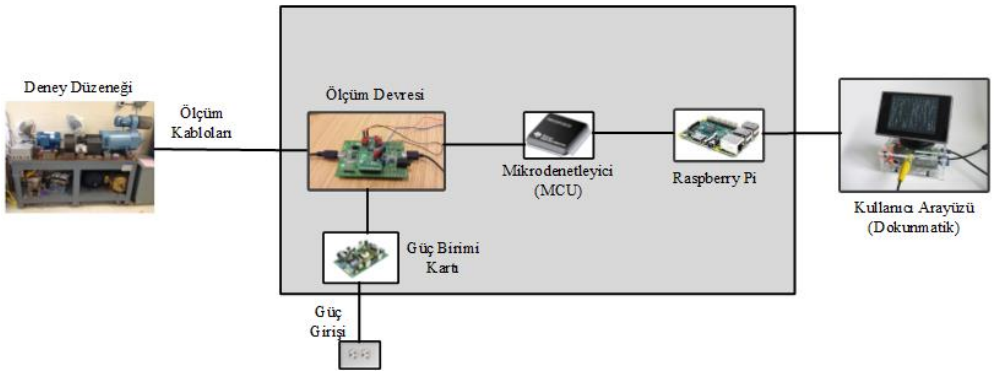
Ek.3. Projedeki Bilimsel Gelişmeler ve Sonuçlar

Proje kapsamında geliştirilen veri toplama, izleme ve yönetimi (data acquisition, monitoring and management) sistemi, enerji sistemlerinden çok sayıda, farklı seviyelerde ve farklı tipte (AC/DC) akım ve gerilimin ölçülmesini, elektrik makinalarında yer alan sensörlerden hız ve moment bilgilerinin okunmasını ve aynı zamanda elektrik makinalarının kontrollü bir şekilde yüklenmesini sağlayan bir sistemdir. Sistemin temel özellikleri şu şekilde sıralanabilir:

- Farklı seviyelerde ve tiplerde akım, gerilim, moment ve hız değişkenlerinin ayarlanabilir hassasiyette ölçülebilmesi
- Ham verilerin ölçümüne ek olarak belli başlı değerlerin hesaplanabilmesi (rms ve ortalama değerler, frekans, aktif güç, reaktif güç, güç faktörü, verim)
- Veri ölçümüne ek olarak elektrik motorlarının kullanıcı kontrolünde yüklenebilmesi
- Harici bir bilgisayar, monitör vb. kullanımına gerek duymadan çalışabilme
- Verilerin gerçek zamanlı izlenmesi
- Verilerin kaydedilebilmesi ve depolanması
- Verilerin kablolu ve kablosuz olarak aktarılması
- Verilerin kullanıcı tarafından işlenmesi/yönetilmesi
- Arayüzün kullanıcı tarafından konfigüre edilebilir olması

Geliştirilen sistemin blok şeması Şekil 1'de görülebilir.

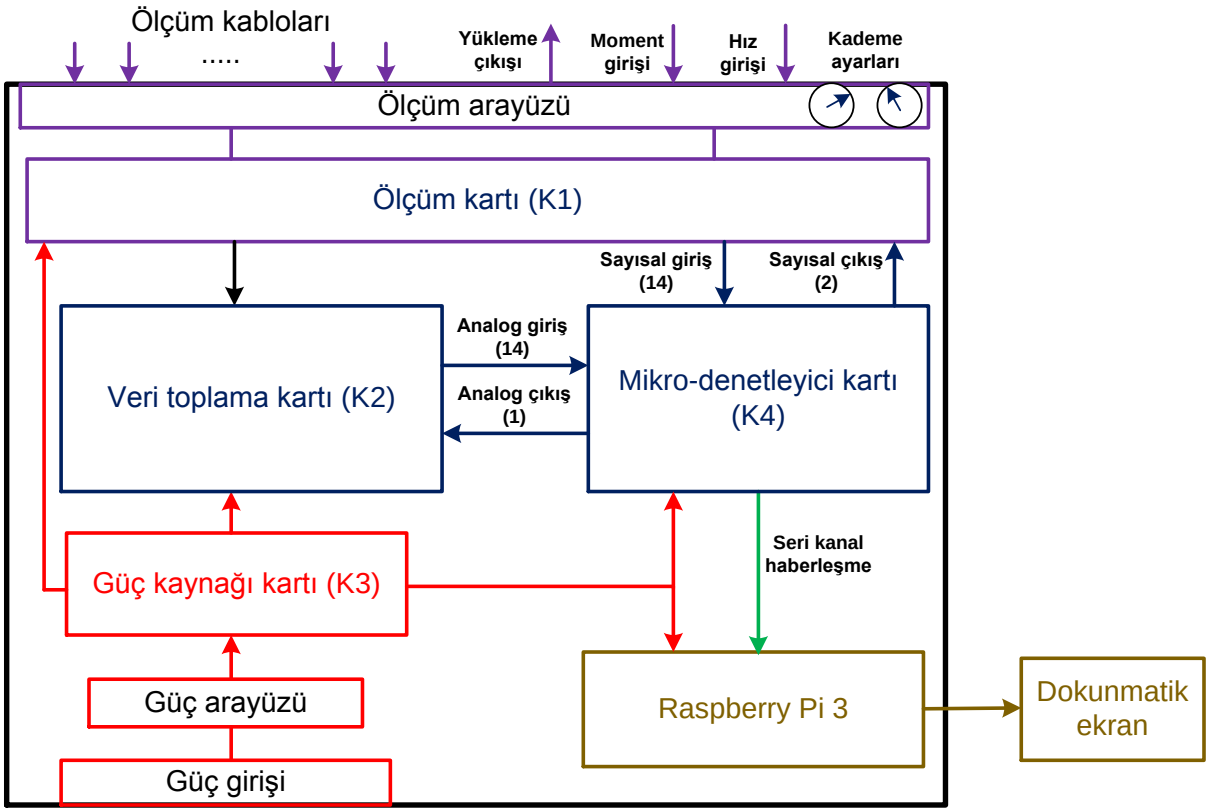


Şekil 1. Geliştirilen sistemin blok şeması

Sistem temelde 4 ana bileşenden oluşmaktadır:

- Ölçüm birimi
- Veri toplama ve işleme birimi
- Bilgisayar ve dokunmatik arayüz birimi
- Güç birimi

Ayrıca sistemin mimari tasarımı şeması Şekil 2'de gösterilmiştir.



Şekil 2. Veri İzleme ve Yönetim Sistemi Mimari Tasarımı Şeması

Ölçüm Birimi

Bu birimde yer alan ölçüm elemanlarının kapsamı şu şekildedir:

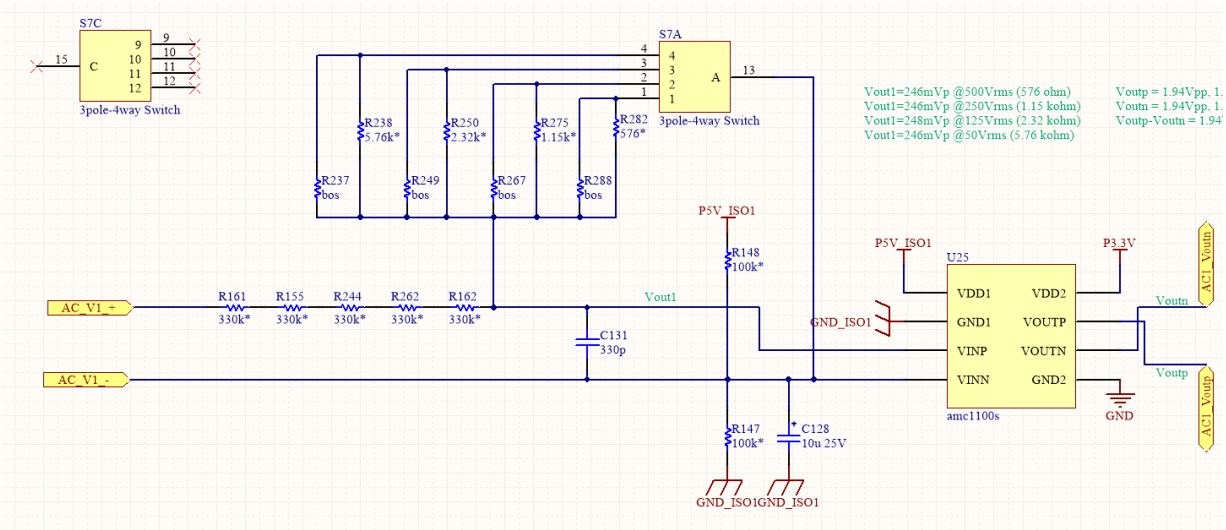
- 4xAC Gerilim (50V, 125V, 250V, 500V)
- 4xAC Akım (5A, 10A, 25A, 50A)
- 2xDC Gerilim (100V, 250V, 500V, 1000V)
- 2xDC Akım (5A, 10A, 25A, 50A)
- 1xMoment (50Nm)
- 1xHız (1800rpm, 3600rpm)
- 1xMoment yükleme çıkışı (0-10V analog sinyal)

Tüm ölçümler kullanıcı tarafından ayarlanacak şekilde kademeli olarak tasarlanmıştır. Bu kademeler ile ölçüm hassasiyeti sürekli olarak en iyi seviyede tutulmaktadır. Ölçüm birimi çıkışında, elde edilen ham analog sinyaller veri toplama kartına işlenmek üzere iletilmektedir.

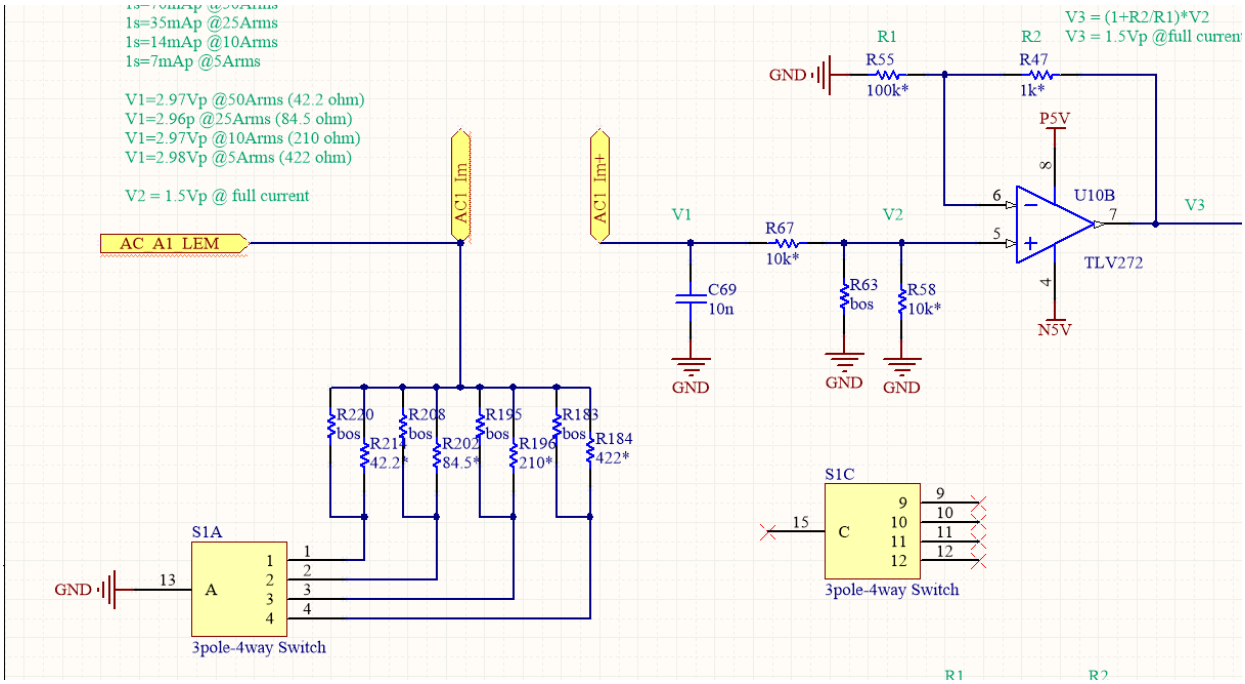
Çoklu ölçüm yapan bir cihazda ölçümlerinden birbirinden izole olması gerekmektedir. Örneğin kullanıcı, devrenin farklı noktalarında yer alan ve eksi uçları (ground)

birbirinden ayrı olan iki gerilimi ölçtüğünde, eğer ölçümler izole edilmemişse, cihaza ölçüm kabloları bağlandığında bu iki nokta ortaklanacak ve devreye zarar verecektir. Ayrıca, ölçümlerin, cihazın düşük gerilim sinyal barındıran diğer kartlarından da izole edilmesi gerekmektedir. Bu da cihazın entegre devrelerinin, mikro denetleyicinin ve bilgisayarın korunmasını sağlar. İzolasyon gereksinimini sağlayacak şekilde, gerilim ve akım ölçümleri için LA-55P adlı LEM ve AMC1100 adlı izolasyon yükselteçleri (isolation amplifier) kullanılmıştır. Kullanıcı tarafından ölçümü ölçeklendirmesini sağlayacak eleman olarak 4 kutuplu ve 3 faz döner anahtarlar kullanılmıştır. Bu anahtarlar, kart üzerinde rezistif gerilim bölücülerin ayarını değiştirecek şekilde ayarlanmıştır. Bunun amacı, ölçüm elemanlarının hassasiyetlerini arttırmaktır. Bu kartta ayrıca, mikro denetleyiciye kullanıcının hangi ölçeği seçtiğini bilgisini gönderecek multiplexer adı verilen entegreler kullanılmıştır.

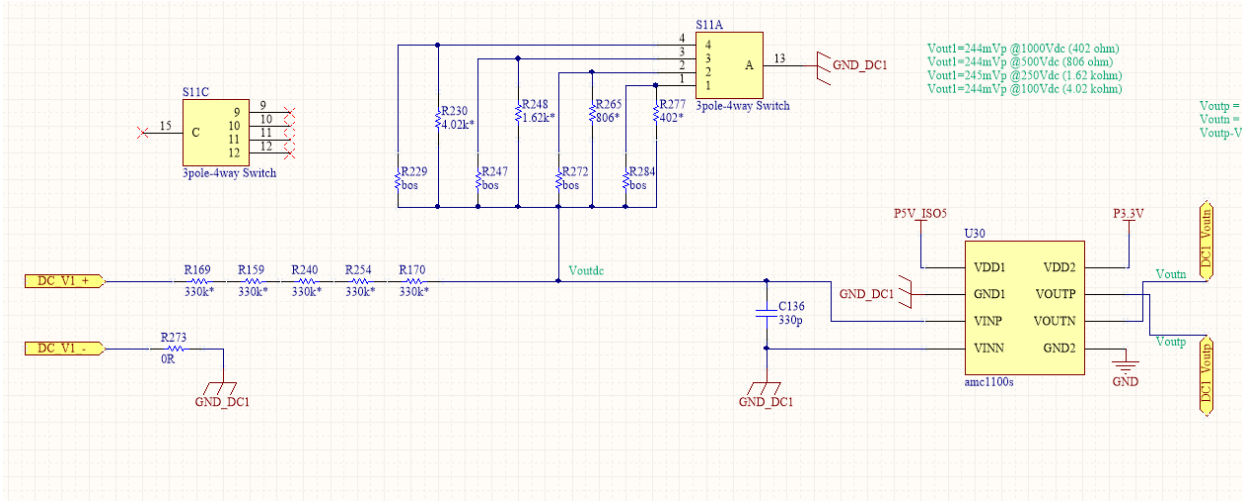
AC gerilim ölçümüne ait devre şeması Şekil 3'te, AC akım ölçümüne ait devre şeması Şekil 4'te, DC gerilim ölçümüne ait devre şeması Şekil 5'te, DC akım ölçümüne ait devre şeması Şekil 6'da, bir multiplexer devre şeması Şekil 7'de gösterilmiştir. ayrıca kullanılan bazı kritik elemanlar Şekil 8'de görülebilir.



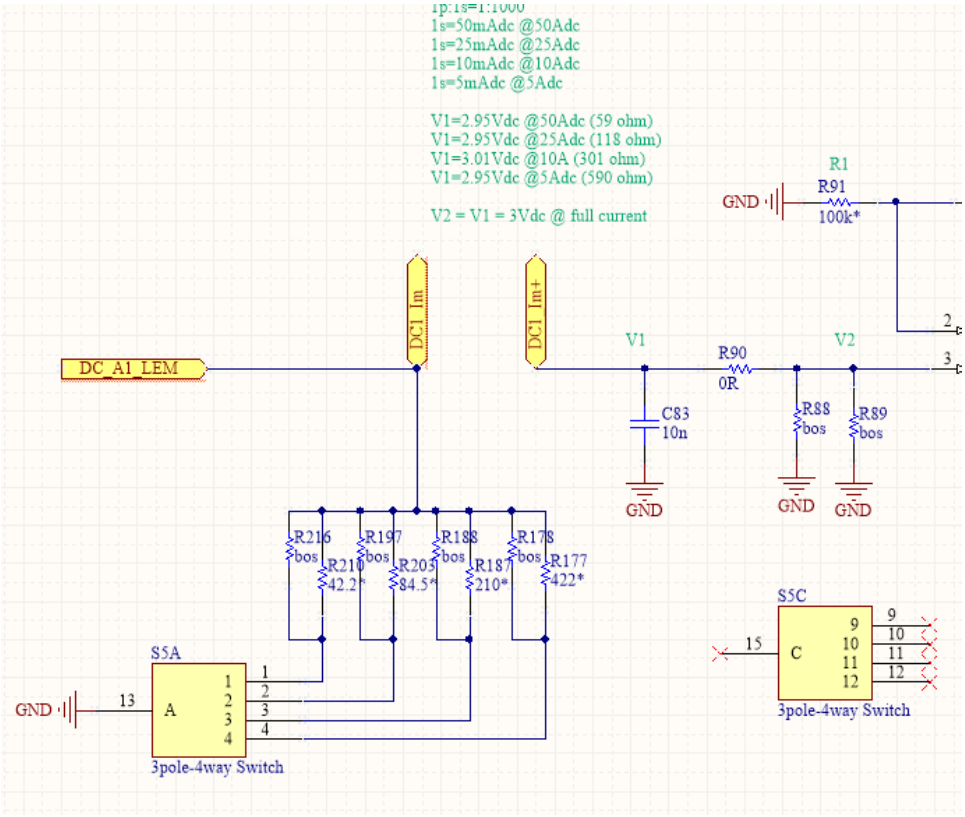
Şekil 3: AC gerilim ölçüm devresi



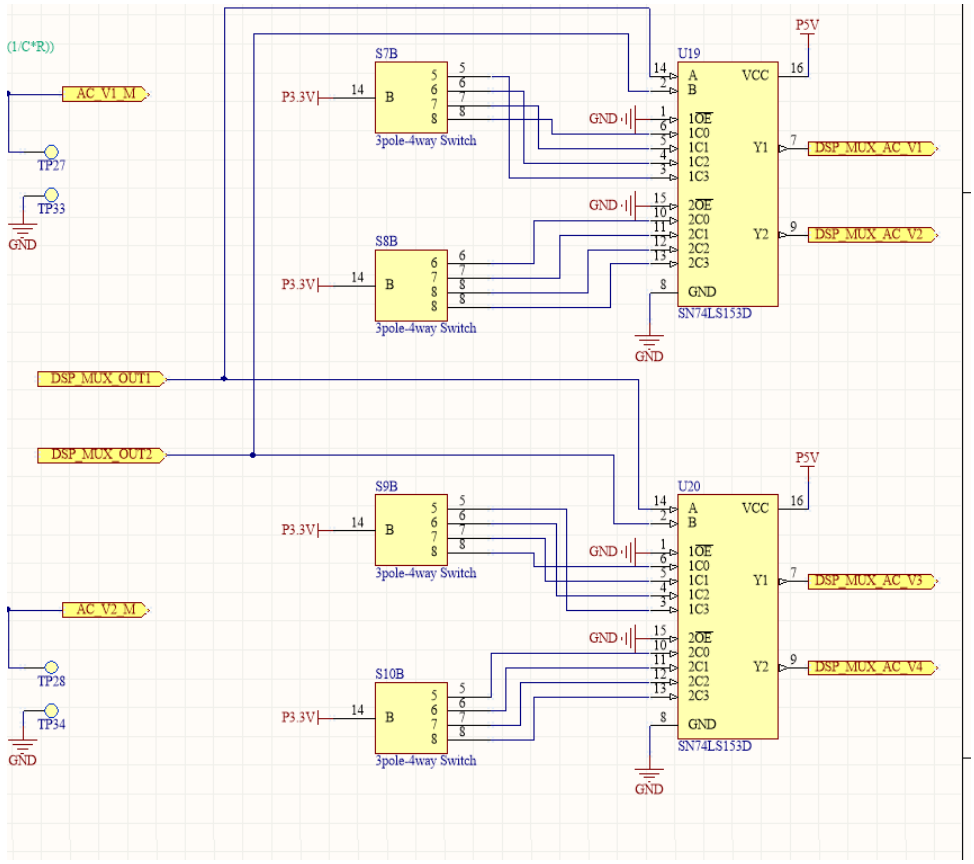
Şekil 4: AC akım ölçüm devresi



Şekil 5: DC gerilim ölçüm devresi



Şekil 6: DC akım ölçüm devresi



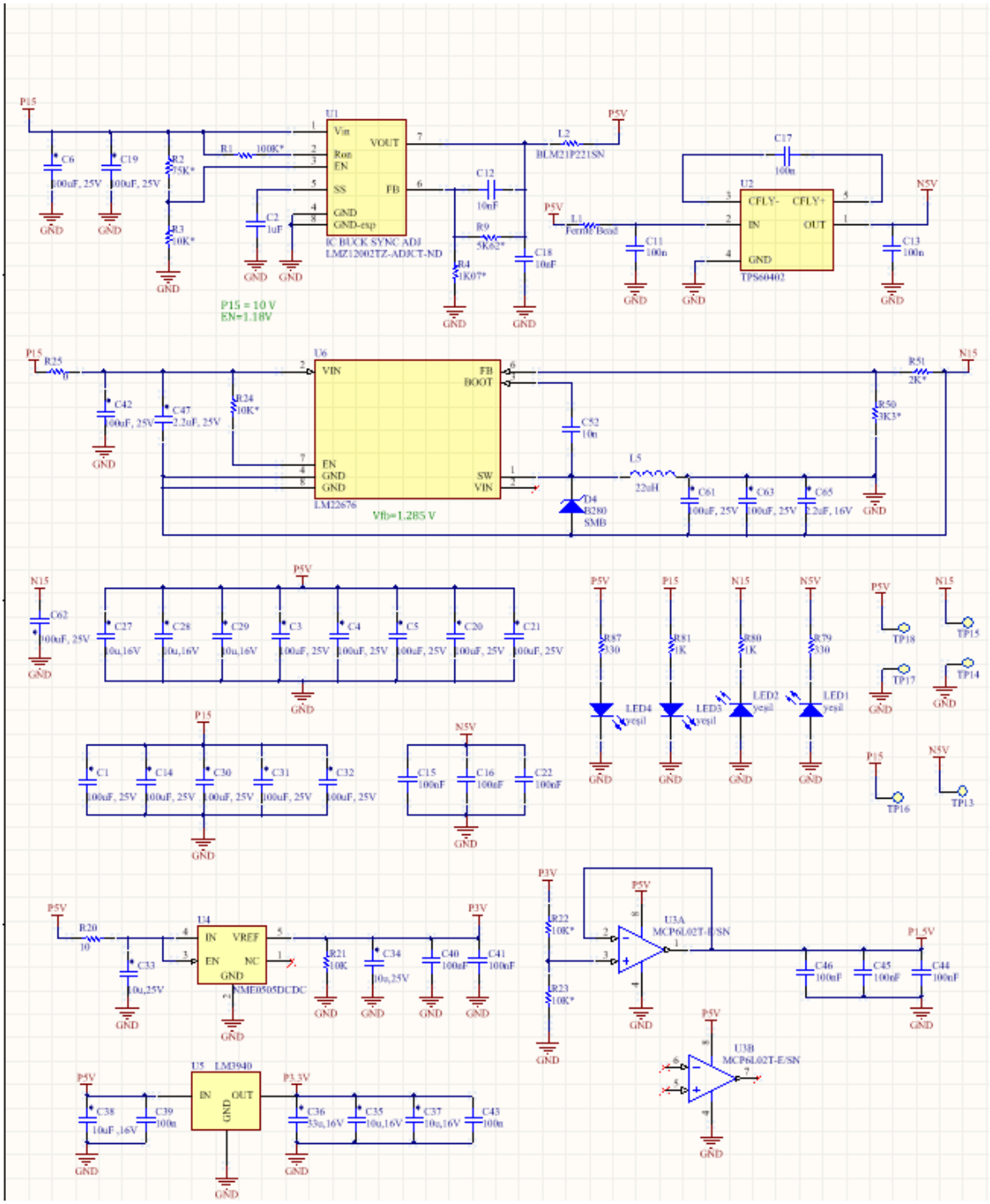
Şekil 7: Multiplexer devresi

Akım LEM'I İzolasyon Yükselteci Döner Anahtar

Şekil 8: Ölçüm devresinde kullanılan kritik elamanlar

Veri Toplama Birimi

Bu birimde yer alan baskı devre kartında, sensörlerden gelen ham veriler mikro-denetleyici tarafından işlenebilecek standart seviyelere (0-3V aralık) dönüştürülmektedir. Bu kart ayrıca diğer kartların düşük seviye besleme gerilimlerini de sağlamaktadır. Güç çevirici devreleri Şekil 9'da gösterilmiştir. Kartta +15V giriş geriliminden (P15), +5V (P5), -5V (N5), -15V (N15), 3.3V(P3V3) ve 1.5V(Vref) gerilim seviyeleri üretilmektedir. P5 gerilimi hem entegre devreleri beslemekte, hem de DSP ve Raspberry Pi'a besleme sağlamaktadır. P15 ve N15 gerilimleri genellikle akım LEM'lerini beslemek için kullanılır. P3V3 DSP seviyesi beslemelerde kullanılmakta olup, 1.5V ise pozitif ve negatif parçaları bulunan (örneğin AC gerilim ve akımlar) sinyallere offset eklemek amacıyla kullanılmaktadır. Offset ekleme gerekliliği, mantığı ve devresi, Texas Instruments firmasının yayınladığı veri sayfasında Şekil 10'daki gibi anlatılmaktadır. Bunun yanında, izole ölçümler için kullanılan izole güç kaynağı devreleri de Şekil 11'de görülebilir. AC gerilim ölçümlerinde, sinyale offset eklemek amacıyla toplayıcı yükselteç (summing amplifier) kullanılmıştır (Şekil 12). DC gerilim ölçümünde ise, izolasyon yükseltecinin çıkışında üretilen pozitif ve negatif birbirine zıt iki sinyalin farkının alınması amacıyla fark yükselteci (difference amplifier) kullanılmıştır (Şekil 13). Hız ve moment sinyal işleme devreleri Şekil 14'te, moment yükleme için kullanılan sayısal/analog dönüştürücü (DAC) devresi ise Şekil 15'te görülebilir.

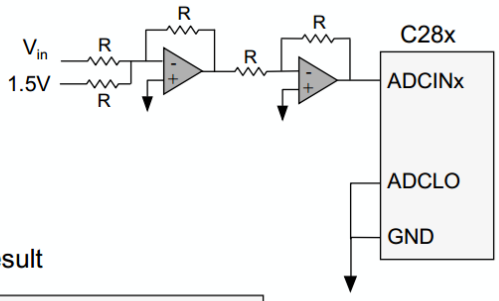


Şekil 9: Güç çevirici devreleri

How Can We Handle Signed Input Voltages?

Example: -1.5 V ≤ V_{in} ≤ +1.5 V

1) Add 1.5 volts to the analog input

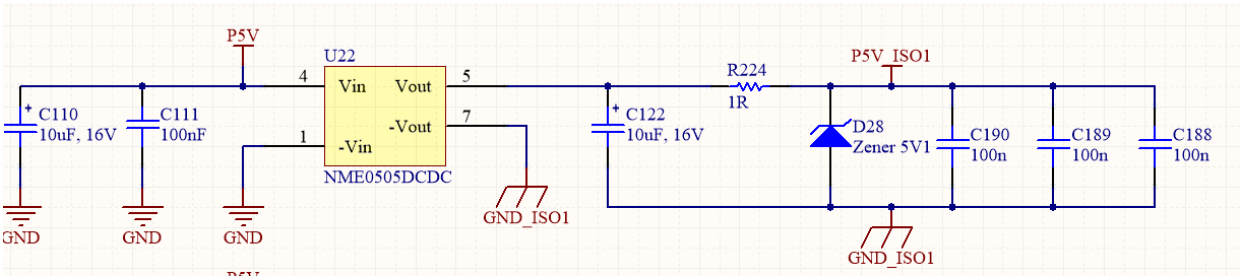


2) Subtract "1.5" from the digital result

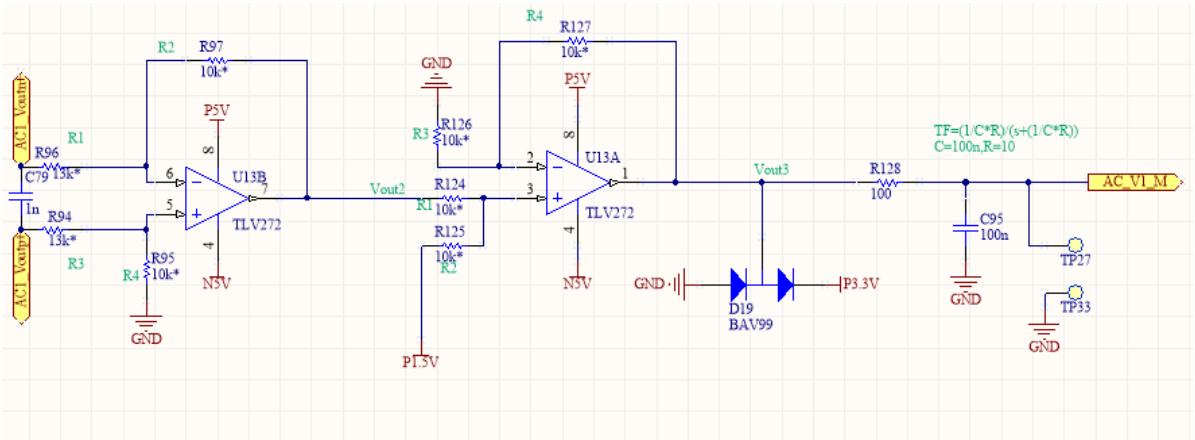
```
#include "DSP2833x_Device.h"
#define  offset  0x07FF
void main(void)
{
   int16 value;            // signed

   value = AdcMirror.ADCRESULT0 - offset;
}
```

8 - 18

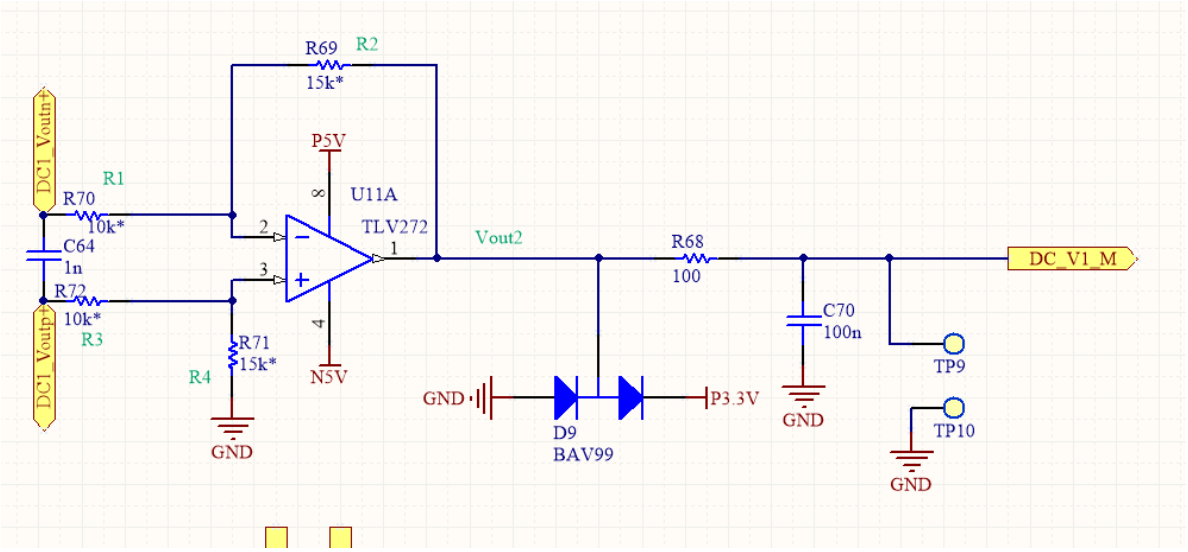
Şekil 10: DSP'de offset ekleme mantığı



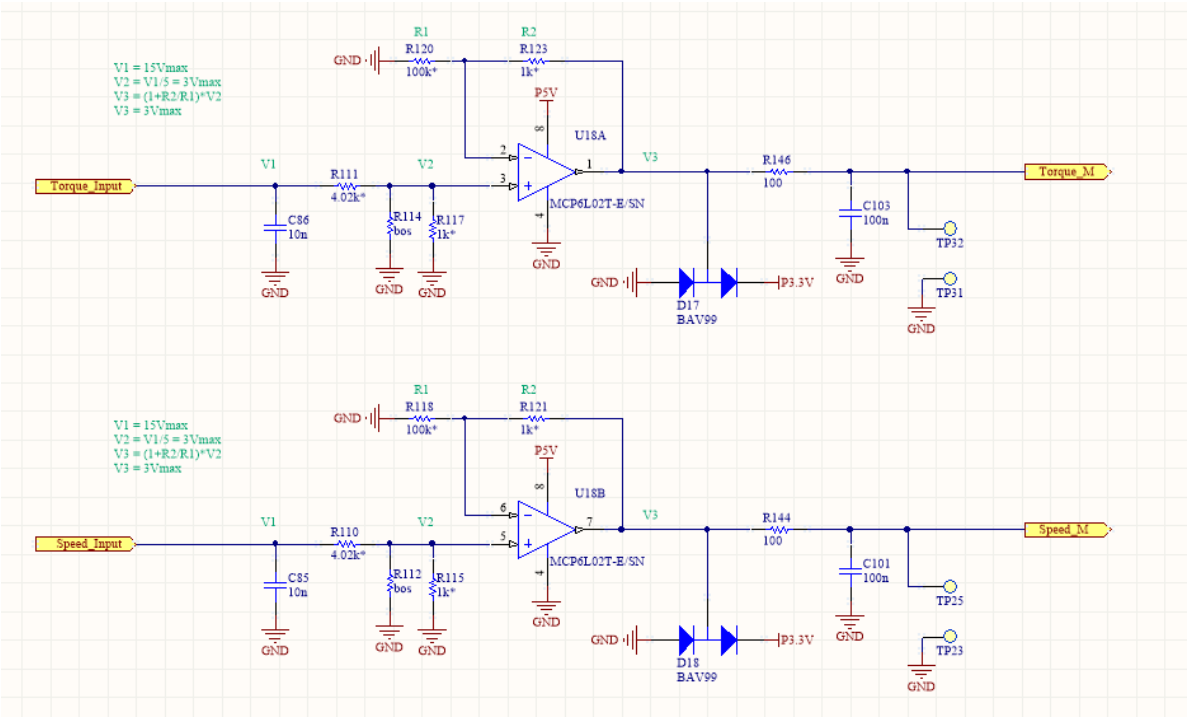
Şekil 11: İzole güç kaynağı devresi



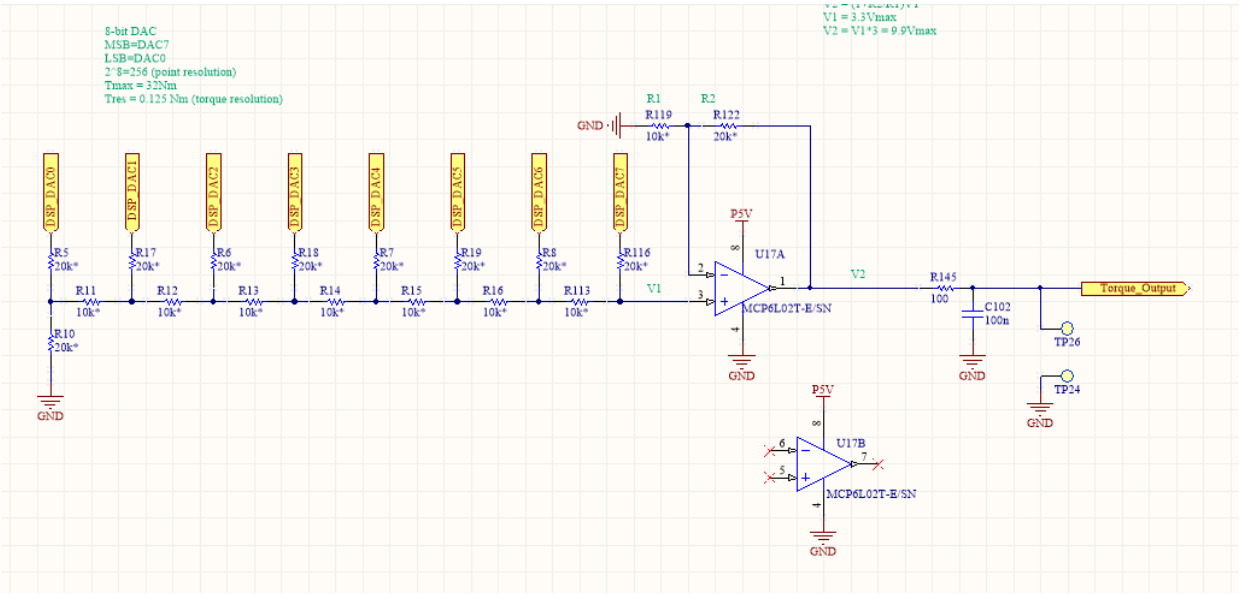
Şekil 12: Toplayıcı yükselteç devresi (summing amplifier)



Şekil 13: Fark yükselteci devresi (difference amplifier)



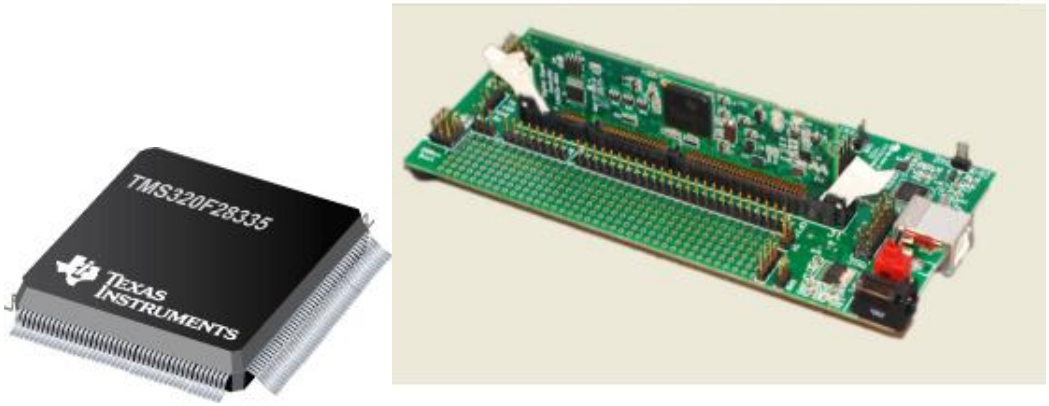
Şekil 14: Moment ve hız sinyal işleme devreleri



Şekil 15: Moment yükleme DAC devresi

Veri İşleme Birimi

Bu birimde mikro-denetleyici olarak Texas Instruments firmasına ait TMS320F28335 adlı DSP kullanılmıştır ve teknik özellikleri Çizelge 1'de, resmi ve projenin ilk versiyonunda kullanılan geliştirme kiti (evaluation modüle, EVM) Şekil 15'te görülebilir. Buna ek olarak bu birim için, işlemci entegresinin kullanıldığı bir DSP kartı da tasarlanmıştır. Bu karta ait şematik dosyalar Şekil 17-21'de görülebilir. Şekil 17'de DSP temel bağlantıları, Şekil 18'de DSP giriş çıkış arayüzü, Şekil 19'da DSP güç kaynağı, Şekil 20'de ise analog arayüz görülebilir.

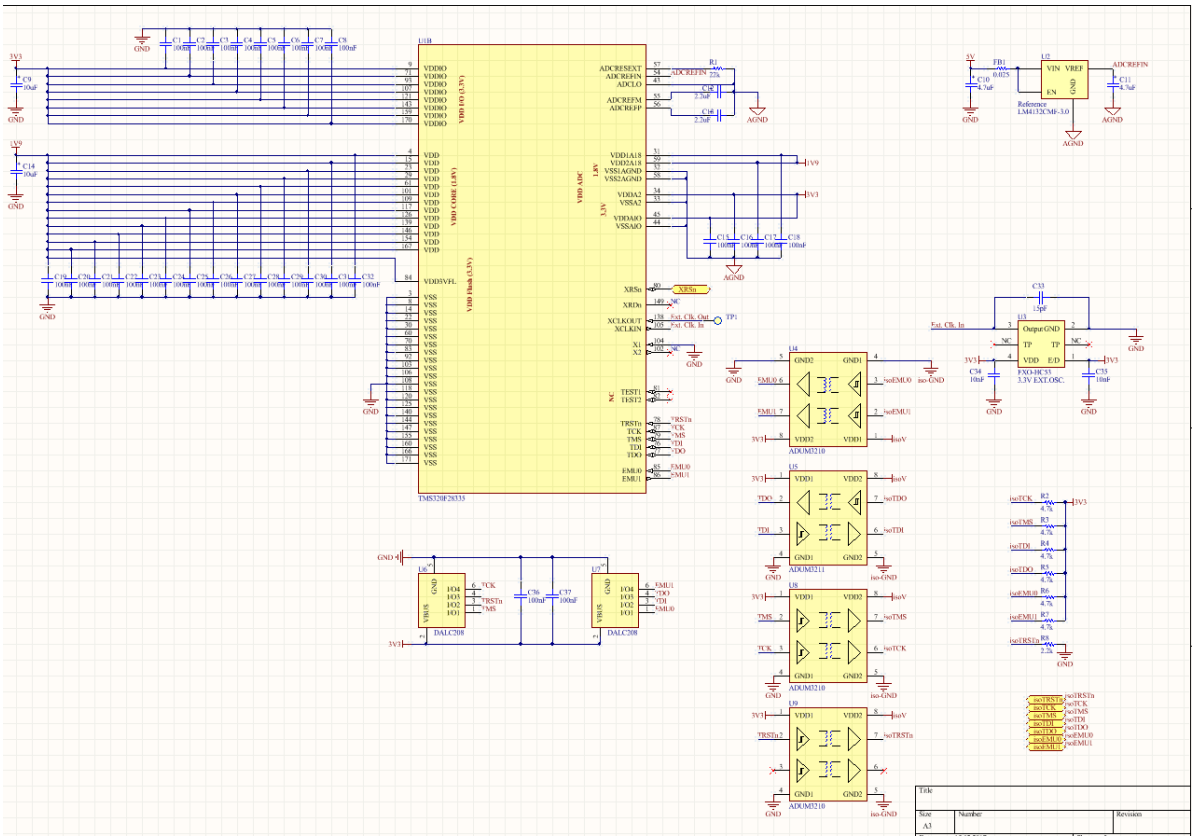


Şekil 16: Kullanılan DSP ve geliştirme kiti

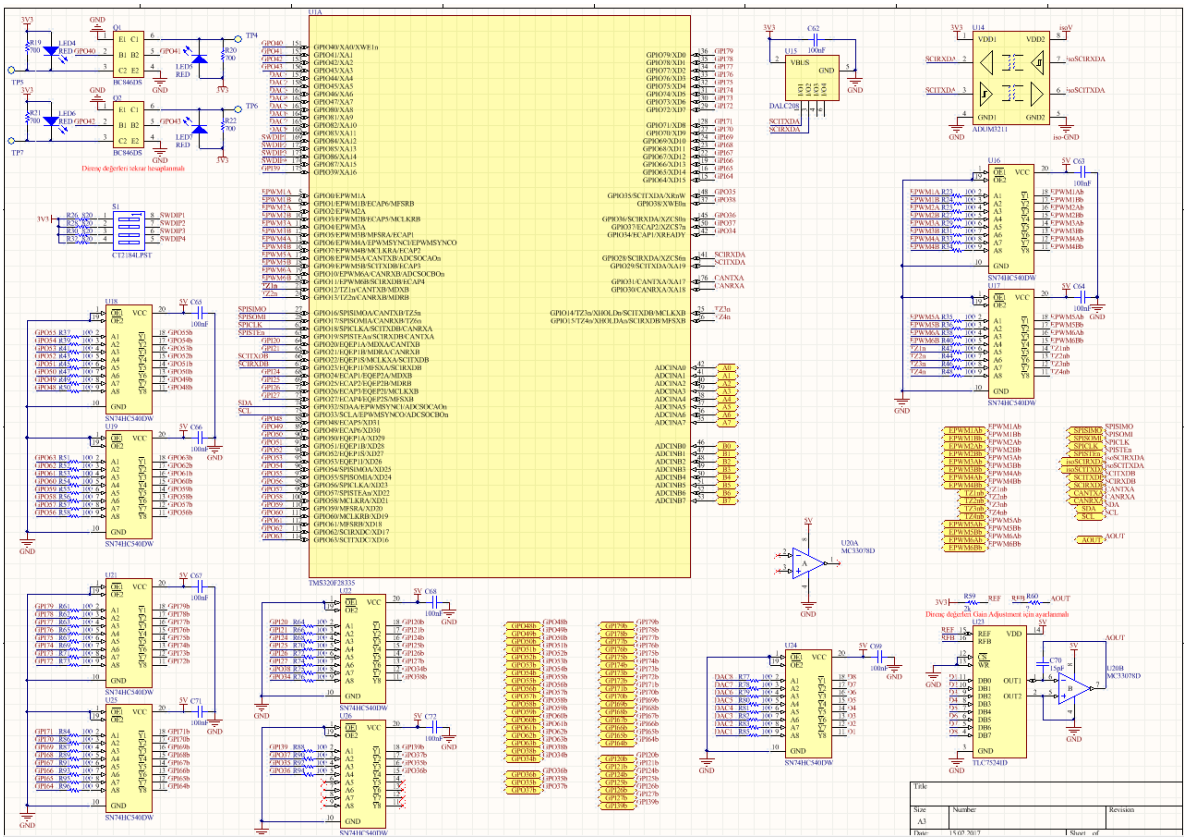
Çizelge 1: TMS320F28335 DSP teknik özellikleri

Clock frekansı	150 MHz
Güç gereksinimleri	3.3V (I/O) ve 1.8V (core)
CPU	32 bit
Hafıza	256Kx16 Flash, 34Kx16 SARAM
PWM	18 kanal
CAN	2 modül
SPI	3 modül
ADC	12-bit, 16 kanal
GPIO	88 kanal

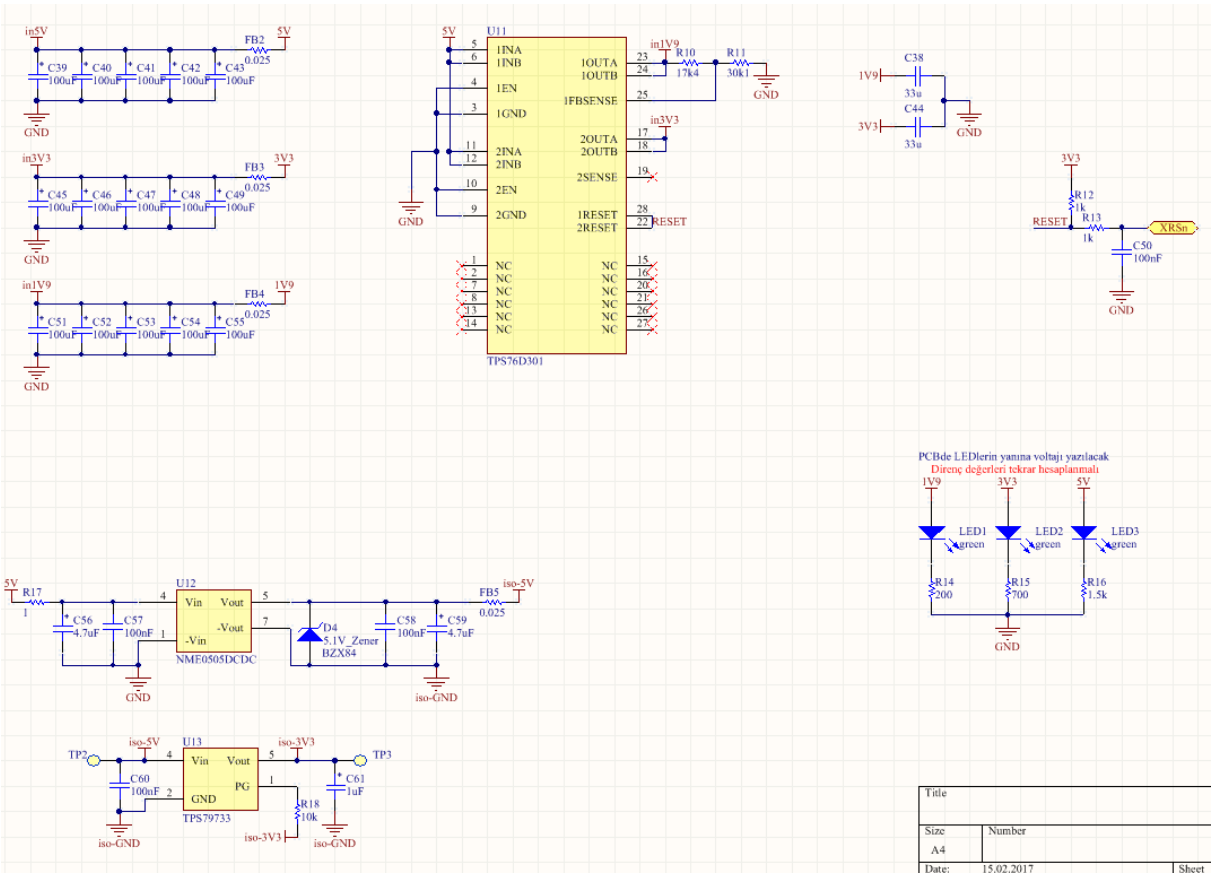
Mikro denetleyicinin temel görevleri, gelen tüm analog sinyalleri sayısala çevirmek (ADC), bilgisayardan gelen moment yükleme bilgisini 8-bit sayısal sinyale çevirmek (DAC), gelen ham verilerden belirli hesaplamalar elde etmek (rms, güç faktörü, aktif güç, verim vb.), ve tüm ham veya işlenmiş verileri daha önceden belirlenen bir frekansta paketlemek ve gerçek zamanlı olarak bilgisayara göndermektir. Bu sayede ölçümler senkronlanabilmektedir ve cihaz hem daha uzun vadeli verileri kullanıcıya gösterebilecekken (örneğin bir gerilimin RMS değerinin 1 saniye aralıklarla değişimi), hem de bir osiloskop gibi çalışabilecek ve örneğin 50 Hz'lik şebeke gerilimini sinüs dalga şekli halinde kullanıcıya gösterebilecektir. Gerçek zamanlı çalışma ve yüksek frekansta veri transferinin bir diğer avantajı da, istenirse FFT fonksiyonu kullanılarak ölçülen tüm gerilim ve akımların belirli bir frekansa kadar harmonik içeriği hesaplanabilecek ve görüntülenebilecektir.



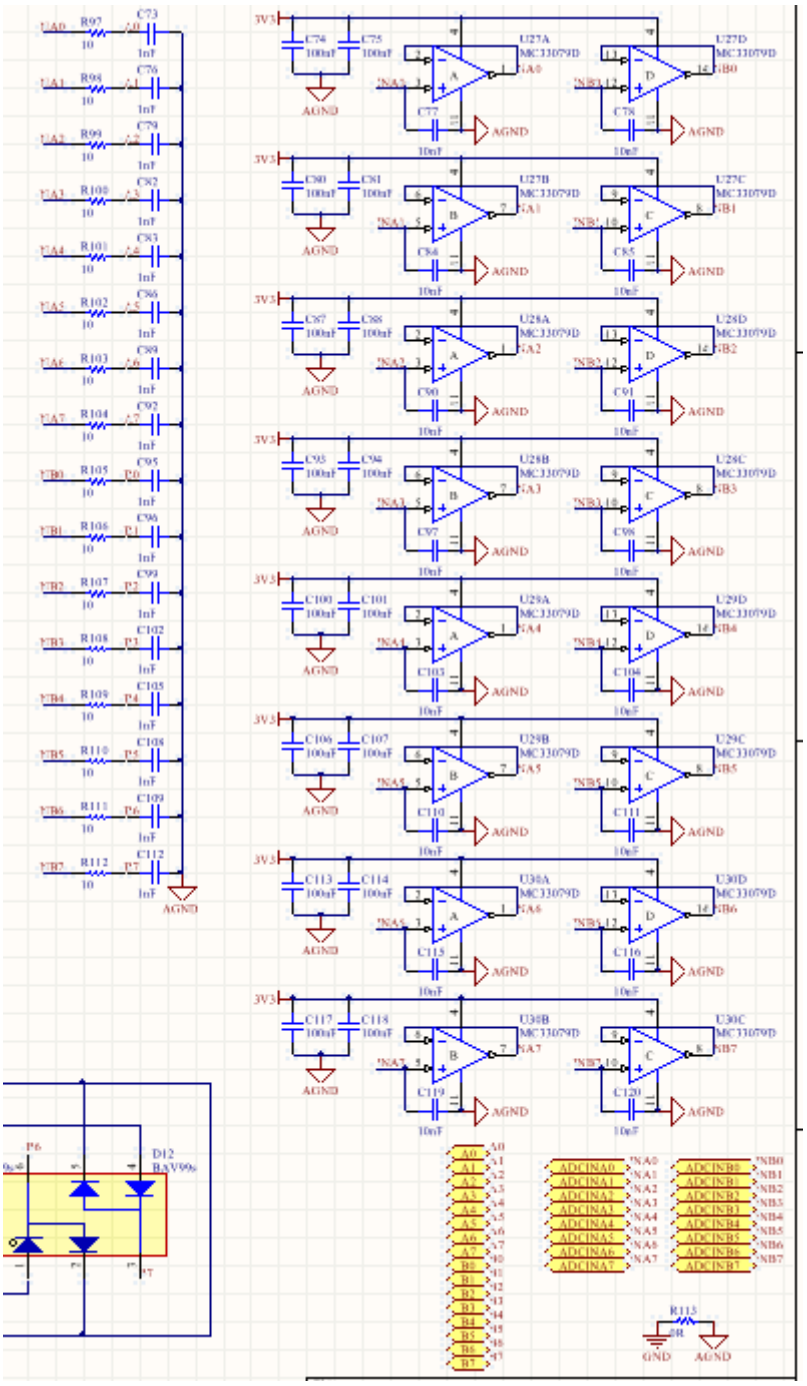
Şekil 17: DSP temel bağlantıları



Şekil 18: DSP giriş çıkışları



Şekil 19: DSP güç kaynağı



Şekil 20: DSP analog arayüzü

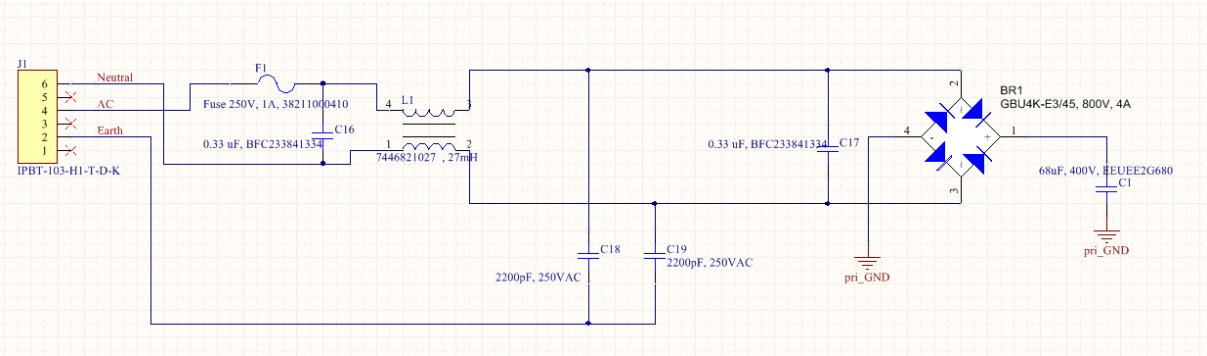
Güç Kaynağı Birimi

Geliştirilen cihazın doğrudan 230V elektrik şebekesinden beslenmesini sağlamak amacıyla bir güç kaynağı kartı tasarlanmış ve üretilmiştir. Güç kaynağı birimi, 130V – 260V aralığında tek faz AC gerilimi 15V besleme gerilimine dönüştürmektedir. Nominal çıkış gücü 60W olarak tasarlanmıştır. Cihazın içerisinde yer alan birimlerin besleme gerilimi ve güç ihtiyaçları Çeizelge 2'de görülebilir.

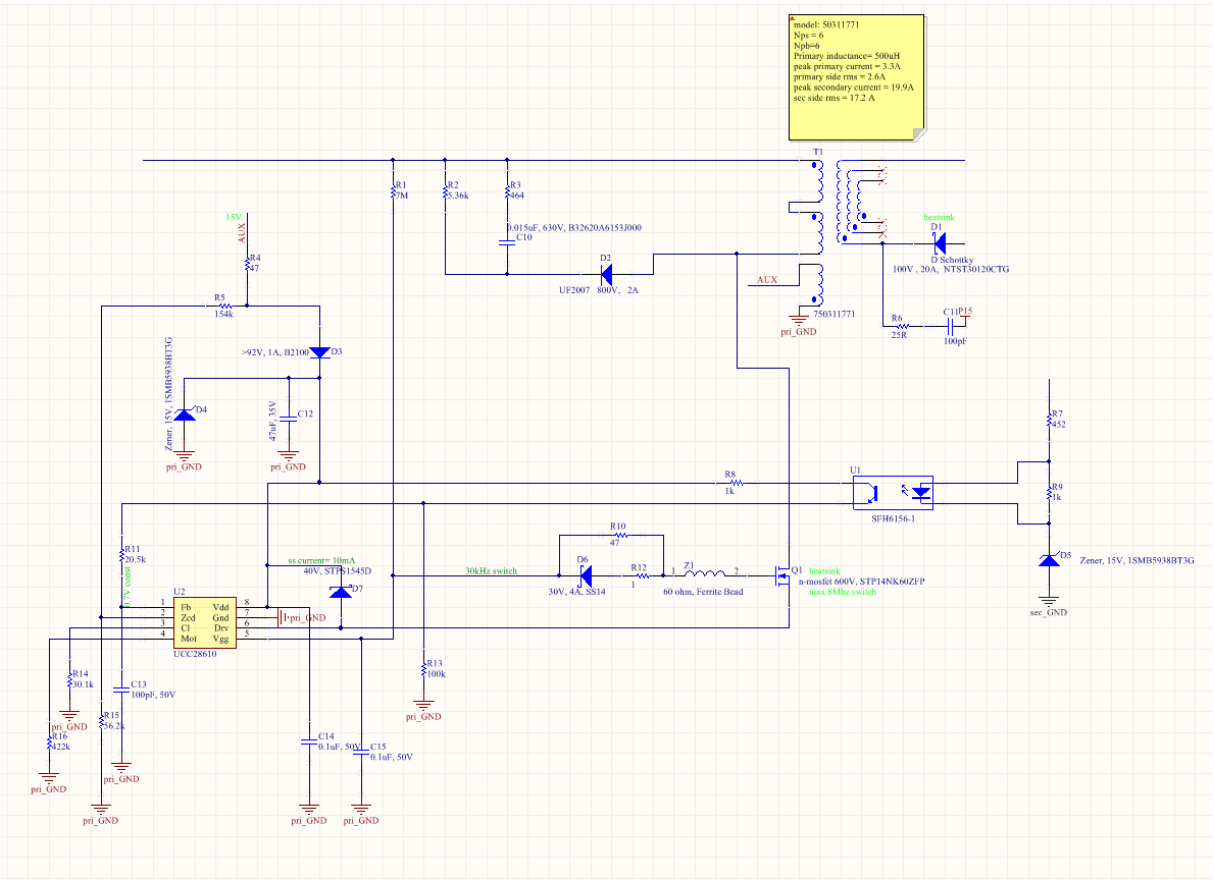
Çizelge 2: Geliştirilen cihazın parçalarının besleme gerilimi ve güç ihtiyaçları

Parça	Gerilim	Güç
Ön panel	5 V	3 W
Ölçüm kartı (K1)	15V	15 W
Veri toplama kartı (K2)	15 V	10 W
Mikro-denetleyici kartı (K4)	5 V	6 W
Raspberry Pi 3	5 V	5 W
Dokunmatik ekran	5 V	6 W
Kullanıcı çıkışı	5 V	5 W
TOPLAM		49 W

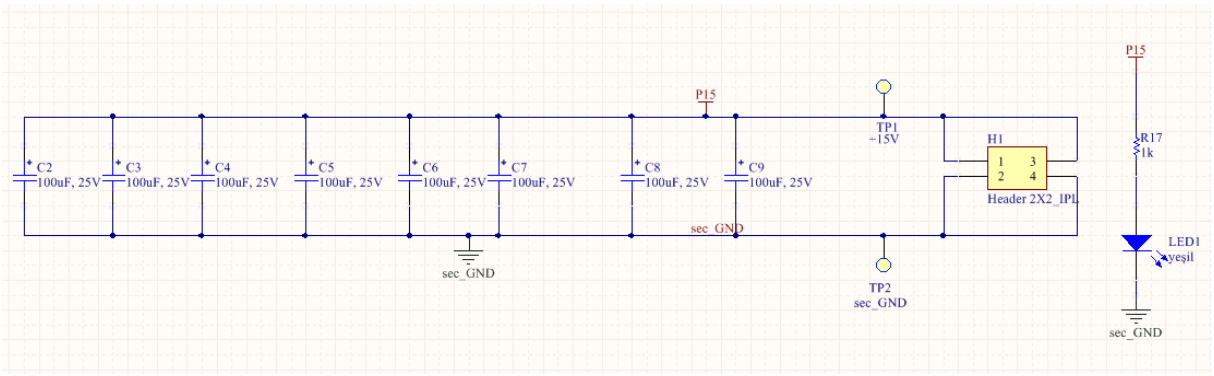
Güç kartında yer alan temel elemanlar, AC giriş fitreleri, AC/DC doğrultucu çevirgeç, flyback çevirici transformatörü, flyback denetleyici devresi, DC/DC çevirici güç elemanları ve DC çıkış filtreleridir. Giriş devresi Şekil 21'de, flyback dönüştürücü devresi Şekil 22'de, çıkış devresi ise Şekil 23'te gösterilmiştir. AC filtresi olarak standart kondansatör filtreleri ve bir de EMI filtresi kullanılmıştır. Flyback dönüştürücüde kullanılan denetleyici entegresi ve transformatör Şekil 24'te görülebilir.



Şekil 21: Güç kaynağı kartı giriş devresi



Şekil 22: Güç kaynağı kartı flyback dönüştürücü devresi



Şekil 23: Güç kaynağı kartı çıkış devresi

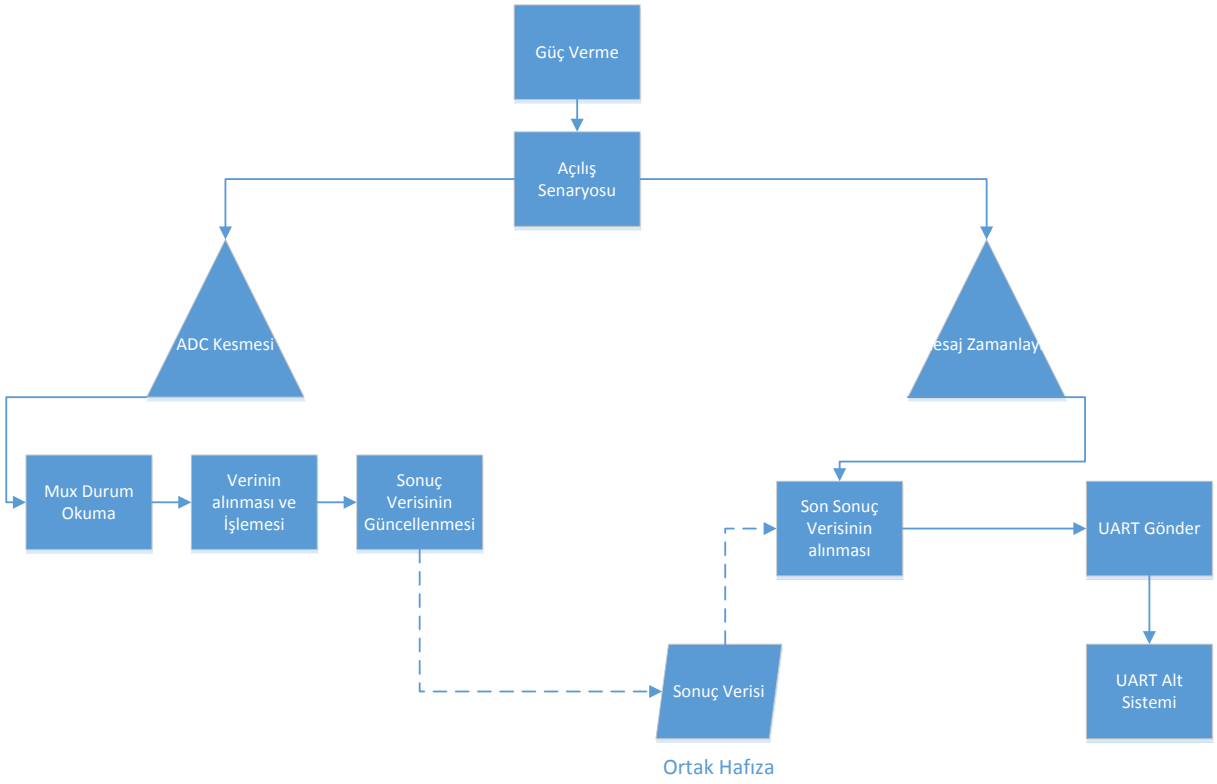
Şekil 24: Güç kaynağı kartı filtre, flyback denetleyici ve transformatör

Mikro-denetleyici Gömülü Yazılımı

İşlemci gömülü yazılımının temel işlevleri şu şekildedir:

- Analog sinyallerin ölçülmesi
- Kullanıcı ölçeklendirmesine göre ölçülen verilerin düzenlenmesi
- Elde edilen ham verilerden yenilemeli (recursive) algoritmalar koşturarak frekansın, güç faktörünün ve RMS değerlerinin hesaplanması
- Zamanlama ve senkronizasyon
- Elde edilen verilerin UART (Universal asynchronous receiver/transmitter) üzerinden kullanıcı arayüzüne aktarılması

Yukarıda sıralanmış gereksinimleri yerine getirmek üzere bir sistem mimarisi ortaya atılmış ve gerçeklenmiştir. Bu mimari Şekil 25'te gösterilmiştir

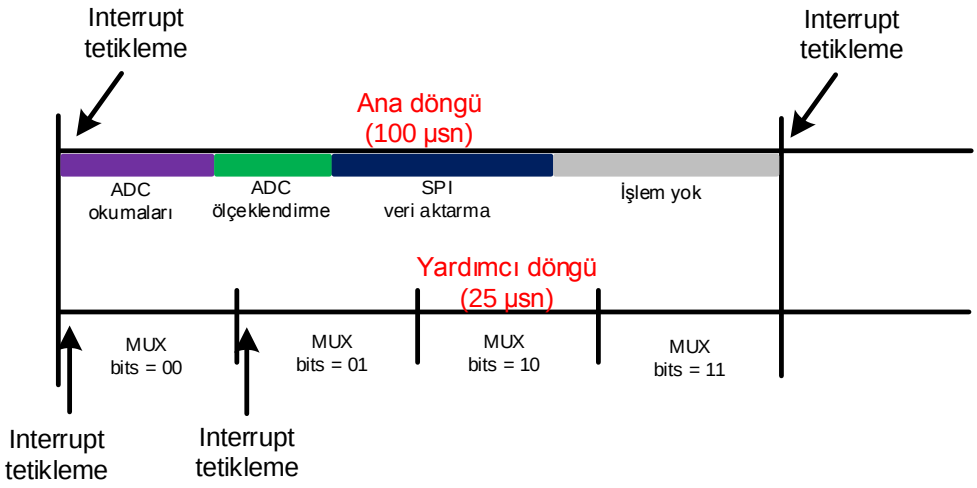


Şekil 25: İşlemci gömülü yazılımı sistem mimarisi akış şeması

Sistemde örnekleme frekansı 10 kHz olarak belirlenmiş ve DSP yazılımında ana döngü her 100 mikrosaniyede bir çalışacak şekilde ayarlanmıştır. Bu amaçla, ePWM modüllerinin interrupt'ları kullanılmıştır. 10 kHz frekansına ayarlanan ePWM modülünün sayacı her sıfır noktasına geldiğinde (sayaç sıfırlandığında) bu ana

interrupt tetiklenmektedir. tüm işlemler bu süre içinde yapılmakta (Raspberry Pi'a veri aktarımı dahil) ve işlem bittiğinde bir sonraki tetikleme beklenmekte ve işlem tekrarlanmaktadır. bu sayede gerçek zamanlı ölçüm (timestamp) sağlanmış olur.

Daha önce, ölçeklendirme bilgilerinin her bir kanal için multiplexerlar aracılığı ile alındığı belirtilmiştir. Bu amaçla, her bir ölçümün hangi ölçekte olduğunu anlamak için, 2 adet bit ile 4 farklı ölçek sırayla denenmektedir. Bu denemeler sonucunda tüm kanallar için ölçek bilgisi alınmış olup, ADC ile alınan veriler yazılım üzerinde ölçeklenmektedir. İşlem bir ana döngüde 4 kez tekrarlandığından dolayı, 4 katı frekansta çalışan (40 kHz) başka bir ePWM modülü interrupt'ı kullanılmıştır (yardımcı döngüler). Alınan veriler paketlenip UART ile gönderilmeden önce ölçeklendirmesi yapılmakta ve sonra gönderilmektedir. Döngüler ve yapılan işlemler Şekil 26'da anlatılmaktadır.

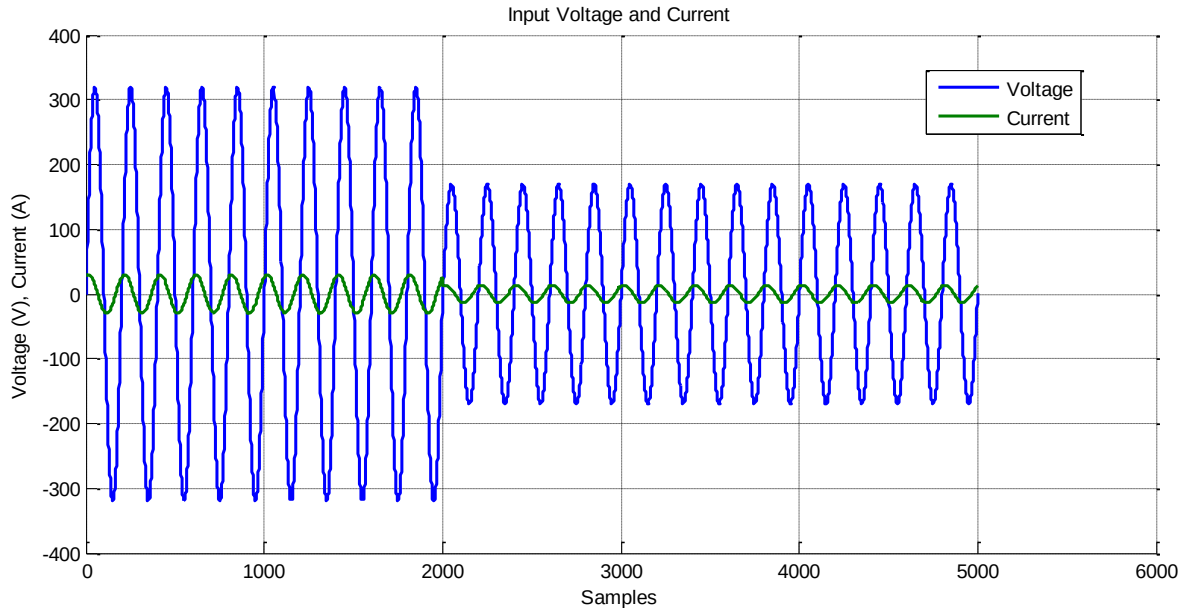


Şekil 26: Mikro-denetleyici yazılımdaki döngüler

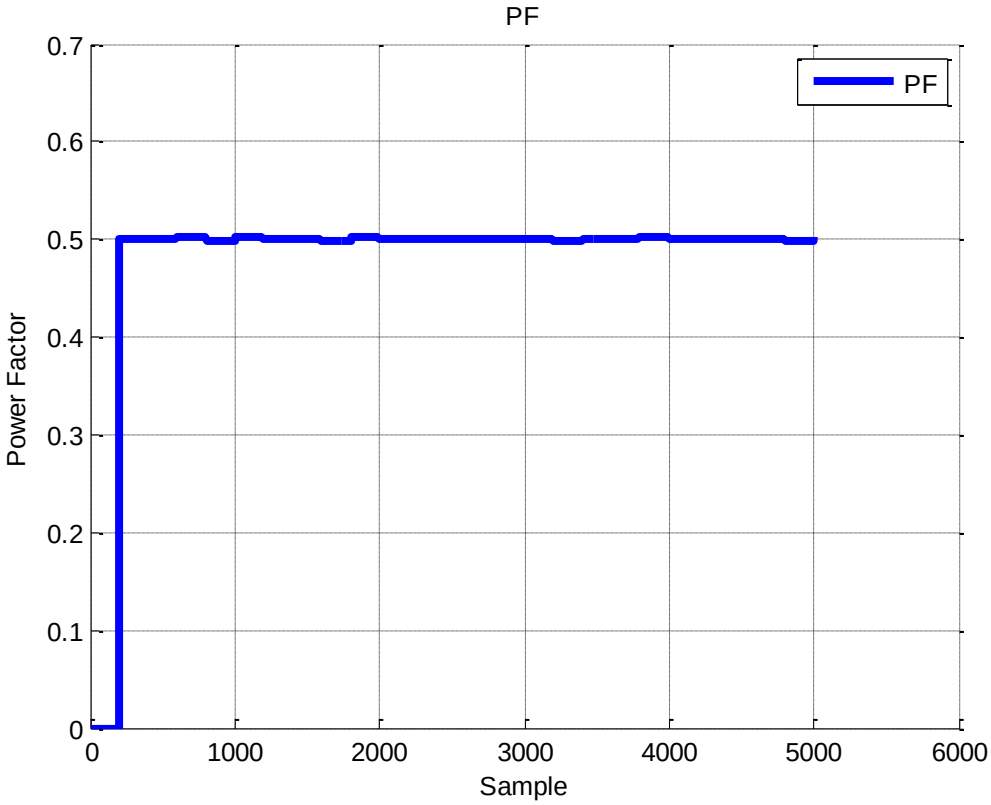
Şekil 25'te gösterilen sistem mimarisinde başta sıralanan gereklerin farklı kısımları gerçeklenmiş ve denenmiştir. 1. ve 2. gereksinimler "ADC Kesmesi" olarak gösterilen blok içerisinde, 3. gereksinim "verilerin alınması ve işlenmesi" bloğunda, 4. Gereksinim ise "UART Altsistemi" bloğunda gerçeklenmiştir. Şekil 25'de gösterilen yazılım mimarisinde yazılımın başlaması kontrol kartına güç verilmesi ile başlamaktadır. Bu işlemin ardından iki adet kesme yaratılarak yazılımın işlevlerini bekleme olmadan yerine getirilmesi sağlanmıştır. "ADC Kesmesi" olarak gösterilen blok **100 mikro saniye** olarak belirlenen zamana aralıkları ile çalışarak örneklenen 12 adet kanaldan verileri almaktadır. Bu kanallar içerisinde RMS, güç faktörü ve frekans değerlerinin kestirileceği kanallar için gereken algoritmaları çalıştırmaktadır.

Bu algoritmaların çalıştırılması sonucunda ortaya çıkan sonuçlar F28335 entegresinin belleğinin belirli bir bölgesinde saklanmakta ve her iterasyon sonrasında (50 Hz bir sinyal için 20 mSec) güncellenmektedir. “ADC Kesmesi” içerisinde hesaplanan ve bellekte güncellenen bu veriler, 50 mSec’de bir çalışan bir mesaj gönderme kesmesi tarafından alınmakta ve UART üzerinden kullanıcı arayüzü bilgisayarına gönderilmektedir. “UART Altsistemi” olarak belirlenen blok bu proje için özel olarak ortaya konulmuş bir yazılım kütüphanesidir. Bu kütüphane kendi içerisinde kesme rutinlerini çağırarak gönderilmek istenen verinin UART altsistemi üzerinden F28335 işlemcisi üzerinde en düşük seviyede işlemci kullanımı ile gönderilmesini sağlamaktadır.

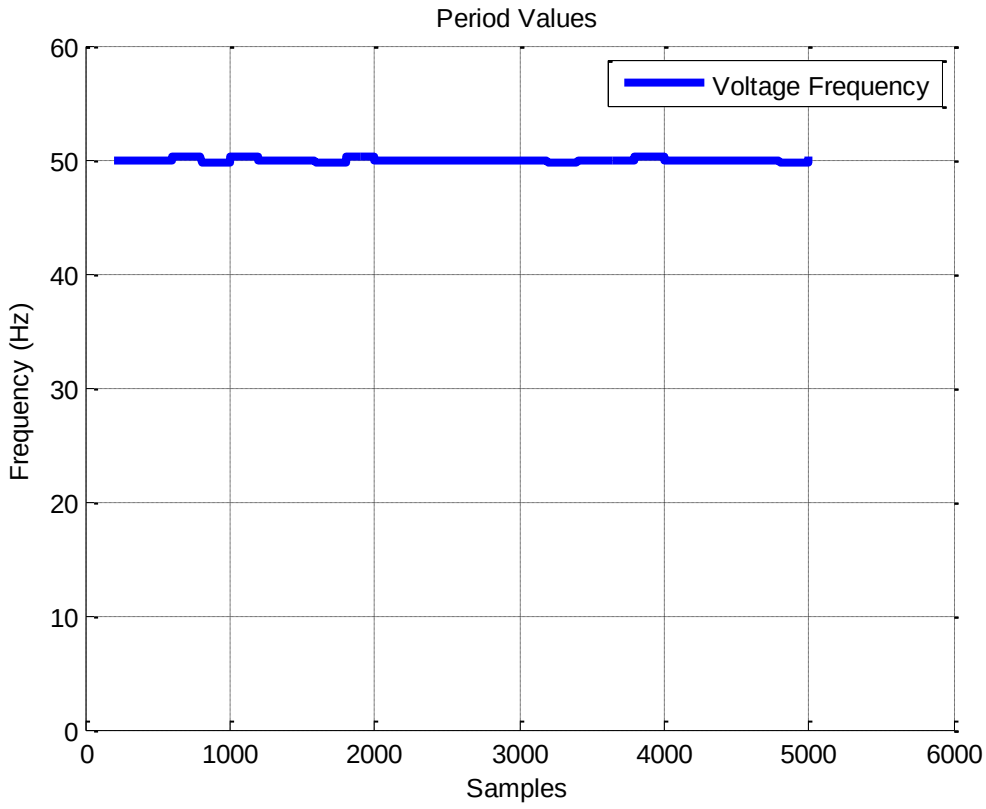
Geliştirilen algoritmalar MATLAB’ta test amaçlı bir akım gerilim ikilisi ile denenmiştir ve sonuçlar Şekil 27-30’da sunulmuştur. Sisteme girdi olarak verilen akım ve gerilim dalga şekilleri Şekil 27’dedir. Algoritma çıktısı olarak elde dilen güç faktörü Şekil 28’de, gerilim frekansı Şekil 29’da, RMS değerleri ise Şekil 30’da görülebilir.



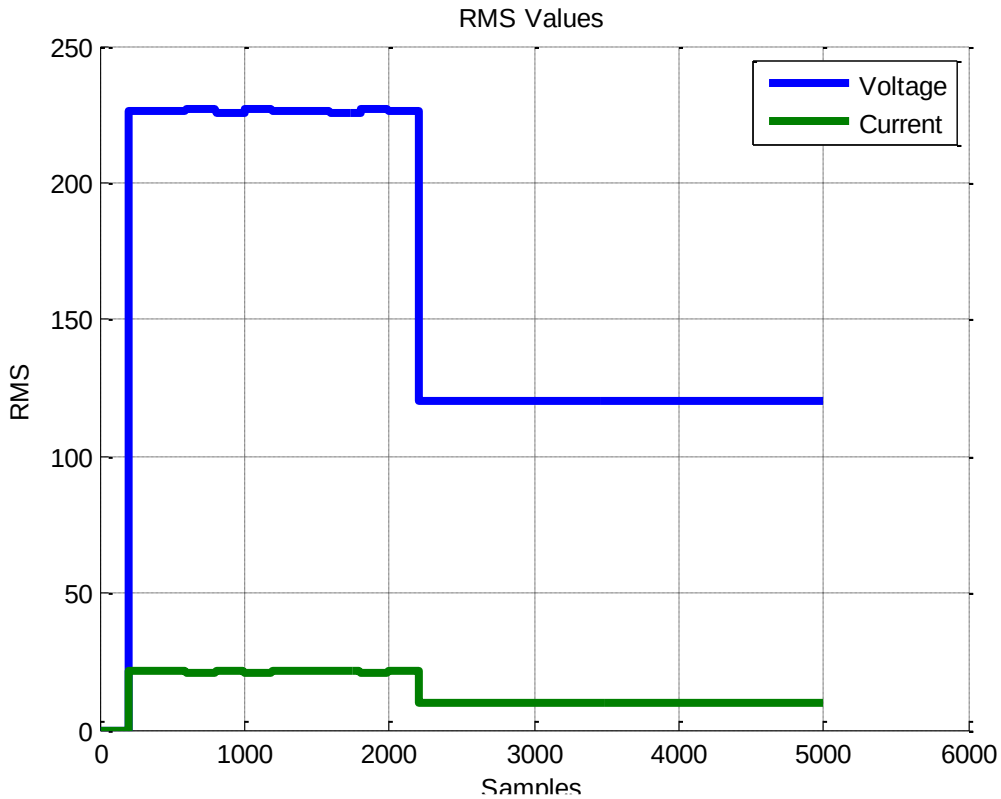
Şekil 27: İşlemci yazılımına girdi olarak verilen gerilim ve akımlar



Şekil 28: Algoritma sonucu elde edilen güç faktörü



Şekil 29: Algoritma sonucu elde edilen gerilim frekansı



Şekil 30: Algoritma sonucu elde edilen RMS değerleri

Bu algoritmalar bahsi geçen gömülü işlemcide işletim sistemi kullanılmadan gerçeklenmiş ve benzer sonuçlar ürettiği kendi içerisinde gerçeklenmiş bir simülatör aracılığı ile gözlemlenmiştir. C dilinde gerçeklenen işlemci yazılımına ait kodun belirli kısımlarından örnekler **Appendix A** kısmında verilmiştir. Gerçeklenen kodun çalışma ve gerçek zamanlılık davranışları incelenmiş ve aşağıda çıkarılmış olan gerekler ile sonuçlar ortaya konulmuştur.

Gerekler:

- Algoritma işlem yükünü bir örnekleme süresi içerisinde bitirmelidir. (100 uSec)
- Gerçeklenen kod ölçüm sonuçlarının belirlenen hızda kullanıcı arayüzü bilgisayarına iletmelidir.

Sonuçlar:

- Gerçeklenen örnekler arası yenilemeli (recursive) algoritmanın verilen sürenin maksimum %20'sinde işlemlerini tamamladığı gözlemlenmiştir.
- Gereklenen UART arayüzünün 15200 Baudrate seviyesinde istenilen verileri saniyede 20 kere gönderebildiği ve bu işlemin toplamda %0.8 işlemci kaynağı kullanılarak gerçekleştirildiği ölçülmüştür.

Bilgisayar ve Dokunmatik Arayüz Birimi

Raspberry Pi 3 (Şekil 31) adı verilen 40$ fiyatında, kredi kartı büyüklüğündeki tek kart bilgisayar (single board computer) prototip içerisinde kullanılarak, cihazın veri izlemek için harici bir bilgisayara bağlanmasına gerek duymaması sağlanmıştır. Bu bilgisayarın görevi, gelen sayısal verilerin önceden tasarlanan bir arayüz üzerinde gösterilmesi ve işlenmesidir. Bilgisayar arayüzü ile yapılabilecek işlemler şunlardır:

- Gerçek zamanlı veri izleme
- Akım ve gerilimlerin RMS, ortalama, vb. değerlerinin gösterimi
- Güç faktörü, aktif güç, reaktif güç, görünür güç gibi değerlerin gösterimi
- İstenilen zaman aralığında veri kaydedilmesi
- Hız ve moment gibi mekanik verilerin izlenmesi
- Makinaların moment çıkışı aracılığı ile zaman bağlı karakteristik girilerek yüklenmesi

Şekil 31: Kullanılan Raspberry Pi 3 tek kart bilgisayar ve dokunmatik ekran

Raspberry Pi Arayüz Yazılımı

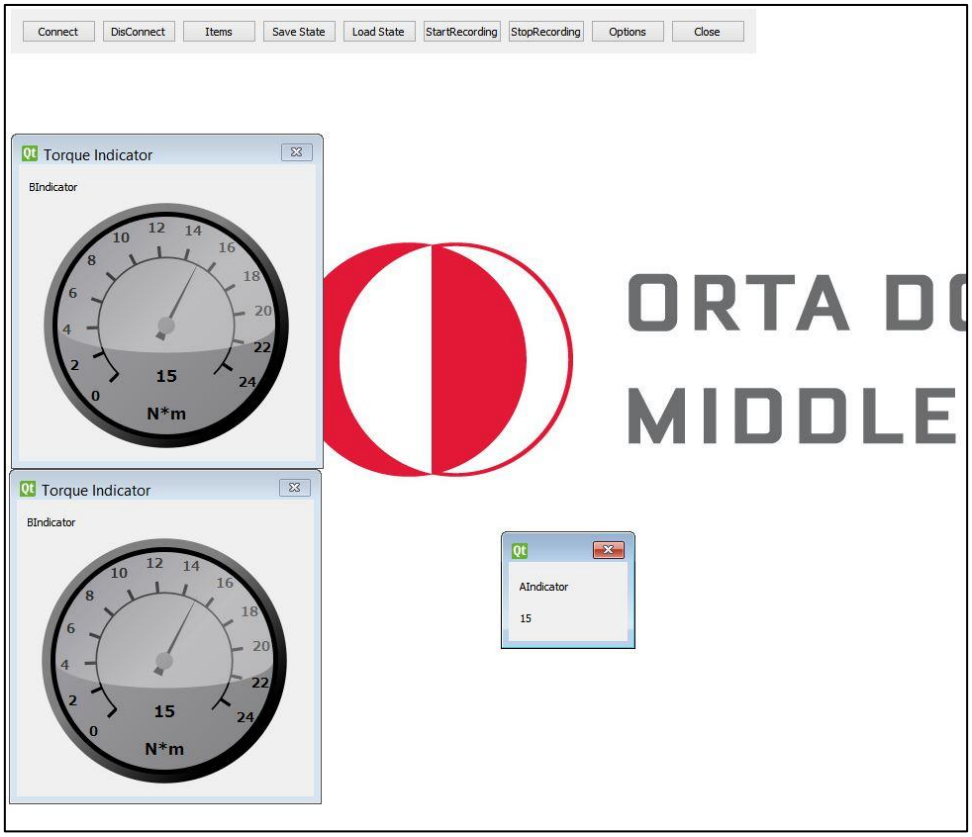
Projede kullanılan arayüz için çeşitli gerekler ortaya çıkarılmıştır.

- Gömülü sistem tarafından UART üzerinden gönderilen verilerin kullanıcıya sunulması.
- Alınan ölçüm verilerinin kaydedilmesi.
- Alınan verilerin gösterilme şeklinin konfigüre edilebilir olması.

Bilgisayar üzerinde gerçeklenen arayüz yazılımının çalışma ortamı QT olarak seçilmiştir. Bu seçimin yapılmasında aşağıdaki faktörler etkili olmuştur:

- İşletim sistemi bağımsız kodlama
- İstenilen işletim sistemi için derleme
- Hazır altyapılar ile çok hızlı bir şekilde yetenek gerçekleme
- Basit ve anlaşılabilir C++ kodlama
- Görevler arası iletişim (Inter Process communication) için basit sinyal-giriş yapısı (Signal-Slot)
- Geniş açık kaynak kodlu topluluk desteği
- GPLv3 olarak kullanabilme

Yukarıda bahsedilen gereksinimler QT çerçeve altyapısı (framework) içerisinde gerçeklenmiştir. Bu gerçeklenen arayüze ilişkin ekran görüntüsü Şekil 32'de görülebilir.



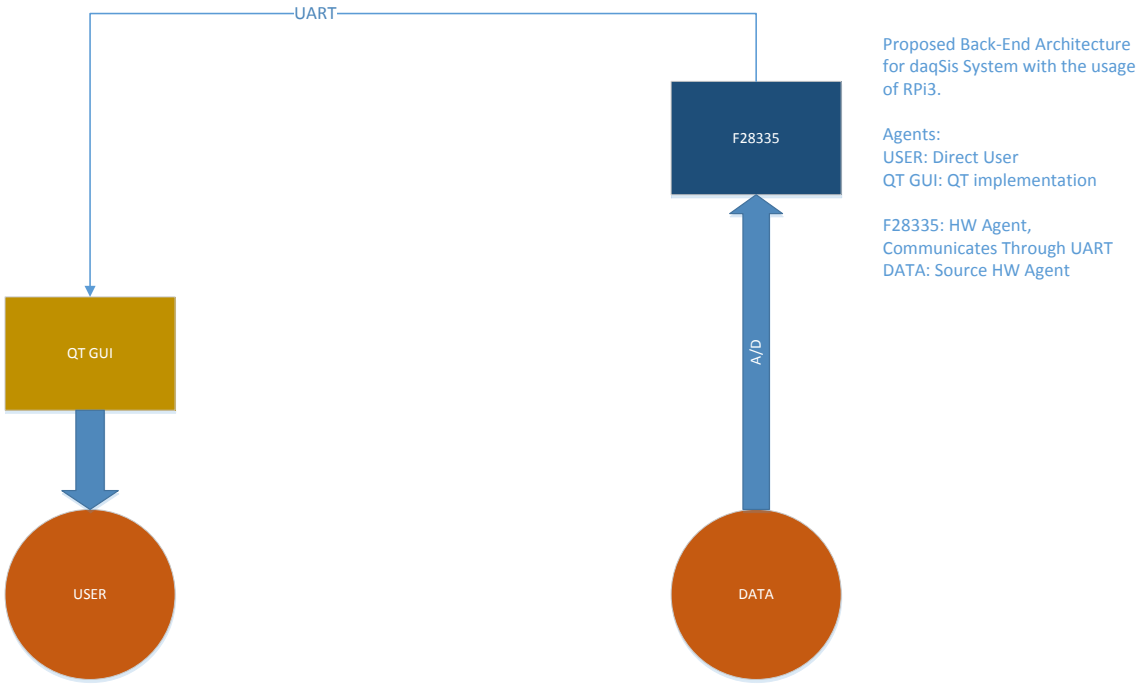
Şekil 32: QT ile geliştirilen bilgisayar arayüzü örnek ekran görüntüsü

Şekil 32'de de görülebileceği üzere verilerin gösterilmesi için iki adet gösterim tipi tercih edilmiştir. Bunlardan bir tanesi Gösterge (Gauge) tipi diğeri ise dijital tip (Indicator) göstergedir. Bu göstergeler gömülü sistem tarafından alınan verileri

konfigüre edilebilir arayüz üzerinde gösterebilmektedir. Sistemin diğer yeteneklerine en üstte bulunan basma tuşları (Push Button) sayesinde ulaşılabilmektedir. BU fonksiyonlar sırası ile:

- **Connect:** Gömülü yazılıma bağlanma
- **Disconnect:** Gömülü yazılımla bağlantıyı koparma
- **Items:** Mevcut göstergelerin listesi
- **Save State:** Mevcut gösterge konfigürasyonunu kaydetme
- **Load State:** Kaydedilmiş bir konfigürasyonu yükleme
- **StartRecording:** Veri kayıt başlatma
- **StopRecording:** Veri kayıt bitirme
- **Options:** Ayarlar
- **Close:** Kapatma

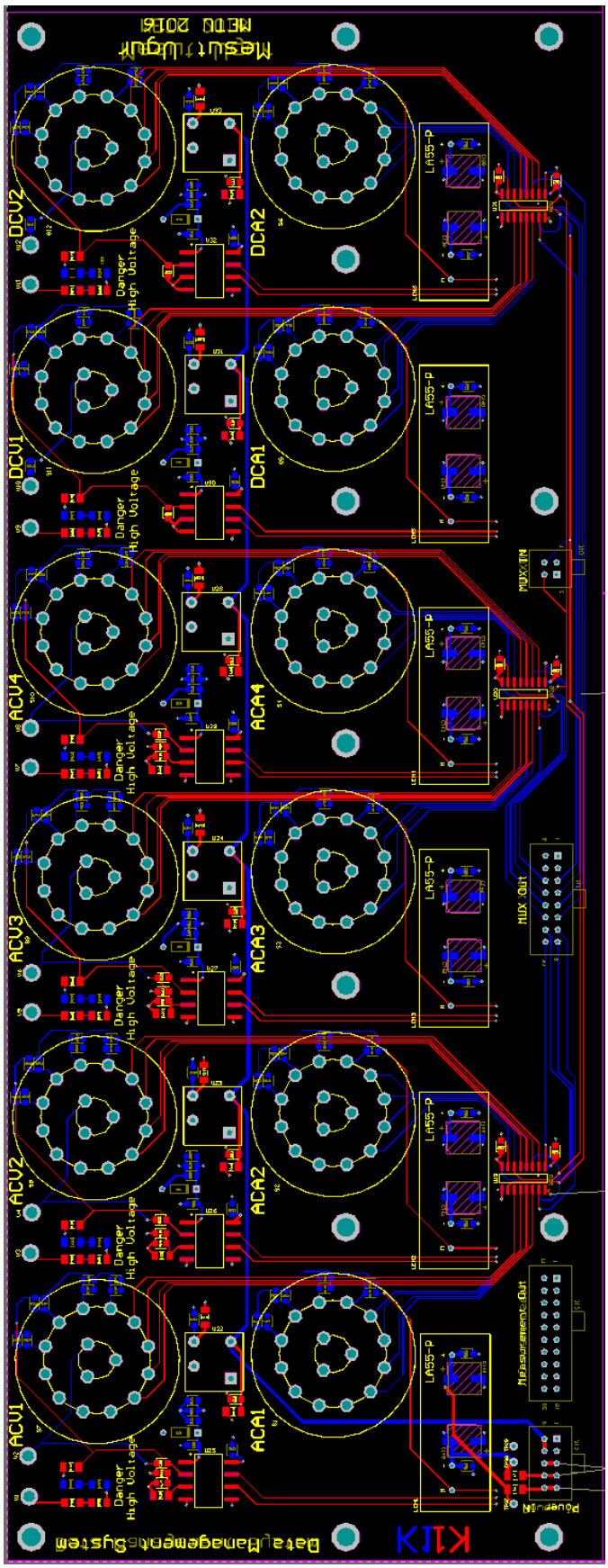
Sistemin işleyişine ilişkin şema ise Şekil 33'te gösterilmiştir. Bu şemadan da anlaşılacağı üzere arayüz yazılımı UART üzerinden verileri F28335 DSP entegresinden almakta ve kullanıcıya göstermektedir. F28335 entegresi de analog-dijital (A/D) çeviricileri sayesinde verileri gerçek fiziksel ortamdan okuyup, gerekli algoritmaları koşturmakta ve bunları arayüz yazılımına göndermektedir.



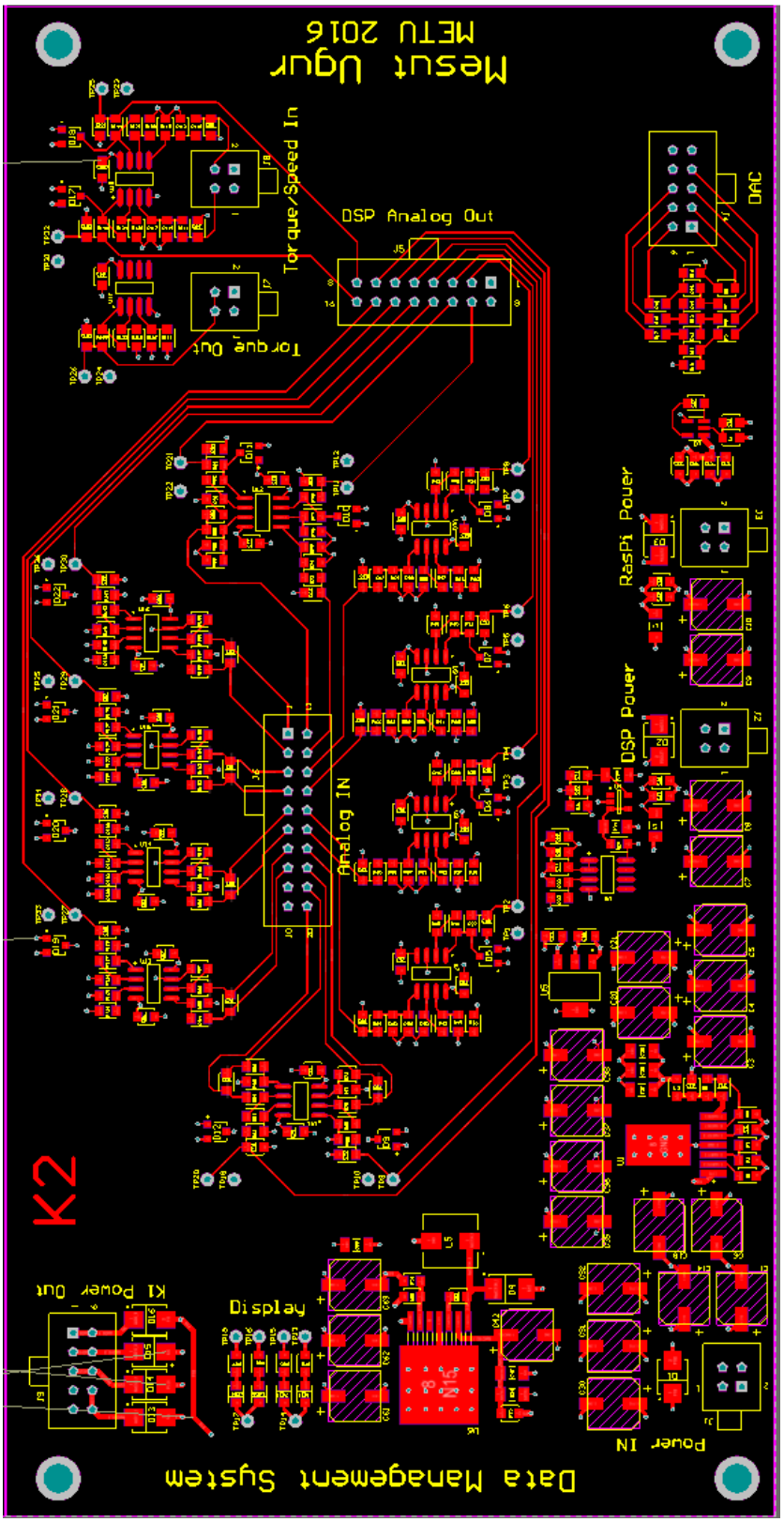
Şekil 33: Sistemin yazılımsal olarak genel işleyiş şeması

Geliştirilen donanımların layout görüntüleri

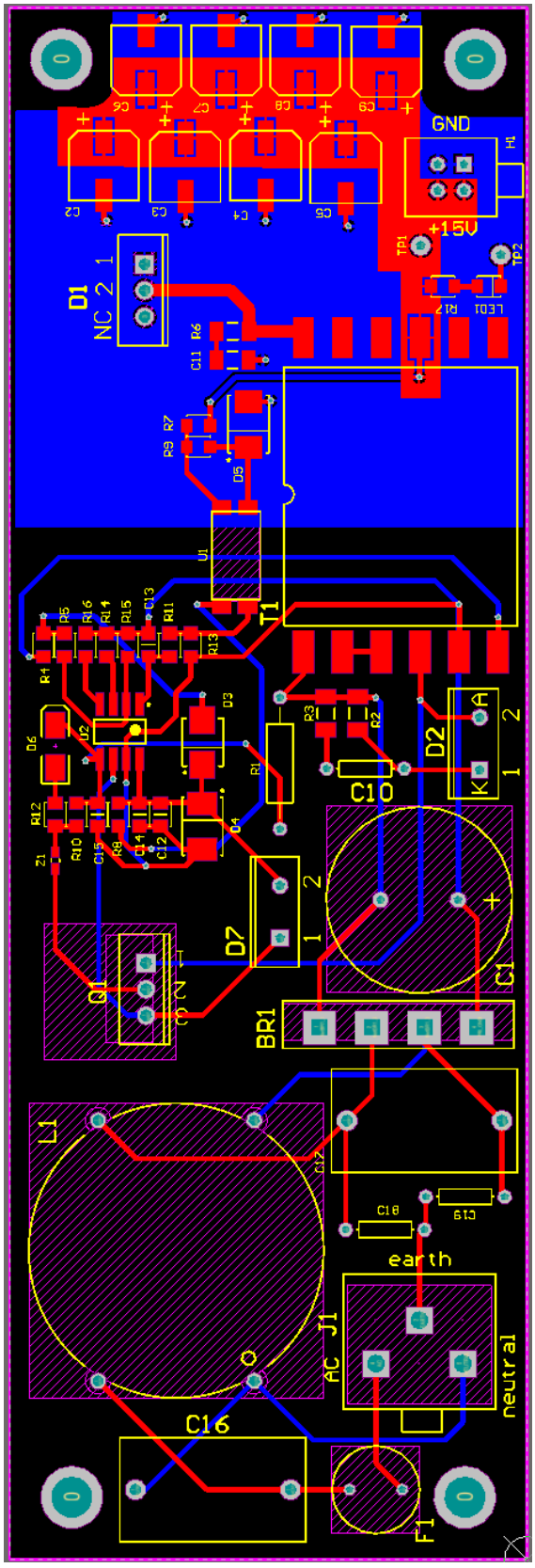
Ölçüm kartı K1:



Veri toplama kartı K2:

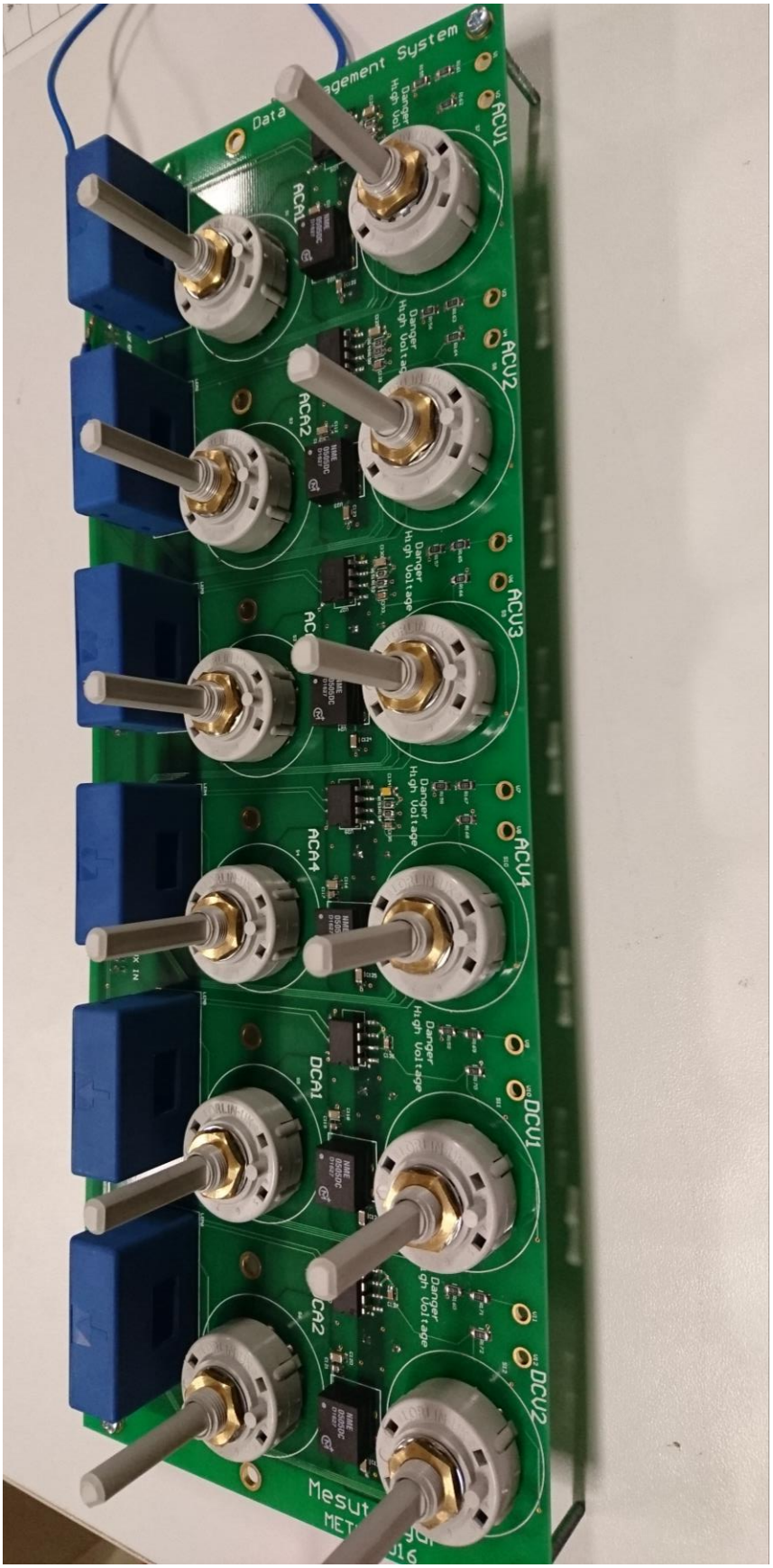


Güç kaynağı kartı K3:

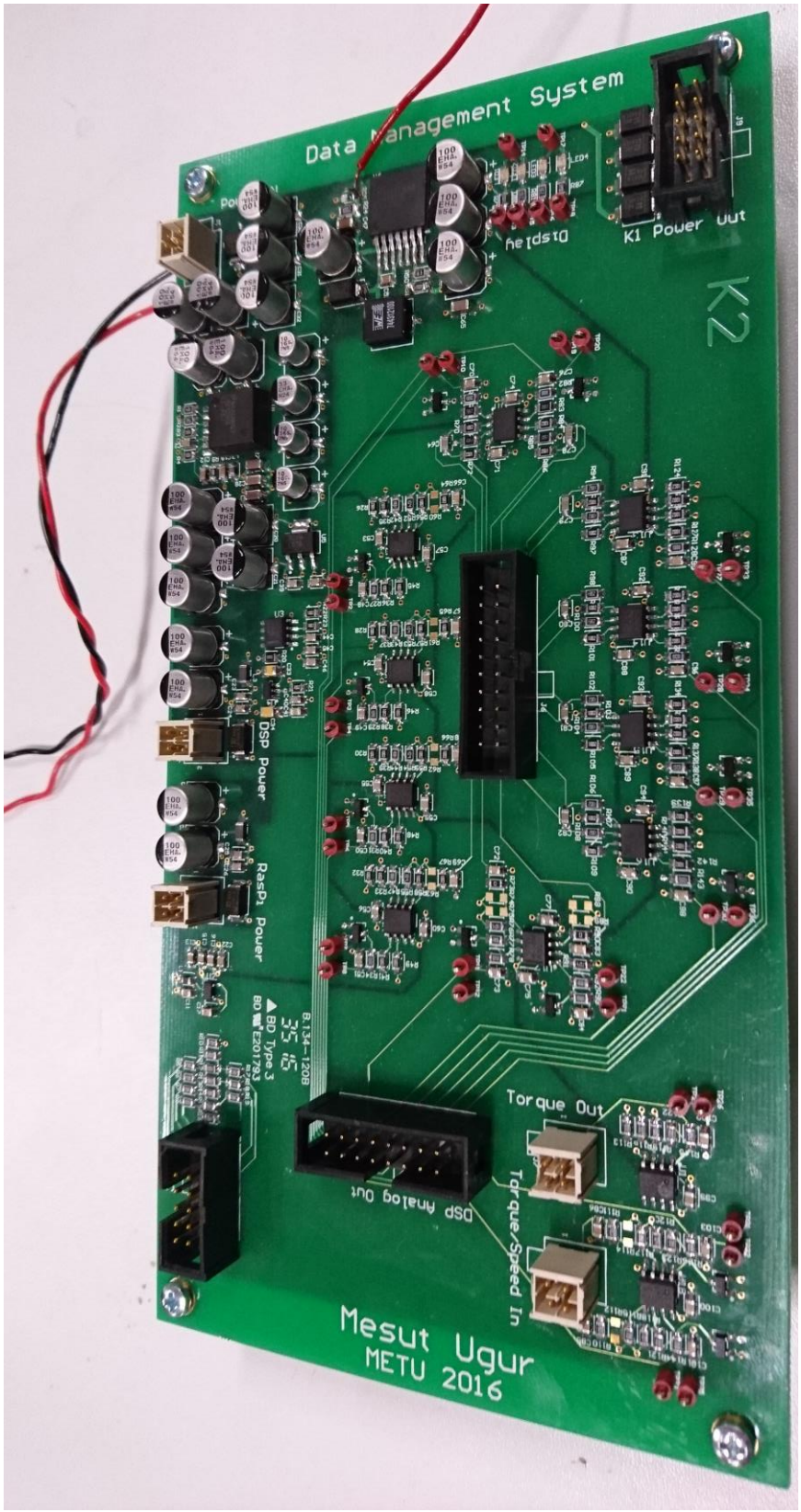


Üretilen kartların resimleri

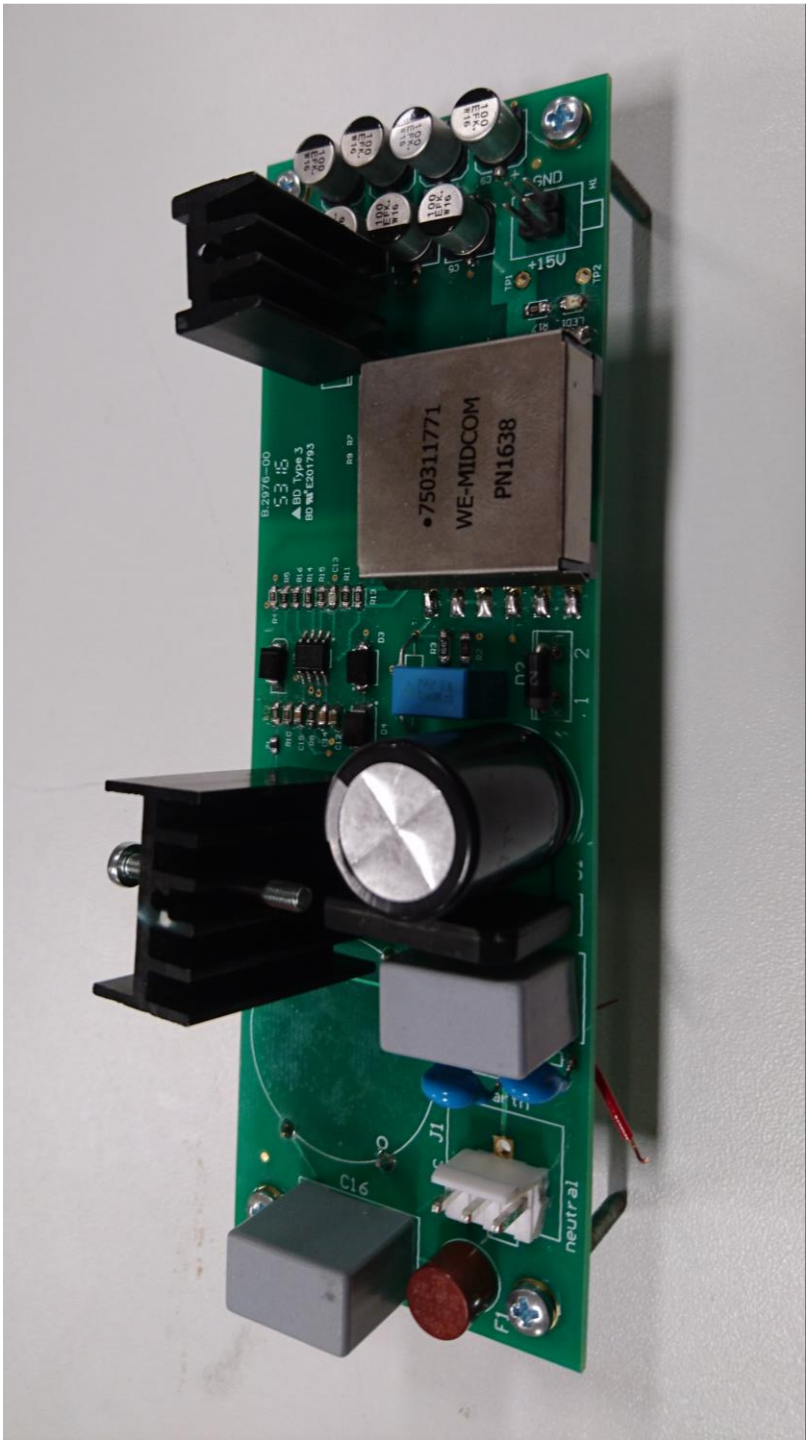
Ölçüm kartı K1:



Veri toplama kartı K2:



Güç kaynağı kartı K3:



Appendix A: Mikro-denetleyici gömülü yazılımı, örnek kodlar

UART sürücü header dosyası:

```
#ifndef UARTDRIVE_28335_H_
#define UARTDRIVE_28335_H_

#define TXINTWORDNUM 4
#define BUFFARRLENGTH 256
#define CURRMSGSIZE 54

#define CPU_FREQ    150E6
#define LSPCLK_FREQ  CPU_FREQ/4
#define SCI_FREQ    1152E2
#define SCI_PRD     (LSPCLK_FREQ/(SCI_FREQ*8))-1

/*Parameter definitions*/
extern int currCondOfUart;
extern char UartCont[256];
extern char *uartBuffWritePtr;
extern char *uartBuffReadPtr;
extern int currCondOfUart;
extern int condBuffer;
extern int condUart;
extern int prevCond;
extern int cntr;
extern int endOfMsgFlag;
extern int contrFlag;
extern int flag;
extern int writeFlag ;
extern int messageFlag ;
extern int contrFlag;
extern int iaGKYCont[3];
extern int *iaPtrContWrite;
extern int *iaPtrContCheck;
extern int currCpuCount;
extern int waitCpuCount;
extern char caGKY[54];
extern char *caPtrGKYWrite;
extern char *caPtrWrite;
extern Uint32 BeginCountTmr;
extern Uint32 EndCountTmr;
extern Uint32 DifferenceCountTmr;
extern Uint32 BeginCountTx;
extern Uint32 EndCountTx;
extern Uint32 DifferenceCountTx;
extern Uint32 TotalTimeTx;
/*Declerations*/

int UartSend(char*,int );
int BufferStatus(int);
int UartTxStatus(void);
void scia_fifo_init(void);
__interrupt void sciaTxFifoIsr(void);
__interrupt void sciaRxFifoIsr(void);
void initParam();
void InitUART();
void iaGKYControl(void);  //To initialize Control Array

#endif /* UARTDRIVE_28335_H_ */
```

UART sürücü source dosyası:

```
#include "DSP28x_Project.h"
#include "UartDrive_28335.h"

int currCondOfUart;
char UartCont[256];
char *uartBuffWritePtr;
char *uartBuffReadPtr;
int currCondOfUart;
int condBuffer;
int condUart;
int prevCond;
int cntr;
int endOfMsgFlag;
int contrFlag;
int flag;
int writeFlag ;
int messageFlag ;
int contrFlag;
int iaGKYCont[3];
int *iaPtrContWrite;
int *iaPtrContCheck;
int currCpuCount;
int waitCpuCount;
char caGKY[54];
char *caPtrGKYWrite;
char *caPtrWrite;
Uint32 BeginCountTx;
Uint32 EndCountTx;
Uint32 DifferenceCountTx;
Uint32 TotalTimeTx;
extern void error(void);
struct GKY_Cont{
          int reEntry_Flag; //Baska unitenin TX i kullanmamasi icin reentrancy flagi
          /*
           *       reEntry_Flag
           *                                    0 ise TX su an kullanilmiyor
           *                                    1 ise TX su an kullaniliyor
           */
          char BuffArr[256];
          int currBufferSize; // Buffer current situation
          char* MsgStart; //From which address the message is starting ;
          int MsgSize; // Message Size (Before sending current message)
          int currMsgSize; // Current message size
}Control_Struct;
int UartSend(char* BuffWriteArray,int lengthOfData)
{
          messageFlag = 1 ;  // Check the Uart Module and prevent to enter other interrupts
such as Timer
          if(prevCond != 2 )
          {
                    condBuffer=BufferStatus(lengthOfData);

                    int tempSize=0;
                    int restSize;

                    if(condBuffer == 1 )  //Buffer is available write to Buffer
                    {
                              if(uartBuffWritePtr+lengthOfData <= Control_Struct.BuffArr + 256
) // If uartBuffWrite pointer when writing the messages will not exceed the Buffer Array
                              {
                                        memcpy(uartBuffWritePtr ,BuffWriteArray , lengthOfData
);
                                        uartBuffWritePtr = uartBuffWritePtr + lengthOfData;
                              }
                              else  // If there is area in B7uffer but writePtr is close to end
of the Buffer array
                              {

                                        memcpy(uartBuffWritePtr  , BuffWriteArray , tempSize);
                                        uartBuffWritePtr = Control_Struct.BuffArr ;
                                        restSize = lengthOfData - tempSize ;
                                        memcpy(uartBuffWritePtr ,BuffWriteArray+tempSize,
restSize );
                                        uartBuffWritePtr = uartBuffWritePtr + restSize ;
```

```
                                }
                                Control_Struct.currBufferSize = Control_Struct.currBufferSize -
lengthOfData;
                        }
                        else
                        {
                                messageFlag = 0 ;
                                return -1 ;
                        }

                        condUart = UartTxStatus();
                        if(condUart == 1)
                        {
                                Control_Struct.currMsgSize = lengthOfData;
                                Control_Struct.reEntry_Flag = 0; //Change reentry flag to 0
                                contrFlag = 1;
                                SciaRegs.SCIFFTX.bit.TXFFIENA =1; // enable the TX interrupt
                                Control_Struct.MsgSize = lengthOfData ;
                                messageFlag = 0 ;
                                return 1;
                        }
                        else
                        {
                                prevCond = 2;
                                messageFlag = 0 ;
                                return 2;
                        }
                }
                else
                {
                        condUart = UartTxStatus();
                        if(condUart == 1)
                        {
                                prevCond = 0;
                                Control_Struct.currMsgSize = lengthOfData;
                                Control_Struct.reEntry_Flag = 0; //Change reentry flag to 0
                                contrFlag = 1;
                                SciaRegs.SCIFFTX.bit.TXFFIENA =1; // enable the TX interrupt
                                Control_Struct.MsgSize = lengthOfData ;
                                messageFlag = 0 ;
                                return 1;
                        }
                        else  // To do double check
                        {
                                prevCond = 2;
                                messageFlag = 0 ;
                                return 2;
                        }
                }

}
int BufferStatus(int lengthOfData)
{

                /* Buffer Status Checker
                 *
                 *  1 ==> Write to Buffer
                 * -1 ==> Do not write to Buffer
                 */
                if(Control_Struct.currBufferSize >= lengthOfData)
                {
                        return 1;
                }
                else
                {
                        return -1;
                }
}

int UartTxStatus(void)
{
                /* Uart Status Checker
                 *  -1 ===> TX is not available wait a while
                 *   1 ===> TX available start to write to TX
                 */
                if(Control_Struct.reEntry_Flag == 1)
                {
```

```
                return 1;
        }
        else
        {
                return -1;
        }
}

__interrupt void sciaTxFifoIsr(void)
{
        BeginCountTx = CpuTimer1.RegsAddr->TIM.all;
        /* Send first type array of the message
           Then send the content of the array
        */
        while(cntr>0)
        {
                if(writeFlag == 1)
                {
                        SciaRegs.SCITXBUF = *uartBuffReadPtr >> 8 ;
                        writeFlag = 0 ;
                }
                else
                {
                        SciaRegs.SCITXBUF = *uartBuffReadPtr;
                        uartBuffReadPtr++;
                        if(uartBuffReadPtr == Control_Struct.BuffArr + 256)
                        {
                                uartBuffReadPtr = Control_Struct.BuffArr ;
                        }
                        writeFlag = 1 ;
                }
                        cntr--;
        }
        if(endOfMsgFlag == 1 )
        {
                Control_Struct.currBufferSize = Control_Struct.currBufferSize +
Control_Struct.currMsgSize ;
                Control_Struct.reEntry_Flag = 1;
                SciaRegs.SCIFFTX.bit.TXFFIENA=0; //Disable TX interrupt
                cntr =TXINTWORDNUM;
                endOfMsgFlag = 0;
                contrFlag = 0;
        }

        if(Control_Struct.currMsgSize <=TXINTWORDNUM/2 && contrFlag == 1)
        {
                cntr = 0;//Control_Struct.currMsgSize;
                endOfMsgFlag = 1;
        }

        if(Control_Struct.currMsgSize > TXINTWORDNUM/2 )
        {
                cntr = TXINTWORDNUM;
                Control_Struct.currMsgSize = Control_Struct.currMsgSize-TXINTWORDNUM/2 ;
                Control_Struct.currBufferSize = Control_Struct.currBufferSize +
TXINTWORDNUM/2 ;
        }

        SciaRegs.SCIFFTX.bit.TXFFINTCLR=1;    // Clear SCI Interrupt flag
        PieCtrlRegs.PIEACK.bit.ACK9 = 1;
        //PieCtrlRegs.PIEACK.all|=0x100;      // Issue PIE ACK
        EndCountTx = CpuTimer1.RegsAddr->TIM.all;
        DifferenceCountTx = BeginCountTx - EndCountTx ;
        TotalTimeTx = TotalTimeTx +  DifferenceCountTx ;
}

__interrupt void sciaRxFifoIsr(void)
{
        SciaRegs.SCIFFRX.bit.RXFFOVRCLR=1;   // Clear Overflow flag
        SciaRegs.SCIFFRX.bit.RXFFINTCLR=1;   // Clear Interrupt flag
        //PieCtrlRegs.PIEACK.bit.A=0x100;       // Issue PIE ack
}

void initParam()
{
        contrFlag = 1;
        writeFlag = 1;
```

```
          messageFlag = 0 ;
          condBuffer = 0;
          condUart = 0 ;
          prevCond = 0 ;
          cntr = TXINTWORDNUM;
          endOfMsgFlag = 0 ;
          currCondOfUart = 1;
          uartBuffWritePtr = Control_Struct.BuffArr; //Initially is pointing start point of
the buffer array
          uartBuffReadPtr = Control_Struct.BuffArr;
          iaGKYControl(); //Initialize the control array
          currCpuCount = 0;
          waitCpuCount = 0;

}

void scia_fifo_init()
{
          SciaRegs.SCICCR.all =0x0007;   // 1 stop bit,  No loopback
          // No parity,8 char bits,
          // async mode, idle-line protocol
          SciaRegs.SCICTL1.bit.RXENA =1;  // enable RX, internal SCICLK,
          SciaRegs.SCICTL1.bit.TXENA = 1;
          // Disable RX ERR, SLEEP, TXWAKE
          SciaRegs.SCICTL2.bit.TXINTENA =1; //From the beginning dont start the TX interrupt
          SciaRegs.SCICTL2.bit.RXBKINTENA =1;
          SciaRegs.SCIHBAUD = 0x0000;
          SciaRegs.SCILBAUD = SCI_PRD;
          SciaRegs.SCICCR.bit.LOOPBKENA =0; // Enable loop back
          SciaRegs.SCIFFTX.all=0xC828;

          SciaRegs.SCIFFRX.all=0x0028;
          SciaRegs.SCIFFCT.all=0x00;

          SciaRegs.SCICTL1.bit.RXENA =1;     // Relinquish SCI from Reset
          SciaRegs.SCICTL1.bit.TXENA = 1;
          SciaRegs.SCICTL1.bit.SWRESET = 1;
          SciaRegs.SCIFFTX.bit.TXFFIENA = 0;
          SciaRegs.SCIFFTX.bit.TXFIFOXRESET=1;
          SciaRegs.SCIFFRX.bit.RXFIFORESET=1;
}

void iaGKYControl(void)
{
          Control_Struct.reEntry_Flag = 1; //Default value of reentry flag
          Control_Struct.currBufferSize = BUFFARRLENGTH; // Current buffer size
          Control_Struct.currMsgSize = 0;  // Current message size
          /*                            reEntry_Flag
           *                                    0 ==> TX is working , is not available,under
other operation
           *                                    1 ==> TX is not working,is available
           */
}

void InitUART()
{
          //Enable interrupt for tx and rx
          IER |= M_INT9;
          // Enable TINT0 in the PIE: Group 1 interrupt 7
          PieCtrlRegs.PIEIER9.bit.INTx1=1;     // PIE Group 9, int1
          PieCtrlRegs.PIEIER9.bit.INTx2=1;     // PIE Group 9, INT2
          InitSciGpio(); // Initialize SCI GPIO
          initParam(); //Initialize the parameters
          scia_fifo_init(); // Initialize SCI-A
          // Enable global Interrupts and higher priority real-time debug events:
          ERTM;   // Enable Global realtime interrupt DBGM
          EINT;   // Enable Global interrupt INTM
}

void InitSciGpio()
{
          InitSciaGpio();
}
void InitSciaGpio(){
          EALLOW;

          //
```

```
          // Enable internal pull-up for the selected pins
          // Pull-ups can be enabled or disabled disabled by the user.
          // This will enable the pullups for the specified pins.
          //
          GpioCtrlRegs.GPAPUD.bit.GPIO28 = 0;  // Enable pull-up for GPIO28 (SCIRXDA)
          GpioCtrlRegs.GPAPUD.bit.GPIO29 = 0;    // Enable pull-up for GPIO29 (SCITXDA)

          //
          // Set qualification for selected pins to asynch only
          // Inputs are synchronized to SYSCLKOUT by default.
          // This will select asynch (no qualification) for the selected pins.
          //
          GpioCtrlRegs.GPAQSEL2.bit.GPIO28 = 3;  // Asynch input GPIO28 (SCIRXDA)

          //
          // Configure SCI-A pins using GPIO regs
          // This specifies which of the possible GPIO pins will be SCI functional
          // pins.
          //
          GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 1;   // Configure GPIO28 to SCIRXDA
          GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 1;   // Configure GPIO29 to SCITXDA

          EDIS;
}
```

Analog veri okuma ve hesaplama

```
#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File
#include <stdlib.h>
#include <string.h >
#include <stdio.h>
#include "UartDrive_28335.h"

void error(void);
__interrupt void cpu_timer0_isr(void);
__interrupt void cpu_timer1_isr(void);
__interrupt void cpu_timer2_isr(void);
void caGKYVeriArray(void); //To initialize the Message Array

Uint32 BeginCountTmr;
Uint32 EndCountTmr;
Uint32 DifferenceCountTmr;
Uint32 OverallTime;
char TestCont[256];
struct GKY_Veri{
          char Gond_Kod;   //Gonderici kodu
          char Alc_Kod;       //Alici kodu
          short int Msg_Id;            //Mesaj Idsi
          short int Msg_Uznlg;         //Mesaj uzunlugu
          float Volt_Rms1;   // AC Voltaj RMS1
          float Volt_Rms2;   // AC Voltaj RMS2
          float Volt_Rms3;   // AC Voltaj RMS3
          float Volt_Rms4;   // AC Voltaj RMS4
          float Volt_Dc1;              // DC Voltaj 1
          float Volt_Dc2;              // DC Voltaj 2
          float Akim_Rms1;   // AC Akim RMS1
          float Akim_Rms2;   // AC Akim RMS2
          float Akim_Rms3;   // AC Akim RMS3
          float Akim_Rms4;   // AC Akim RMS4
          float Akim_Dc1;              // DC Akim 1
          float Akim_Dc2;              // DC Akim 2
          float Moment;                //Moment
          float Hiz;                             //Hiz
          float Ac_Guc1;               // AC Guc 1
          float Ac_Guc2;               // AC Guc 2
          float Ac_Guc3;               // AC Guc 3
          float Ac_Guc4;               // AC Guc 4
          float Ac_GucFktr1; // AC Guc Faktoru 1
          float Ac_GucFktr2; // AC Guc Faktoru 2
          float Ac_GucFktr3; // AC Guc Faktoru 3
          float Ac_GucFktr4; // AC Guc Faktoru 4
          float Dc_Guc1;               // DC Guc 1
          float Dc_Guc2;               // DC Guc 2
          float Meka_Guc;              // Mekanik Guc
};
void main(void)
{
```

```
          InitSysCtrl();
          DINT;
          InitPieCtrl();

          IER = 0x0000;
          IFR = 0x0000;
          flag =0;
          InitPieVectTable();
          caGKYVeriArray();
          EALLOW;  // This is needed to write to EALLOW protected registers
          PieVectTable.TINT0 = &cpu_timer0_isr;
          PieVectTable.XINT13 = &cpu_timer1_isr;
          PieVectTable.TINT2 = &cpu_timer2_isr;
          PieVectTable.SCIRXINTA = &sciaRxFifoIsr;
          PieVectTable.SCITXINTA = &sciaTxFifoIsr;
          EDIS;    // This is needed to disable write to EALLOW protected registers
          IER |= M_INT1;
          IER |= M_INT13;
          IER |= M_INT14;
          PieCtrlRegs.PIECTRL.bit.ENPIE = 1;   // Enable the PIE block
          PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
          InitCpuTimers();

#if (CPU_FRQ_150MHZ)
          ConfigCpuTimer(&CpuTimer0, 150, 50000);
          ConfigCpuTimer(&CpuTimer1, 150, 100000);
          ConfigCpuTimer(&CpuTimer2, 150, 1000000);
#endif

#if (CPU_FRQ_100MHZ)
          ConfigCpuTimer(&CpuTimer0, 100, 100000);
          ConfigCpuTimer(&CpuTimer1, 100, 100000);
          ConfigCpuTimer(&CpuTimer2, 100, 1000000);
#endif

          CpuTimer0Regs.TCR.all = 0x4000; // Use write-only instruction to set TSS bit = 0
          CpuTimer1Regs.TCR.all = 0x4000; // Use write-only instruction to set TSS bit = 0
          CpuTimer2Regs.TCR.all = 0x4000; // Use write-only instruction to set TSS bit = 0

          InitUART();
          CpuTimer0Regs.TCR.bit.TSS = 0; // Start the timer

          for(;;);
}
void error(void)
{
          __asm("     ESTOP0"); // Test failed!! Stop!
          for(;;){};
}

__interrupt void cpu_timer0_isr(void)
{
          OverallTime = TotalTimeTx + DifferenceCountTmr ;
          CpuTimer0.InterruptCount++;
          BeginCountTmr = CpuTimer1.RegsAddr->TIM.all;

          if(messageFlag == 0 )
          {
                    currCondOfUart = UartSend(TestCont,256);
          }
          if(currCondOfUart == -1 )
          {
                    error();
          }
          PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
          EndCountTmr = CpuTimer1.RegsAddr->TIM.all;
          DifferenceCountTmr = BeginCountTmr - EndCountTmr ;
          TotalTimeTx = 0 ;
}

__interrupt void cpu_timer1_isr(void)
{
          CpuTimer1.InterruptCount++;
          EDIS;
}
__interrupt void cpu_timer2_isr(void)
{  EALLOW;
```

```
CpuTimer2.InterruptCount++;
EDIS;
}
void caGKYVeriArray(void){
          struct GKY_Veri GKY_Verileri;
          caPtrGKYWrite = caGKY;
          GKY_Verileri.Gond_Kod = 12;
          GKY_Verileri.Alc_Kod = 33;
          GKY_Verileri.Msg_Id = 47;
          GKY_Verileri.Msg_Uznlg = 54;
          GKY_Verileri.Volt_Rms1 = 220;
          GKY_Verileri.Volt_Rms2 = 220;
          GKY_Verileri.Volt_Rms3 = 220;
          GKY_Verileri.Volt_Rms4 = 220;
          GKY_Verileri.Volt_Dc1 = 310;
          GKY_Verileri.Volt_Dc2 = 310;

          GKY_Verileri.Akim_Rms1 = 15;
          GKY_Verileri.Akim_Rms2 = 15;
          GKY_Verileri.Akim_Rms3 = 15;
          GKY_Verileri.Akim_Rms4 = 15;
          GKY_Verileri.Akim_Dc1 = 20;
          GKY_Verileri.Akim_Dc2 = 20;

          memcpy(caPtrGKYWrite+ 0, &(GKY_Verileri.Gond_Kod),
sizeof(GKY_Verileri.Gond_Kod));
          memcpy(caPtrGKYWrite+ 1, &(GKY_Verileri.Alc_Kod),
sizeof(GKY_Verileri.Alc_Kod));
          memcpy(caPtrGKYWrite+ 2, &(GKY_Verileri.Msg_Id),
sizeof(GKY_Verileri.Msg_Id));
          memcpy(caPtrGKYWrite+ 3, &(GKY_Verileri.Msg_Uznlg),
sizeof(GKY_Verileri.Msg_Uznlg));
          memcpy(caPtrGKYWrite+ 4, &(GKY_Verileri.Volt_Rms1),
sizeof(GKY_Verileri.Volt_Rms1));
          memcpy(caPtrGKYWrite+ 6, &(GKY_Verileri.Volt_Rms2),
sizeof(GKY_Verileri.Volt_Rms2));
          memcpy(caPtrGKYWrite+ 8, &(GKY_Verileri.Volt_Rms3),
sizeof(GKY_Verileri.Volt_Rms3));
          memcpy(caPtrGKYWrite+10, &(GKY_Verileri.Volt_Rms4),
sizeof(GKY_Verileri.Volt_Rms4));
          memcpy(caPtrGKYWrite+12, &(GKY_Verileri.Volt_Dc1),
sizeof(GKY_Verileri.Volt_Dc1));
          memcpy(caPtrGKYWrite+14, &(GKY_Verileri.Volt_Dc2),
sizeof(GKY_Verileri.Volt_Dc2));
          memcpy(caPtrGKYWrite+16, &(GKY_Verileri.Akim_Rms1),
sizeof(GKY_Verileri.Akim_Rms1));
          memcpy(caPtrGKYWrite+18, &(GKY_Verileri.Akim_Rms2),
sizeof(GKY_Verileri.Akim_Rms2));
          memcpy(caPtrGKYWrite+20, &(GKY_Verileri.Akim_Rms3),
sizeof(GKY_Verileri.Akim_Rms3));
          memcpy(caPtrGKYWrite+22, &(GKY_Verileri.Akim_Rms4),
sizeof(GKY_Verileri.Akim_Rms4));
          memcpy(caPtrGKYWrite+24, &(GKY_Verileri.Akim_Dc1),
sizeof(GKY_Verileri.Akim_Dc1));
          memcpy(caPtrGKYWrite+26, &(GKY_Verileri.Akim_Dc2),
sizeof(GKY_Verileri.Akim_Dc2));
          memcpy(caPtrGKYWrite+28, &(GKY_Verileri.Moment),
sizeof(GKY_Verileri.Moment));
          memcpy(caPtrGKYWrite+30, &(GKY_Verileri.Hiz),             sizeof(GKY_Verileri.Hiz));
          memcpy(caPtrGKYWrite+32, &(GKY_Verileri.Ac_Guc1),
sizeof(GKY_Verileri.Ac_Guc1));
          memcpy(caPtrGKYWrite+34, &(GKY_Verileri.Ac_Guc2),
sizeof(GKY_Verileri.Ac_Guc2));
          memcpy(caPtrGKYWrite+36, &(GKY_Verileri.Ac_Guc3),
sizeof(GKY_Verileri.Ac_Guc3));
          memcpy(caPtrGKYWrite+38, &(GKY_Verileri.Ac_Guc4),    sizeof(GKY_Verileri.Ac_Guc4));
          memcpy(caPtrGKYWrite+40,
&(GKY_Verileri.Ac_GucFktr1),sizeof(GKY_Verileri.Ac_GucFktr1));
          memcpy(caPtrGKYWrite+42,
&(GKY_Verileri.Ac_GucFktr2),sizeof(GKY_Verileri.Ac_GucFktr2));
          memcpy(caPtrGKYWrite+44,
&(GKY_Verileri.Ac_GucFktr3),sizeof(GKY_Verileri.Ac_GucFktr3));
          memcpy(caPtrGKYWrite+46,
&(GKY_Verileri.Ac_GucFktr4),sizeof(GKY_Verileri.Ac_GucFktr4));
          memcpy(caPtrGKYWrite+48, &(GKY_Verileri.Dc_Guc1),
sizeof(GKY_Verileri.Dc_Guc1));
```

```
          memcpy(caPtrGKYWrite+50, &(GKY_Verileri.Dc_Guc2),
sizeof(GKY_Verileri.Dc_Guc2));
          memcpy(caPtrGKYWrite+52, &(GKY_Verileri.Meka_Guc),
sizeof(GKY_Verileri.Meka_Guc));

          int l;
          for(l=0;l<256;l++)
          {
                    TestCont[l] =l;
          }
}
```