Modül 1: Code Composer Studio V5 Kurulumu, Proje Oluşturma ve Debug İşlemleri

Giriş

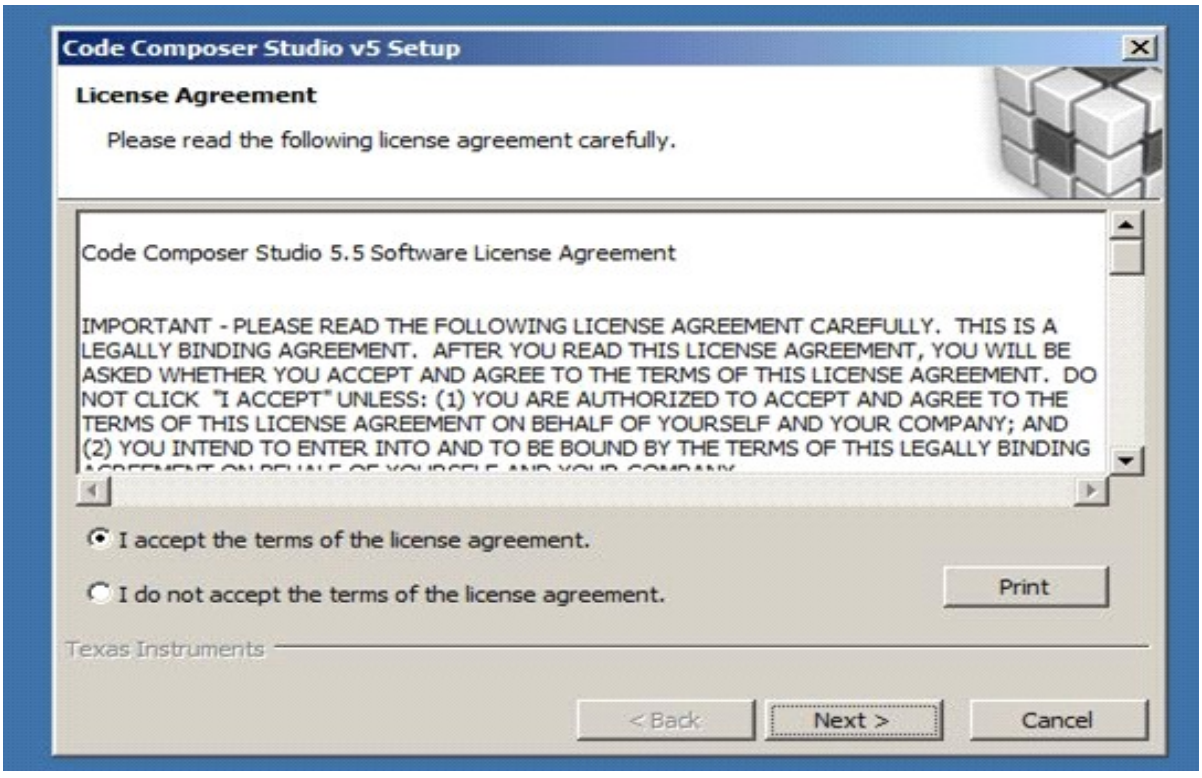
Tümleşik geliştirme ortamları genel olarak programın yazıldığı, derleme, bağlama işlemlerinin yapıldığı, programlama, debug ve simülasyon işlemlerine izin veren bir ortam olarak tanımlanabilir. Son zamanlarda popülerliği artmıştır. Mikrodenetleyicilerden, masaüstü programlama, mobil programlama gibi tüm programlama işlemleri için birçok geliştirme ortamları bulunmaktadır.

Code Composer Studio (CCS) Texas Instruments firmasının kendi işlemci ve denetleyicileri için geliştirdiği (IDE) tümleşik geliştirme ortamıdır.

CCS IDE Kurulumu

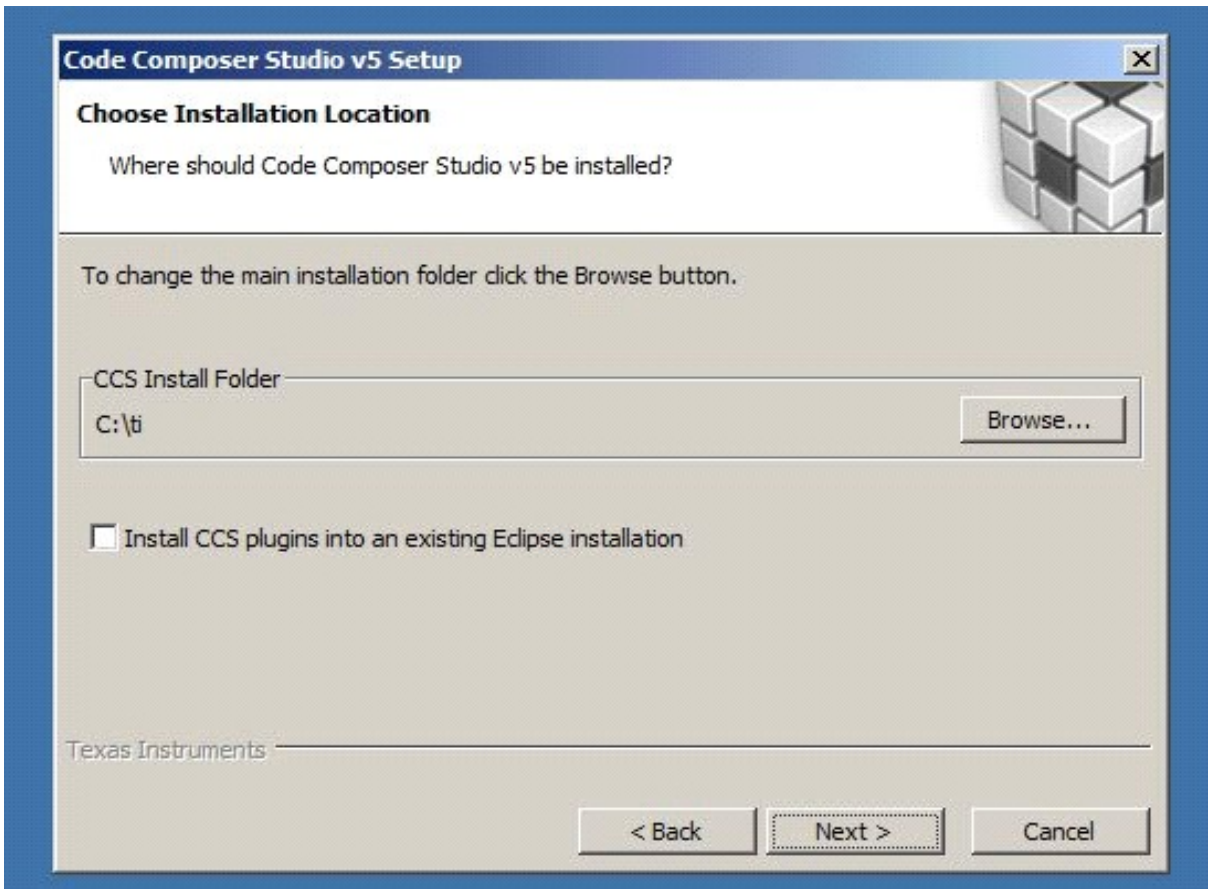
http://www.ti.com/tool/ccstudio-msp430 Adresinden MSP430 için CCS geliştirme ortamının en son sürümü indirebilirsiniz. İndirme işlemi için üyelik gerekmektedir. Kısa bir işlem ile ücretsiz üye olabilirsiniz.

İndirdiğiniz kurulum dosyasını çalıştırıp aşağıda resimlerde görüldüğü gibi seçenekleri seçerek kurulumu başlatabilirsiniz. Kurulum işlemi internet üzerinden gerçekleştiği için biraz uzun sürmektedir.



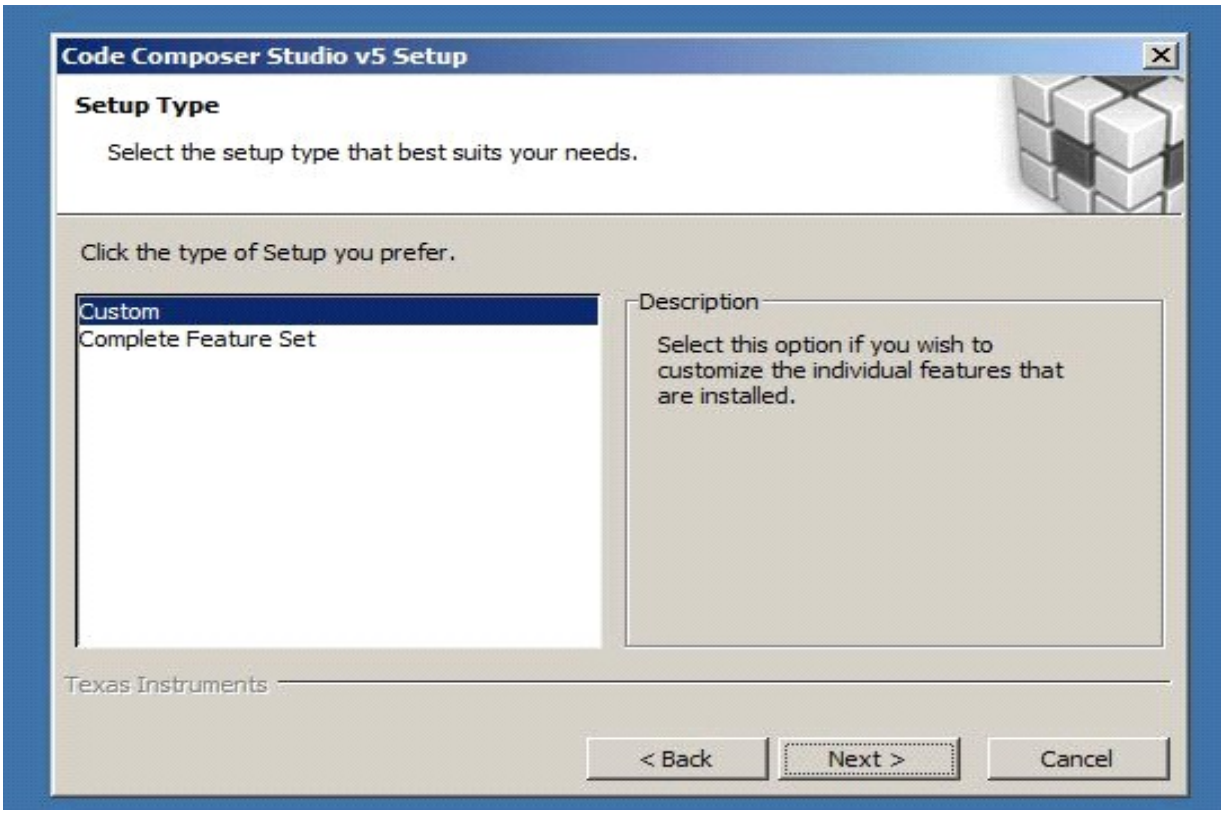
Şekil 1: CCS IDE Kurulum

Koşulları kabul edip Next diyoruz.



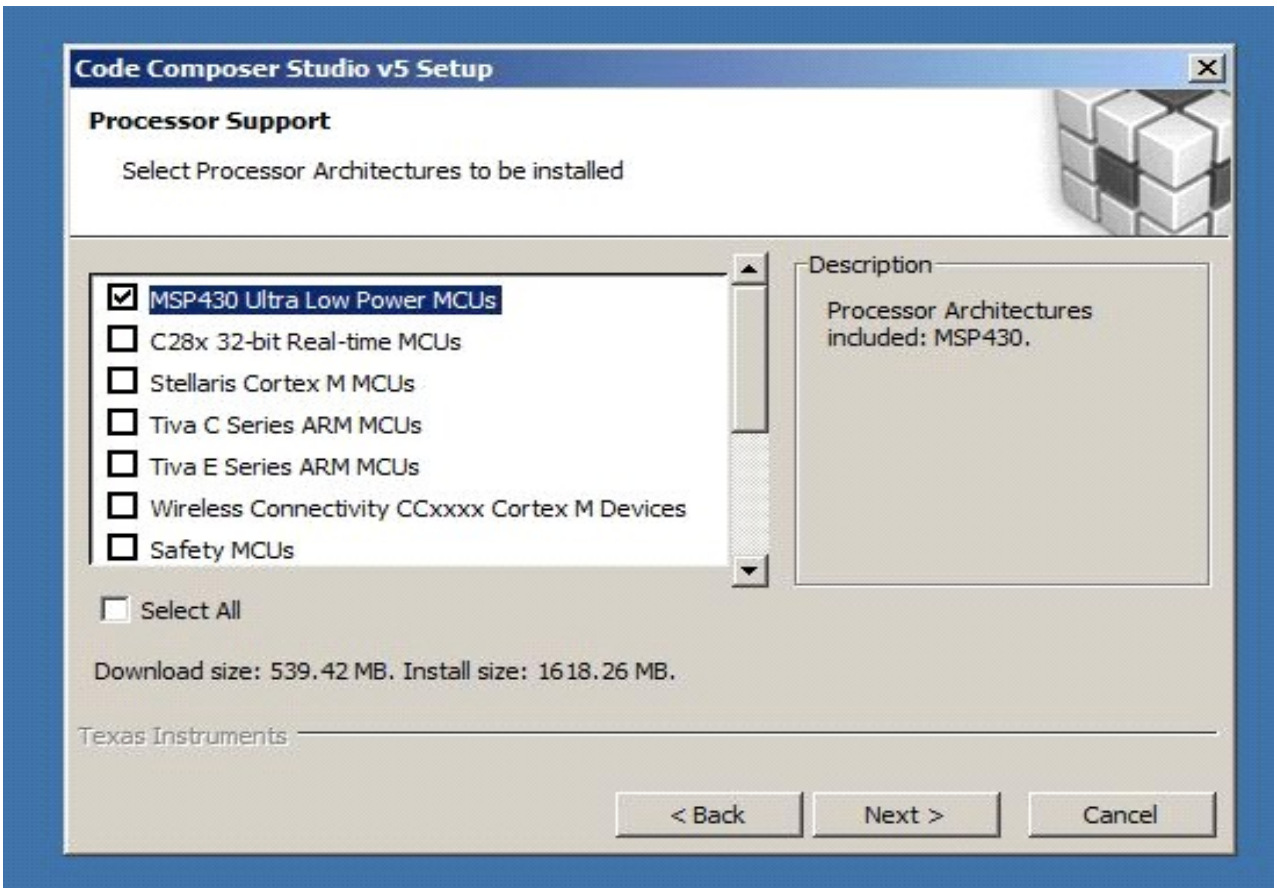
Şekil 2: CCS IDE Kurulum

Kurulacak yeri belirleyip Next diyoruz. Varsayılan C:\ti kalmasında fayda var.



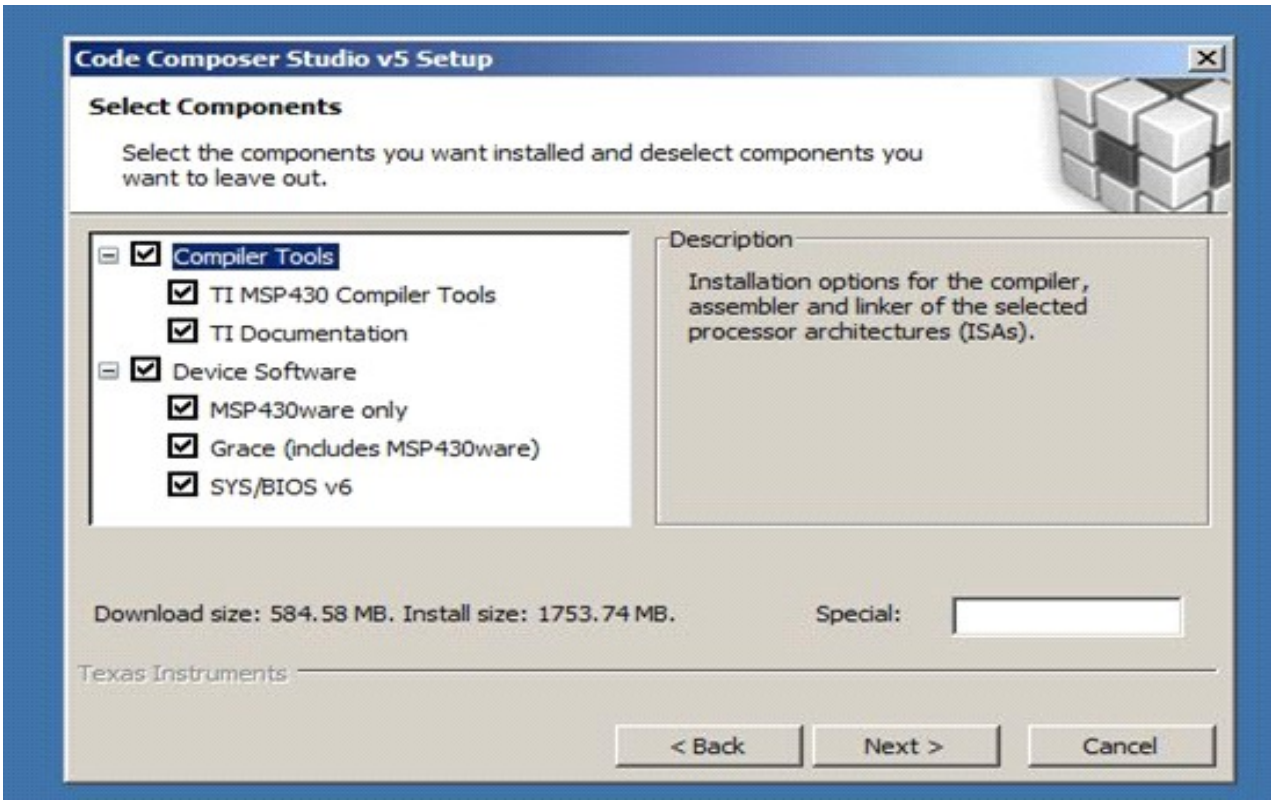
Şekil 3: CCS IDE Kurulum

Custom seçeneğini seçip Next diyoruz



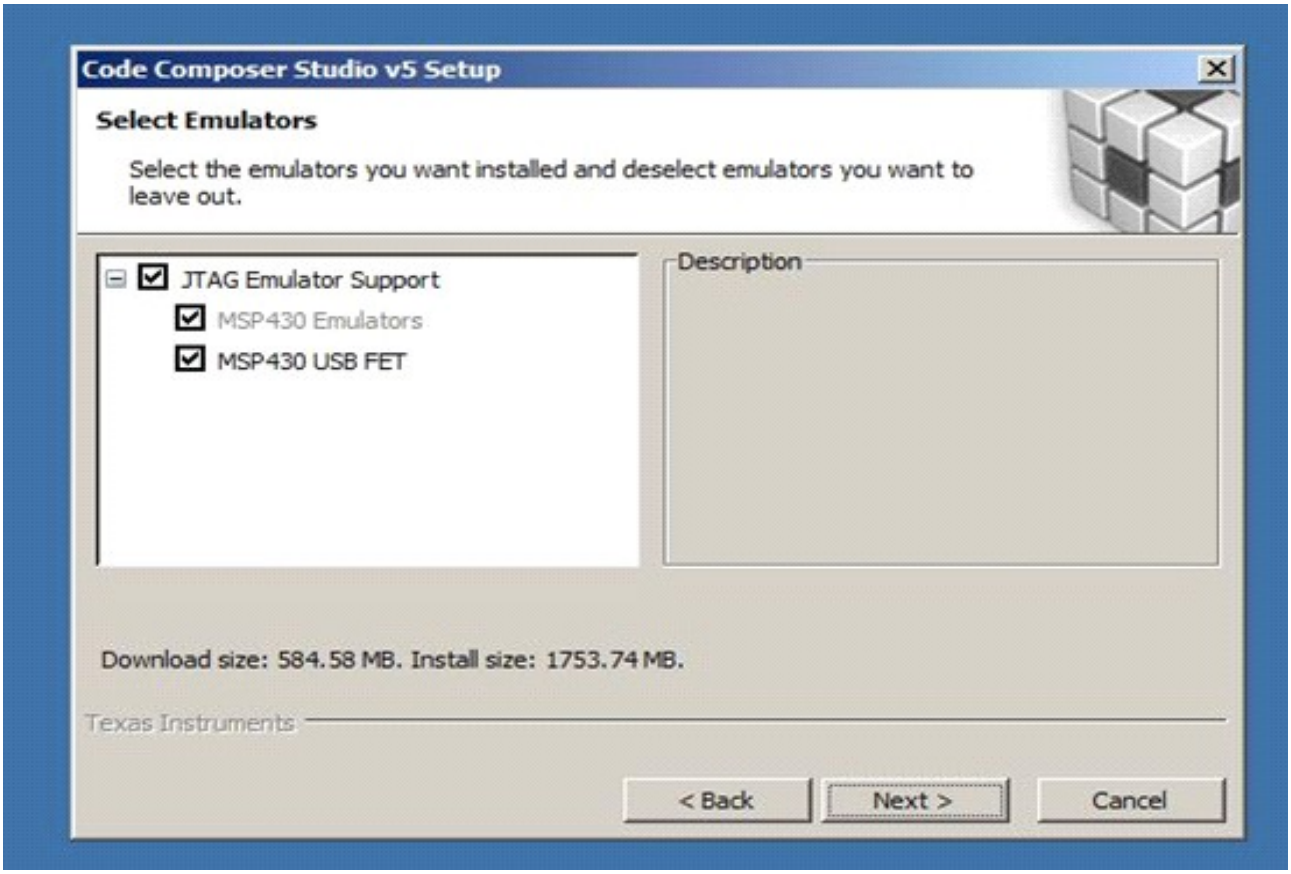
Şekil 4: CCS IDE Kurulum

Biz sadece MSP430 ile ilgilendiğimiz için MSP430 seçip Next diyoruz.



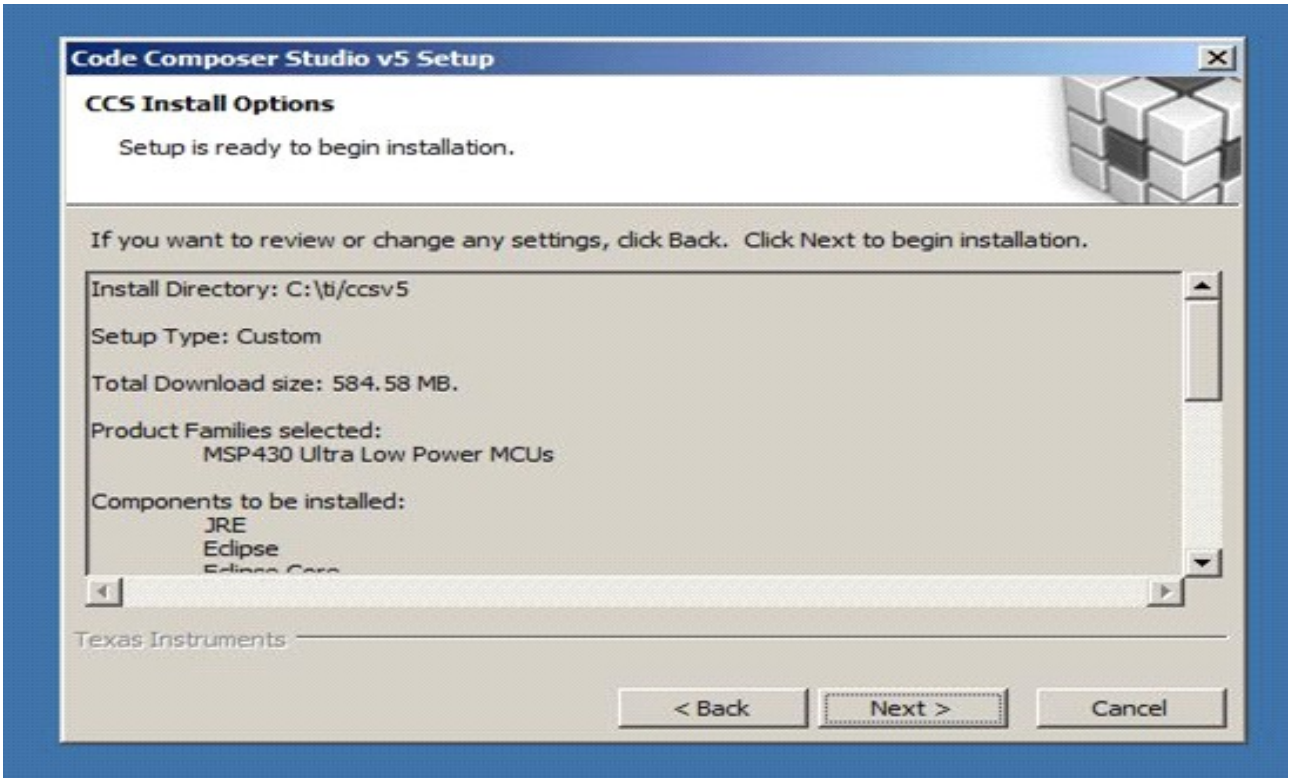
Şekil 5: CCS IDE Kurulum

MSP430 ile ilgili tüm araçları seçip Next diyoruz.



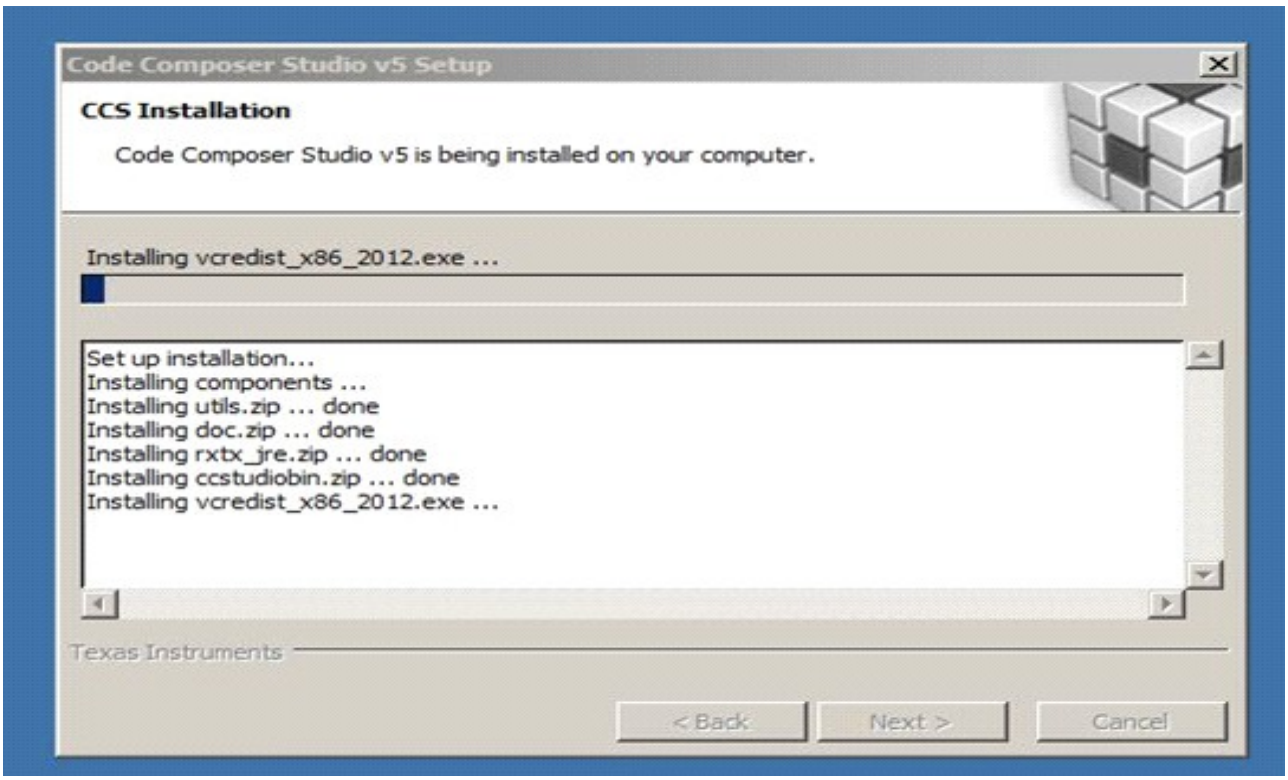
Şekil 6: CCS IDE Kurulum

Debug ve emülatör araçlarını seçip Next diyoruz.



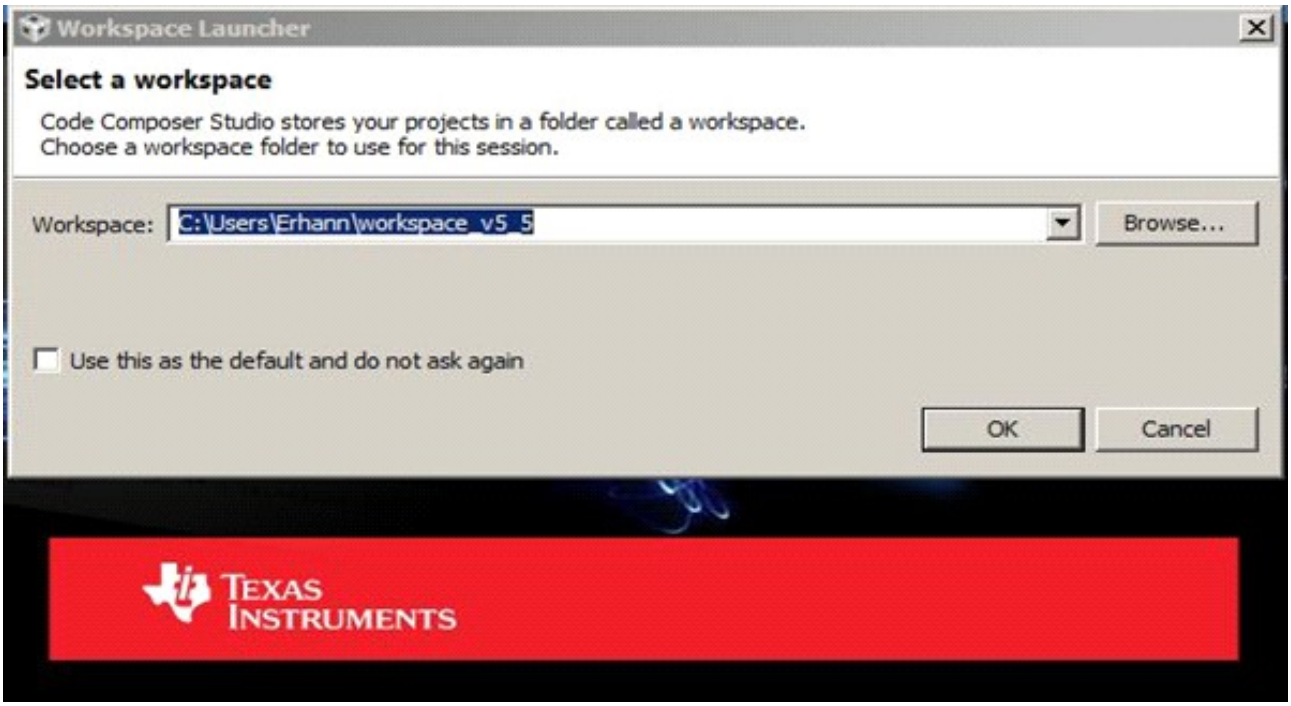
Şekil 7: CCS IDE Kurulum

Son olarak seçtiklerimizi gözden geçiriyoruz Next diyoruz.



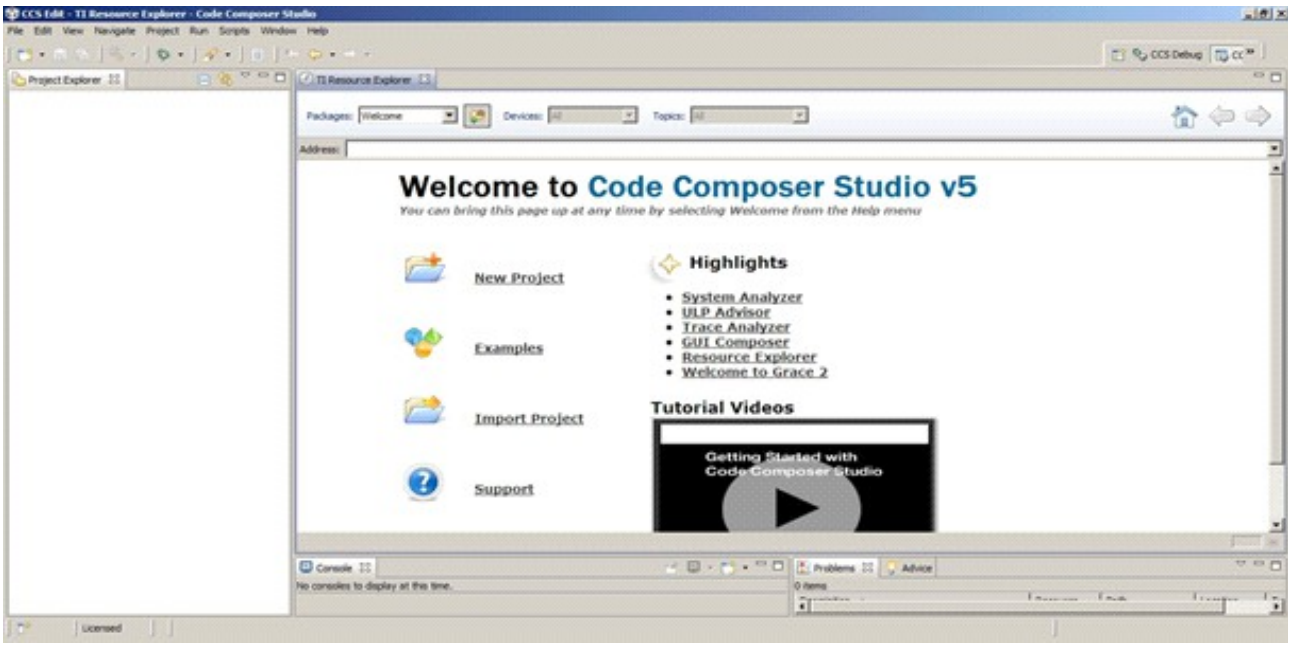
Şekil 8: CCS IDE Kurulum

Bu aşamadan sonra internet üzerinden indirme yaparak kurulum başlayacaktır. Kurulum uzun sürmektedir. Dosyaları internetten indirdiği için internet hızınız kurulumun hızını belirler.



Şekil 9: CCS IDE Kurulum

Kurulumun işlemini bitirip çalıştırdığınızda size çalışma alanınızın kayıt yerini sormaktadır. Varsayılan olarak çalıştığınız projeler çalışma alanında tutulur. Çalışma alanınızı işletim sisteminden farklı bir diske kaydederek silinmesini önleyebilirsiniz. Programı her çalıştırdığınızda bu işlem sorulur. Kutucuğu işaretlerseniz artık sormaz.



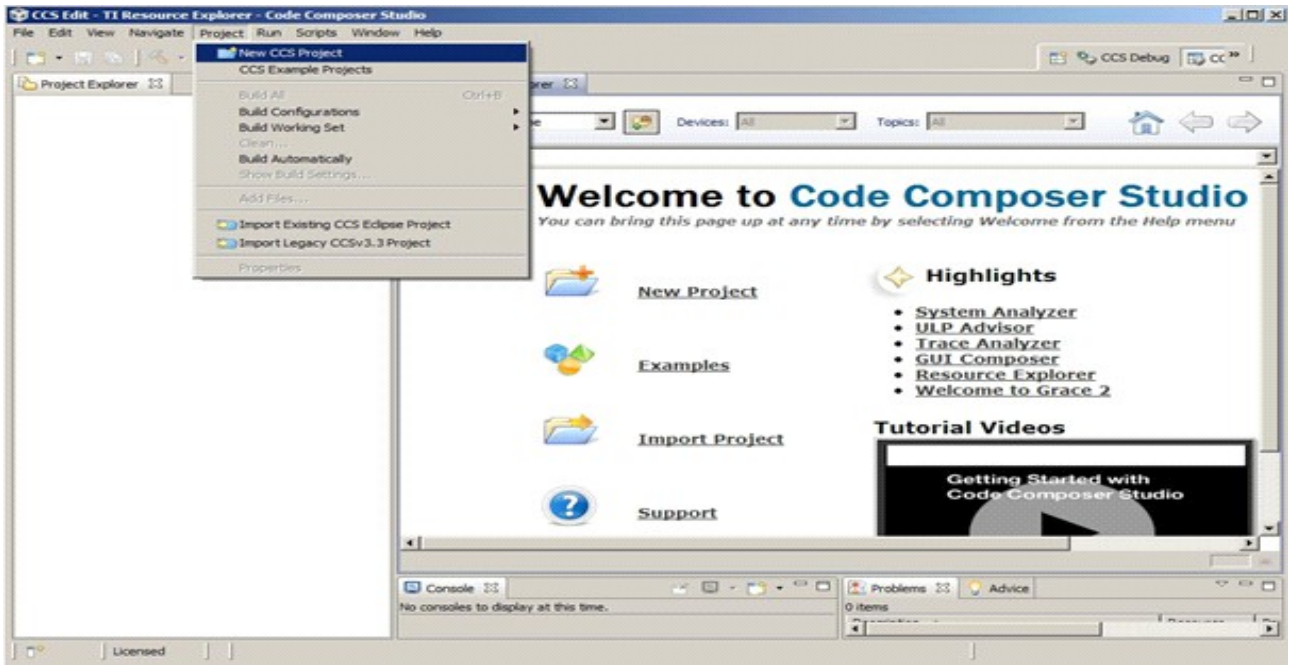
Şekil 10: CCS IDE Kurulum

Programı çalıştırdığınızda şekil 10'da ki gibi bir ekran gelmektedir. İlk çalıştırmada lisans işlemi gerektirirse Free License seçeneğini kullanarak adımları izleyin.16 KB, zaman kısıtlamasız olarak programı lisanslayabilirsiniz.

Bu aşamaya kadar geliştirme ortamımızın kurulumu bitmiştir. Proje oluşturarak kullanmaya başlayabilirsiniz.

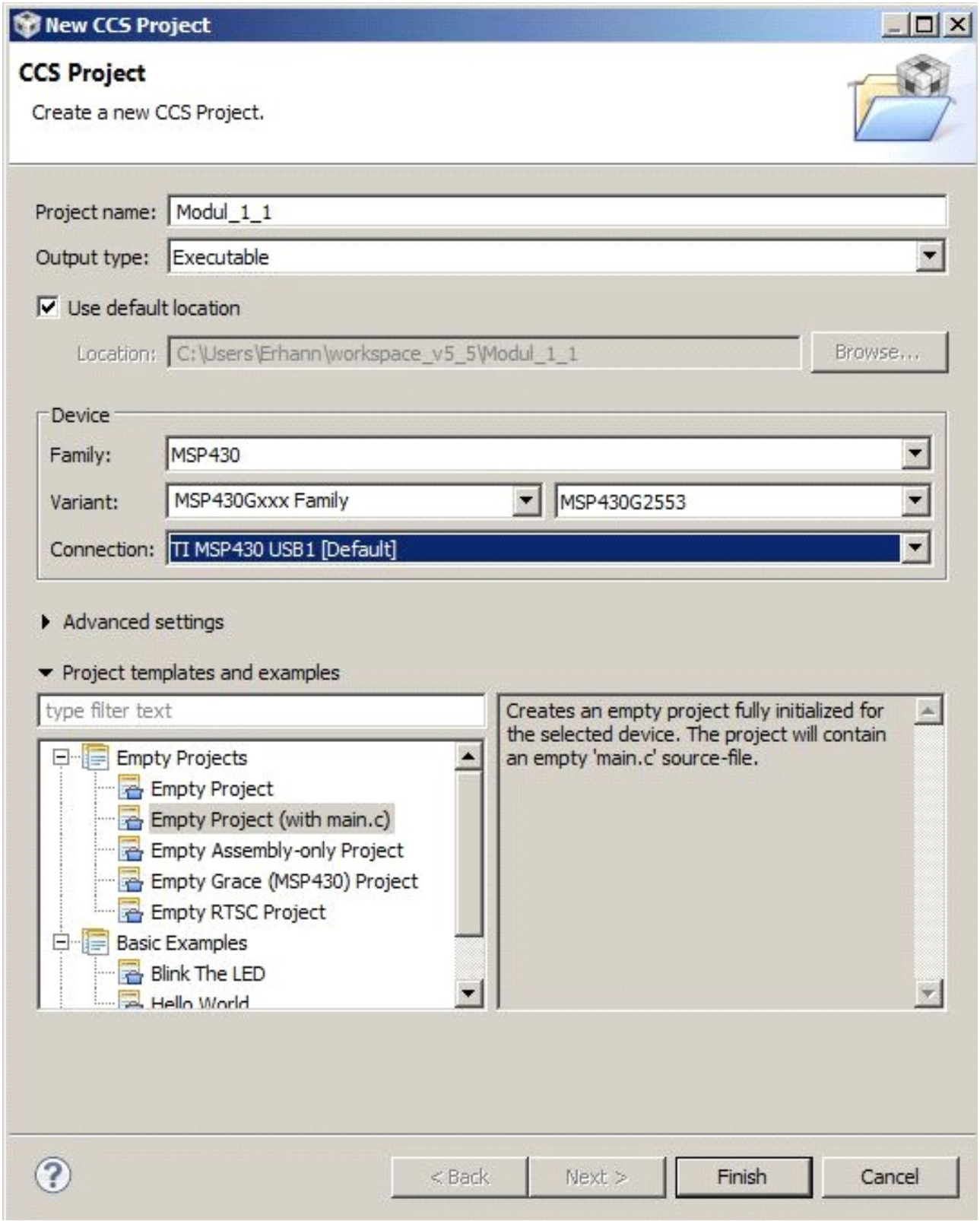
Proje Oluşturma

Programlama işlemine geçmeden önce yeni bir proje oluşturmak gerekmektedir. Proje oluşturarak daha düzenli bir şekilde çalışabilirsiniz.



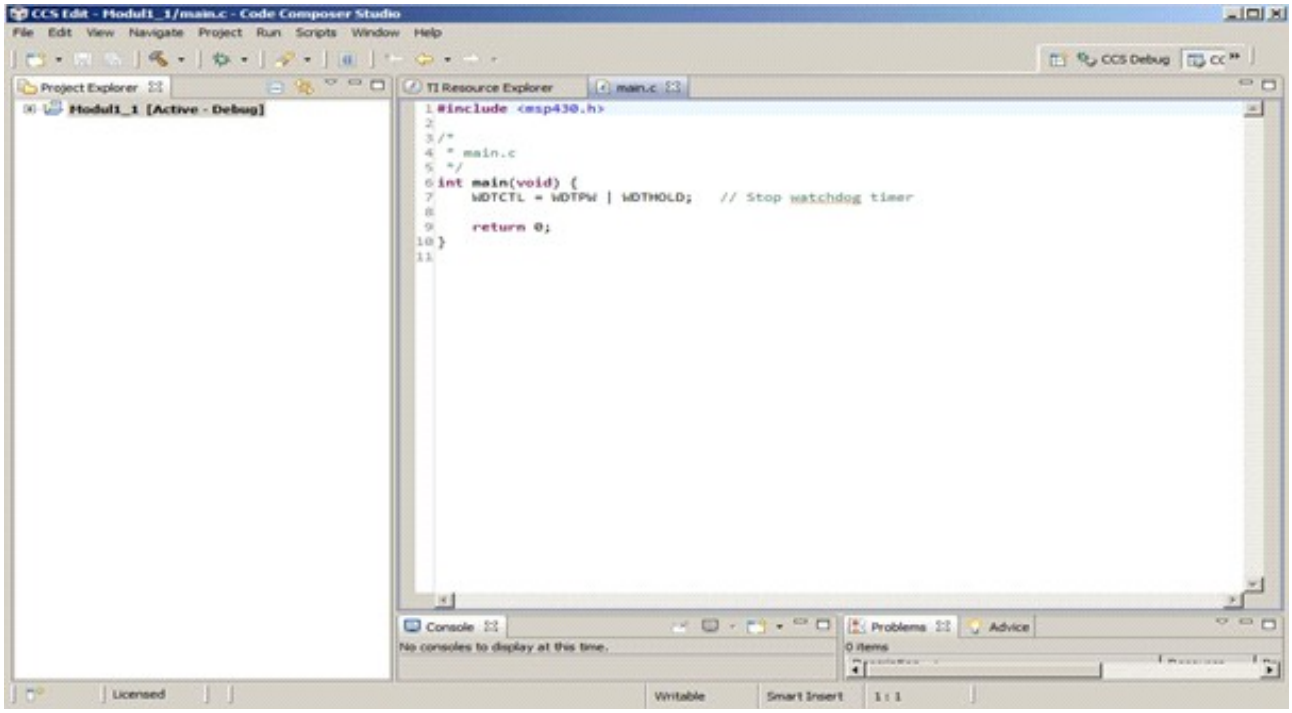
Şekil 11: CCS IDE Proje Oluşturma

Proje oluşturmak için Project menüsünden New CCS Project seçeneğine tıklıyoruz.



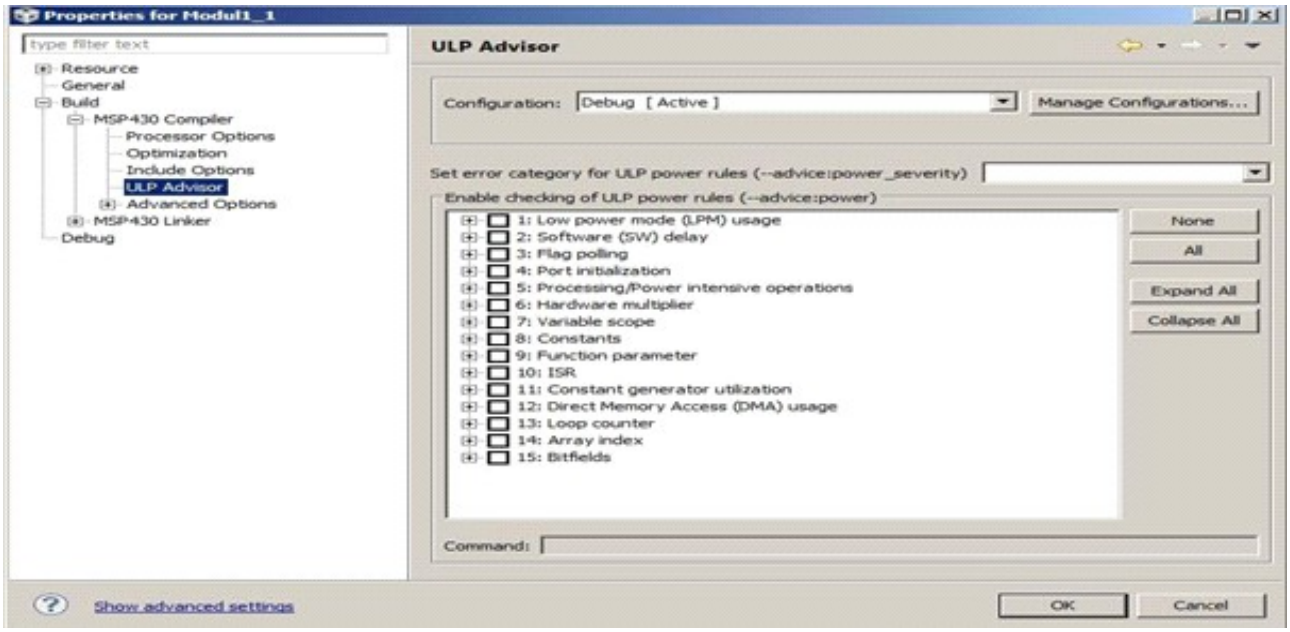
Şekil 12: CCS IDE Proje Oluşturma

Karşımıza çıkan ekranı Şekil 12'de ki gibi düzenleyip Finish diyoruz. Şekilde görüldüğü gibi Proje ismi Modul1_1, Denetleyici MS430G2553 seçilmiştir. Varsayılan USB üzerinden bağlanılan Debugger seçilmiştir. Ayrıca sadece main.c dosyası bulunan boş bir proje seçilmiştir. Projemiz verdiğiniz isim ile açılışta belirlediğimiz çalışma ortamına kaydedilir. İstenir ise kayıt yeri değiştirilebilir.



Şekil 13: CCS IDE Proje Oluşturma

Bu aşamadan sonra içerisinde birkaç satır başlangıç kodları bulunan main.c dosyası ile projemiz hazırdır. Bu aşamadan sonra kod yazımına geçilebilir.



Şekil 14: CCS IDE Proje Oluşturma

Projemizi oluşturduktan sonra Project menüsünden Properties seçeneğine tıklıyoruz Açılan pencere ULP Advisor sekmesine gelip None butonuna basarak tüm kutucukların işaretini kaldırıyoruz. Bu işlem ile ULP Advisor aracını devre dışı bırakıyoruz. ULP Advisor, geliştirme ortamında bulunan düşük güç tüketimi ve kod verimliliği sağlayan optimizasyon aracıdır. Başlangıç seviyesinde kullanımına gerek yoktur. Aynı zamanda yine properties pencersindeyken general sekmesini tıklayıp Optimization Level seçeneğini kapalı değilse kapalı konuma getirmenezde fayda var. Gecikme vs. amaçlı yazdığımız bazı döngüler optimizasyon aracı tarafından gereksiz görülüp kaldırılıp programınızın çalışmasını etkileyebilir.

Yazdığımız kodları derlemek için Ctrl + B kısa yolunu kullanabilir yada Project menüsünden Build All tuşuna basabilirsiniz. Ayrıca Project menüsünden Build Automatically seçeneğini seçerseniz kodunuz her kaydettiğinizde otomatik derlenir. Programınızda hata yok ise sorunsuz bir şekilde derlenir.

Uygulama 1.1 Adım Adım Debug İşlemi

Debug işlemine geçmeden önce genel olarak debug hakkında bilgi vermek yerinde olacaktır.

Debug, kelime anlamı olarak ayıklama, böcek ayıklama gibi anlamlara gelmektedir. İlk bilgisayarlar zamanında cihazların içine girip çalışmasını engelleyen böcekleri ayıklamak işlemi olarak tanımlandığı söylenmektedir. Günümüzde ise yazdığımız kodları denetlemek hataları, yanlışları gidermek için yaptığımız işlemler diyebiliriz.

Mikrodenetleyicilerden, masaüstü programlama, web programlama, mobil uygulamalara kadar her yazılım gereken yerde debug yapılabilmektedir. Geliştirme ortamları dahili olarak debug işlemini yapabilmektedir. Mikrodenetleyicilerin bilgisayar ortamından bağımsız kendi donanımları olduklarından bilgisayar üzerinden debug yapabilmek için harici bir debugger cihazına ihtiyaç duyarlar. Masaüstü yada web gibi programlamalarda geliştirilen ortamda programın kullanıldığı ortamda bilgisayarın kendisi olduğu için debugger donanımına ihtiyaç duyulmaz.

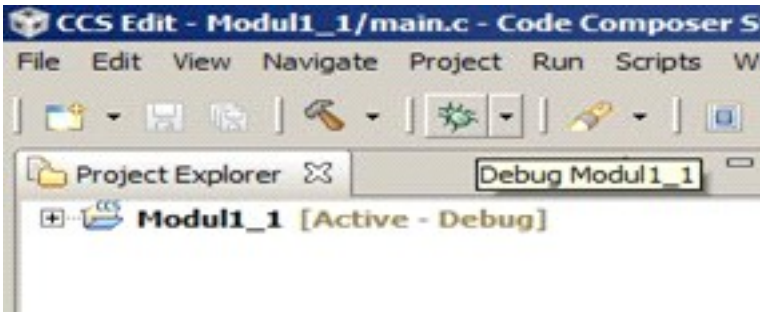
Kullanacağımız MPS430'u debug yapmak için geliştirme kartımız üzerinde bulunan Launhpad'in debugger donanımından faydalanacağız.

Debug işlemi sayesinde yazdığımız kodları işlemci üzerinde adım adım çalıştırabilir, kodlarımızın düzgün çalışıp çalışmadığını kontrol etmek için program içerisine Break Point denilen kontrol noktaları koyabiliriz. Bu sayede daha hızlı program geliştirme yapabilir, yazdığımız kodlarda oluşan sorunları daha hızlı çözebiliriz.

Debug işlemi için örnek kodumuz aşağıdaki gibidir.

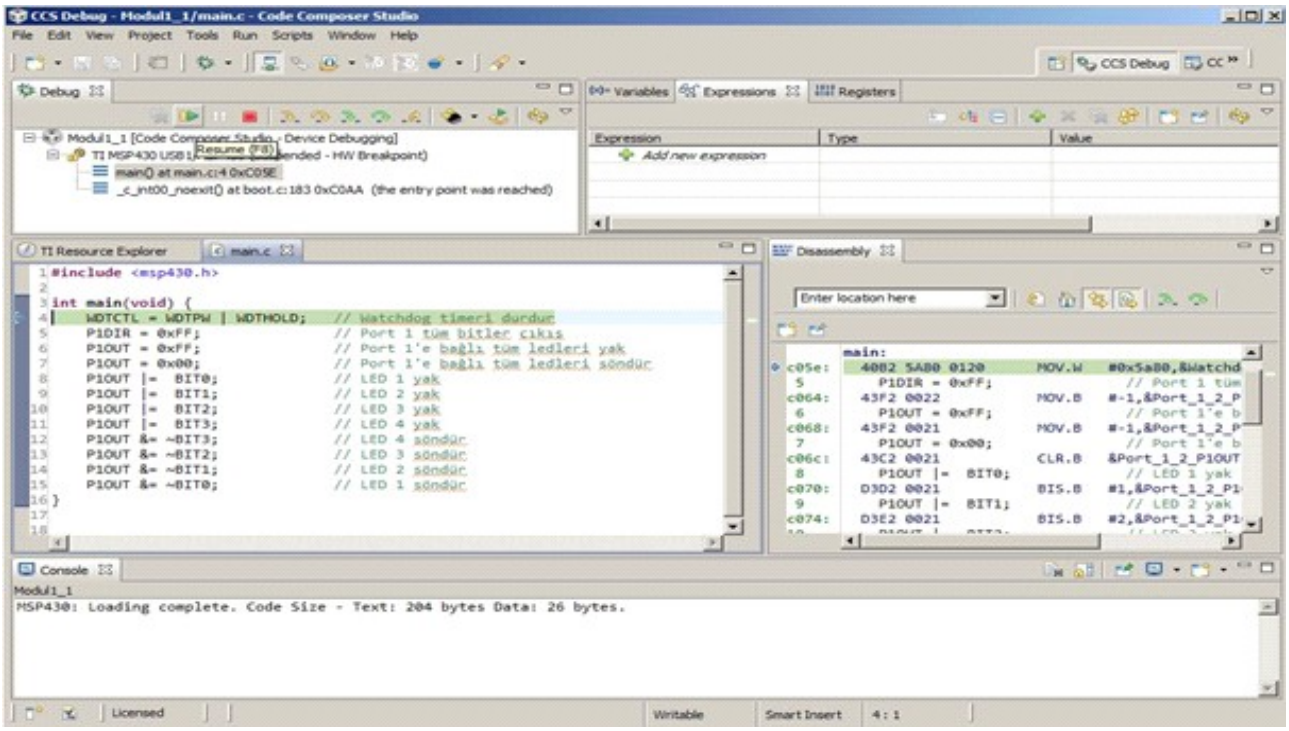
```
#include <msp430.h>

void main(void) {
WDTCTL = WDTPW | WDTHOLD;       // Watchdog timeri durdur
P1DIR = 0xFF;                   // Port 1 tüm bitler çıkış
P1OUT = 0xFF;                   // Port 1'e bağlı tüm ledleri yak
P1OUT = 0x00;                   // Port 1'e bağlı tüm ledleri söndür
P1OUT |=  BIT0;                 // LED 1 yak
P1OUT |=  BIT1;                 // LED 2 yak
P1OUT |=  BIT2;                 // LED 3 yak
P1OUT |=  BIT3;                 // LED 4 yak
P1OUT &= ~BIT3;                 // LED 4 söndür
P1OUT &= ~BIT2;                 // LED 3 söndür
P1OUT &= ~BIT1;                 // LED 2 söndür
P1OUT &= ~BIT0;                 // LED 1 söndür
}
```



Şekil 15: Debug İşlemi

Oluşturduğumuz projeye kodlarımızı ekleyip sorunsuz bir şekilde derledikten sonra Şekil 15'te görülen debug(yeşil böcek) butonuna tıklıyoruz. Geliştirme kartımız ile bilgisayar arasında herhangi bir bağlantı sorunu yada başka bir sorun yoksa Şekil 16'da ki debug arayüzü gelmesi gerekir.



Şekil 16: Debug İşlemi

Şekil 16'da projemize ait debug arayüzü görülmektedir. Ekrandan kısaca bahsedecek olursak, kod kısmında 4. Satırda bulunan mavi ok şuan işletilecek olan kodu gösterir. Onun sağ tarafında aynı şekilde C kodlarının assembly karşılığı ve hangi kodun işletileceği mavi ok ile belirtilir. Assembly kodların üzerinde ise programda kullanılan değişkenleri, RAM ve SFR değerlerini gözlemleyebilmek için alan bulunur.

Diğer önemli kısım ise debug butonlarıdır. Resume butonundan itibaren soldan sağa sırasıyla kısaca açıklayalım.

Resume: Kodun işlemci üzerinde çalışmasını başlatır.

Suspend: Çalışma anında ara vermek için kullanılır.

Terminate: Debug işlemini sonlandırır.

Step Into: Kodu detaylı olarak adım adım çalıştırır.

Step Over: Kodu dallanmadan çalıştırır. Örneğin fonksiyonların içine girmez.

Assembly Step Info: Kodu assembly olarak işletir. Step Info ile aynı özelliktedir.

Assembly Step Over: Kodu assembly olarak dallanmadan işletir. Step Over ile aynı özelliktedir.

Step Return: Step Into ile girilen fonksiyondan geri döner.

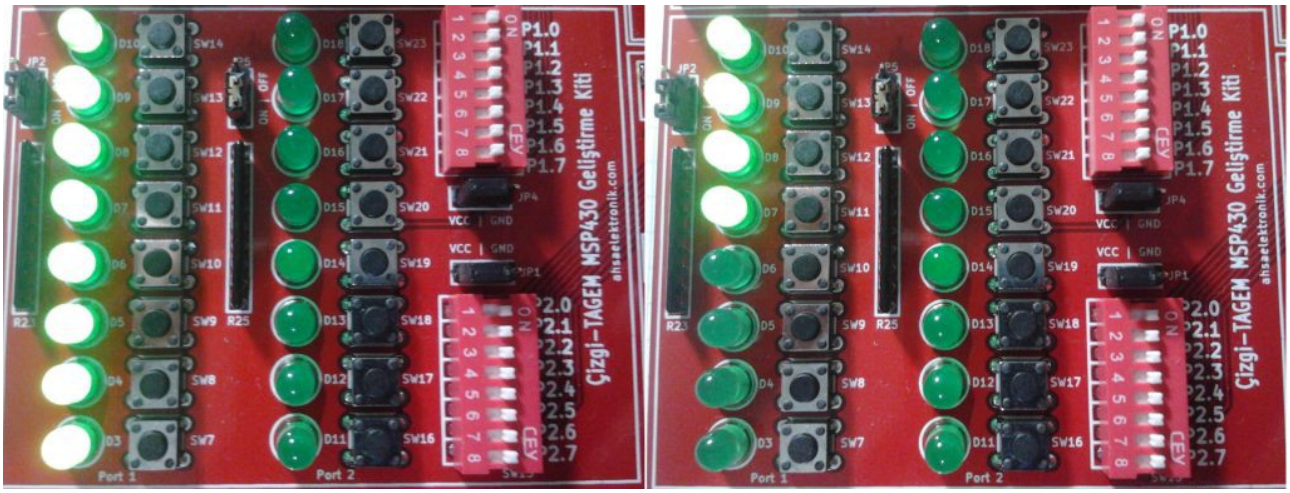
Reset: İşlemciyi yazılımsal veya donanımsal olarak resetler.

Restart: Debug işlemini tekrar başlatır.

Refresh: Yenileme yapar.

Şimdi geliştirme kartımız üzerindeki JP2 atlamasını on konumuna getirip SW4 üzerindeki anahtarları on konumuna getirelim. Yani ledlerimizi Port1'e bağlayalım. Diğer kullanılmayan birinlerin atlama ve anahtarlarını off konumuna getirmekte fayda var. Sonrasında Step Into butonuna basarak debug işlemine başlayalım.

Mavi ok 7. Satırı gösterdiğinde tüm ledlerin yandığını görebilirsiniz. Bunun nedeni öncesinde 6. Satır icra edilmiştir ve 6. Satırdaki komut port1'in tüm çıkışlarını aktif yapar. Aynı şekilde toplam 12 adımda debug işlemini tamamlayıp ledlerin değişimlerini gözlemleyebilirsiniz.



Şekil 17: Debug İşlemi

Şekil 17'de ki resimlerde debug işlemi sırasında değişen led durumları görülmektedir. Görüldüğü gibi adım adım çalıştırılarak ledler farklı düzenlerde yakılmıştır. Adım adım debug yapma işlemi bize yazdığımız kod parçasının istediğimiz gibi çalışıp çalışmadığını anlamamızı sağlar.

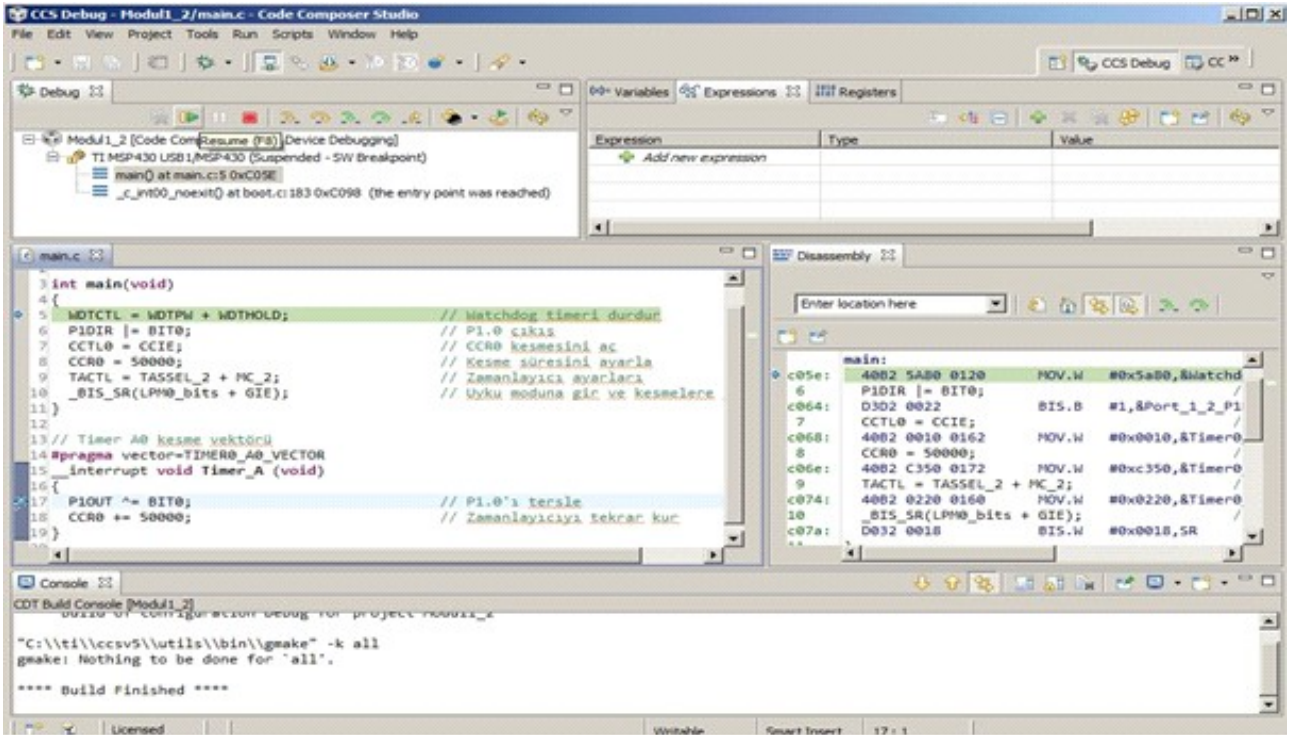
Uygulama 1.2 Break Point ile Debug İşlemi

Şimdi ise break point kullanarak yapılan debug işlemine geçelim. Debug işlemi için kodlarımız aşağıdaki gibidir. Yeni bir proje açıp kodları projenize eklemek yeterlidir. Program Timer kesmesi kullanarak yaklaşık 50 ms aralıklarla kesme üretip Port1'in 0. Bitine bağlı ledin durumunu değiştirmektedir.

```
#include <msp430.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;      // Watchdog timeri durdur
  P1DIR |= BIT0;                 // P1.0 çıkış
  CCTL0 = CCIE;                  // CCR0 kesmesini aç
  CCR0 = 50000;                  // Kesme süresini ayarla
  TACTL = TASSEL_2 + MC_2;       // Zamanlayıcı ayarları
  _BIS_SR(LPM0_bits + GIE);      // Uyku moduna gir ve kesmelere izin ver
}
// Timer A0 kesme vektörü
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
  P1OUT ^= BIT0;                 // P1.0'ı tersle
  CCR0 += 50000;                 // Zamanlayıcıyı tekrar kur
}
```

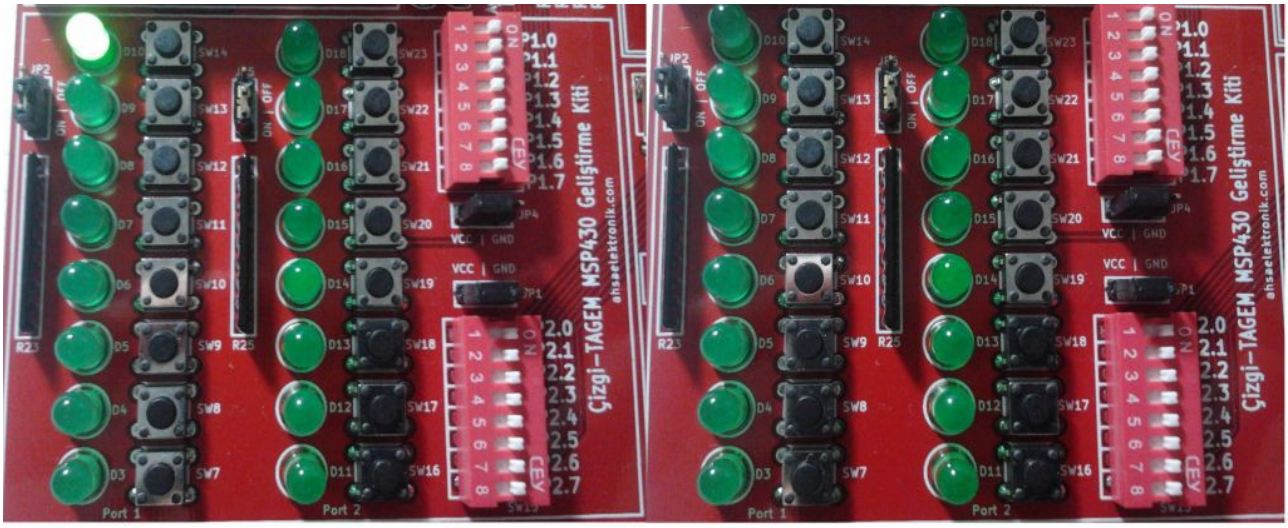
Yeni projemizi oluşturup kodlarımızı derledikten sonra debug butonuna basarak debug işlemine geçebiliriz.



Şekil 18: Debug İşlemi

Debug ekranı geldikten sonra 17. Satır numarası üzerine Mouse ile iki kere tıklayarak 17. Satıra break point eklenir. Bunun anlamı kodumuz çalışmaya başladığında 17. Satıra gelince işlemci durdurulacak demektir. Kodlar incelenirse 17. Satırın zamanlayıcı kesme programının başlangıcı olduğu görülür.

Break pointimizi ekledikten sonra Resume tuşuna basarak debug işlemini başlatalım. Resume tuşuna basar basmaz çok kısa bir sürede işlemcinin 17. Satırda durduğunu görebilirsiniz. Bunun nedeni zamanlayıcının yaklaşık 50 mili saniyede bir kesme üreterek break pointe gelmesidir. Bir sonraki adımda geliştirme kiti üzerinde bulunan Port1.0'a bağlı ledin durumunun değiştiğini görebilirsiniz.



Şekil 19: Debug İşlemi

Uygulamayı çalıştırmadan önce geliştirme kartı üzerinde bulunan jp2 atlamasını on, sw4 anahtarlarını üzerinde bulunan 1 numaralı anahtarı on konumuna getirmek gerekir. Ayrıca diğer kullanılmayan birimlerin anahtar ve atlamaları off konumda tutulmalıdır.

Şekil 19'da resimlerde debug işlemi yapılarak Port1.0'a bağlı ledin değişimi görülebilir.

Break point ile debug yapma işlemi bize yazdığımız kodların işletilip işletilmediğini öğrenmemizi sağlar. Örneğin, yazdığımız ve ana programda çağrılması gereken bir fonksiyonun gerçekte çağrılıp çağrılmadığını yada kullandığımız bir kesme işleminin gerçekleşip gerçekleşmediğini ilgili kısımlara break point koyarak anlayabiliriz. Hatta break point sonrasında programı adım adım çalıştırılarak kodlu detaylı bir şekilde inceleyebiliriz.

Bu modül kapsamında geliştirme ortamının tanıtılması, proje oluşturma, debug yapma gibi işlemler anlatılmıştır. Diğer modüllerin temeli niteliğinde olan bu modülün iyi anlaşılması ve uygulanması diğer modülleri iyi anlamak ve uygulamak açısından önemlidir.