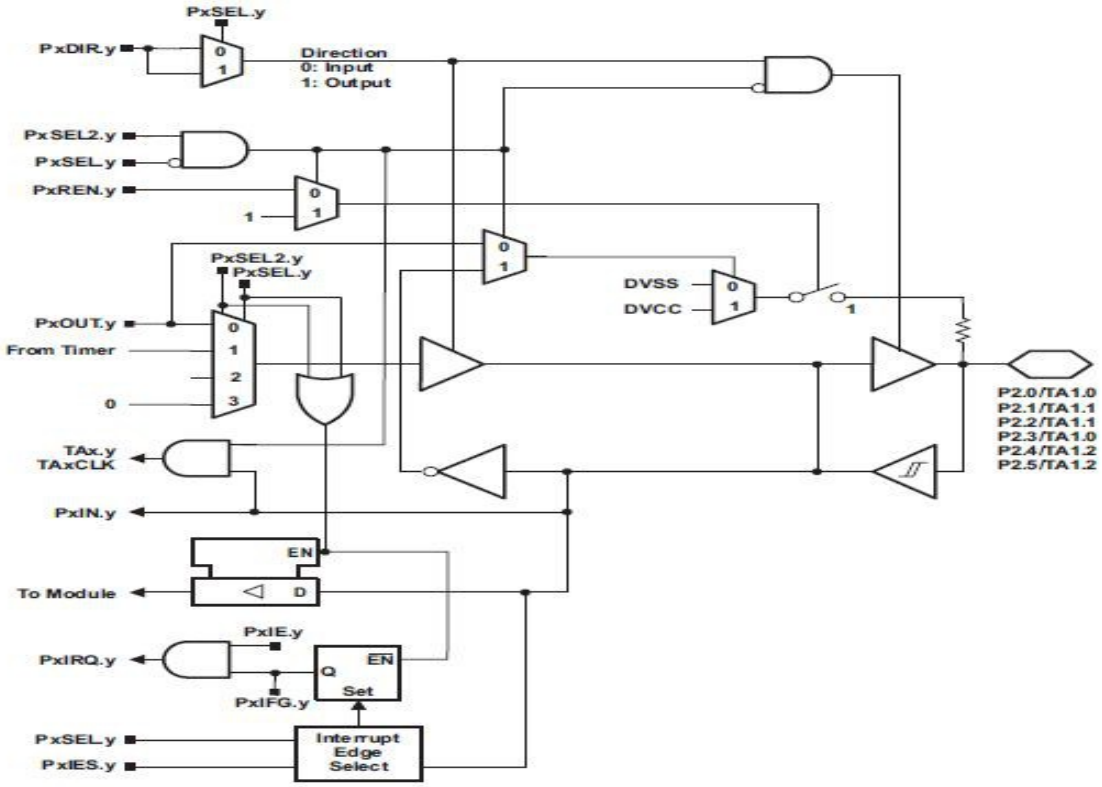
Modül 2: MSP430 Port Yapısı, Uygulamaları, Güç Tasarruf Modları ve Temel Saat Birimi

Giriş

Port, kelime anlamı ile liman yani giriş çıkış yapılan yer anlamına gelmektedir. Portlar mikrodenetleyicilerde de benzer anlam ifade etmektedirler. Portlara mikrodenetleyicilerin Dünya'ya açılan kapıları diyebiliriz. Mikrodenetleyiciden dış Dünyaya bir sinyal göndermek yada dışarıdan bir sinyal almak için portlar kullanılır. I/O (Input/Output), giriş/çıkış, şeklinde de bahsedilebilir. Portları olmayan mikrodenetleyici yoktur diyebiliriz. Mikrodenetleyi üreten firmaların isimlendirmelerine göre portlar, PortA,B,C..., Port0,1,2... GPIO0,1,2... şeklinde isimler alabilmektedir. Hangi mikrodenetleyici olursa olsun portlar büyük oranda benzerlik göstermektedir. Aşağıda genel olarak mikrodenetleyici portlarının özellikleri yer almaktadır.

- Giriş/Çıkış olarak kullanılabilme
- Push/Pull, Open Drain gibi yapılara izin verme
- Sink ve Source olarak akım akıtıp verebilme
- 2-20 mA çıkış akımı verebilme
- Girişinde sinyal değişikliği olunca kesme üretebilme
- Dahili Pull-Up, Pull-Down dirençleri bulunması
- Kesme oluşturabilme

MSP430 Port Yapısı



Şekil 1: MSP430 Örnek Port Yapısı

MSP430 Mikrodenetleyicilerde gelişmiş bir port yapısı bulunmaktadır. Şekil 1'de Port2.0-5 pinlerine ait port iç yapısını görebilirsiniz. Portlar aynı zamanda mikrodenetleyici içerisindeki çevresel birimlerin dış dünyaya ile bağlantı kurmasınıda sağlar. Şekil 1'de bu işlem görülebilir. Örneğin Port2.0-5 pinleri aynı zamanda mikrodenetleyici içerisinde bulunan zamanlayıcı ile ortaklaşa kullanılabilmektedir. MSP430 denetleyicilerin port özellikleri kısaca aşağıdaki gibidir.

- Her pin bağımsız olarak giriş veya çıkış olarak ayarlanabilir
- Port1 ve Port2 pinleri kesme üretebilir
- Birbirinden bağımsız giriş ve çıkışı kaydedicisi
- Ayrı ayrı programramlanabilir Pull-Up, Pull-Down dirençleri
- Bazı MSP430 denetleyicilerde pin osilatör özelliği (Dokunmatik uygulamalar için)

Özelliklerinden ve iç yapısından görüldüğü gibi MSP430'un gelişmiş ve esnek bir port yapısı vardır.

Bu modüller kapsamında Portlar dahil diğer çevresel birimlerin teorik olarak çalışmasını, kaydedicilerinin özelliklerini anlatmak yerine çevrsel birimlere ait özellikleri geliştirme kartı üzerinde bulunan uygun donanımlar ile birleştirip kart üzerinde uygulamak daha faydalı olacaktır. Modüllerin tümünde bu şekilde bir yol izlenip MSP430 işlemciler hakkında uygulama becerisi aktarılamaya çalışılmıştır. Okuyucunun temel C programlama ve elektronik bilgisi olduğu varsayılmaktadır.

MSP430 denetleyiciler hakkında teorik bilgi edinmek için, bizim uygulamalarımızda kullandığımız MSP430G2553'ü de kapsayan **MSP430x2xx Family user Guide.pdf** adlı dokümanı TI'in sitesinden indirip inceleyebilirsiniz. Dili ingilizce olmakla birlikte yaklaşık 650 sayfadır.

Bu açıklamadan sonra Port işlemlerini anlatan örnek uygulamalarımıza geçelim.

Uygulama 2.1 Port Okuma Yazma

Bu uygulamada en temel port okuma/yazma uygulaması yapılmıştır. Port1 çıkış olarak ayarlanmış ve Port1'e bağlı ledler port çıkışının durumunu göstermektedir. Benzer şekilde Port2 giriş olarak tanımlanmıştır. Port2'ye veri girmek için Port2'ye bağlı butonlara basmak yeterlidir. Uygulama kodları aşağıdaki gibidir.

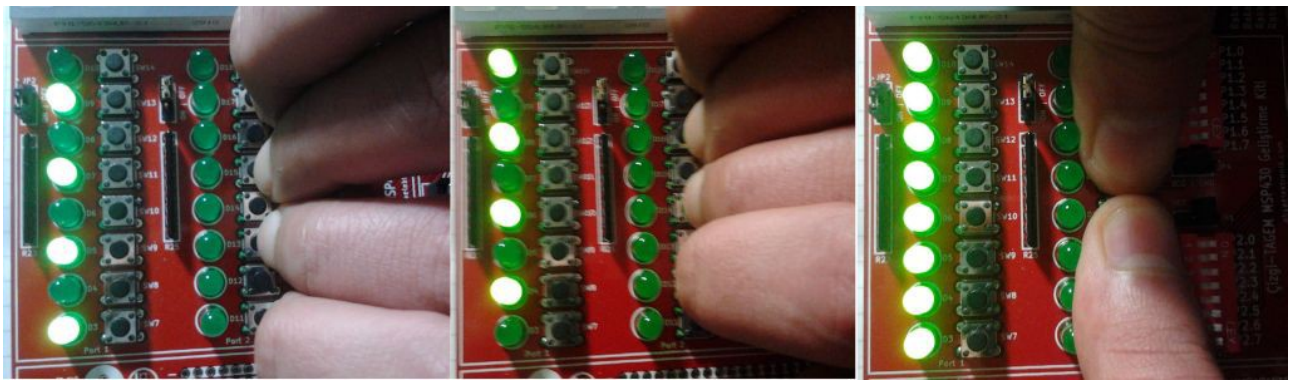
```
#include <msp430.h>

void main(void) {
   WDTCTL = WDTPW | WDTHOLD;    // Watchdog timeri durdur.
   P1DIR = 0xff;                // Port1 tamamı çıkış.
   P2DIR = 0x00;                // Port2 tamamı giriş.
   P2REN = 0xff;                // Port2 dirençleri aktif et.
   P2OUT = 0x00;                // Port2 Pull-Down.
   P2SEL = 0x00;                // Port2 I/O olarak kullanılacak.
   P2SEL2= 0x00;                // Port2 I/O olarak kullanılacak.
   while(1){                    // Sonsuz döngüye gir.
   P1OUT = P2IN;                // Port2'yi oku Port1'e yaz.
   }
}
```

Kodları yeni bir proje oluşturup derleyelim.

Kodları derledikten sonra debug işlemine geçmeden geliştirme kartı üzerindeki atlamaları ayarlayalım. Bu uygulama için jp2 atlaması on, jp5 atlaması off, jp4 atlaması VCC'ye bağlanmalı ve jp1 atlaması boşta bırakılmalıdır. Aynı zamanda sd kart kısmında bulunan jp6 atlamasıda boşta bırakılmalıdır. Aynı zamanda sw4 ve sw15 anahtarları on konumuna getirilmelidir

Geliştirme kartı üzerinde bir çok donanım paylaşımlı olarak aynı portları kullandığı için atlama ayarlarına dikkat etmek gerek. Kart üzerindeki SW kodlu anahtarlar ilgili donanımları açma/kapama içindir. Kullanmadığınız donanımların anahtarını kapatmak alakasız sorunların oluşmasını önler.

Bu genel açıklamadan sonra uygulamamıza geri dönelim. Projeyi derledikten sonra debug işlemini başlatıp Resume butonuna basalım. Her hangi bir sorun yoksa kodlar denetleyiciye atılıp debug işlemi başlayacaktır.



Şekil 2: Uygulama 2.1 Çalışma Görüntüleri

Şekil 2'de görüldüğü gibi uygulama çalıştırıldığında Port2'de hangi butona basılırsa ona karşılık gelen Port1'de ki ledi yakmaktadır. Uygulamanın amacı temel port okuma yazma işlemine örnek olmasıdır.

Kodlar incelenirse, başlangıçta port ayarları yapılıp sonrasında ana döngüye girilerek sürekli olarak Port2'den okunan değer Port1'e yazılmaktadır.

Programda bazı kısımları açıklamak gerekirse P2REN Port2'ye bağlı Pull-Up, Pull-Down dirençlerini aktif eden kaydedicidir. Hangi biti set edilirse(1 yapılırsa) Port2'nin ilgili bitinin direnci aktif olur. Direnci pull-Up yada Pull-Down olarak seçmek için ise yine aynı portun P2OUT kaydedicisini kullanmak gerekir. P2OUT'un hangi biti set edilirse (1 yapılırsa) Port2'in ilgili bitinin Pull-Up özelliği olur. Reset edilirse(0 yapılırsa) Pull-Down özelliği aktif olur. Bu işlemler Port2 giriş olarak ayarlandığında geçerlidir. Çıkış olarak ayarlanırsa P2OUT kaydedicisi normal işlevinde çalışır. Ayrıca P2SEL kaydedicisi ise Portun diğer çevresel birimleri ile olan bağlantılarını kontrol eder.

Port pinlerinin aynı zamanda denetleyicinin diğer çevre birimleri ile paylaşımlı kullandıldığına değinmiştik. P2SEL = 0x00; ve P2SEL2= 0x00; komutları ile Port2'nin tamamı port işlemleri yapacak şekilde ayarlanmıştır. Port1'de aynı şekilde varsayılan olarak port işlemlerine ayarlı olduğu içinde benzer kodları onun içinde yazmaya gerek yoktur. İlerleyen uygulamalarda çevre birimlerinin kullanılmasıyla port işlemleri daha iyi anlaşılabilir.

Port2 için anlatılan bu işlemler diğer portlar içinde benzerdir.

Uygulama 2.2 Buton Kullanımı

Sıradaki uygulamamız ile buton kullanımını gerçekleştireceğiz. Yaptığınız tasarımlarda bulunan buton girişlerini Portlara bağlayarak okuyabilir ona göre programınızın akışını yönlendirebilirsiniz.

Buton kullanım örneğine ilişkin kodumuz aşağıdadır.

```
#include <msp430.h>
unsigned short wSayac;                          // Sayaç değişkeni
void main(void) {
   WDTCTL = WDTPW | WDTHOLD;                    // Watchdog timeri durdur.
   P1DIR |=  BIT0;                              // Port1.0 çıkış.
   P2DIR &= ~BIT0;                              // Port2.0 giriş.
   P2REN |=  BIT0;                              // Port2.0 direnci aktif
   P2OUT |=  BIT0;                              // Port2.0 girişi Pull-Up
   while(1){                                    // Sonsuz döngü
   if(!(P2IN & BIT0)){                          // Butona basıldı mı?
   for(wSayac=0;wSayac<1000;wSayac++)           // Buton arkını önlemek için bekle
   while(!(P2IN & BIT0));                       // Butonun bırakılmasını bekle
   P1OUT ^=  BIT0;                              // Port1.0 bitini tersle
          }
        }
  }
```

Şekil 3: Uygulama 2.2 Çalışma Görüntüleri

Şekil 3'de uygulamaya ait çalışma resimleri görülmektedir. Uygulamayı çalıştırabilmek için jp2 on, jp5 off, jp4 gnd ve sw4, sw15 anahtarları üzerinde bulunan 1 numaralı anahtarlar on konununa getirilmelidir. Debug işlemini yapıp kodları attıktan sonra Port2.0'a bağlı SW23 butonuna her basıp bırakışta Port1.0'a bağlı D10 ledinin durum değiştirdiğini görebilirsiniz.

Uygulamanın kodları basit olmasına rağmen bir çok önemli detayı içermektedir. Örneğin bu programla beraber değişken kullanılmaya başlamıştır. Değişkenler her programlama dilinde olduğu gibi içerisinde sayısal değerleri tutarlar. Programımızda wSayac isminde 16bit Sayaç değişkeni tanımlanmıştır. Değikenin ilk harfi 'w' word tipinde olduğunu kalan kısmı ise kullanıldığı alanla ilgili bir isimdir. Yani wSayac isminde değişken gördüğümüzde bunun word tipinde bir Sayaç değişkeni olduğunu çıkarabiliriz. Aynı şekilde byte,float double, pointer gibi türdeki değişkenler içinde değişken ismi önüne benzer b, f, d, p ön harfleri getirilebilir. Bu sayede programın anlaşılırlığı artar daha rahat kod yazabiliriz.

Kod kısmında ise ^, |, & gibi önemli bit operatörleri bulunmaktadır. Portların bitlerini ayrı ayrı kullanmak istediğimizde bu operatörleri kullanmak kaçınılmazdır. Kodlar üzerinden örnek verelim.

Örneğin P1DIR |= BIT0 komutu ile P1DIR değişkeni BIT0(0x01) sayısı ile lojik or işlemine tabi tutulup sonuç P1DIR değişkenine yazılır. P1DIR değişkenine rasgele bir değer verip elimize kağıt kalem alıp değişkenleri binary sayı şeklinde or işlemine tabi tutarsak sonuçta P1DIR değişkeninin 0. bitinin 1 olduğunu görebiliriz. Yani her hangi bir X değişkeninin y. bitini set etmek için X |= BITy; şeklinde bir komut kullanmamız yeterli. Bu sayede aynı değişken içerisinde bulunan diğer bitlerin durumları değişmez.

Aynı şekilde P2DIR &= ~BIT0; komutuda P2DIR değişkeninin ~BIT0 sayısı(0xFE) and işlemine tabi tutarak sonucu P2DIR değişkenine yazar. Burda tilda (~) operatörü BIT0 sayısını tersler yani sayı içindeki 1'ler 0, 0'lar 1 olur. Sonuçta BIT0(0x01) sayısı tersleme işleminden sonra 0xFE olur. Sonuç olarak bu komut ile P2DIR değişkeninin 0. biti sıfırlanır. Aynı şekilde herhangi bir X değişkeninin y. bitini 0 yapmak için X &= ~y; komutu yeterlidir.

Son olarak P1OUT ^= BIT0; komutuda P1OUT değişkeninin 0. bitinin durumunu tersler. Yani 0 ise 1, 1 ise 0 yapar. Yani herhangi bir X değişkeninin y. bitini terslemek için

X ^= BITy; komutunu kullanmak yeterlidir. 3 komutta da değişkenlerin diğer bitlerinin durumları etkilenmez olduğu gibi kalır. Sonuç olarak bu komutlar ile istediğimiz bir değişkenin istediğimiz bitlerini set,reset edebilir veya tersleyebiliriz.

Bit operatörleri gerek C dilinde gerek ise mikrodenetleyicilerde vazgeçilmezidir. Bit operatörlerinin iyi anlaşılması konuya daha iyi hakim olmak adına önemlidir.

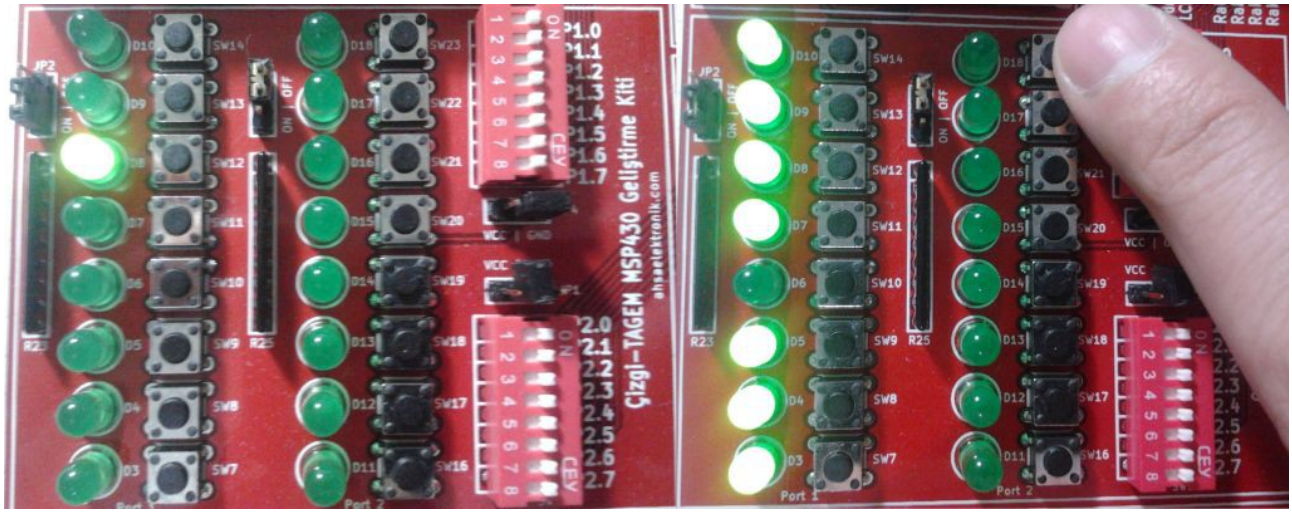
Uygulama 2.3 Kara Şimşek

Bu uygulamızda oldukça bilinen kara şimşek uygulamasını gerçekleştireceğiz. Yani Port1'e bağlı ledleri belirli bir düzende kaydıracağız. Aynı zamanda Port2.0 bitine bağlı buton ile ledlerin durumuna müdahale edeceğiz.

Uygulamanın kodları aşağıdaki gibidir.

```
#include <msp430.h>
unsigned char  bSayac;                  // Sayac değişkeni
unsigned short wGecikme;                // Gecikme değişkeni
unsigned char  bKaraSimsek;             // Kara şimşek değişkeni
void main(void) {
   WDTCTL = WDTPW | WDTHOLD;    // Watchdog timeri durdur.
   P1DIR  =  0xff;                      // Port1 bütün bitler çıkış.
   P2DIR &= ~BIT0;                      // Port2.0 biti giriş.
   P2REN |=  BIT0;                      // Port2.0 direnci aktif.
   P2OUT |=  BIT0;                      // Port2.0 direnci Pull-Up.
   bKaraSimsek = 0x01;                  // Kara şimşek başlangıç değeri.
   while(1){                            // Sonsuz döngü
   for(bSayac=0;bSayac<7;bSayac++){ // 7li for döngüsü
   bKaraSimsek <<= 1;                   // Değişkeni 1 bit sola ötele.
   if(P2IN & BIT0)                      // Buton basılı mı?
   P1OUT =  bKaraSimsek;                // Değilse değeri terslemeden çıkışa aktar.
   else
   P1OUT = ~bKaraSimsek;                // Basılı ise değeri tersleyip çıkışa aktar.
   for(wGecikme=0;wGecikme<10000;wGecikme++);// Ledlerin görünmesi için bir süre
bekle.
        }                               // For döngüsü sonu.

   // 7li for döngüsü
   for(bSayac=0;bSayac<7;bSayac++){
```

```
    bKaraSimsek >>= 1;                  // Değişkeni 1 bit sağa ötele.
    if(P2IN & BIT0)                     // Buton basılı mı?
    P1OUT =  bKaraSimsek;               // Değilse değeri terslemeden çıkışa aktar.
    else
    P1OUT =  ~bKaraSimsek;              // Basılı ise değeri tersleyip çıkışa aktar.
    for(wGecikme=0;wGecikme<10000;wGecikme++);// Ledlerin görünmesi için bir süre
bekle.
    }                                   // For döngüsü sonu.
    }                                   // Sonsuz döngü sonu.
    }                                   // Program sonu.
```



Şekil 4: Uygulama 2.3 Çalışma Görüntüleri

Şekil 4'de uygulama 2.3'e ait çalışma resimleri görülmektedir. Uygulamayı çalıştırabilmek için jp2 atlamasını on, jp5 atlamasını off, jp4 atlamasını GND, sw4 anahtalarını on ve sw15 anahtarları üzerinde bulunan 1 numaralı anahtarı on konumuna getirip debug işlemini başlatalım. Program çalışmaya başlayınca sw23 butonuna basılmaz ise her seferinde Port1'de ki ledlerden biri yanarak sağa ve sola kaymaya devam eder. Port2.0'a bağlı sw23 butonuna basılırsa bu seferde port1'e bağlı sadece bir led sönük kalır ve sağa, sola kaymaya devam eder. Yani buton basılmadan önceki led durumlarının tersi görülür. Bu sayede eğlenceli bir uygulama yapmış olduk.

Uygulama kodlarının önemli kısımlarına değnelim. Bir önceki uygulamada olduğu gibi **if**(P2IN & BIT0) komutu buton durumunu kontrol eder. Yani P2IN giriş değerlerini tutar. Port2.0 bitinin değeri P2IN kaydedicisinin 0. bitinde bulunur. P2IN BIT(0x01) sayısı ile and(&) işlemine tabi tutulursa sonuç P2IN kaydedicisinin 0. bitine bağlı olara 1 yada 0 olur. Yani butona basılıp basılmadığını if sorgusu ile P2IN değişkenine BIT0 sayısı ile and işlemi uygulayarak öğrenebilir. Aynı şekilde herhangi bir X değişkeninin y. bitinin 1 olduğunu sorgulamak için if(X & BITy) 0 olduğunu sorgulamak içinse if(!(X & BITy)) komutunu kullanmak yeterlidir. Burada C programlama dilinden bilindiği gibi ! İşareti tersi yani değili anlamında kullanılır.

Diğer bir komut ise **for**(wGecikme=0;wGecikme<10000;wGecikme++); komutudur. Bilindiği gibi for komutu belirli sayıda tekrarlanan işlemler için kullanılır. Görülüdüğü gibi bizim for döngümüz 10000 kere tekrarlanmaktadır. For işlemi MSP430 assembly komut karşılığı olarak yaklaşık 5 cycle'da işlenmektedir. Yani for komutu çalışınca 10000*5 50000 cycle kadar işlemci boşta bekler. Şuana kadar olan uygulamalarda işlemci saat frekansı hakkında bir şey belirtmedik.

Programda aksi belirtilmez ise MSP430 işlemcimiz varsayılan olarak yaklaşık 1-1.2MHz hızında olan dahili osilatör ile çalışır. Bu durumda 1 cycle süremiz yaklaşık 1us olur. Yani

sonuç olarak işlemcimiz yaklaşık 50ms ledlerin kaymasını gözlemleyebilmek için bekletir. Bu şekilde bekleme işlemleri profesyonelce olmasada özellikle yeni başlayanlar tarafından çokca kullanılmaktadır.

Uygulama 2.4 Kesme Kullanımı

Modül ile ilgili bu son uygulamamızda port kesmesine değineceğiz. Uygulama 2.2'de ki işlemi bu sefer kesme kullanıp yapacağız. Uygulamaya ait kodlar aşağıdaki gibidir.

```
#include <msp430.h>
unsigned char bGecikme;                          // Gecikme değişkeni.
void main(void) {
    WDTCTL = WDTPW | WDTHOLD;                    // Watchdog timeri durdur.
    P1DIR  |=    BIT0;                           // Port1.0 çıkış.
    P2DIR  &= ~BIT0;                             // Port2.0 giriş.
    P2REN |=   BIT0;                             // Port2.0 direnci aktif.
    P2OUT |=   BIT0;                             // Port2.0 direnci Pull-Up.
    P2IE    |=   BIT0;                           // Port2.0 kesmesini aç.
    P2IES   |=   BIT0;                           // Düşen kenar kesmesini seç.
    P2IFG   |=   BIT0;                           // Kesme bayrağını temizle.
    _bis_SR_register(LPM4_bits + GIE);           // LPM4 moduna gir ve kesmeleri aç
}
// Port 2 kesme vektörü
#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void)
{
  P1OUT ^= BIT0;                                 // Port1.0'ı tersle.
  while(!(P2IN & BIT0));                         // Buton bırakılana kadar bekle.
  for(bGecikme=0;bGecikme<250;bGecikme++); // Buton arkını önlemek için biraz bekle.
  P2IFG &= ~BIT0;                                // Kesme bayrağını temizle.
}
```

Anahtar ve atlama ayarlarını değiştirmeden kodlarımızı işlemciye atıp programı çalıştıralım. Uygulama 2.2'de olduğu gibi Port2.0'da ki butona her basıp bırakma işleminde Port1.0'a bağlı ledin durumunun değiştiğini görürüz.

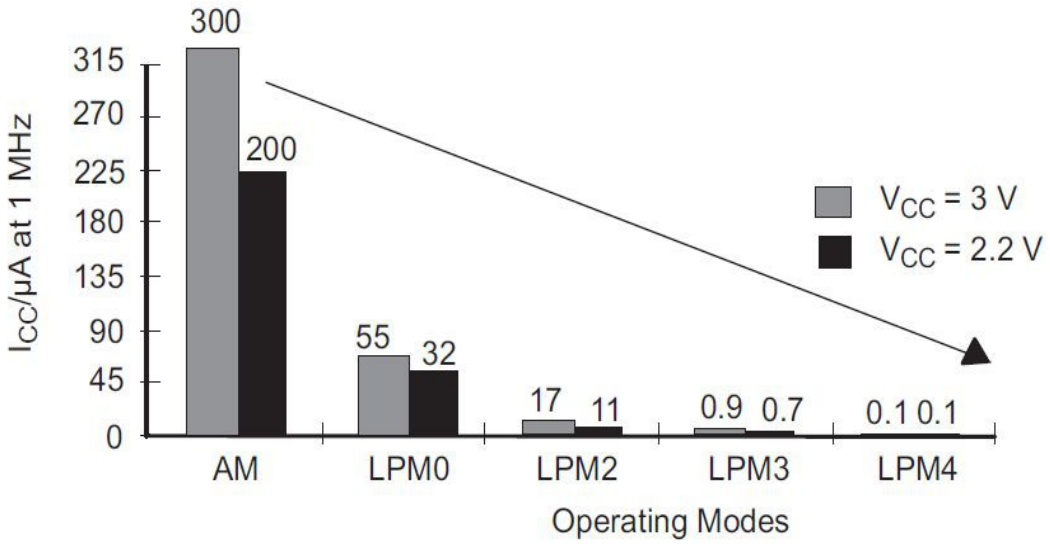
Kodları incelersek; giriş çıkış işlemlerinden sonra Port2.0 girişi düşen kenarda kesme oluşturacak şekilde ayarlanıp **_bis_SR_register**(LPM4_bits + GIE); komutu ile Genel kesmeler izin verilip LMP4 güç tasarruf moduna geçilir. LPM4 modunda işlemcinin tüm saat kaynakları kapatılır ve uykuya geçer. Bu modda güç tüketimi çok azalır. İşlemcinin tekrar uyanması için kesme veya reset gibi kaynaklardan tetiklenmesi lazım. Uygulamamızda butona basma işlemi kesme üreterek işlemciyi uykudan çıkarır ve kesme programını çalıştırır.

Kesme rutininde ise Port1.0 ledinin durumu değiştirilir, butonun bırakılmasını bekledikten sonra bir süre for döngüsü ile buton arkını önlemek için beklenir. Sonrasında yeni kesmelerin oluşumuna izin vermek için kesme bayrağı temizlenip kesme programına son verilir ve işlemci tekrar uyku moduna girer.

Güç tasarruf modları MSP430 denetleyicilerinin çok avantajlı bir özelliğidir. Esnek bir yapıya sahip olduğundan uygulamanıza göre istenilen mod seçilerek güç tasarrufu sağlanabilir. Modülün devamında güç tasarruf modlarına değinilmiştir.

MSP430 Güç Tasarruf Modları

Güç tasarruf modları MSP430 denetleyicilerin uygulamaya göre daha az güç tüketmesini sağlamak üzere tasarlanmıştır. Bu modlar sayesinde denetleyicimiz boşta olduğu zamanlarda uykuya geçirilebilir böyle güç tüketimi asgari seviyeye indirilir. Aynı şekilde kesme veya reset gibi tetiklemeler ile hızlı bir şekilde uyku modundan çıkıp işlemlerine kaldığı yerden devam eder. Aşağıda şekil 5'te örnek olarak MSP430F21x1 serisi denetleyicilere ait tasarruf modlarının akım tüketim oranları görülebilir.



Şekil 5: F21x1 denetleyicilere ait tipik akım tüketimleri

Şekil 5'te görüldüğü AM (Active Mode) yani aktif mod hariç LPM0, LPM2, LPM3, LPM4 olmak üzere denetleyicimizin 4 tane güç tasarruf modu bulunmaktadır. Birde grafikte görünmeyen LPM1 modu vardır. LPM0 ile ufak bir farkı vardır. Grafiktende anlaşılacağı üzere aktif çalışma moduna göre güç tasarruf modları çok büyük oranlarda tasarruf sağlayabilmektedirler.

Uygulamanıza özgü güç tasarruf modunu kullanmak için güç tasarruf modlarının özelliklerini bilmek gerekir. Şekil 6'da tabloda güç tasarruf modlarının özellikleri açıklanmıştır.

SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled, SMCLK, ACLK are active
0	1	0	1	LPM1	CPU, MCLK are disabled. DCO and DC generator are disabled if the DCO is not used for SMCLK. ACLK is active.
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO are disabled. DC generator remains enabled. ACLK is active.
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO are disabled. DC generator disabled. ACLK is active.
1	1	1	1	LPM4	CPU and all clocks disabled

Şekil 6: MSP430 Güç Tasarruf Modları

Şekil 6'da görüldüğü gibi işlemcinin Status kaydedicisinde bulunan SCG0, SCG1,

OSCOFF ve CPUOFF bitlerini değiştirerek istenilen tasarruf moduna geçilebilir.

Modları açıklayacak olursak;

Aktif Mod: İşlemci ve tüm saat kaynakları açık, güç tüketimi azami seviyededir.

LPM0: İşlemci ve MCLK saati kapalı SMCLK ve ACLK açık. Güç tüketimi aktif moda göre daha düşüktür.

LPM1: İşlemci ve MCLK saati kapalı, dahili DCO osilatör ve DC üreteç SMCLK için kullanılmadıysa kapalı. ACLK açık.

LPM2: İşlemci, MCLK, SMCLK ve DCO kapalı. DC üreteç açık kalır. ACLK aktif. Güç tüketimi LPM0'a göre daha düşüktür.

LPM3: İşlemci, MCLK, SMCLK ve DCO kapalı. DC üreteç kapalı kalır. ACLK aktif. Güç tüketimi LPM2'ye göre daha düşüktür.

LPM4: İşlemci ve tüm saat kaynakları kapalı. Güç tüketimi asgari seviyededir. Güç tüketimi LPM3'e göre daha düşüktür.

Uygulamanızda kullandığınız saat kaynakları, çevresel birimler ve uygulamanızın çalışma yapısına göre istediğiniz tasarruf modunu kullanabilirisiz. Başlangıçta biraz karışık gelsede tasarruf modlarını kullandıkça kullanımını daha iyi anlayabilir ve daha az güç tüketen programlar yazabilirsiniz.

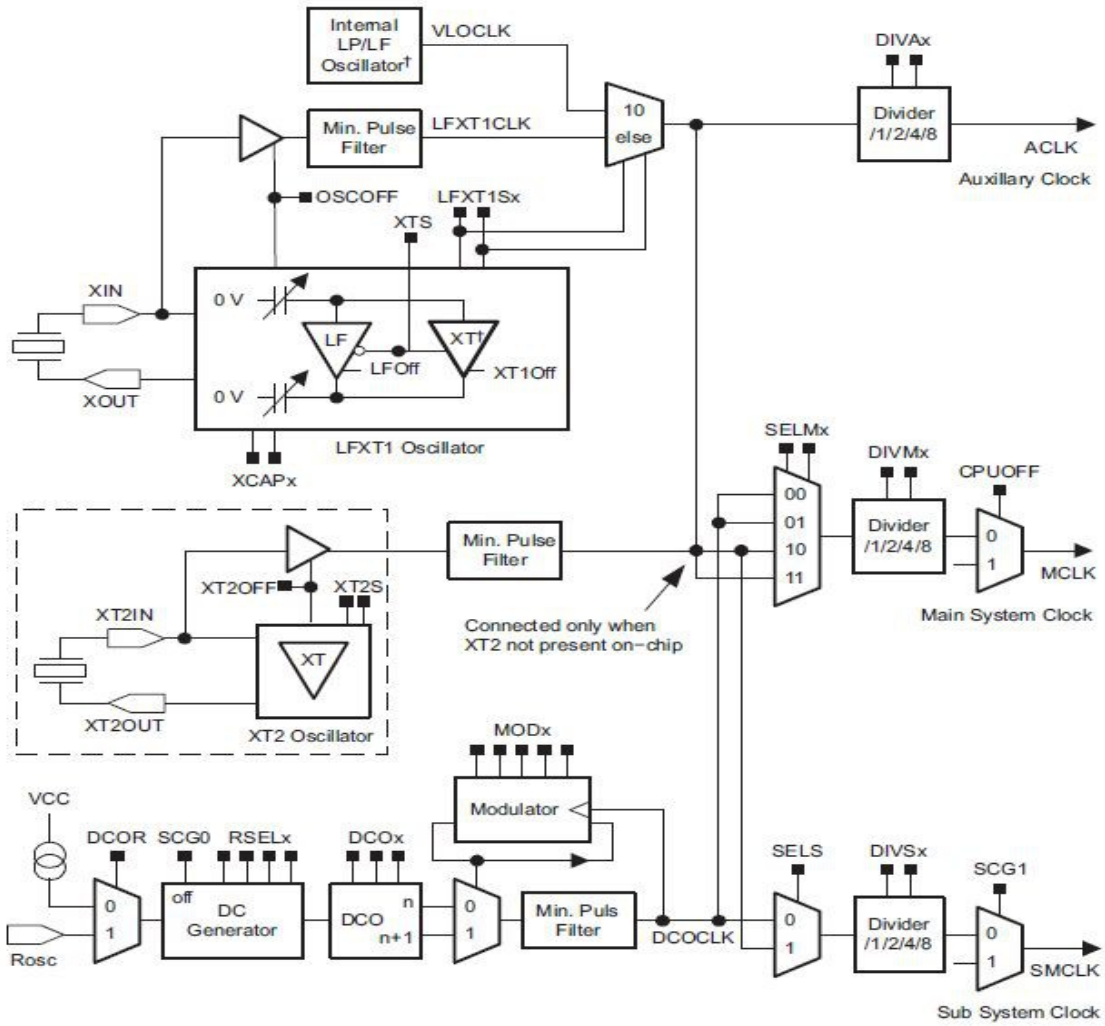
MSP430 Temel Saat Birimi

MSP430 denetleyicilerde tasarruf modları kadar önemli ve birlikte kullanılan diğer bir birimi ise temel saat birimidir. Yeri geldiği için bu modülde her iki (tasarruf modları ve saat birimi) birim hakkında açıklama yapmakta fayda var. Bu birimlerin diğer modüllerde kullanıldığında zorlanmamak için burada anlatılıp kavranmasında fayda var.

Basic Clock Module+ diye bilinen temel saat birimi MSP430 denetleyicilerin gelişmiş saat kaynaklarını ve bu kaynakların bağlandığı saat kanallarını içerir.

MSP430 temel saat birimi gelişmiş, esnek bir yapıya sahiptir. Bu sayede gerek işlemci gerekse diğer çevresel birimleri farklı frekansta ve farklı saat kaynakları ile kullanabiliriz. Bu özellik ile uygulamalarımızın saat ihtiyaçlarını daha kolay elde edebiliriz.

Temel saat biriminin blok diyagramı şekil 7'deki gibidir.



Şekil 7: MSP430 Temel Saat Birimi – MSP430F2xx

Şekil 7'de MSP430F2xx serisi denetleyicilere ait temel saat birimi blok diyagramı görülmektedir. Diyagram aynı zamanda MSP430G2xx denetleyiciler içinde geçerlidir. Diyagram genel bir diyagram olduğu için MSP430 serisine ait bir çok denetleyiciyi kapsamaktadır. Kullandığınız denetleyicinin saat kaynaklarını tam olarak bilmek için referans kılavuzunu inceleyebilirsiniz.

Bizim uygulamalarda kullandığımız MSP430G2553 denetleyicisinde 4 adet saat kaynağı bulunmaktadır. Bunlar aşağıdaki gibidir.

- 16 Mhz'e kadar dahili osilatör, kalibre edilmiş 4 frekans değeri
- Dahili çok düşük güçlü ve düşük frekanslı osialtör (~12kHz)
- 32 kHz kristal osilatör
- Harici saat kaynağı

Denetleyicimizde Rosc(Harici Dirençli Osilatör), XT2 osilatör bulunmamaktadır ve LFXT1 osilatörü HF(Yüksek Frekans) özelliği bulunmamaktadır.

Yukarıda listedekiler denetleyicimizin saat kaynaklarını oluşturmaktadır. Diyagramda görüldüğü gibi birde bu saat kaynaklarının denetleyiciye dağıtıldığı saat kanalları bulunmaktadır.

Bu kanallar MCLK(Main Clock/Ana Saat), SMCLK(Sub Main Clokck/Alt Ana Saat) ve ACLK(Auxiliary Clock/Yardımcı Saat) saat kanallarıdır. Denetleyicimiz içinde bulunan işlemci ve saat kaynağı gerektiren tüm çevresel birimler saat kaynaklarını bu kanallar üzerinden karşılarlar.

MCLK: MCLK saat kanalı işlemcinin saat işaretini sağlayan kanaldır. Diğer çevresel birimlerede saat kaynağı sağlayabilir. Bu kanala giriş olarak denetleyicinin tüm saat kaynakları bağlanabilir.

SMCLK: SMCLK saat kanalı ana sistem kanalının bir alt seviye kanalı olarak düşünülebilir. Saat kaynağı gerektiren tüm çevresel birimlere saat kaynağı sağlayabilir. Bu kaynağa giriş olarak denetleyicinin tüm kaynakları bağlanabilir.

ACLK: ACLK saat kanalı yardımcı saat kanalıdır. Düşük güç tüketimi yada hassas zamanlama gerektiren uygulamalarda kullanılabilir. Denetleyicinin çevresel birimlerine saat kaynağı sağlayabilir. Bu kanala giriş olarak dahili düşük frekanslı osilatör(VLOCLK) yada düşük frekanslı harici kristali osilatör(LFXT1CLK) bağlanabilir.

Diyagramdan görüldüğü üzere saat kanalları çıkşlarında bulunan bölücü devreler sayesinde saat frekanslarını 1,2,4,8 oranında bölebilirler. Bu sayede denetleyicimiz için çeşitli frekans değerleri elde edebiliriz.

Özetleyecek olursak MSP430G2553 denetleyicimizin çeşitli saat kaynakları ilgili saat kanallarına bağlanarak hatta istenirse bu kanal çıkışları bölünerek işlemcimiz ve çevresel birimler için çeşitli saat frekansları sağlanabilir. Tüm bu işlemler Temel Saat Birim ile ilgili kaydediciler ile yapılmaktadır. İstenirse referans kılavuzu incelenerek detaylı bilgi edilinebilir. Uygulama özelliklerine göre yeri geldikçe gerekli ayarlar yapılarak istenilen saat kaynakları elde edilmektedir.

Görüldüğü üzere MSP430 denetleyicileri saat kaynağı konusunda bize bir çok kombinasyon ve esneklik sağlamaktadır. Başlangıçta karmaşık görünmesine rağmen uygulamalarda kullandıkça faydası görülmektedir.

Bu modül kapsamında MSP430 port yapısına ait port yapısı, güç tasarruf modları ve temel saat birimine yer verilmiştir. MSP430 denetleyicilerin en temel birimleri olan bu birimler diğer modüllerdeki uygulamalarda da sıkça kullanılacağından bu birimlerin çalışmasını kavramak diğer uygulamalarda kolaylık sağlayacaktır.