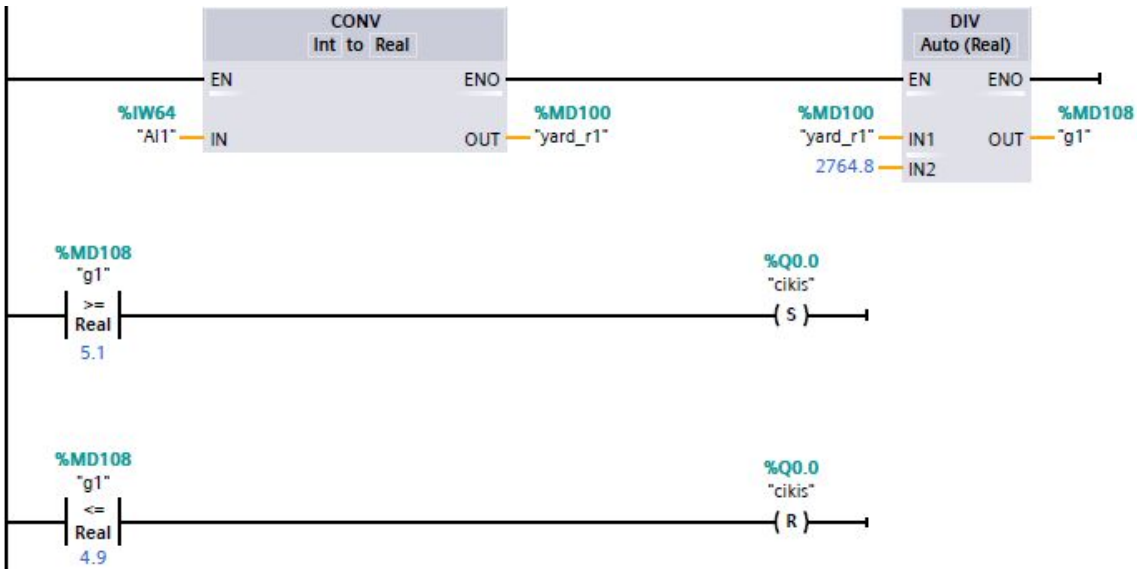
3.8.2. Çözünürlük

PLC'nin yaptığı ADC işlemi belirli bir hataya sahiptir ve buna çözünürlük denir. Çözünürlük, gerilim aralığı ne kadar dar ve tam sayı aralığı ne kadar geniş olursa o kadar yüksektir. Örneğin aynı gerilim aralığında 12 bitlik bir ADC işleminde çözünürlük 10/(2^12 – 1) = ~2,44mV olurken 16 bitlik bir ADC işleminde çözünürlük 10/(2^16 – 1) = ~0,15mV olur. Bu 4 bitlik artış ~2,28mV daha hassasiyet kazandırır. Çözünürlükten düşük gerilim değişikliği ADC işleminde ayrıklaşamayacaktır.

3.8.3. Algoritma Örneği

0-10V aralığındaki kararsız gerilim PLC ile ölçülsün. Ölçümü filtrelemek ve çıkışı kararlaştırmak amacıyla ölçüm >5,1 ise yüksek, <4,9 ise düşük çıkış verilsin. [4,9 5,1] aralığında ise çıkış değişmesin.

PLC bu sinyali tam sayıya dönüştürerek (ADC) okuyacaktır. Öncelikle yapılması gereken bu tam sayının temsil ettiği gerilim karşılığını hesaplamaktır. Bu hesaplama (DAC) yapıldıktan sonra hesaplanan değer >5,1 ise çıkış yüksek, <4,9 ise çıkış düşük olacak; bu aralıkta ise eski halini koruyacaktır.



Kod Parçası 3.10. Analog sinyali filtreleme algoritması

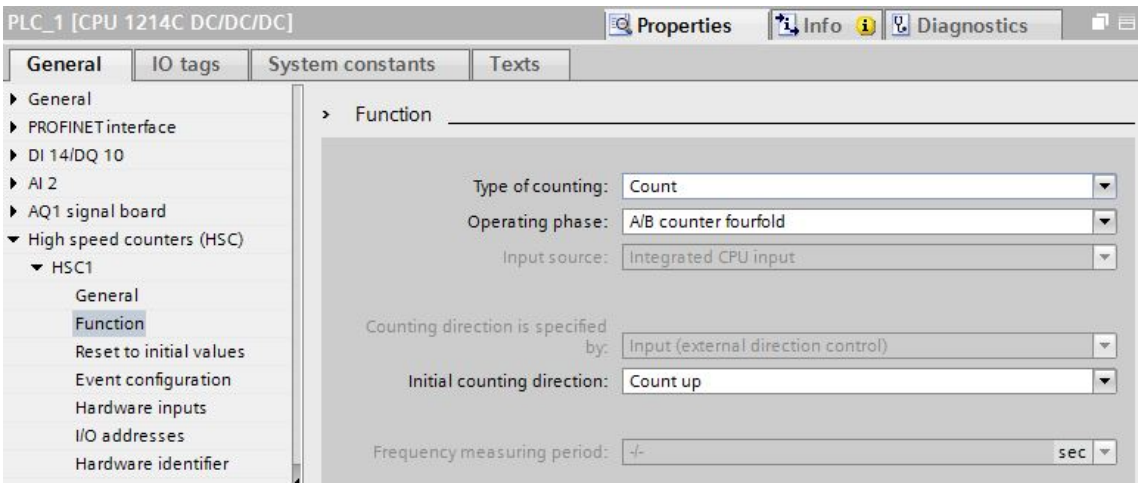
3.9. Artımsal Enkoder

Artımsal enkoder doğrusal veya dairesel hareketi elektrik sinyaline dönüştürür. 3 kanallı olan artımsal enkoderin bu kanalları A, B ve Z kanalları olarak adlandırılır. Kabaca yarık sayılarına göre ayrılırlar. Örneğin 500 yarıklı bir artımsal enkoder, tur başına bu sayıda pulse üretir. Buna göre pulse başına 0,72 derece döner. 0,72 dereceden az dönüşler algılanamaz.

A ve B kanallarından 90 derece faz farkıyla pulse üretilirken Z kanalından tur sayısı elde edilir. Eğer sadece devirle ilgileniliyorsa tek kanaldan ölçüm yeterliyken yönle de ilgileniliyorsa çift kanaldan ölçüm yapılmalıdır. Ek olarak pulse sayarken normal sayıcıdan ziyade HSC (Yüksek Hızlı Sayıcı) kullanılmalıdır.

3.9.1. Pulse Sayısından Açıya Geçiş

500 pulse bir artımsal enkoderin tur başına bu sayıda pulse ürettiğini biliyoruz. Eğer HSC ayarlarında `Operating Phase` olarak `A/B counter fourfold` ayarlanırsa sayım 4 katına çıkacaktır. Bu durumda tur başına 2000 sayım gerçekleşecektir.



Program Çıkısı 3.1. `Operating phase` ayarı

Bundan dolayı sayım başına da 0,18 derece düşecektir. Sayım başına düşen dereceyi bulduktan sonra sayıcının değerini bu değerle çaparak açıya ve yönüne ulaşabiliriz.

3.9.2. Açıdan Hıza Geçiş

Hızın hesaplanabilmesi için tanımı gereği bir miktar zamanın geçmesi gerekir. Hız – zaman grafiği belirli olan bir hareketlinin belirli bir zamandaki anlık hızını bulmak için grafiğin o zamandaki eğimini hesaplamak gerekir. Eğim de türevle hesaplanabilir:

$$\omega(t) = \lim_{t_0 \to 0} \frac{\theta(t + t_0) - \theta(t)}{t_0}$$

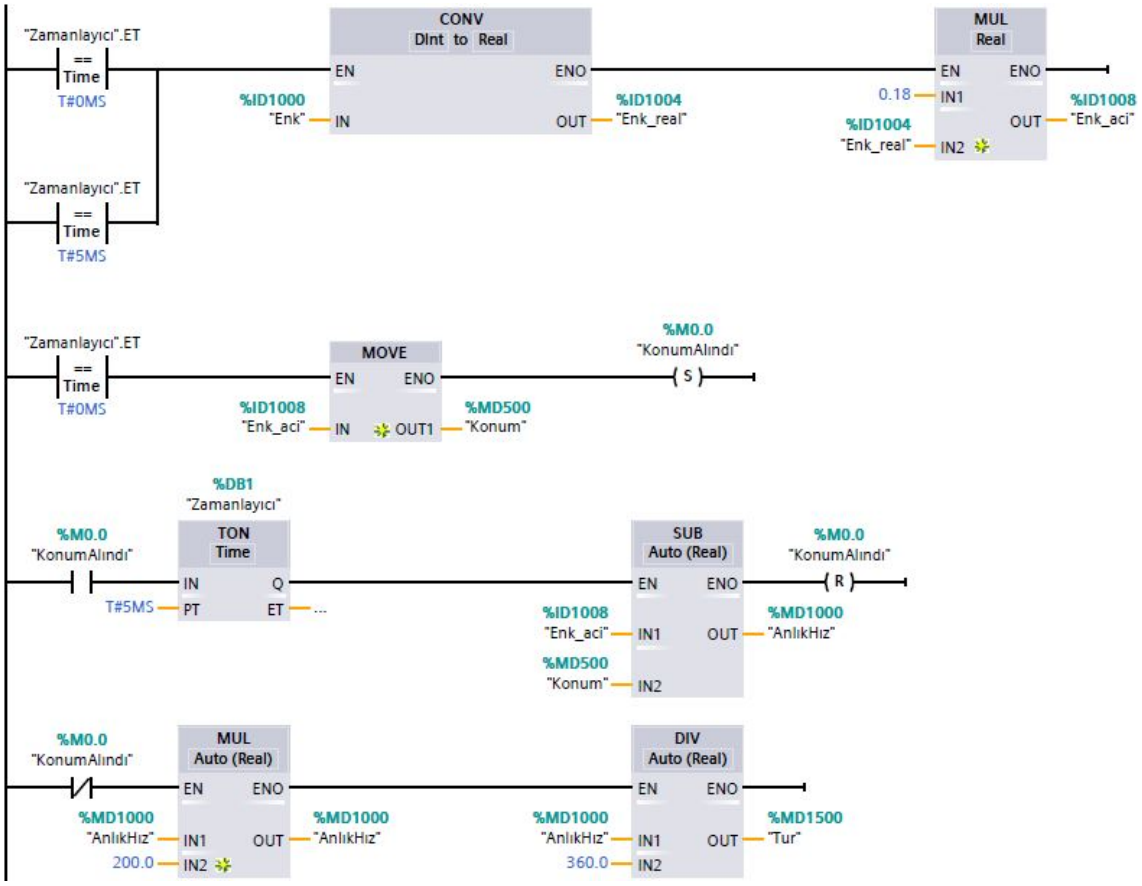
Matematiksel Denklem 3.2. Konumun ileri türevinden hıza geçiş

Bu denklem PLC'de kullanarak hesap yapılabilir fakat geçen zamanı 0'e götürmek mümkün olmayacağından nümerik türev kullanılarak geçen zaman çok küçük bir değer alınabilir.

Tarama süresi de düşünülerek geçen zaman `5ms` kabul edilirse PT değeri `5ms` olan bir zamanlayıcı kullanılması gerekir. ET değeri `0ms` iken sayıcıdan gelen sayım açıya

çevrilsin ve çevrilen bu değer saklansın. Saklandıktan sonra zamanlayıcı çalışmaya başlasın. Zamanlayıcı çıkış verdikten sonra gelen sayım tekrar açıya çevrilsin ve çevrilen bu değer, saklanan değerden çıkarılsın. Bu şekilde bu süre zarfındaki açı değişimi hesaplanmış olur.

Bu hesaplamanın ardından devir değişimini geçen süreye bölündüğünde nümerik olarak anlık hız derece/saniye cinsinden hesaplamış olur. Hesaplanan bu değer 360 dereceye bölünerek de devir/saniye cinsinden saniyedeki tur sayısı hesaplanmış olur.

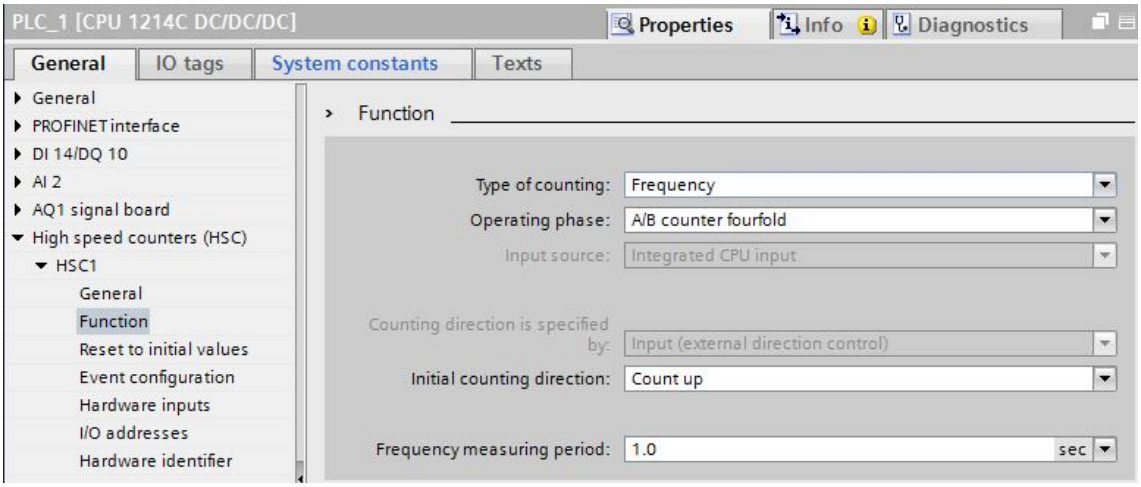


Kod Parçası 3.12. Pulse sayısından hıza geçiş algoritması

Her ne kadar mantıklı bir algoritma da olsa tarama süresinden dolayı sağlıklı bir hesaplama yapılamayabilir. Zira 5ms seçilen geçen süre 1ms seçildiğinde algoritmanın çalışmadığı görüldü. Bu sorun kesme kullanılarak çözülebilir, buna ek olarak

devir/saniye biriminde saniyedeki tur sayısını hesaplamak yerine daha çok kullanılan RPM biriminde dakikadaki tur sahibini hesaplamak kullanışlı olabilir.

Bu algoritmayı kullanmak yerine HSC ayarlarında sayma tipini Frequency olarak değiştirildiğinde bu sefer doğrudan açı/saniye biriminde hız hesaplaması yapıldığı görüldü. Buradan çıkan değerle algoritma sonucu çıkan değer karşılaştırıldı ve ~60RPM fark olduğu görüldü.



Program Çıktısı 3.2. Sayma tipinin Frequency olarak ayarlanması

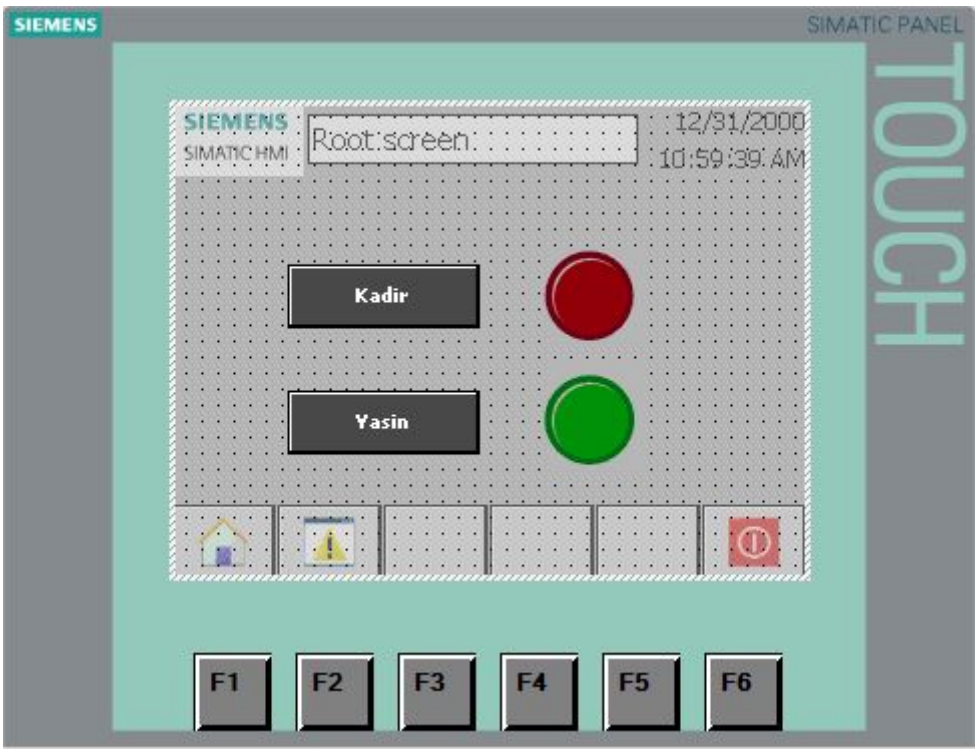
3.10. HMI Programlamaya Giriş

HMI (İnsan Makine Arayüzü), otomasyon sistemiyle insan arasında etkileşim sağlayan ekrandır. Dokunmatik panel, operatör paneli gibi isimlerle de anılır. Bu ekran vasıtasıyla bilgisayar olmaksızın sistemin kontrolü sağlanabilmektedir.

Staj dönemi boyunca HMI programlama basit düzeyde öğrenilmiştir.

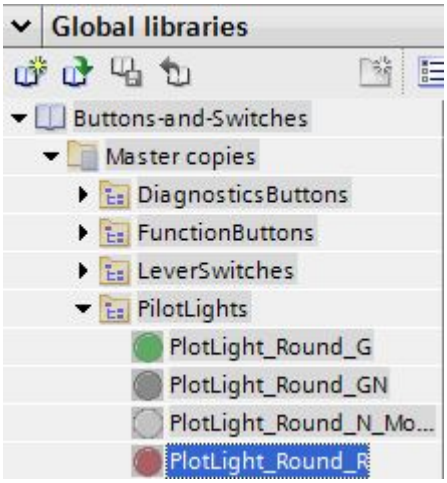
3.10.1. Basit HMI Uygulaması

2'şer tuş ve ışığın olduğu bir uygulamada tuşların sağında ışıklar konumlandırılsın. Tuşa basılı tutulduğunda ışık yansın, bırakıldığında ışık sönsün.



Program Çıktısı 3.3. HMI ekran

Işıklara global kütüphanede aşağıdaki dizininden erişilebilir:

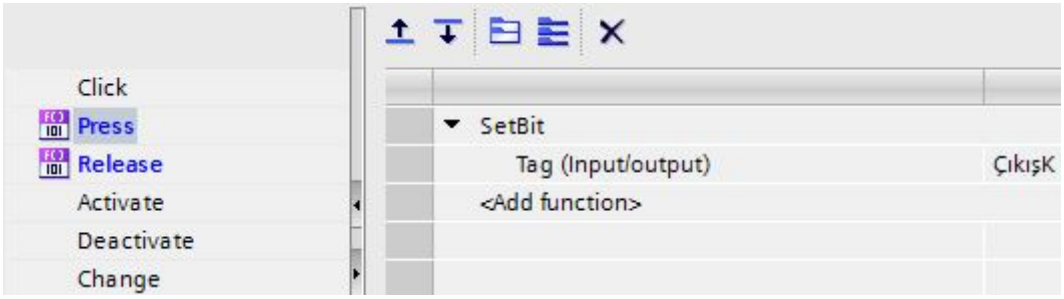


Program Çıktısı 3.4. Işıkların bulunduğu dizin

Tuşlar ve ışıklar eklendikten sonra ışıkları `bool` bir ifadeyle ilişkilendirmek gerekir. Bu ilişki kurulduktan sonra tuşların olayları kullanılarak ışıklar kontrol edilebilir.

PLC'deki olayları C#'taki olaylara benzetmek mümkündür. Windows Forms uygulamalarındaki `Button` sınıfının sahip olduğu birçok olay vardır. Bu olaylar işletim sistemi tarafından çağrılır. Örneğin form penceresindeki tuşa tıkladığı zaman `Click` olayı, çift tıkladığı zaman `DoubleClick` olayı tetiklenmiş olur.

Bu algoritmanın kurulması için `Press` ve `Release` olayları kullanıldı. Bu olaylara sırasıyla `SetBit` ve `ResetBit` fonksiyonları bağlandı ve bu fonksiyonlara parametre olarak ışıklarla ilişkilendirilen `bool` ifadeler verildi.



Program Çıktısı 3.5. `Press` olayına `SetBit` fonksiyonunun bağlanması

4. Arduino'da Seri Port Haberleşmesi

Staj dönemi boyunca `2018 TÜBİTAK Uluslararası İHA Yarışması`'nda kullanılmak üzere C# Windows Forms uygulaması üzerinde çalışıldı. Bu uygulamada staj defterinin 1. ve 2. bölümde bulunan kütüphanelerin kullanılması planlandı. Program basitçe İHA'ya ait verileri takip edebilmek amacıyla tasarlandı ve ilk olarak coğrafi konumları seri port üzerinden programa iletme üzerinde odaklanıldı.

Bu işlemi simüle etmek için `Arduino – PC` haberleşmesi üzerinde çalışıldı. `Arduino`'da enlem ve boylamı temsil edecek `single` değişkenler kullanıldı. Bu değişkenler 4 baytlık bayt dizisine dönüştürülerek seri porta yazıldı. C# uygulamasından da yazılan bu 4 baytlık dizi okunarak `single` dönüşümü yapıldı.

```
union
{
  double db;                          //Sayının double hali.
  byte by[4];                         //Sayının bayt[] hali.
} enlem;
union
{
  double db;                          //Sayının double hali.
  byte by[4];                         //Sayının bayt[] hali.
} boylam;

void setup()
{
  pinMode(A0,INPUT);                  //Arduino girişi seçilir.
  Serial.begin(9600);                 //Seri haberleşme başlar.
  enlem.db = 41.029198242468411;      //Örnek enlem.
  boylam.db = 28.889928460121155;     //Örnek boylam.
}

void loop()
{
  enlem.db = enlem.db + 0.00001;      //Enlem artıyor.
  boylam.db = boylam.db + 0.00001;    //Boylam artıyor.
  for (int i = 0; i < 4; i++)
    Serial.write(enlem.by[i]);        //Enlemler yazılıyor.
  for (int i = 0; i < 4; i++)
    Serial.write(boylam.by[i]);       //Boylamlar yazılıyor.
}
```

Kod Parçası 4.1. Arduino seri porta yazma algoritması

C# uygulamasında seri porttan okunacak bayt sayısı 8 için uzunluğu 8 olan bayt dizisi tanımlanır. `SerialPort` nesnesinin `DataReceived` olayı kullanılarak seri porttaki 8 bayt bu diziye aktarılır.

`single`, tek hassasiyetli kayan noktalı sayı demektir ve hafızada 4 baytlık yer kaplar. Buna göre 8 baytlık dizide 2 tane `single` bulunur. Her bir `single` sayının 4 baytı kullanarak dönüşüm yapılır ve haberleşme tamamlanmış olur.

```
byte[] b_koordinatlar = new byte[8];       //byte[] tanımlandı.
private void seriport_DataReceived(object sender,
SerialDataReceivedEventArgs e)
        {
            for (int i = 0; i < 8; i++)    //baytlar okunuyor.
                b_koordinatlar[i] = (byte)seriport.ReadByte();
        }
private void backgroundWorker_işaret_DoWork(object sender,
DoWorkEventArgs e)
        {
            float enlem   =
BitConverter.ToSingle(b_koordinatlar, 0);  //single dönüşümü
            float boylam  =
BitConverter.ToSingle(b_koordinatlar, 4);  //single dönüşümü
        }
```

Kod Parçası 4.2. C# uygulaması seri porttan okuma algoritması

5. Qt Programlama

Qt, çok platformlu bir geliştirme ortamıdır. Qt ortamında C++ kullanarak projeler oluşturulmaktadır. C++'dan farklı olarak form oluşturmak ve tasarlamak kolaydır. Qt ortamında program yazmak için Qt Creator programı kullanılır.

5.1. Merhaba Dünya Uygulaması

Merhaba Dünya uygulaması için Qt Konsol Uygulaması oluşturulur. Oluşturulduktan sonra `main.cpp` dosyası düzenlenir.

```
#include <QCoreApplication>     //Konsol uygulaması
#include <QDebug>               //Debug ekranı

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);   //main p.metreleri alınır
    qDebug() << "Merhaba Dunya";      //Ekrana çıktı alınır.
    return a.exec();                  //Uygulama çalışır.
}
```

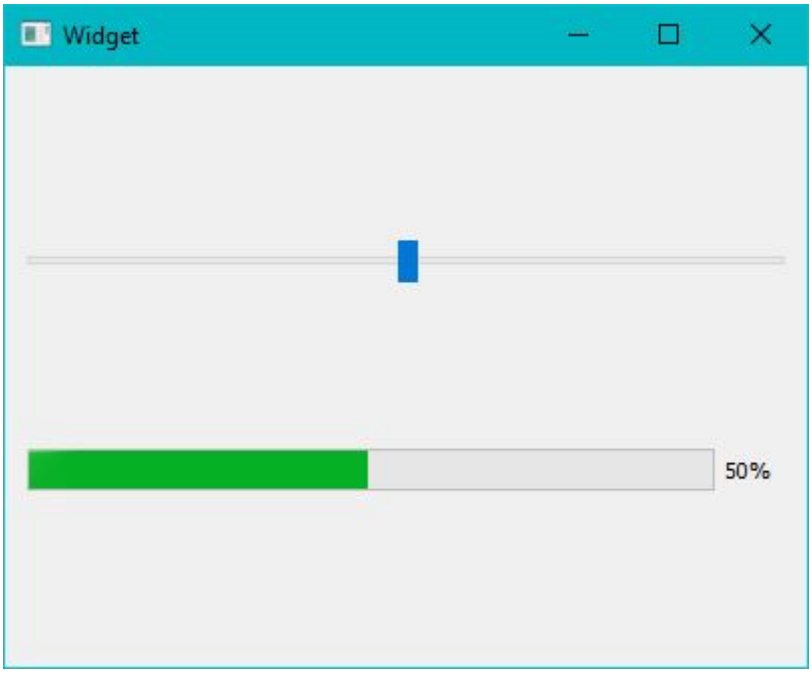
Kod Parçası 5.1. Merhaba Dünya

Program çalıştıktan sonra fark edilebilecek ilk farklılık kodun çalıştıktan sonra çıktı ekranının hemen kapanmamasıdır. Çünkü `main.cpp`'de döndürülen değer programı çalıştıran fonksiyonun döndürdüğü tam sayıdır. Bu fonksiyon da basitçe sonsuz döngü olarak düşünülebilir, bundan dolayı fonksiyon sonlanmadan ekran da kapanmayacaktır.

5.2. Sinyaller ve Slotlar

Sinyaller ve slotlar, C#'taki olaylara benzetilebilir. Windows Forms uygulamalarındaki `Button` sınıfının sahip olduğu birçok olay vardır. Bu olaylar işletim sistemi tarafından çağrılır. Örneğin form penceresindeki tuşa tıkladığı zaman `Click` olayı, çift tıkladığı zaman `DoubleClick` olayı tetiklenmiş olur.

Tetiklenen bu olaylar kendilerine bağlı fonksiyonları çalıştırırlar. Örneğin Qt Widget Uygulaması oluşturalım ve forma `QProgressBar` ve `QSlider` sınıflarından nesneler ekleyelim. Daha sonrasında `QSlider` nesnesinin `valueChanged(int)` sinyalini `QProgressBar` nesnesinin `setValue(int)` slotuna bağlayalım.



Program Çıktısı 5.1. `QSlider` ve `QProgressBar` içeren form

Sinyal ve slot birbirine bağlandıktan sonra artık `QSlider` nesnesinin değeri değiştikçe `valueChanged(int)` sinyali yayımlanacak ve bu sinyale bağlı `setValue(int)` slotu çalışacaktır.

Aslında bir sinyalin yayımlanması için herhangi bir slota bağlı olmasına gerek yoktur. C#'taki gibi nesneye ait tüm sinyaller, slotlardan bağımsız kullanıcının işlemine göre işletim sistemi tarafından yayımlanacaktır.

Buradaki örnekte göründüğü gibi kod olarak yazarak da bağlayabileceğimiz bu 2 nesne, sinyal ve slot yardımıyla kod yazmadan birbirine bağlanmış oldu. C# gibi Qt de olay güdümlü programlamaya uygun bir ortamdır ve programlamayı kolaylaştırır.

5.3. `QList` ve `QListIterator` Sınıfları

`QList` sınıfı, C#'taki `List` sınıfına benzer. Konteyner sınıfı olup belirli bir tipte oluşturulurlar.

Nesne oluşturduktan sonra `operator<<()` kullanılarak listeye ekleme yapılabilir.

```
QList<int> liste1;
liste1 << 1 << 2 << 3 << 4 << 5 << 6 << 7 << 8 << 9 << 10 << 11;
```

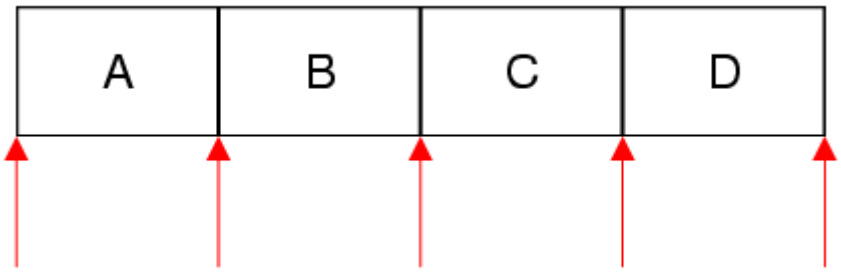
Kod Parçası 5.2. 11 elemanlı bir `QList<int>` nesnesi

`operator[]()` kullanılarak belirli bir indisteki eleman erişilebilir. `at()` methodu kullanıldığında salt okunur olarak erişilmiş olur.

```
QList<QString> liste2;
     liste2 << "Yasin" << "Turker" << "Kadir";
     for (int i = 0; i < liste2.length(); i++)
          if (liste2.at(i) == "Turker")
               cout << "Bu listede Turker " << i + 1 << ".
sirada bulunuyor." << endl;
```

Kod Parçası 5.3. Listede `QString` araması yapmak

QList nesnesinde iterasyon yapabilmek için QListIterator nesnesi oluşturulabilir. Bu nesne, liste içindeki elemanların arasında bulunan işaretçi gibi düşünülebilir. Bu iterasyon şekli Java'ya benzemektedir ve Java-stili iterasyon olarak da anılır.



Şekil 5.1. Java-stili iterasyon

Bu nesneyi kullanarak (Kod Parçası 5.2)'te oluşturulan dizide iterasyon yapabiliriz:

```
QListIterator<int> iterator(liste1);  //liste1 iterasyonu
while (iterator.hasNext())            //Sonraki eleman var mı?
     qDebug() << iterator.next();   //Sonrakini yazdır ve
sonrakini işaret et
```

Kod Parçası 5.4. QListIterator kullanarak iterasyon

Sağa doğru iterasyon yapılabildiği gibi sola doğru da yapmak mümkün fakat öncelikle sondan başlanacağının belirtilmesi gerekiyor.

```
iterator.toBack();                   //Sondan iterasyon yapılacak
while (iterator.hasPrevious())        //Önceki eleman var mı?
     qDebug() << iterator.previous(); //Öncekini yazdır ve
öncekini işaret et
```

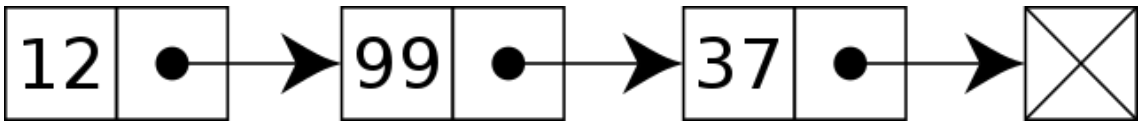
Kod Parçası 5.5. Geriye doğru iterasyon

Önceki elemanı işaret etmeden okumak için peekPrevious() methodu kullanılabilir. Aynısı sonraki eleman için de geçerlidir.

5.3.1. QList ile QLinkedList Arasındaki Farklar

`QList` nesnesinde elemanlar hafızada tek bir parça olarak bulunur, dolayısıyla hızlı bir şekilde iterasyon yapılabilir durumdalardır. Fakat yeni bir eleman yerleştirilmek istendiğinde, listenin yeni uzunluğuna göre yeni bir hafıza alanı ayrılması gerekir.

`QLinkedList` nesnesinde elemanlar hafızada tek bir parça halinde bulunmaz, her bir eleman hafızanın farklı bir alanında bulunabilir. Her eleman bir sonraki elemanın işaretçisini de tutar. Bu durumda elemanların iterasyonu diğerine göre yavaş olacaktır, zira işaretçiler hafızadaki farklı alanları işaret edecektir. Fakat yeni bir eleman eklenmek istediğinde tüm elemanlar hafızada farklı alanlarda olduğundan yeni eleman da hafızanın farklı bir alanında olacak, böylece uzun bir hafıza alanı ayrılmasına gerek kalmayacaktır.



Şekil 5.2. Linked list algoritması

5.3.2. `QMutableListIterator` Sınıfı

QListIterator kullanarak QList üzerinde iterasyon yapmak mümkün olsa da sadece okuma işlemi yapılabilmektedir. Diğer bir değişle eleman yazmak veya değiştirmek mümkün değildir. İterasyon yaparken aynı zamanda düzenleme yapmak için bu sınıfı kullanmak gerekir.

```
QMutableListIterator<QString> iterator(liste2);
while (iterator.hasNext())
     if (iterator.next() == "Turker")
          iterator.remove();
```

Kod Parçası 5.6. Listenin `QMutableListIterator` ile düzenlenmesi

5.4. QMap Sınıfı

Bu sınıf, 2 farklı veriyi bir eleman olacak şekilde listelemeye yarayan bir gruptur. Örneğin `QMap<int, QString>` nesnesi, `QString` ve `int` 2'lileri türünden veri saklamaya yarar. İlk tip, `key`; son tip, `value` olarak anılır.

```
QMap<int, QString> map;
map.insert(1, "Yasin");
map[2] = "Turker";
map.insert(3, "Kadir");
```

Kod Parçası 5.7. 3 elemanlı bir `QMap<int, QString>` nesnesi

(Kod Parçası 5.7)'de 3 elemanlı bir nesne tanımlanmıştır. İlk tip kimlik numarası, son tip isim olarak düşünülebilir. `insert()` methodu kullanılarak aynı `key`de farklı bir `value` eklenemez. Bunun için `insertMulti()` methodu kullanılır.

Bu sınıf için de iterasyon yapabilecek `QMapIterator` ve `QMutableMapIterator` sınıfları bulunur, kullanımı `QList` sınıfındaki gibidir.

5.4.1. QMap ile QHash Arasındaki Farklılıklar

Aynı işi yapan bu 2 sınıf arasında algoritmik farklar bulunur. `QHash` sınıfı içerisinde `QMap` sınıfına göre daha hızlı arama yapılabilmektedir. `QMap` nesnesinde iterasyon yaparken, elemanlar her zaman `key`e göre sıralıdır, `QHash` nesnesinde ise keyfi bir sıralama vardır.

5.5. QThread Sınıfı

Şuana kadar yazdığımız kodlar programlama mantığınca sırayla çalıştırılır.

```
for (int i = 1; i < 100; i += 2)
     qDebug() << i;
for (int i = 0; i < 100; i += 2)
     qDebug() << i;
```

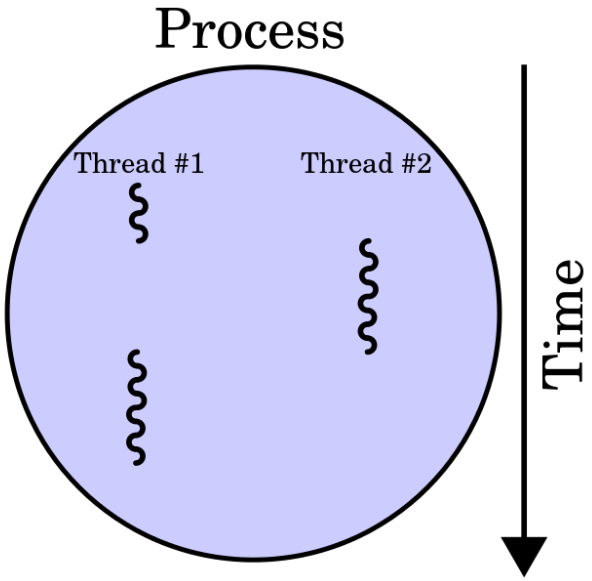
Kod Parçası 5.8. Sırayla çalıştırılacak 2 döngü

(Kod Parçası 5.8)'deki 2 döngü sırasıyla çalıştırılacaktır. İlk döngü tek sayıları basacak, son tek sayı basıldıktan sonra son döngü çift sayıları basacaktır. Bu iki döngü kısa sürede sonuçlanacağı için bize bir sorun yaratmayacaktır, fakat özellikle GUI programlarken uzun sürebilecek bazı işlemler sırasında GUI yanıt vermeyi durduracaktır, çünkü yukarıda bahsedilen programlama mantığınca bu işlem bitmeden başka bir işleme geçmeyecektir.

Bu durum antivirüs gibi karmaşık programlarda veya işletim sistemlerinde büyük bir problemdir. Zira bu durumda antivirüs bilgisayarı tararken yanıt vermeyi durduracaktır.

GUI'nin ana ipliğinde bu işlemleri yapmak yerine ana ipliği meşgul etmeyecek şekilde ondan bağımsız bir iplik oluşturulur ve bu işlemler bu iplik üzerinden yürütülürse GUI yanıt vermeye devam edecektir.

İşletim sistemi her iplik için belirli bir süre ayırır. Örneğin 2 ipliği olan bir işlemde ilk iplik belirli bir süre çalışır, kalınan yer saklanır, diğer ipliğe geçilir ve bu şekilde döngü devam eder.



Şekil 5.3. Çok iplikli bir işlemde ipliklerin zamana göre çalışmaları

QThread sınıfı, iplik işlemlerini Qt ortamında mümkün kılan sınıftır. Bu sınıf kullanmak için birçok yol vardır. Burada en sık kullanılan yöntemden bahsedilecektir.

5.5.1. İşçi Nesnelerini İpliğe Taşımak

```
#pragma once
#include <QObject>

class Worker : public QObject
{
     Q_OBJECT
          QThread *thread;           //İplik
public:
     Worker(unsigned long delay);    //Oluşturucu method
     void m_Start();                 //İpliği başlatan method
private:
     unsigned long delay;            //Gecikmeyi tutan değişken
public slots:
     void m_process();               //İşin bulunduğu slot
signals:
     void m_finished();    //İş bittiğinde yayımlanacak sinyal
};
```

Kod Parçası 5.9. Worker.h

İşi yapacak bir sınıf tanımlanıp bu sınıfta oluşturucu method, başlatma methodu, gecikmeyi tutacak değişken, işin bulunduğu slot ve iş bittiğinde yayımlanacak bitti sinyali; işin taşınacağı iplikle beraber tanımlanır.

Header dosyası düzenlendikten sonra kaynak dosyasının düzenlenmesine geçilir. İplik çalışmaya başlamadan önce yapılması gereken ayarların yapılabilmesi için oluşturucu method düzenlenir. Gecikme ayarlanır, iplik oluşturulduktan sonra nesne, ipliğe taşınır. Taşındıktan sonra ipliğin başlama sinyali nesnenin slotuna, nesnenin bitti sinyali ipliğin çıkış slotuna bağlanır.

Bu ayarlar yapıldıktan sonra yapılacak iş başlamaya hazırdır.

İşi simüle etmek amacıyla döngüyle sayı yazdırılsın. `main.cpp` dosyasında bu sınıftan 2 nesne türetilsin.

```
#include "Worker.h"
#include <qdebug.h>
#include <qthread.h>

Worker::Worker(unsigned long delay)
{
	this->delay = delay;				//Gecikme alınır.
	thread = new QThread();			//İplik referansı alınır.
	this->moveToThread(thread);		//Nesne ipliğe taşınır.
	connect(thread, &QThread::started, this,
&Worker::m_process);
	connect(this, &Worker::m_finished, thread, &QThread::quit);
	connect(this, &Worker::m_finished, this,
&Worker::deleteLater);
	connect(thread, &QThread::finished, thread,
&QThread::deleteLater);		//Sinyaller ve slotlar bağlanır.
	qDebug() << "Worker OK";
}
void Worker::m_Start()
{
	qDebug() << "Thread is starting";
	thread->start();				//İplik başlar.
	qDebug() << "Thread was started";
}
void Worker::m_process()
{
	for (int i = 0; i < 100; i++)
	{
		qDebug() << thread->currentThreadId() << i;
		thread->msleep(delay);
	}
	qDebug() << "Signal is emitting OK";
	emit m_finished();	//İş bittiğinde sinyal yayımlanır.
}
void Worker::m_End()
{
	qDebug() << "Thread OK";
}
```

Kod Parçası 5.10. `Worker.cpp`

```
#include <QtCore/QCoreApplication>
#include <qdebug.h>
#include <qthread.h>
#include "Worker.h"

int main(int argc, char *argv[])
{
	QCoreApplication a(argc, argv);
	Worker *worker1 = new Worker(100);	//1. işçi
	worker1->m_Start();			//1. işçi başlar
	Worker *worker2 = new Worker(200);	//2. işçi
	worker2->m_Start();			//2. işçi başlar
	qDebug() << "Main OK";
	return a.exec();
}
```

Kod Parçası 5.11. `main.cpp`

İki işte de 100 sayı yazdırılmasına rağmen ilk işte sayı başına `100ms` sürerken, son işte sayı başına `200ms` sürmektedir. Buna göre ilk işin bitmesi için `10s` gerekliyken son işin bitmesi için `20s` gerekir. İplik kullanılmadan yapılan bir işlem toplamda `30s` sürecekken 2+1 iplikli işlemde iplikler eş zamanlı olduğundan `20s` sürecektir.