Modül 7: MSP430 USCI Birimi ve UART Uygulamaları

Giriş

Elektronik sistemlerde diğer önemli bir konu ise haberleşmedir. Elektronikte haberleşme denetleyicili yada elektronik bir sistemin çevresinde, yakınında yada uzağında bulunan diğer sistemler, çevre birimleri yada benzer elektronik sistemler ile bilgi alışverişi yapabilmesidir. Haberleşme geniş bir konu olduğu için mikrodenetleyiciler için genelleştirirsek, haberleşme mikrodenetleyicinin çevresinde bulunan diğer çevre birimleri yada diğer mikrodenetleyicili sistemler ile bilgi alışverişi yapabilmesidir.

Mikrodenetleyicilerin haberleşme yapabilmesi için bazı haberleşme protokolleri bulunmaktadr. Bunlardan bir kaçı; SPI, I2C, RS232, RS422, RS485 gibi haberleşme protokolleridir. Bu protokoller gerek donanım olarak gerekse haberleşme yapısı olarak tanımlanmıştır. Bu sayede aynı protokolü kullanan iki sistem kolaylıkla birbiriyle haberleşebilir.

Haberleşme protokolleri kullanım alanlarına uygun olarak kullanmak gereklidir. Örneğin SPI protokolü mikrodenetleyicinin yakın çevresinde bulunan sensör, LCD, hafıza elemanı gibi birimlerle çoklu olarak kısa mesafede hızlı bilgi alışverişi yapmak için kullanılır. Aynı şekilde I2C protokolü ise düşük hızda çevre birimleri ile haberleşme yapmak için kullanılır. SPI ve I2C protokolleri senkron olarak çalışırlar. Yani alıcı ile verici eş zamanlı olarak haberleşirler.

RS232 protokolü ise genellikle daha uzak iki noktada ki sistemlerin haberleşmesinde kullanılır. Örneğin Bilgisayar ile mikrodenetleyicinin haberleşmesinde çok kullanılır. RS422 ve RS485 gibi protokoller ise RS232 protokolünün geliştirilmiş halidir. Bu protokoller ile birden çok cihaz aralarında yüzlerce metre mesafe olmasına rağmen hızlı bir şekilde haberleşebilirler. RS232, RS422, RS485 gibi protokoller asenkron çalışırlar. Yani alıcı ile verici arasında senkronizasyonu sağlamak için bir işaret(saat işareti) yoktur.

Sonuç olarak haberleşme sistemlerinin kavranması uygulama ve tecrübe gerekmektedir. Uygulamalarda yeri geldikçe gereken sistemler kullanılabilir. Bizde bu modül kapsamında MSP430 denetleyiciler içerisinde bulunan USCI biriminin UART özelliğini kullanarak RS232 uygulamaları gerçekleştireceğiz.

UART, (Universal Asynchronous Receiver Transmitter/Evrensel Asenkron Alıcı Verici) birimleri mikrodenetleyicilere RS232, RS422, RS485 gibi protokollerin haberleşme yapılarına göre haberleşme imkanı sağlar. Mikrodenetleyiciler içerilerinde bulunan UART birimi ve uygun donanım protokolünü sağlayan entegreler(SN75176, MAX232 vb.) ile RS232, RS422, RS485 protokolleri ile haberleşen sistemler ile haberleşilebilir. Günümüzde çoğu mikrodenetleyiciler içerisinde gerek senkron (SPI, I2C) gerekse asenkron(RS232, RS422, RS485) haberleşme için destek veren haberleşme birimleri barındırırlar. MSP430 USCI birimide benzer şekilde senkron ve asenkron iletişimi sağlayan haberleşme birimidir.

MSP430 USCI Birimi

USCI yani Universal Serial Communication Interface(Evrensel Seri Haberleşme Arayüzü) MSP430 denetleyiciler içerisinde bulunan ve bir çok seri haberleşme protokolünü destekleyen gelişmiş bir birimdir. MSP430G2553 denetleyicisi içerisinde USCI_A0 ve USCI_B0 olmak üzere iki adet USCI birimi bulunmaktadır. USCI birimlerini özellikleri aşağıdaki gibidir.

USCI_A0 Birimi Özellikleri

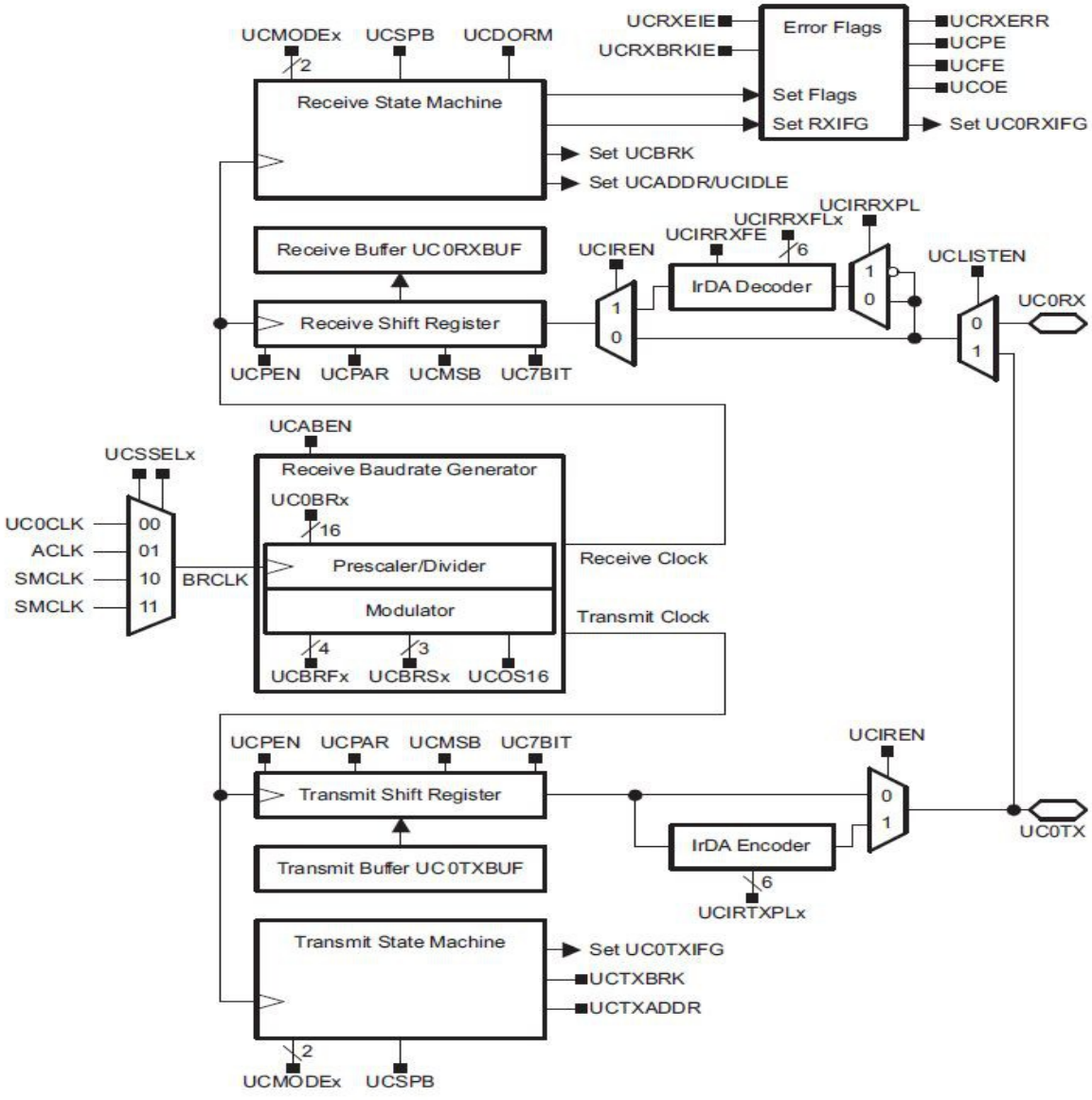
- UART haberleşme özelliği
- Kızılötesi haberleşmeler için darbe şekillendirme özelliği
- LIN haberleşmesi için otomatik baud tespit özelliği
- SPI haberleşme özelliği

USCI_B0 Birimi Özellikleri

- I2C haberleşme özelliği
- SPI haberleşme özelliği

Görüldüğü gibi her iki biriminde kendine özgü özellikleri bulunmaktadır. USCI birimi tüm özellikleriyle kapsamlı bir konu olduğu için biz bu modül kapsamında USCI_A0 biriminin UART özelliği üzerinde durup uygulamalarını gerçekleştireceğiz.

USCI_A0 UART Birimi



Şekil 1: USCI Birimi UART Modu Blok Diyagramı

Şekil 1'de USCI biriminin UART modunda ki blok diyagramı görülmektedir. UCAxCTL0 kaydedicisinde ki UCSYNC biti sıfır yapılarak UART moduna geçilir. UART modunda MSP430 denetleyici UCAxRXD ve UCAxTXD pinleri üzerinden çevre birimleri ile haberleşir.

İfadelerde geçen x USCI_Ax biriminin numarasını temsil etmektedir. Farklı denetleyicilerde birden fazla USCI_A birimi olabilir. Bizim kullandığımız MSP430G2553 denetleyicisinde bir tane USCI_A birimi olduğu için x yerine sıfır olduğunu düşünebilirsiniz.

USCI_A0 biriminin özellikleri;

- 7,8 bit veri transferi tek, çift parity yada parity yok desteği
- Bağımsız gönderme ve alma kaymalı kaydedicileri
- Ayrı ayrı alma ve gönderme tampon kaydedicileri
- LSB veya MSB ilk transfer seçimi
- Çoklu işlemcili haberleşme desteği
- Veri geldiğinde tasarruf modlarından otomatik uyanma
- Bağımsız kesme oluşturabilen alma ve gönderme özelliği

Görüldüğü gibi USCI biriminin tek başına UART modunun bile bir çok fonksiyonu bulunmaktadır. USCI biriminin tüm özellikleri kaydedici yapılarını vs. kapsayan bilgiler için kullanma kılavuzunu inceleyebilirsiniz. UART uygulamaları ile çalışma yapısı daha iyi anlaşılabilir.

UART Uygulamaları

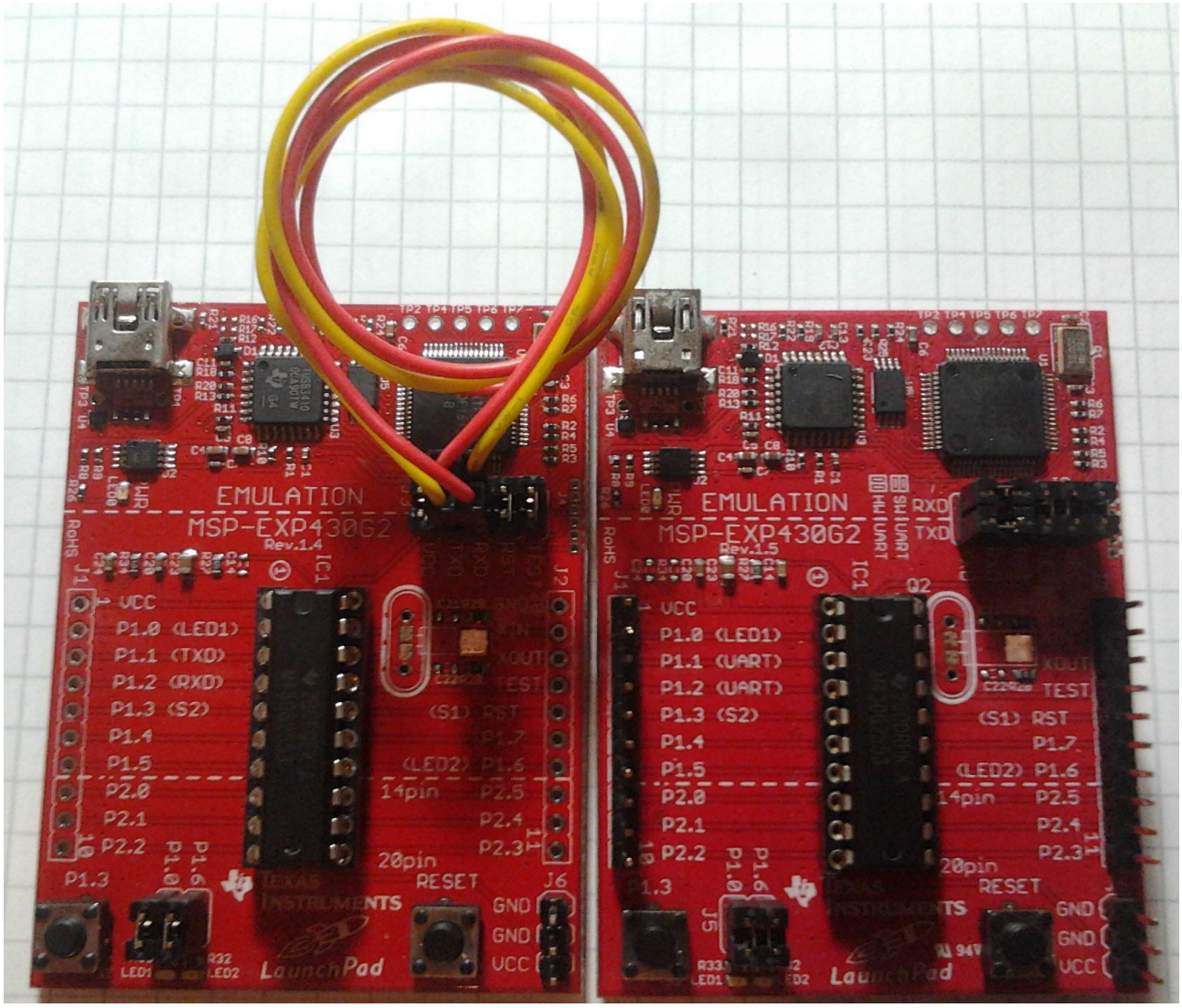
MSP430 geliştirme kartımız ile UART uygulamaları yapabilmek için MSP430 launchpad kartımız üzerinde bulunan USB-UART dönüştürücü birimini kullanacağız. Bu sayede başka ek bir donanım ve bağlantıya gerek olmadan hem debug/program işlemlerimizi yapıp hemde bilgisayar ile denetleyicimizi UART üzerinden haberleştirebileceğiz.

Normal şartlarda USB-UART dönüştürücümüz olmasaydı denetleyicimizi bilgisayar ile haberleştirmek için bilgisayarın seri portunu kullanmamız gerekecekti. Bilgisayar seri portu ile MSP430 denetleyicimizin lojik gerilim seviyeleri farklı olduğundan haberleşme için denetleyici çıkışımıza max232 gibi seviye uyumlaştırıcı entegreleri kullanarak devreler kurmamız gerekecekti. Geliştirme kartımız üzerinde bulunan USB-UART dönüştürücü sayesinde seviye dönüştürme işlemi ile uğraşmaktan kurtulmuş oluruz.

Bilgisayar kısmında denetleyicimizden veri alıp gönderebilmek için basit haberleşme programlarına ihtiyaç duyarız. Bunlardan en bilineni windows xp ile birlikte gelen hyper terminal programıdır. Xp işletim sistemi eski bir sistem olduğu için yeni sistemlerde hyper terminal desteği kalmadığından başka terminal programları kullanmamız gerekmetedir. Biz bu uygulamamızda Terminal v1.9b isimli programı kullanıp haberleşme işlemini gerçekleştireceğiz. Terminal v1.9b programını internetten aratarak kolayca indirebilirsiniz.

RS232(UART) protokolü haberleşme hızı, bit sayısı, parity işlemleri gibi ayarlamalar gerektirmektedir. Bu tanımlamaların uygulamanıza göre avantajları/dezavantajları bulunmaktadır. Bu ayarlamalar tecrübe gerektiren işlemler olduğu için biz uygulamalarımızda bilinen ve genellikle herkes tarafından kullanılan ayarlamaları yapıp uygulamalarımızı gerçekleştireceğiz. Uygulamalarımıza geçmeden önce geliştirme kartımız üzerinde kullandığımız launchpad'in revizyonuna göre değişklik yapmamız gerekecek.

MSP430 Launchpad 1.4 revizyonu çıktığında donanımsal UART özelliği bulunan MSP430G2553 denetleyicisi o zamnalar bulunmadığı için UART işlemleri diğer denetleyiciler ile zamanlayıcı birimleri kullanılarak yarı donanımsal olarak yapılmaktaydı. Bu durumda kullanılan zamanlayıcının çıkışları launchpad üzerinde bulunan USB-UART dönüştürücüye direk bağlı olduğu için sorunsuz haberleşebiliyordu. Fakat o zamanlar yeni çıkan MSP430G2553'ün donanımsal UART pinleri ile USB-UART dönüştürücü pinleri ile ters bağlantılı olduğu için rev 1.4 kartlar üzerinde MSP430G2553'ün donanımsal UART ile USB-UART dönüştürücü üzerinden bağlantı yapabilmek için bir kaç değişiklik yapmak gereklidir. Aynı şekilde ileride üretilen rev 1.5 launchpadlerde bu işlem atlama ayarları ile yapılabilmektedir. Konuyu özetleyen resim şekil 2'de görülebilir.



Şekil 2: MSP430 Launchpad UART AyarlarınınYapılması

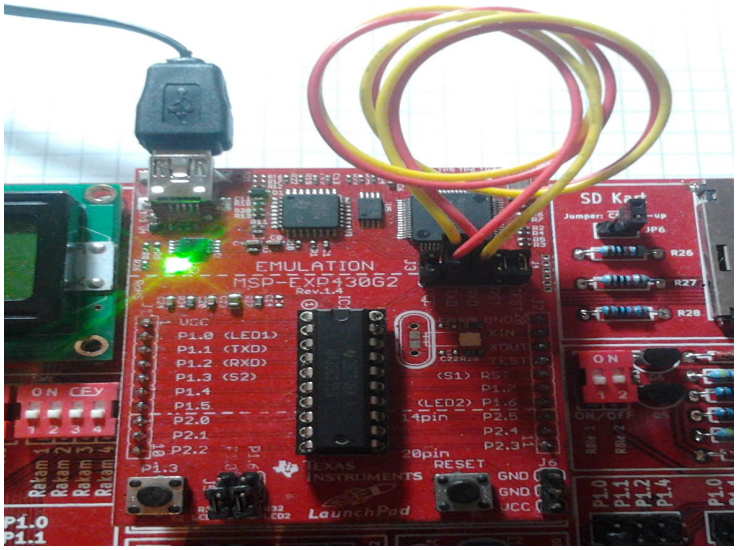
Şekil 2'de MSP430 launchpad rev 1.4 ve rev 1.5 için donanımsal UART kullanmak için gerekli bağlantı görülmektedir. Görüldüğü gibi rev 1.4 eski üretim olduğu için MSP430G2553 kullanıp donanımsal UART uygulaması yapmak için şekil 2'de ki gibi atlamaları 2 kablo ile çapraz bağlamak gereklidir. Benzer şekil rev 1.5 launchpadler de UART atlamalarını HW-UART konumuna getirmek yeterlidir. Kullandığınız kartın revizyonuna göre gerekli atlama ayarlarını yaparak MSP430G2553 denetleyicinizin donanımsal UART özelliğini kullanarak bilgisayar ile haberleşme yapabilirsiniz. Launcpadimizin donanımsal UART ayarlarını yapıp geliştirme kartımıza bağladıktan sonra uygulamalarımıza geçebiliriz.

Uygulama 7.1 UART Uygulaması

Bu uygulamamızda USCI_A0 birimimizin UART çalışma yapısını gösteren basit bir uygulama gerçekleştireceğiz. Yaptığımız uygulama ile bilgisayardan gelen veriyi aynı geldi gibi geri göndereceğiz. Böylece basit bir echo(yankı) uygulaması yapacağız. Uygulamanın kodları aşağıdaki gibidir.

```
#include <msp430.h>                    // MSP430 başlık dosyası
void main(void)
{
  WDTCTL  = WDTPW | WDTHOLD;           // Watchdog timeri durdur.
  DCOCTL = 0;                          // Dahili osilatör ayarlarını en düşük yap
  BCSCTL1 = CALBC1_1MHZ;               // Dahili osilatörü 1MHz'e ayarla
  DCOCTL = CALDCO_1MHZ;                // Dahili osilatörü 1MHz'e ayarla
  P1SEL = BIT1 + BIT2 ;                // P1.1 = RXD, P1.2=TXD
  P1SEL2 = BIT1 + BIT2 ;               // P1.1 = RXD, P1.2=TXD
  UCA0CTL1 |= UCSSEL_2;                // UART Ayarları, Saat kaynağı SMCLK
  UCA0BR0 = 104;                       // 1MHz 9600 bud ayarı
  UCA0BR1 = 0;                         // 1MHz 9600 baud ayarı
  UCA0MCTL = UCBRS0;                   // UART Baud hassas ayar
  UCA0CTL1 &= ~UCSWRST;                // USCI birimini hazırla
  IE2 |= UCA0RXIE;                     // USCI_A0 RX kesmesini aç
  __bis_SR_register(LPM0_bits + GIE);  // Genel kesmeleri aç LPM0 moduna gir
}
//  USCIAAB0 RX Kesme vektörü
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
  while (!(IFG2&UCA0TXIFG));           // TX tamponu hazır olana kadar bekle
  UCA0TXBUF = UCA0RXBUF;               // Alınan karakteri geri gönder
}
```

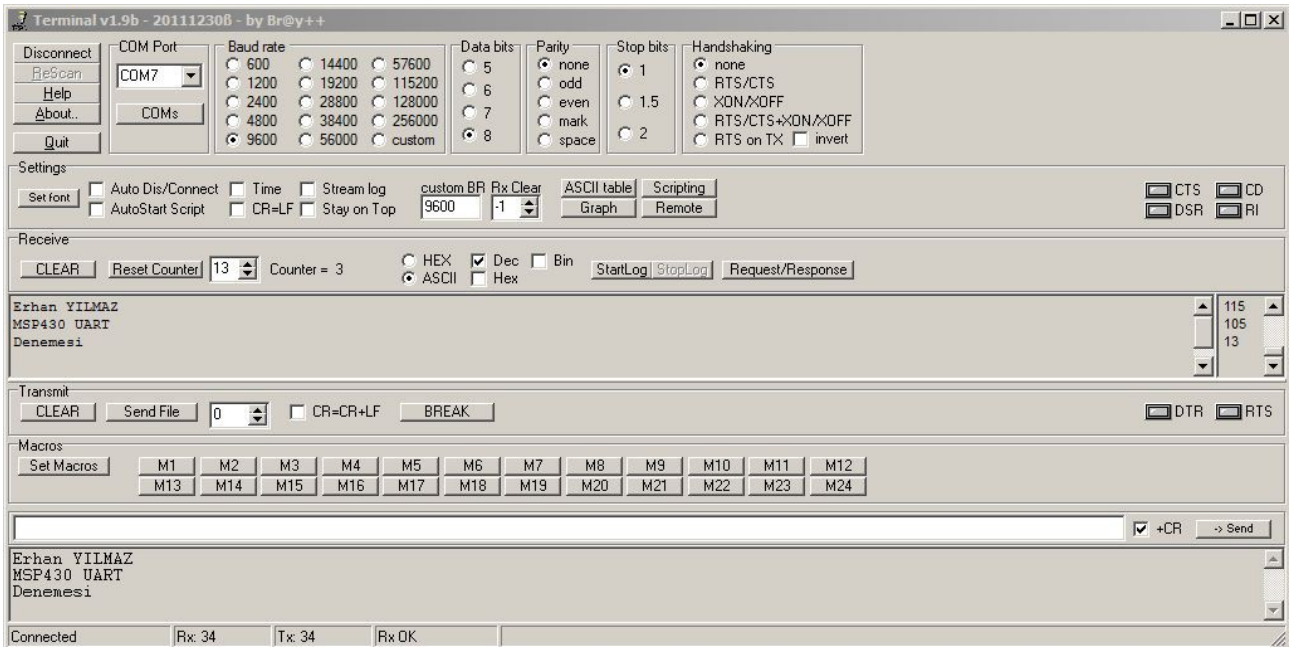
Uygulamanın kodlarından anlaşıldığı gibi yine işlem yapılmayan zamanlarda işlemcimiz tasarruf modundadır. Başlangıçta ilk ayarlar ve USCI birimi için UART ayarları yapılır. UART haberleşme hızı 9600 baud olarak, 8 data biti, parity biti olmaksızın 1 stop biti olarak ayarlanmıştır. Aynı şekilde karşı taraftaki bilgisayarda benzer ayarlar yapılarak haberleşme işlemi başlatılır.



Şekil 3: Uygulama 7.1 Çalışma Görüntüsü

Şekil 3'te uygulamamızın bağlantı yapısı görülebilir. Kodlarımızı derleyip kartımıza yüklemeden önce kart ayarlarını yapmamız gerekmektedir. Kart ayarları için görüldüğü gibi 2 adet kablo(Rev 1.4 için) ile gerekli atlamalar yapılıp kullanılmayan diğer birimlerin anahtarları ile kapatılması yeterlidir.

Gerekli ayarları yapıp kodlarımızı yükledikten sonra herhangi bir sorun yoksa uygulamamız çalışacaktır. Uygulamamızı denemek için Bilgisayar tarafından veri göndermemiz gerekmektedir.



Şekil 4: Terminal v1.9b Programı ve Çalışması

Şekil 4'te terminal programımız üzerinden giden deneme veriler ve MSP430G2553 denetleyicimizden gelen cevaplar görülmektedir. Görüldüğü gibi Launchpad üzerinde bulunan USB-UART dönüştürücümüz kendini COM7 portu olarak tanıttığı için COM7 portu üzerinden bağlantı kurulmuştur. Bağlantı kurulmadan önce haberleşme ayarlarının 9600 baud, 8 bit data, No parity, 1 stop biti şeklinde yapıldığına dikkat edin.

Bu uygulama ile temel anlamda bilgisayar ile denetleyicimiz arasında haberleşme işlemini gerçekleştirmiştik olduk.

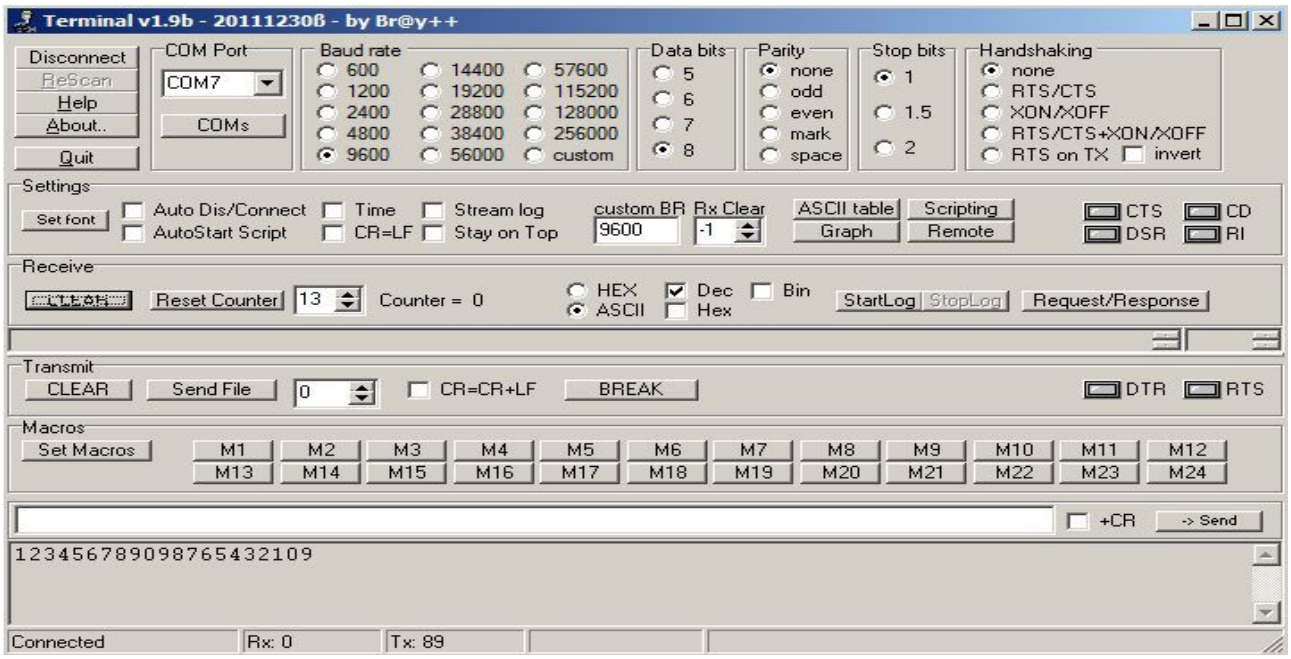
Uygulama 7.2 UART ile Bilgisayar Üzerinden Kontrol

Bu uygulamamızda bilgisayarımızda bulunan klavyeden bastığımız sayılara göre Port 2 üzerinde bulunan ledleri yakıp söndüreceğiz. Uygulama kodları aşağıdaki gibidir.

```
#include <msp430.h>                 // MSP430 başlık dosyası

void main(void)
{
  WDTCTL  = WDTPW | WDTHOLD;        // Watchdog timeri durdur.
  DCOCTL = 0;                       // Dahili osilatör ayarlarını en düşük yap
  BCSCTL1 = CALBC1_1MHZ;            // Dahili osilatörü 1MHz'e ayarla
  DCOCTL = CALDCO_1MHZ;             // Dahili osilatörü 1MHz'e ayarla
  P1SEL = BIT1;                     // P1.1 = RXD
  P1SEL2 = BIT1;                    // P1.1 = RXD
  P2DIR  = 0xff;                    // Port2 çıkış
  P2OUT  = 0x00;                    // Port2 çıkışını sıfırla
```

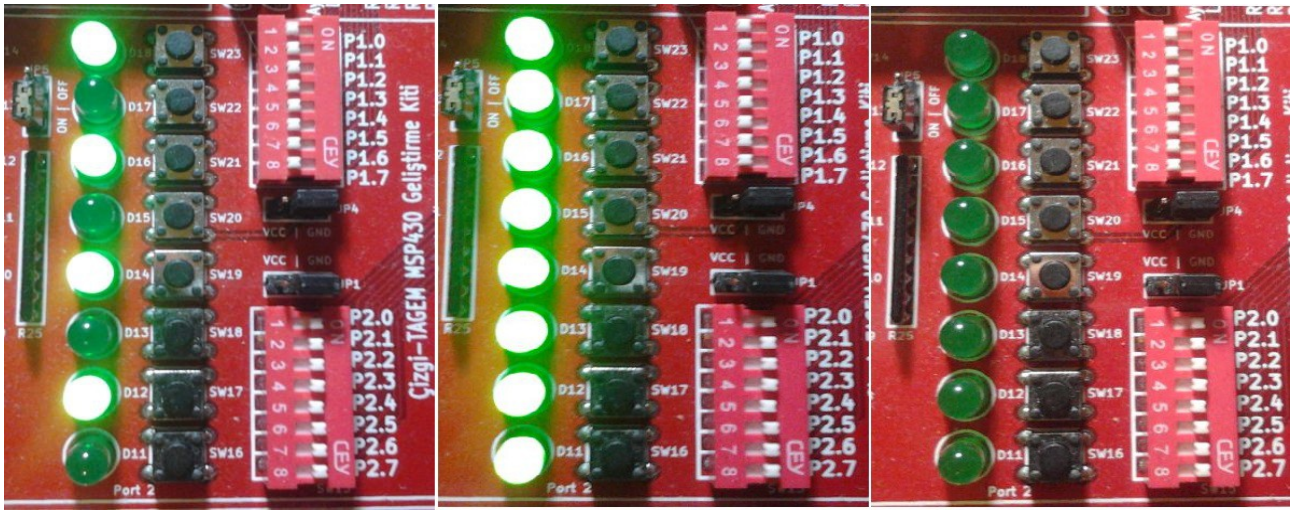
```
  P2SEL  = 0x00;                            // Port2 port işlemleri için kullanılacak
  P2SEL2 = 0x00;                            // Port2 port işlemleri için kullanılacak
  UCA0CTL1 |= UCSSEL_2;                     // UART Ayarları, Saat kaynağı SMCLK
  UCA0BR0 = 104;                            // 1MHz 9600 bud ayarı
  UCA0BR1 = 0;                              // 1MHz 9600 baud ayarı
  UCA0MCTL = UCBRS0;                        // UART Baud hassas ayar
  UCA0CTL1 &= ~UCSWRST;                     // USCI birimini hazırla
  IE2 |= UCA0RXIE;                          // USCI_A0 RX kesmesini aç
  __bis_SR_register(LPM0_bits + GIE); // Genel kesmeleri aç LPM0 moduna gir
}

//  USCIAAB0 RX Kesme vektörü
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
  unsigned char bAlinanKarakter;        // Gelen veriyi tutan değişken
  bAlinanKarakter = UCA0RXBUF;          // Gelen veriyi al
  switch(bAlinanKarakter){              // Gelen veriyi kontrol et
  case '0':                             // 0 ise
    P2OUT = 0x00;                       // Port2 çıkışını sıfırla
    break;
  case '1':                             // 1 ise
  P2OUT ^= BIT0;                        // Port2.0'ı tersle
        break;
  case '2':                             // 2 ise
    P2OUT ^= BIT1;                      // Port2.1'i tersle
    break;
  case '3':                             // 3 ise
    P2OUT ^= BIT2;                      // Port2.2'yi tersle
    break;
  case '4':                             // 4 ise
    P2OUT ^= BIT3;                      // port2.3'ü tersle
    break;
  case '5':                             // 5 ise
    P2OUT ^= BIT4;                      // Port2.4'ü tersle
    break;
  case '6':                             // 6 ise
    P2OUT ^= BIT5;                      // Port2.5'i tersle
    break;
  case '7':                             // 7 ise
    P2OUT ^= BIT6;                      // Port2.6'yı tersle
    break;
  case '8':                             // 8 ise
    P2OUT ^= BIT7;                      // Port2.7'yi tersle
    break;
  case '9':                             // 9 ise
    P2OUT = 0xff;                       // Port2 çıkışını set et
    break;
  }
}
```

Kodları inceleyecek olursak; Başlangıçta ilk ayarlar ve USCI birimi için UART ayarları yapılmaktadır. Bir önceki uygulama ile aynı ayarlar yapılmıştır. Yalnız bu uygulamada veri gönderme olmadığı için TXD pini kullanılmamıştır. Bilgisayardan veri geldiği zaman denetleyici gelen veriyi değerlendirir. ilgili sayılar gelmesi halinde ledlerin durumunu tersler. Örneğin 1 sayısı gelirse Port2.0'a bağlı ledin durumu terslenir. Aynı şekilde 2 sayısı gelirse port2.1'e bağlı ledin durumu terslenir. Bu şekilde ardışıl olarak devam eder. Eğer 0 sayısı gelirse port2'ye bağlı tüm ledler söndürülür, 9 sayısı gelirse port2'ye bağlı tüm ledler yakılır. Bu şekilde ledlerin kontrol yapılır.



Şekil 5: Terminal v1.9b Programı ve Çalışması

Şekil 5'te görüldüğü gibi gerekli ayarlamalar yapılarak terminal programı çalıştırılır ve ledleri kontrol etmek için gerekli sayılar gönderilerek uygulamanın çalışması test edilir.



Şekil 6: Uygulama 7.2 Çalışma Görüntüleri

Uygulama kodlarımızı derleyip geliştirme kartımıza yüklemeden önce Port2'ye bağlı ledleri aktif etmek için jp5 atlamasını ve sw15 anahtarlarını on konumuna getirelim. UART haberleşmesi için bir önceki uygulama ayarları aynen kalmalıdır. Kullanılmayan birimler anahtarları ile kapatılmalıdır. Gerekli ayarlamaları yaptıktan sonra herhangi bir sorun yoksa kodlarımızı denetleyiciye yükledikten sonra uygulama çalışmaya başlayacaktır.

Uygulama çalıştıktan sonra sırasıyla bilgisayarda 0-9 arası rakamlar göndererek port2 üzerinde ki ledlerin her seferinde değiştiğini görebilirsiniz. Bu uygulama ile bilgisayardan gelen veriler doğrultusunda ledlerin durumları değiştirilerek basit genel amaçlı bir kontrol uygulaması yapılmıştır.

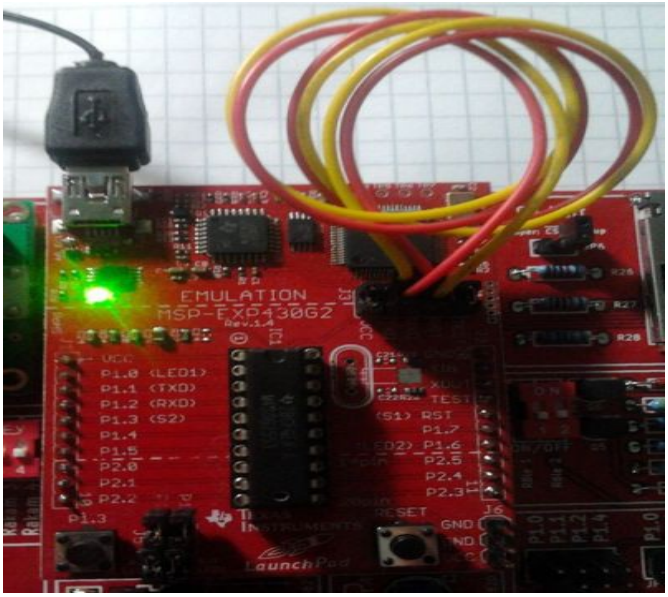
Uygulama 7.3 UART ile Sıcaklık Bilgisi Aktarımı

Bu modül ile ilglili son uygulamamızda MSP430 denetleyicimiz içersinde bulunan sıcaklık sensörü ile sıcaklık değerini ölçüp UART üzerinden bilgisayara göndereceğiz. Uygulama kodları aşağıdaki gibidir.

```
#include <msp430.h>                          // MSP430 başlık dosyası
unsigned short wGecikmeSayac=0;              // Gecikme sayacı
unsigned long wSicaklik;                     // Sıcaklık değerini tutan değişken
unsigned char bTXSayac=0;                    // Giden veri sayacı
unsigned char bYazi[15]={"SICAKLIK=xx°C\n\r"};  // Gönderilecek yazı
void main(void)
{
  WDTCTL  = WDTPW | WDTHOLD;                 // Watchdog timeri durdur.
  DCOCTL = 0;                                // Dahili osilatör ayarlarını en düşük yap
  BCSCTL1 = CALBC1_1MHZ;                     // Dahili osilatörü 1MHz'e ayarla
  DCOCTL = CALDCO_1MHZ;                      // Dahili osilatörü 1MHz'e ayarla
  P1SEL =  BIT2 ;                            // P1.2=TXD
  P1SEL2 = BIT2 ;                            // P1.2=TXD
  UCA0CTL1 |= UCSSEL_2;                      // UART Ayarları, Saat kaynağı SMCLK
  UCA0BR0 = 104;                             // 1MHz 9600 bud ayarı
  UCA0BR1 = 0;                               // 1MHz 9600 baud ayarı
  UCA0MCTL = UCBRS0;                         // UART Baud hassas ayar
  UCA0CTL1 &= ~UCSWRST;                      // USCI birimini hazırla
  TA0CCTL0 = CCIE;                           // Timer0 CCR0 ayarları
  TA0CCR0  = 5000-1;                         // Timer0 kesme periyodu 5ms
  TA0CTL   = TASSEL_2 + MC_2;                // Timer0 ayarları
  ADC10CTL1 = INCH_10 + ADC10DIV_3;          // Sıcaklık sensörü seçilir. ADC10CLK/4
  // ADC10 ayarları, dahili 1.5V referans, kesmeleri aç
  ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10IE;
  __bis_SR_register(GIE);                    // Genel kesmeleri aç
  while(1){                                  // Sonsuz döngü
  __bis_SR_register(LPM0_bits);              // İşlemciyi uykuya sok
  wSicaklik = ADC10MEM;                      // Çevrim sonucunu kaydet
  wSicaklik = ((wSicaklik - 673) * 423) / 1024;  // Çevrim sonucubu santigrat dereceye çevir
  wSicaklik %=100;                           // Sonucun 0-99 arasında olmasını sağla
  bYazi[9] = wSicaklik/10+0x30;              // Sıcaklık değeri birler basamağı(ascii)
  bYazi[10]= wSicaklik%10+0x30;              // Sıcaklık değeri onlar basamağı(ascii)
  IE2 |= UCA0TXIE;                           // Gönderme kesmesini aç
  UCA0TXBUF = bYazi[0];                      // Göndermeye başla
  bTXSayac=1;                                // Gönderme sayacını ayarla
 } }
// USCIAB0TX kesme vektörü
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void)
{
  UCA0TXBUF = bYazi[bTXSayac];               // sıradaki karakteri gönder
```

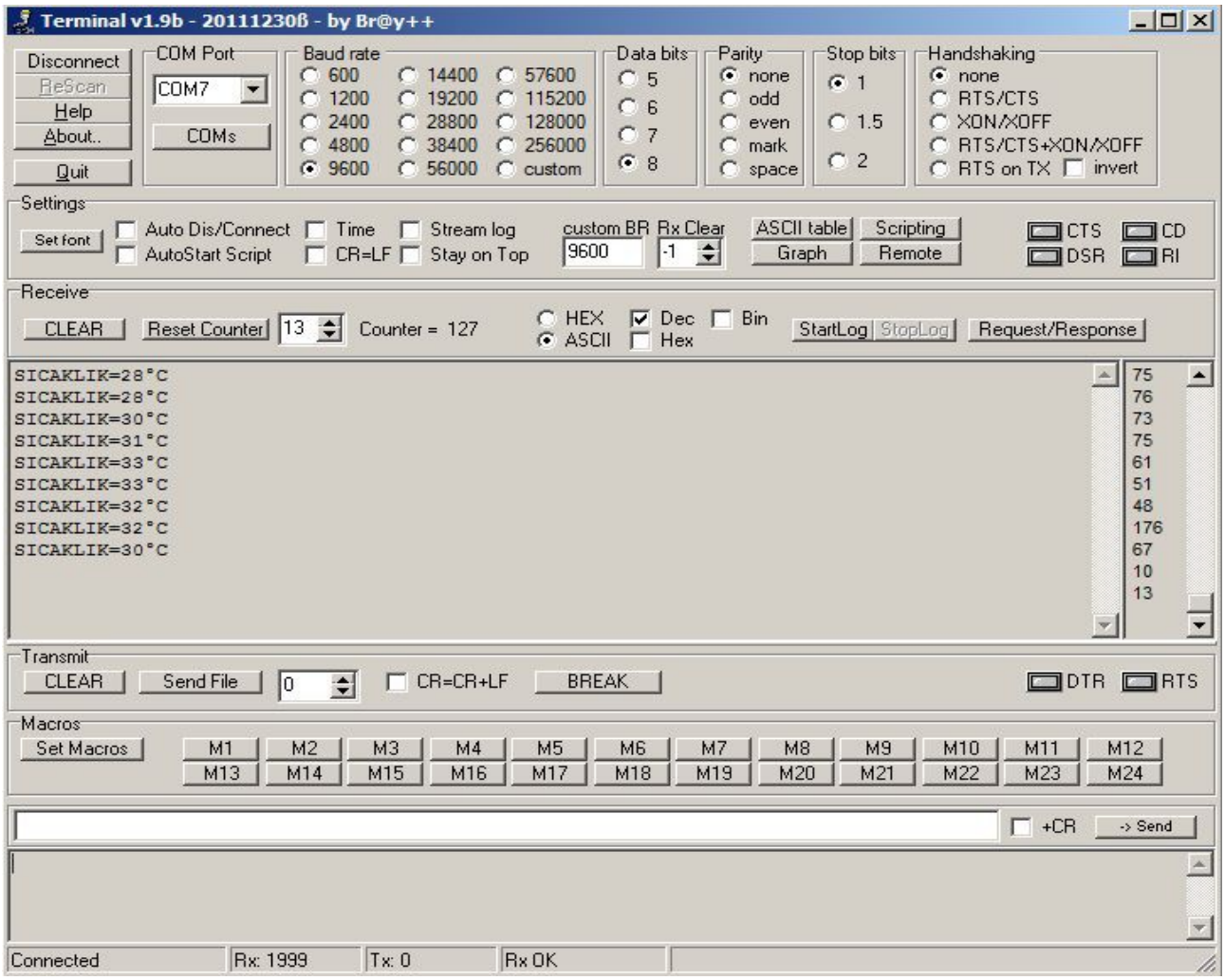
```
  if (++bTXSayac>=15){                    // Tüm karakterler gitti mi?
  IE2 &= ~UCA0TXIE;                       // Evet ise gönderme kesmesini kapat
  bTXSayac=0;                             // Gödnerme sayascını sıfırla
  }
}
// Timer_A0 CCR0 kesme vektörü
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
  if(++wGecikmeSayac>=600){               // 3 saniye oldu mu?
  wGecikmeSayac=0;                        // Zaman sayacını sıfırla
  ADC10CTL0 |= ENC + ADC10SC;             // AD çevrimi başlat
 }
  TA0CCR0 += 5000;                        // Timeri yeniden kur.
}
// ADC10 kesme vektörü
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);      // İşlemciyi uykudan uyandır.
}
```

Kodlardan anlaşılacağı üzere uygulamada işlem yapılmayan zamanlarda işlemci uyutularak güç tasarrufu sağlanmaktadır. Başlangıçta işlemci ilk ayarları yapılır. Sonrasında sırasıyla UART, zamanlayıcı ve ADC ayarları yapılır ve kesmeleri aktif edilir. Sonrasında kesmelere izin verilerek ana döngüye girilir. Ana döngüde işlemci ilk olarak uyku moduna girer. Bu arada her 5 ms'de bir zamanlayıcı kesmesi oluşur. Oluşan kesme sayısı 600'a ulaştığında 3 saniye geçmiş olur. 3 saniye geçince ADC çevrimi başlatılır. ADC işlemi bitince sıcaklık değeri okunur ve okunan sıcaklık değeri bilgisayara gönderilmek üzere basamaklarına ayrılarak ascii karakter haline getirilir ve gönderilecek dizideki yerlerine yazılır. Sıcaklık değeri elde edildikten sonra elde edilen değerin bilgisayara gönderme işlemi başlatılır. Sırasıyla her karakter gittiğinde kesme oluşturulur ve bir sonraki karakter gönderilir. Tüm karakterler gönderildiğinde ise gönderme işlemi tamamlanır işlemci tekrar uyku moduna girer.



Şekil 7: Uygulama 7.3 Çalışma Görüntüsü

Geliştirme kartımızın ayarları bir önceki uygulama ile aynı kalması yeterlidir. Kodlarımızı derleyip şekil 7'de ki gibi bağlantımızı yaptıktan sonra herhangi bir sorun yoksa uygulamamız çalışmaya başlayıp her 3 saniyede bir sıcaklık bilgisini UART üzerinden bilgisayara gönderecektir.



Şekil 8: Uygulama 7.3 Çalışma Görüntüsü

Şekil 8'de görüldüğü gibi bilgisayar tarafında ki ayarlar önceki uygulamada olduğu gibi 9600 baud, 8 bit data, No parity, 1 stop biti şeklinde yapılarak bağlantı sağlandığı takdirde herhangi bir sorun yoksa sıcaklık bilgileri gelecektir. Şekil 8'de görüldüğü gibi sıcaklık bilgileri her 3 saniyede bir satır satır gelmektedir.

Bu sayede ölçtüğümüz sıcaklık değerini bilgisayara aktararak basit bir bilgisayar destekli sıcaklık ölçüm uygulaması yapmış olduk. İstenirse uygulama geliştirilebilir. Aynı zamanda bilgisayar tarafında da çeşitli kontrol uygulamaları geliştirilerek uygulamamız ile birlikte kullanılabilir.

Bu modül kapsamında USCI birminin UART özelliği incelenerek asenkron haberleşme uygulamaları yapılmıştır. Asenkron haberleşme mikrodenetleyici sistemlerde sıkça kullanılan bir haberleşme tipi olduğundan konunun kavranması mikrodenetleyici sistemlerde uygulama geliştirmek adına önemlidir.