Modül 4: MSP430 Timer_A Zamanlayıcısı ve Uygulamaları

Giriş

Zamanlayıcılar mikrodenetleyicilerin en önemli birimlerinden biridir. Aynı şekilde Portlarda olduğu gibi içerisinde zamanlayıcı olmayan denetleyici yoktur diyebiliriz.

Zamanlayıcıların temel işlevi periyodik olarak belirli zamanlarda mikroişlemciyi uyararak zamana bağlı işlemlerin yapılmasını sağlamaktadır. Aynı şekilde zamanlayıcılar sayıcı olarakta kullanılıp girişinde ki darbelerin yada işaretlerin sayısını sayabilirler.

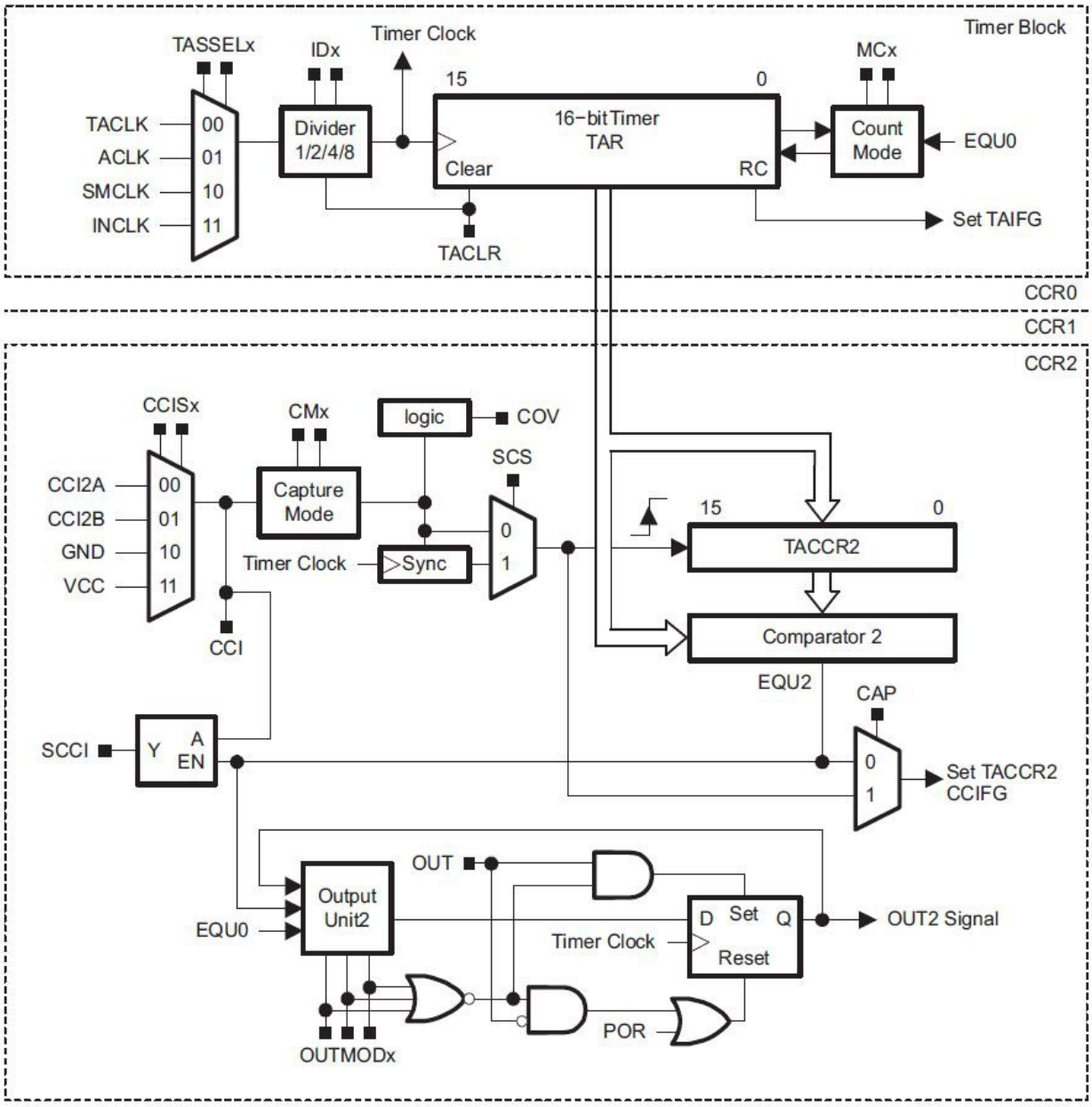
Günümüzde teknolojinin gelişmesiyle birlikte mikrodenetleyicilerde bulunan zamanlayıcılarda gelişmiş bir çok özellikle donatılmıştır. Örneğin MSP430 denetleyicilerde bulunan zamanlayıcılarda, periyodik zaman üretmenin yanı sıra, aşağı-yukarı sayma, darbe üretimi, çoklu kanal gelişmiş PWM üretimi, darbe süresi ölçümü, kenar yakalama vb. bir çok özellik bulunmaktadır. Modül kapsamında Timer_A'nın özelliklerine değinilip çalışması ile ilgili uygulamalar yapılacaktır.

MSP430 Timer_A Zamanlayıcısı

Timer_A içerisinde 3 adet Capture/Compare(Yakala/Karşılaştır) birimi bulunan 16 bitlik zamanlayıcı/sayıcıdır. Timer_A gelişmiş donanımı sayesinde çoklu capture/compare işlemleri, PWM çıkışları ve aralık zaman(periyodik) üretme işlemleri yapabilir. Aynı zamanda bu işlemler sonuçlarına göre kesme üretme özelliğine sahiptir.

Timer_A özellikleri aşağıdaki gibidir.

- Asenkron 16 bit Zamanlayıcı/Sayıcı, 4 çalışma modu
- Değiştirebilen konfigüre edilebilen saat kaynağı
- 2 veya 3 konfigüre edilebilen capture/compare kaydedicileri
- Konfigüre edilebilir PWM yetenekli çıkışlar
- Asenkron girş ve çıkış mandallama(latching)
- Kesmeleri hızlı çözümleyebilmek için kesme vektör kaydedicisi



Şekil 1: Timer_A Birimi Bloak Diyagramı

Şekil 1'de Timer_A biriminin genel yapısı görülmektedir. Şekilde görüldüğü gibi üst kısımda Timer_A'nın ana kısmı bulunmaktadır. Onun altında ise CCR2(Capture/Compare Kaydedicisi 2) biriminin blok diyagramı görülmektedir. CCR0 ve CCR1 birimleride aynı olduğu için tekrar çizilmemiştir.

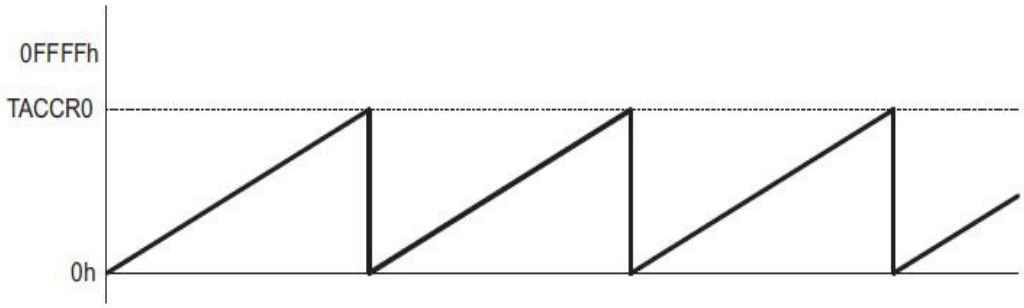
Timer_A Çalışması

Timer_A şekil 1'de görüldüğü gibi bir çok saat kaynağı girişine sahip istenirse bu saat girişi 1, 2, 4, 8'e bölünebilir. Elde edilen bu saat giriş 16 bit TAR kaydedicisini sayma moduna göre saydırır. Taşma durumunda ise kesme oluşur.

Timer_A'nın 4 adet çalışma modu vardır.

Stop Mod: Bu modda Timer çalışmaz.

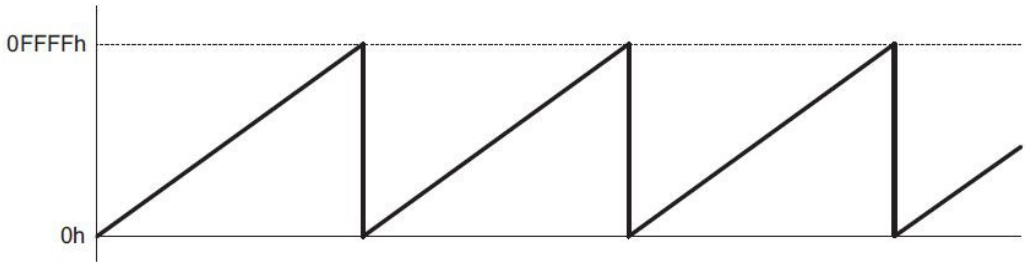
Up Mod: Timer sürekli olarak sıfırdan TACCR0 değişkeninde bulunan değere kadar yukarı sayar.



Şekil 2: Up(Yukarı) Mod

Up modda iken timer TACCR0 değerine geldiğinde CCIFG(Capture/Compare Kesme Bayrağı) bayrağı set edilir, istenirse kesme oluşturulur. Timer 0 değerine geldiğinde ise TAIFG(Timer_A Kesme Bayrağı) bayrağı set edilir. İstenirse kesme oluşturulur.

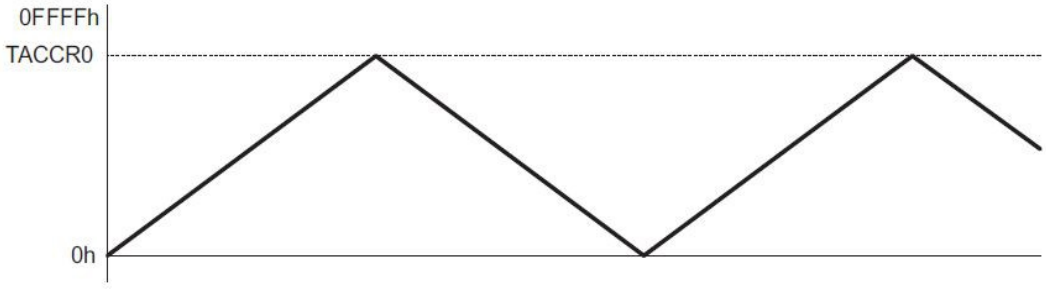
Continuous Mod: Bu modda Timer sürekli olarak 0'dan 0xFFFF değerine kadar sayar.



Şekil 3: Continuous(Sürekli) Mod

Continuous modda iken timer 0 değerine geldiğinde TAIFG bayrağı set edilir. İstenirse kesme oluşturulur.

Up/Down Mod: Bu modda Timer sürekli olarak sıfırdan TACCR0 değerine ileri sayar. Ordan tekrar sıfıra doğru geri sayar.



Şekil 4: Up/Down(Yukarı/Aşağı) Mod

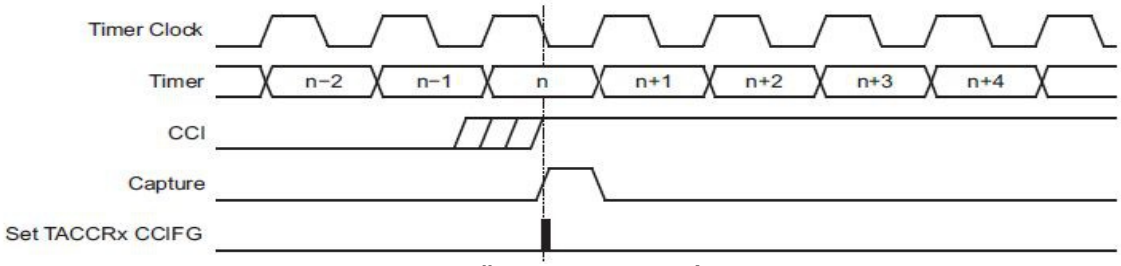
Up/Down modunda iken timer TACCR0 değerine ulaşınca CCIFG bayrağı set edilir. İstenirse kesme oluşturulur. Aynı şekilde timer 0 değerine gelince TAIFG bayrağı set edilir. İstenirse kesme oluşturulur.

Timer_A'nın esnek bir çalışma yapısı olduğu için uygulamanıza göre istediğiniz şekilde çalıştırabilirsiniz. İlk aşamada çalışma modları karışık gibi görünsede uygulamalarda kullanıldıkça daha iyi pekiştirilmektedir.

Capture/Compare Blokları

Capture/Compare bloklarından her biri timerdan veri yakalama yada periyodik zaman üretme için kullanılabilir. Her CC biriminin kendine ait giriş/çıkışı vardır. Bu giriş/çıkışlar denetleyicinin kılavuzunda belirtilmiş port pinleri ile ilişkilendirilmiştir. Detaylı anlamak için ayrı ayrı incelemekte fayda var**.**

Capture Mod: Bu modda timer ayarlandığı gibi çalışmasına devam eder. Çalışma esnasında Capture girişinden ayarlamaya göre düşen kenar, yükselen kenar veya her iki kenardan biri geldiğinde capture birimi yakalama işlemi yapar. Yani TimerA'nın o anki saymakta olduğu değeri CCR kaydedicisine saklar. Aynı zamanda CCIFG bayrağı set edilir. İstenirse kesme oluşturulur.



Şekil 5: Örnek Capture İşlemi

Şekil 5'te capture işlemi daha iyi anlaşılabilir. Capture işlemi sayesinde denetleyici girişine uygulanan işaretlerin darbe genişliklerini hatta periyodlarını bulabiliriz. Capture özelliği sayesinde örnek olarak kızıl ötesi kumanda işaretlerinin algılanması gibi asenkron haberleşme işlemleri yapılabilir.

Compare Mod: Compare modda CCR kaydedicisi belirli bir değere set edilir. Timer normal çalışmasına başlar. Timer sayarken CCR kaydedicisi içerisindeki değere geldiği anda karşılaştırma işlemi sağlanmış olur. Yani TAR kaydedicisi ile CCR kaydedicisinin değerleri eşit olur. Bu durumda CCR birimine bağlı çıkış istenirse Set, Reset, veya Toggle yapılabilir. Aynı zamanda CCIFG bayrağı set edilir istenirse kesme oluşturulur.

Compare biriminin en önemli özelliği PWM üretebilmesidir. PWM üretimi için CCR0 kaydedicisine periyot değeri yazılır. Bu durumda CCR1 ve CCR2 değişkenlerine PWM duty oranları yazılarak istenilen PWM değeri elde edilebilir. 3 CCR kaydedicisine sahip bir Timer_A birimi frekansı aynı 2 adet PWM işareti üretebilir. PWM frekansları timer çalışma frekansı ve CCR0 değeri ile orantılıdır. PWM frekansı arttıkça çözünürlüğü düşmektedir. Aynı durum tersi içinde geçerlidir.

Hangi durumda hangi çözünürlükte yada frekansta PWM kullanılacağı uygulamadan uygulamaya değişmekte ve bunların belirlenmesi tecrübeye bağlıdır. Biz bu modül kapsamında deneme amaçlı belirli modlarda uygulamalar yapıp PWM elde edeceğiz. Timer_A ile ilgili daha detaylı bilgi için kullanma kılavuzu okunup detaylı bilgi sahibi olunabilir. Ayrıca bizim kullandığımız MSP430G2553 denetleyicisinde Timer0_A3 ve Timer1_A3 olmak üzere 2 adet Timer_A bulunduğu için 4 kanala kadar PWM üretebiliriz. Benzer şekilde 6 kanal Capture/Compare işlemi yapabiliriz.

Uygulama 4.1 Periyodik Zamanlayıcı Uygulaması

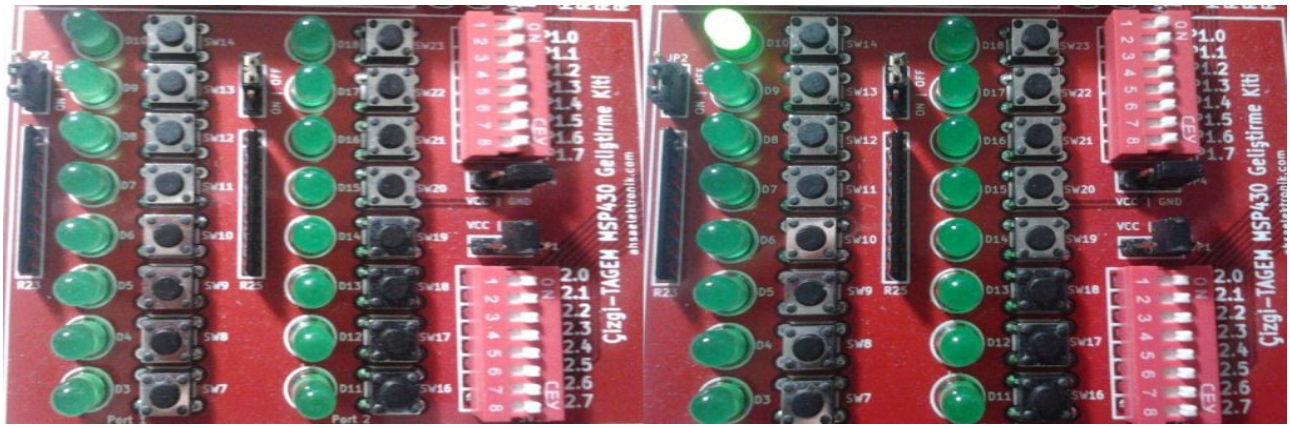
Timer_A ile ilgili bu kadar teorik bilgiden sonra uygulamasını yapmak faydalı olacaktır. Timer_A birimi yapısı itibari ile karmaşık olduğu için teorik olarak anlaşılması zor olabilir. Uygulamalar ile çalışması daha iyi anlaşılabilir.

Bu uygulamada zamanlayıcımızı 50 mili saniyede bir kesme üretecek şekilde ayarlayacağız. Üretilen kesmeleri sayarak her 500 mili saniyede bir Port1.0'a bağlı Ledin durumunu değiştireceğiz. Böylece ledimizi periyodik olarak yanıp sönecektir. Geriye kalan zamanlarda işlemcimizi uyku moduna sokarak güç tasarrufu elde edeceğiz.

Uygulama kodları aağıdaki gibidir.

```
#include <msp430.h>
unsigned char bSayac=0;                  // Kesme sayaç değişkeni
void main(void) {
    WDTCTL = WDTPW | WDTHOLD;            // Watchdog timeri durdur.
    BCSCTL1 = CALBC1_1MHZ;               // Dahili osilatörü 1MHz'e ayarla.
    DCOCTL = CALDCO_1MHZ;                // Dahili osilatörü 1MHz'e ayarla.
    P1DIR |= BIT0;                       // Port1.0 çıkış.
    CCTL0 = CCIE;                        // CCR0 kesmesini aç
    CCR0 = 50000;                        // CCR0 50 ms ayarla.
    TACTL = TASSEL_2 + MC_2;             // Zamanlayıcı ayarları.
    _BIS_SR(LPM0_bits + GIE);            // LPM0 moduna gir kesmeleri aç.
}
#pragma vector=TIMER0_A0_VECTOR          // Timer0_A0 kesme vektörü
__interrupt void Timer_A (void)
{
  if(++bSayac==10){                      // Sayaç 10 olmuş mu?
  P1OUT ^= BIT0;                         // Port1.0'ı tersle.
  bSayac = 0; }                          // Sayacı sıfırla.
  CCR0 += 50000;                         // CCR0'ı tekrar kur.
}
```

Kodları incelersek, başlangıçta watchdog timer ve osilatör ayarları yapılır. Dahili osilatör 1MHz'e kalibre edilerek hassas zaman aralıkları üretilmesi sağlanır. Sonrasında timer ve CCR0 ayarlamaları yapılır. Bu uygulamada timer normal çalışmada CCR0(50000) değerine kadar sayar ve kesme oluşturur. Yani yaklaşık 50ms'de bir kesme oluşur. Oluşan bu kesmeler bSayac değişkeni ile sayılarak 10 olması halinde port1.0'a bağlı ledin durumu değiştirilir. Yani yarım saniye aralıklar ile ledin durumu değiştirilir.



Şekil 6: Uygulama 4.1 Çalışma Görüntüleri

Yeni bir proje oluşturup kodlarımızı projeye dahil edelim. Geliştirme kartı üzerinde kullanmadığımız donanımların anahtarlarının kapalı olduğundan emin olalım. Uygulamayı kart üzerinde çalıştırabilmek için jp2 atlamasını on konumuna getirip sw4 anahtarlarından 1 numaralı anahtarı on konumuna getirelim. Kodlarımızı debug edip işlemciye attıktan sonra Port1.0 çıkışında bulunan ledin yaklaşık yarım saniye aralıklar ile yanıp söndüğünü görürüz. Bu uygulama ile zamanlayıcıların en temel işlemini gerçekleştirmiş olduk.

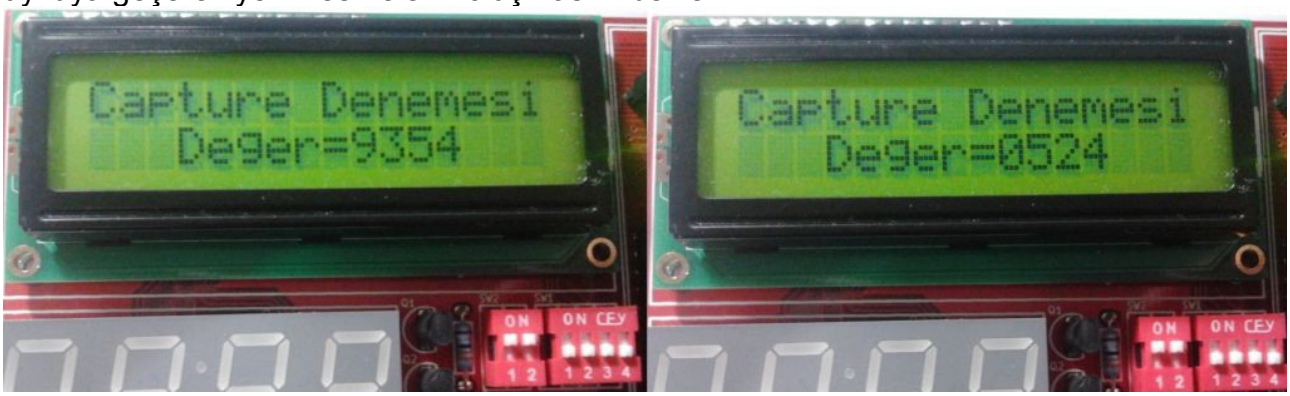
Uygulama 4.2 Capture(Yakalama) Uygulaması

Bu uygulamamızda Timer_A'nın capture(Yakalama) özelliğini içeren bir örnek yapacağız. Timer_A normal olarak up modda 0-9999(CCR0=9999) arası saymaya devam edecek. CCR1 birimi capture moduna alınarak Port1.2(TA0.1) bitinden düşen kenar gelmesini yani butona basılmasını bekleyecek. Düşen kenar oluştuğu anda capture işlemi icra edilerek yakalanan değer LCD ekranda görüntülenecektir. Uygulamanın kodları aşağıdaki gibidir.

```
#include <msp430.h>                          // MSP430 başlık dosyası
#include "LCD.h"                             // LCD başlık dosyası
void BCDCevir(unsigned short);               // Fonksiyon prototipi
unsigned char bBCDDeger[4];                  // BCD değerleri tutan dizi
unsigned short wDeger;                       // Yakalanan değeri tutan değişken
void main(void) {
    WDTCTL  = WDTPW | WDTHOLD;               // Watchdog timeri durdur.
    BCSCTL1 = CALBC1_1MHZ;                   // Dahili osilatörü 1MHz'e ayarla.
    DCOCTL = CALDCO_1MHZ;                    // Dahili osilatörü 1MHz'e ayarla.
    LCD_Ayarla();                            // LCD başlangıç ayarları.
    P1DIR  &= ~BIT2;                         // P1.2 Giriş.
    P1REN  |=  BIT2;                         // P1.2 Direnç aktif.
    P1OUT  |=  BIT2;                         // P1.2 pull-Up.
    P1SEL  |=  BIT2;                         // P1.2 Timer için kullanılacak.
    P1SEL2 &= ~BIT2;                         // P1.2 Timer için kullanılacak.
    TA0CCR0  = 9999;                         // Timer 0-9999 arasında sayacak.
    TA0CTL = TASSEL_2 + MC_1;                // Up mode, SMCLK seçildi.
    TA0CCTL1 = CM_2 + CAP + CCIE;            // Düşen kenar yakalama, kesme aktif.
    _BIS_SR(GIE);                            // Kesmeleri aç
  while(1){                                  // Sonsuz döngü
  _BIS_SR(LPM0_bits);                        // Uykuya gir
    LCD_Git_XY(1,1);                         // İmleci 1.satır 1.sütuna götür.
    LCD_Yazi_Yaz("Capture Denemesi");        // Ekrana Capture Denemesi yazdır.
    LCD_Git_XY(2,4);                         // İmleci 2.satır 4.sütuna götür.
    LCD_Yazi_Yaz("Deger=");                  // Ekrana Deger= yazdır.
    BCDCevir(wDeger);                        // Sayıyı BCD'ye çevir
    LCD_Karakter_Yaz(bBCDDeger[3]);          // Binler basamağını yazdır.
    LCD_Karakter_Yaz(bBCDDeger[2]);          // Yüzler basamağını yazdır.
    LCD_Karakter_Yaz(bBCDDeger[1]);          // Onlar basamağını yazdır.
    LCD_Karakter_Yaz(bBCDDeger[0]);          // Birler basamağını yazdır.
}}
// 0-9999 Arası girilen sayıyı BCD'ye çevirip dizide saklayan fonksiyon
void BCDCevir(unsigned short Sayi){
        unsigned char bSayac;                // Sayaç değişkeni.
        for(bSayac=0;bSayac<4;bSayac++){
        bBCDDeger[bSayac] = Sayi%10 + 0x30;
        Sayi /= 10;
 }}
// Timer_A3 Kesme vektörü (TA0IV Kesme Kaydedicisi)
#pragma vector=TIMER0_A1_VECTOR
__interrupt void Timer_A(void)
{
  switch( TA0IV ){
  case  2:                                   // CCR1 kesmesi
```

```
  wDeger = TA0CCR1;                    // Yakalanan değeri al.
  __bic_SR_register_on_exit(CPUOFF);   // İşlemciyi uykudan uyandır.
  break;
  case  4: break;                      // CCR2 kesmesi.
  case 10: break;                      // Taşma kesmesi.
 }}
```

Timer0_A1 kesme vektörü CCR1, CCR2 ve timer taşma kesmelerinde kullanıldığı için hangi kesmenin geldiğini öğrenmek için kodlarda görünen swtich case yapısını kullanmak gereklidir. Kodları incelersek, görüldüğü gibi kesme kullanılarak yakalama işlemi yapılmıştır. Yakalama işleminin yapılması ardından yakalanan değer wDeger değişkenine kopylanır ve __bic_SR_register_on_exit(CPUOFF) komutu ile işlemci uykudan uyandırılarak ana programa döndürülür. Uygulamada diğer bir önemli kod ise BCDYazdir fonksiyonudur. Bu fonksiyon ile 0-9999 arası girilen sayı 4 basamak olarak BCDye çevrilir bBCDDeger dizisinde saklanır. LCDye yazdırma işlemi bittikten sonra işlemci tekrar uykuya geçerek yeni kesmelerin oluşmasını bekler.



Şekil 7: Uygulama 4.2 Çalışma Görüntüleri

Kodlarımızı yüklemeden önce geliştirme kartımız üzerindeki sw2 anahtarlarını on, jp2 atlamasını off, jp1, jp4 atlamalarını gnd, sw15 anahtalarını off ve sw4 anahtarlarında bulunan 3 numaralı anahtarı on konumuna getirip diğerlerini off konumuna getirmeniz gerekmektedir. Kodlarımızı debug yapıp kartımıza yükledikten sonra ilk başta ekranda hiç bir görüntü olmaması lazım. Sonrasında Port1.2'ye bağlı sw12 butonuna her basışta ekranda şekil 7'de ki gibi görüntüler oluşması gerekir. Uygulamamız her yakalama işlemi geldiğinde 0-9999 arası sayan timerin o anki değerini yakalayarak ekranda görüntüler. Bu sayede capture işlemi yapılmış olur. Capture birimi ile darbe yakalama, süre ölçme gibi işlemler yapılabilir.

Uygulama 4.3 Timer_A PWM Üretim Uygulaması

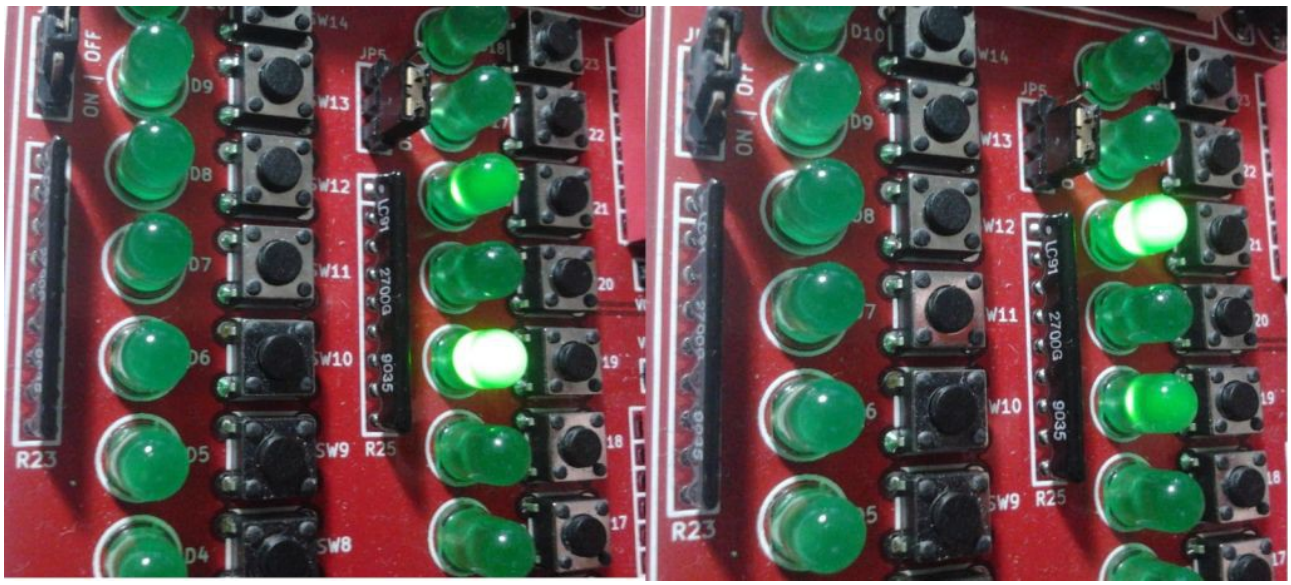
Bu uygulamamızda son olarak Timer_A'nın PWM üretme(Compare) özelliğini göreceğiz. Uygulamada Timer1_A3 ile P2.2 ve P2.4 pinlerinden 2 adet PWM işareti üreteceğiz. Timer0_A3'ü ise belirli bir değere kurup zaman aralıkları oluşturacağız. Oluşturduğumuz bu aralıklarda PWM değerlerini değiştirip PWM çıkışlarına bağlı ledlerin parlaklığını değiştiren led animasyon gibi bir uygulama yapacağız. Uygulama kodları aşağıdaki gibidir.

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

unsigned short wPWM=1000;            // PWM değişkeni, başlangıç değeri 1000
unsigned char bYon=0;                // Yön değerini tutan değişken.

void main(void) {
   WDTCTL  = WDTPW | WDTHOLD;        // Watchdog timeri durdur.
```

```
    BCSCTL1 = CALBC1_1MHZ;          // Dahili osilatörü 1MHz'e ayarla.
    DCOCTL = CALDCO_1MHZ;           // Dahili osilatörü 1MHz'e ayarla.
    P2DIR     |= BIT2 + BIT4;       // P2.2, P2.4 çıkış.
    P2SEL    |= BIT2 + BIT4;        // P2.2, P2.4 timer için kullanılacak.
    TA1CCR0 = 1000;              // Timer1 PWM periyodu 1000+1 cycle yaklaşık 1kHz
    TA1CCTL1 = OUTMOD_7;            // Timer1 CCR1 PWM ayarları
    TA1CCTL2 = OUTMOD_7;            // Timer1 CCR2 PWM ayarları
    TA1CCR1  = 0;                   // Timer1 CCR1 başlangıç değeri
    TA1CCR2  = 0;                   // Timer1 CCR2 başlangıç değeri
    TA1CTL      = TASSEL_2 + MC_1;  // Timer1 ayarları
    TA0CCTL0 = CCIE;                // Timer0 CCR0 ayaları
    TA0CCR0  = 20000;               // Timer0 kesme periyodu ~20ms
    TA0CTL   = TASSEL_2 + MC_2;     // Timer0 ayarları
    _BIS_SR(LPM0_bits + GIE);       // Kesmeleri aç uykuya gir.
}
// TimerA0 CCR0 kesme vektörü
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
  if(bYon)                          // Yön yukarı mı?
  {
          wPWM += 10;               // PWM değerini arttır
          if(wPWM>1000)             // PWM 1000 den büyük mü?
          {
                  wPWM=1000;        // Evet ise PWM değerini 1000 yap
                  bYon=0;           // Yönü değiştir.(Aşağı)
          }
  } else                            // Yön aşağı mı?
  {
         wPWM -= 10;                // PWM değerini azalt.
         if(wPWM == 0)              // PWM 0 oldu mu?
         {
                 wPWM=0;            // Evet ise PWM değerini 0 yap.
                 bYon=1;            // Yönü değiştir. (Yukarı)
         }
  }
  TA1CCR1 =  wPWM;                  // TA1CCR1'e PWM değerini yükle.
  TA1CCR2 =  1000- wPWM;            // TA1CCR2'ye PWM değerinin tersini yükle.
  TA0CCR0 += 20000;                 // TA0CCR0'ı tekrar kur.
}
```

Kodları incelersek, başlangıçta osilatör ve ilk ayarlar yapılır. Sonrasında PWM çıkışı alacağımız P2.2 ve P2.24 pinleri çıkış yapılarak Timer ile ilişkilendirilir. Sonrasında TimerA1 compare kanalları PWM üretecek şekilde ayarlanır. TA0CCR0 değişkeni 1000 yapılarak zamanlayıcının 1000+1 saykılda turunu tamamlaması sağlanır. Bu sayede PWM periyodu 1001 saykıl yani ~1001 us olur. Yani sonuçta 2 kanal içinde PWM frekansı ~1kHz olur.

Sonrasında Timer_A0 CCR0 kanalı kullanılarak yaklaşık 20 ms aralıklar ile kesme üretilir. Her kesme üretiminde PWM kannalarından biri 10 arttırılır diğeri ise benzer şekilde 10 azaltılır. Bu sayede PWM çıkışlarına bağlı ledlerden birisi parlaklığını arttırırken diğer azaltır. İşlem tepeye(PWM=1000) geldiğinde ters işlem başlatılır. Yani tamamen yanık olan led sönmeye tamamen sönük olan led yanmaya başlar. Bu şekilde animasyon halinde ledler yanıp sönerler.



Şekil 8: Uygulama 4.3 Çalışma Görüntüleri

Geliştirme kartı üzerindeki kullanmadığınız birimlerin anahtar vve atlamalarını off konumuna getirdikten sonra jp5 atlamasını on konumuna, sw15 anahtarları üzerinde bulunan 3 ve 5 numaralı anahtarları on konumuna getirelim. Kodlarımızı derleyip debug işlemi ile kartımıza yükledikten sonra uygulama çalışmaya başlayacaktır.

Uygulama çalışmaya başlayınca şekil 8'de görüldüğü gibi D14 ve D16 ledleri P2.2, P2.4 PWM çıkışlarına bağlı olduklarından PWM değerlerinin değişmesi ile ledlerin parlaklıklarıda değişecektir. Bu sayede sırasıyla bir led yavaş yavaş yanarken diğeri yavaş yavaş sönecektir. Ledin biri tamamen yandığında diğeri tamamen sönmüş olacaktır. Sonrasında ters işlem başlayarak diğeri yanıp diğeri sönmeye başlyacaktır. Bu şekilde program çalışmasına devam eder. Bu sayede görsel bir led animasyon uygulaması yapmış oluruz.

Bu modül ile MSP430 denetleyicilerde bulunan TimerA zamanlayıcı/sayıcısını inceleyip çalışması ile ilgili örnek uygulamalar yapılmıştır. TimerA'nın çalışmasının ve iç yapısının iyi kavranması ileri ki modüllerde kullanılan zamanlama işlemlerini anlamada kolaylık sağlar.