

ASSEMBLY İLE ARM PROGRAMLAMA

1. GİRİŞ

Assembly, Türkçe adıyla çevirici dil, programlama için kullanılan alt seviyeli bir dildir. C, C++, Java, Python ve benzeri bu diller yüksek seviyeli diller olarak adlandırılır. Yüksek seviyeli bir dille yazılmış kod, çalıştırıldığında öncelikle assembly diline ve daha sonrasında ise makine diline çevrilerek işlemciye aktarılır. Her işlemcinin kendine ait bir buyruk kümesi mimarisi, dolayısıyla kendilerine özel bir assembly dili bulunur. Örnek olarak 8051 assembly'si ile ARM assembly'si farklıdır.

Bu lab boyunca alt seviye dil olan assembly dili ile STM32F4 kartımızda bulunan ARM Cortex M4 çekirdeğini programlayacağız.

2. ARM ASSEMBLY DİLİ

Örnek olarak aşağıda verilen C ile yazılmış kodun ARM assembly dilindeki çevirisine bakalım.

```
int total;
int i;

total = 0;
for (i = 10; i > 0; i--) {
    total += i;
}
```

```
        MOV    R0, #0           ; R0 accumulates total
        MOV    R1, #10          ; R1 counts from 10 down to 1
again    ADD    R0, R0, R1
        SUBS   R1, R1, #1
        BNE    again
halt     B      halt            ; infinite loop to stop computation
```

R0 ve R1 gibi değişkenlere yazmaç adı verilir. ARM işlemcisinde ulaşılabilir R0'dan R15'e kadar 16 adet yazmaç bulunmaktadır. Her bir yazmaç 32-bitlik değer saklar. Anlık değerler ifade edilirken “#<değer>” şeklinde kullanılmaktadır.

Bir sonraki sayfada ARM assembly dilinde bulunan komutları görebilirsiniz. “reg” ile başlayanlar R0,R1,...,R15 yazmaçları, “arg” olanlar ise anlık değerleri temsil etmektedir. Örnek olarak yukarıda assembly diline çevrilen buyrukları inceleyebilirsiniz.

0. AND regd, rega, argb	$regd \leftarrow rega \& argb$
1. EOR regd, rega, argb	$regd \leftarrow rega \wedge argb$
2. SUB regd, rega, argb	$regd \leftarrow rega - argb$
3. RSB regd, rega, argb	$regd \leftarrow argb - rega$
4. ADD regd, rega, argb	$regd \leftarrow rega + argb$
5. ADC regd, rega, argb	$regd \leftarrow rega + argb + carry$
6. SBC regd, rega, argb	$regd \leftarrow rega - argb - !carry$
7. RSC regd, rega, argb	$regd \leftarrow argb - rega - !carry$
8. TST rega, argb	set flags for $rega \& argb$
9. TEQ rega, argb	set flags for $rega \wedge argb$
10. CMP rega, argb	set flags for $rega - argb$
11. CMN rega, argb	set flags for $rega + argb$
12. ORR regd, rega, argb	$regd \leftarrow rega argb$
13. MOV regd, arg	$regd \leftarrow arg$
14. BIC regd, rega, argb	$regd \leftarrow rega \& \sim argb$
15. MVN regd, arg	$regd \leftarrow \sim argb$

Şekil 1: ARM Buyrukları

2.1 Dallanma Buyrukları (Branch Instructions)

Öncelikle C dilinden ARM assembly diline çevrilmiş kodu inceleyelim.

```
a = 40;
b = 25;
while (a != b) {
    if (a > b) a -= b;
    else      b -= a;
}
```

```
again    MOV R0, #40      ; R0 is a
         MOV R1, #25      ; R1 is b
         CMP R0, R1
         BEQ halt
         BLT isLess
         SUB R0, R0, R1
         B again
isLess   SUB R1, R1, R0
         B again
halt     B halt
```

S

Assembly dilinde yazılmış olan kodda görüldüğü üzere herhangi bir if, while için belirtilen şartlarda öncelikle “CMP,TST,CMN,TEQ” ve “SUBS” adlı buyruklar kullanılmalıdır. Daha sonra ise “BEQ, BLT,BGT,BLE,BGE,BNE” gibi buyruklar kullanılarak if, while benzeri komutlar tamamlanır.

CMP, İngilizce compare kelimesinin kısaltması, iki tane yazmacın değerini kıyaslar. Daha sonra gelecek olan branch(dallanma) buyrukları ise yapılacak eylemi belirler.

BİL362L - Mikroişlemciler Laboratuvarı

Lab 1 Öncesi

Örnek olarak assembly diliyle yazılan yukarıdaki kodda öncelikle “CMP R0, R1” buyruğu ile

Buyruk	Kullanımı	Açıklama
BEQ	BEQ <gidilecek etiket>	Eğer CMP ile karşılaştırılan yazmaçlar birbirine eşitse <gidilecek etiket>’e atla.
BLT	BLT <gidilecek etiket>	Eğer CMP ile karşılaştırılan yazmaçların ilki ikincisinden küçükse (işaretli karşılaştırma) <gidilecek etiket>’e atla.
BGT	BGT <gidilecek etiket>	Eğer CMP ile karşılaştırılan yazmaçların ilki ikincisinden büyükse (işaretli karşılaştırma) <gidilecek etiket>’e atla.
BLE	BLE <gidilecek etiket>	Eğer CMP ile karşılaştırılan yazmaçların ilki ikincisinden küçük eşitse (işaretli karşılaştırma) <gidilecek etiket>’e atla.
BGE	BGE <gidilecek etiket>	Eğer CMP ile karşılaştırılan yazmaçların ilki ikincisinden büyük eşitse (işaretli karşılaştırma) <gidilecek etiket>’e atla.
BNE	BNE <gidilecek etiket>	Eğer CMP ile karşılaştırılan yazmaçlar birbirine eşit değilse <gidilecek etiket>’e atla.

R0 ile R1 yazmaçları karşılaştırılmıştır. Daha sonrasında ise “BEQ halt” buyruğu ile eğer R0 ile R1 birbirine eşitse halt etiketinin bulunduğu satırdan, değilse bir sonraki satırdan kod devam edecektir. “BLT isLess” satırında ise eğer R0, R1’den küçükse “isLess” etiketinin bulunduğu satırdan, değilse bir sonraki satırdan kod devam edecektir.

“B again” buyruğu ise herhangi bir duruma bakmazsınız “again” etiketinin bulunduğu satıra atlar. Basitçe, “B <gidilecek etiket>” buyruğu, 8051 mimarisindeki bulunan jump buyruğunun aynısıdır.

2.2 Bellek İşlemleri

ARM mimarisinde bellek işlemleri “LDR” ve “STR” buyruklarıyla yapılmaktadır. “LDR” buyruğu veriyi bellekten hedef yazmaca yazmaya, “STR” buyruğu ise yazmaçtaki değeri belleğin verilen adresine yazar. Bu buyrukları bir örnek üzerinde inceleyelim. Aşağıda verilen kod, belleğin belli bir kısmındaki sayıların toplamını R4 yazmacına kaydeden işlemi yapmaktadır.

```
addInts MOV R4, #0
addLoop LDR R2, [R0]
        ADD R4, R4, R2
        ADD R0, R0, #4
        SUBS R1, R1, #1
        BNE addLoop
```

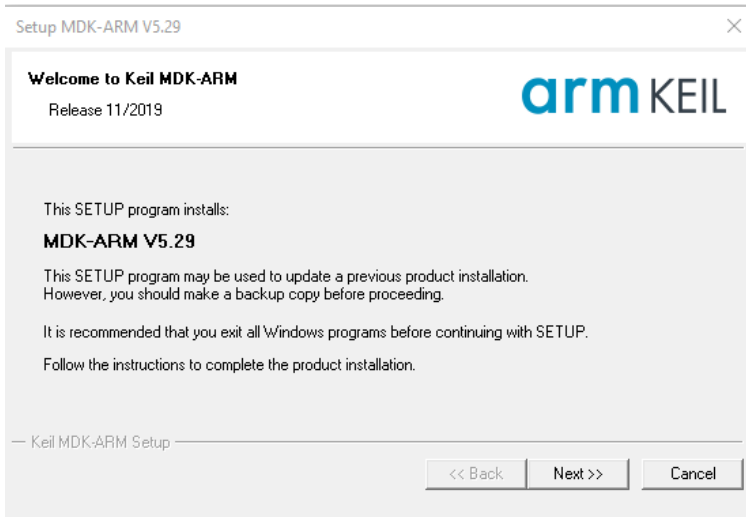
“LDR” ve “STR” buyrukları 32-bitlik değerlerle işlem yapmaktadır. Aynı zamanda ARM mimarisinde bellek için bayt adresleme kullanılır. (Bu nedenle yukarıdaki kodda yeni bir değere erişecekken 4 arttırdık [4-bayt = 32 bit]).

8-bitlik değerlerle bellek işlemi yapmak için “LDRB” ve “STRB” buyrukları bulunmaktadır. Aşağıdaki kodda kullanımını görebilirsiniz. Eğer fark ettiyseniz 8-bit değerlerle işlem yaptığımız için bu sefer adresi 4 değil sadece 1 arttırdık [8-bit = 1-bayt].

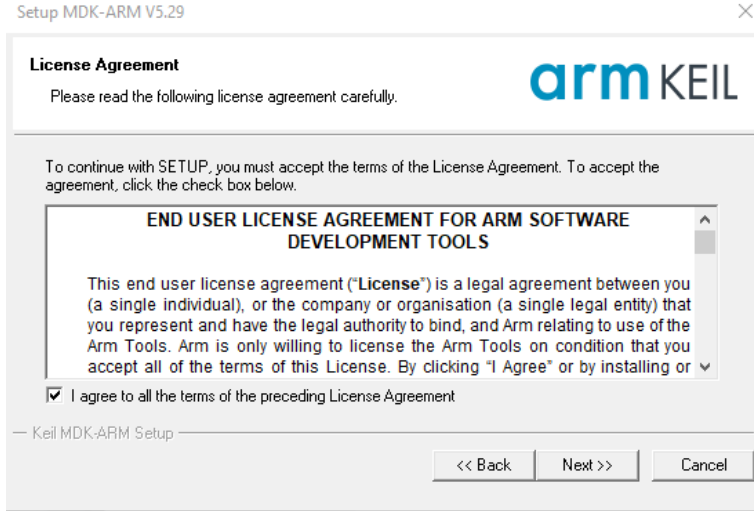
```
strcpy  LDRB R2, [R1]
        STRB R2, [R0]
        ADD R0, R0, #1
        ADD R1, R1, #1
        TST R2, R2      ; repeat if R2 is nonzero
        BNE strcpy
```

3. ARM KEIL v5.29 KURULUMU

[Buradan](#) indireceğiniz “MDK529” yükleme dosyasıyla kartımızın üstündeki ARM’ı programlayacağız. Dosyayı indirdikten sonra karşınıza şöyle bir pencerenin açılması gerekiyor.



Bu pencerede “Next >>” butonuna basarak aşağıdaki pencereye erişeceksiniz.



Setup MDK-ARM V5.29

License Agreement

Please read the following license agreement carefully.

To continue with SETUP, you must accept the terms of the License Agreement. To accept the agreement, click the check box below.

END USER LICENSE AGREEMENT FOR ARM SOFTWARE DEVELOPMENT TOOLS

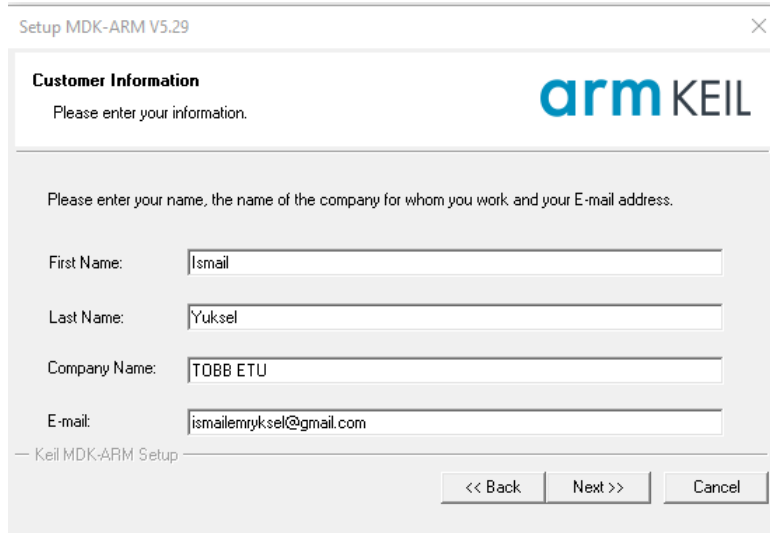
This end user license agreement (“License”) is a legal agreement between you (a single individual), or the company or organisation (a single legal entity) that you represent and have the legal authority to bind, and Arm relating to use of the Arm Tools. Arm is only willing to license the Arm Tools on condition that you accept all of the terms of this License. By clicking “I Agree” or by installing or

☒ I agree to all the terms of the preceding License Agreement

— Keil MDK-ARM Setup —

<< Back Next >> Cancel

Burada “I agree to all the terms of the preceding License Agreement” kutucuğunu işaretledikten sonra “Next >>” butonuna basarak bir sonraki adıma geçebilirsiniz.



Setup MDK-ARM V5.29

Customer Information

Please enter your information.

Please enter your name, the name of the company for whom you work and your E-mail address.

First Name: Ismail

Last Name: Yüksel

Company Name: TOBB ETU

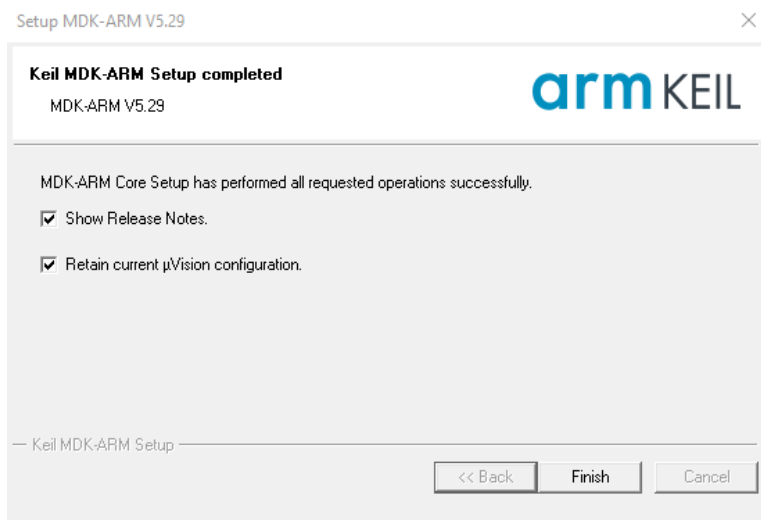
E-mail: ismailmryksel@gmail.com

— Keil MDK-ARM Setup —

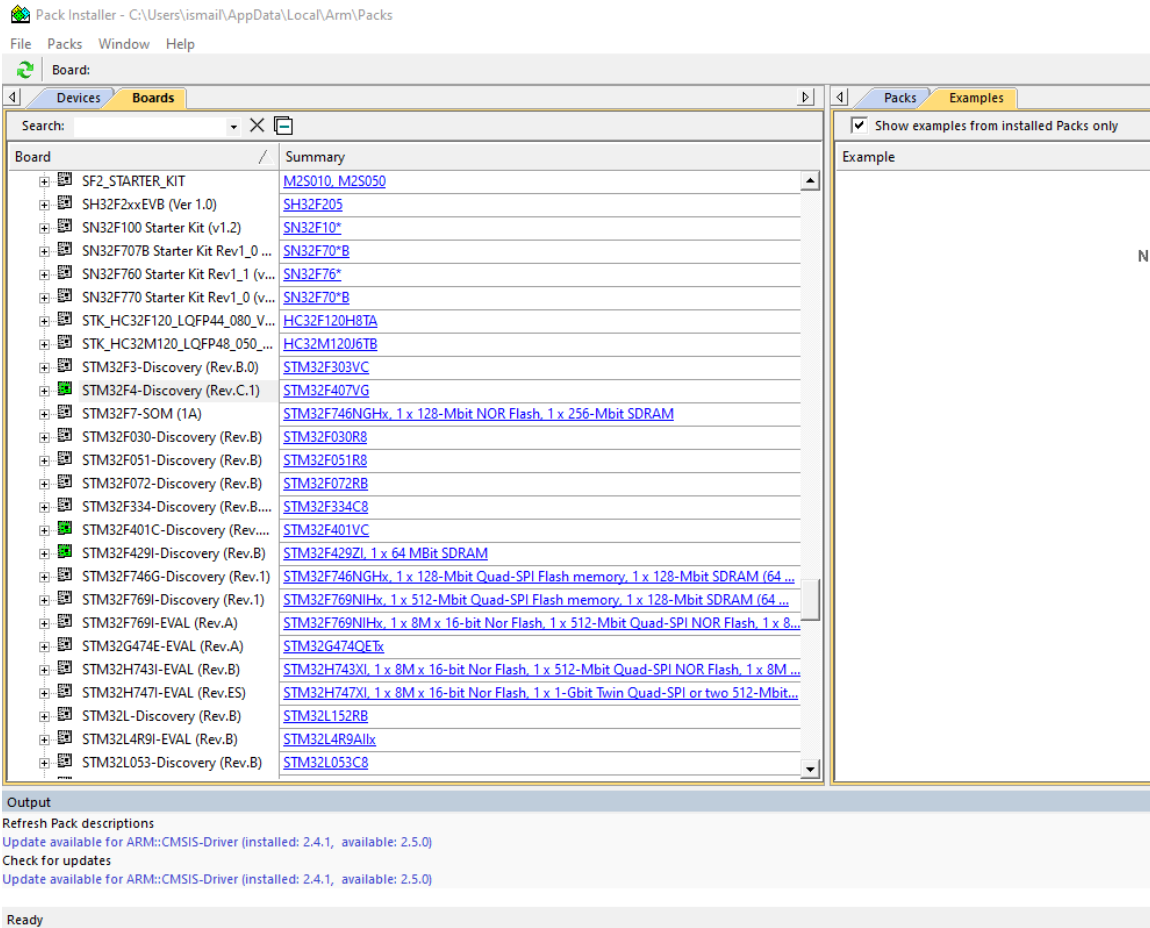
<< Back Next >> Cancel

Daha sonra açılan bu pencerede “Company Name” olarak “TOBB ETU”yü yazıp diğerlerine kendi adınızı, soyadınızı ve mail adresinizi girmelisiniz. “Next >>” butonuna tıklayarak yüklemeyi başlatabilirsiniz.

Yüklemeyi tamamladıktan sonra aşağıda gördüğünüz pencere açılacaktır.



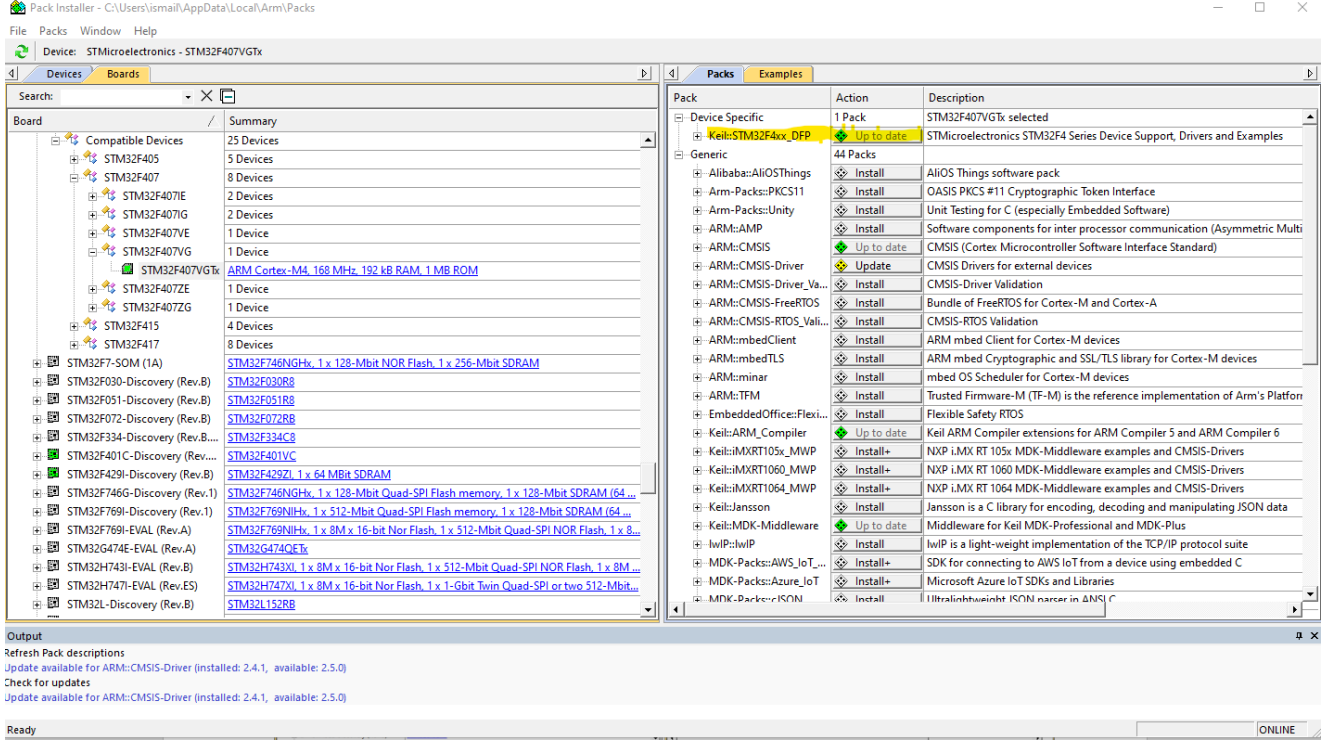
“Finish” butonuna tıkladıktan sonra yüklenecek kütüphaneyi seçeceğiniz pencere açılacaktır.



BİL362L - Mikroişlemciler Laboratuvarı

Lab 1 Öncesi

Bu pencerede sol tarafta yer alan “STM32F4-Discovery (Rev.C.1)” sekmesini açın. Açtıktan sonra “Compatible Devices” adlı bir sekme daha göreceksiniz. O sekme de açtığınızda aşağıdaki gibi bir pencere karşınıza çıkacaktır.



Bu pencerede “STM32F407VG” sekmesini açtığınızda karşınıza çıkan “STM32F407VGx” kartını seçin. Seçtiğinizde sağ taraftaki bölüm gözükecektir. Bu bölümde “Keil-STM32F4xx_DFP” adlı kütüphaneyi seçip Install butonuna tıklayın. Bu sayede ihtiyacınız olan kütüphaneyi yüklemiş olacaksınız. Yükleme bittikten sonra bu pencereyi kapatın.

4. KEIL v5.29 KULLANARAK KARTIN PROGRAMLANMASI

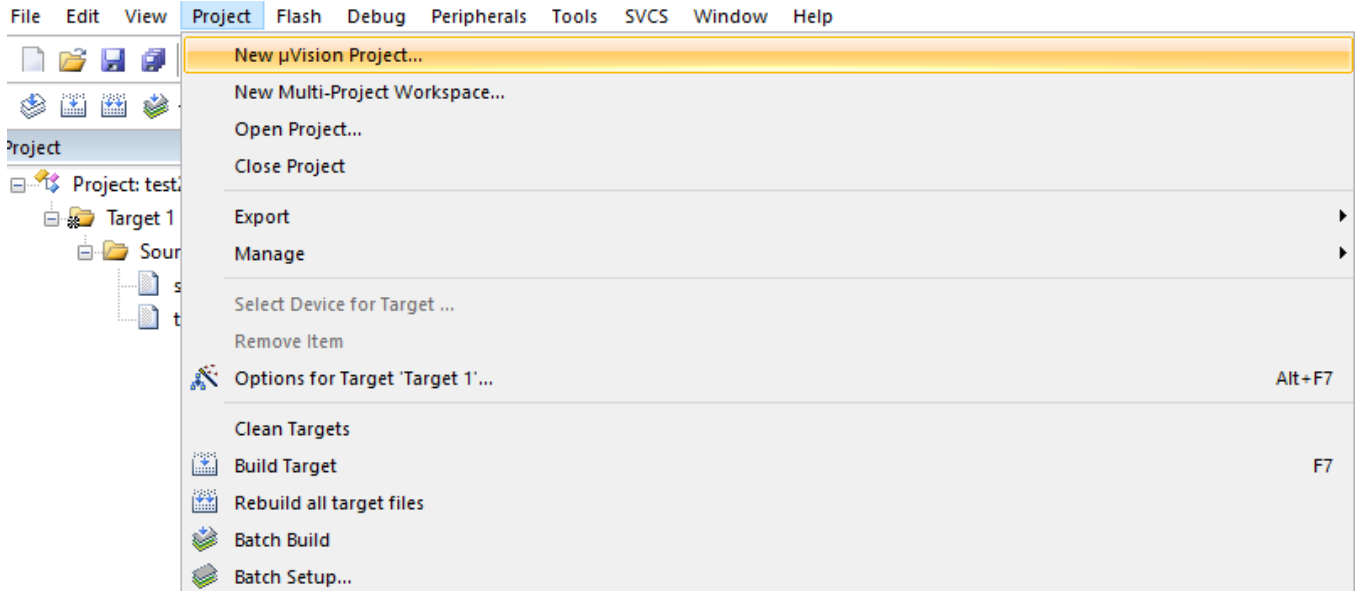
Bu bölümde yüklemiş olduğunuz KEIL v5.29’da nasıl proje oluşturulacağı, yazılan kodun nasıl karta atılacağı ve yazılan kodun doğru çalışıp çalışmadığını gösteren “Debug” arayüzü anlatılacaktır.

KEIL ile hem C hem de assembly dilinde kodlamalar yapabilirsiniz. Bu hafta KEIL’de assembly dilinde kodlama yapacağız. Kullandığımız kart nedeniyle, herhangi bir kod yazdığımızda onu karta atmamız için, kartın açılması için gereken kodu da yüklememiz gerekmektedir. Bu dosyaya Piazza’da General Resources sekmesinden “startup_stm32f407xx.s” adıyla ulaşabilirsiniz.

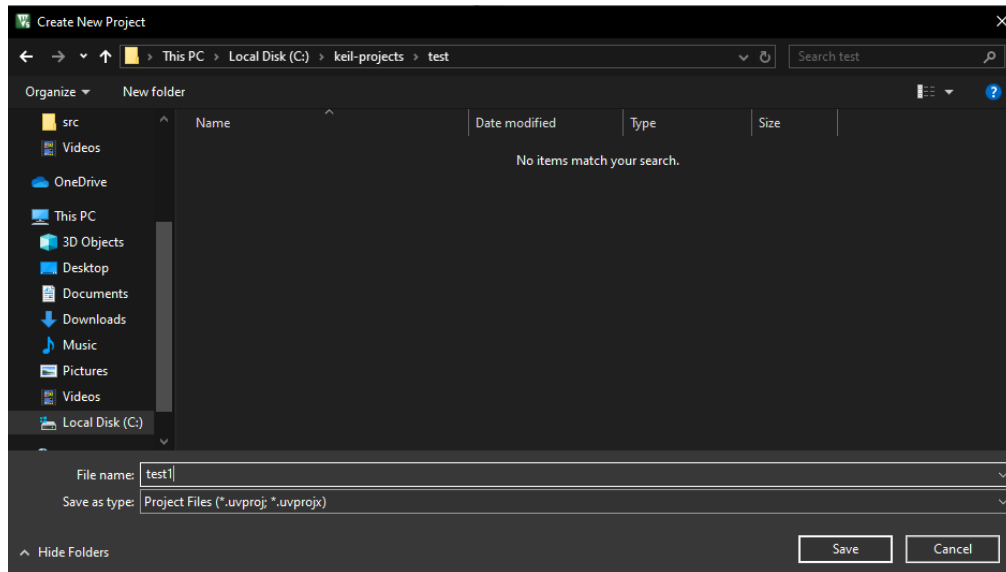
Assembly’den bağımsız olarak başlangıç dosyasının çalışması için asıl dosyamıza eklememiz gereken birkaç satır bulunuyor. Bu durumdan “Assembly kodunun yazılması” başlığının altında değinilecektir.

4.1 ASSEMBLY PROJESİNİN OLUŞTURULMASI

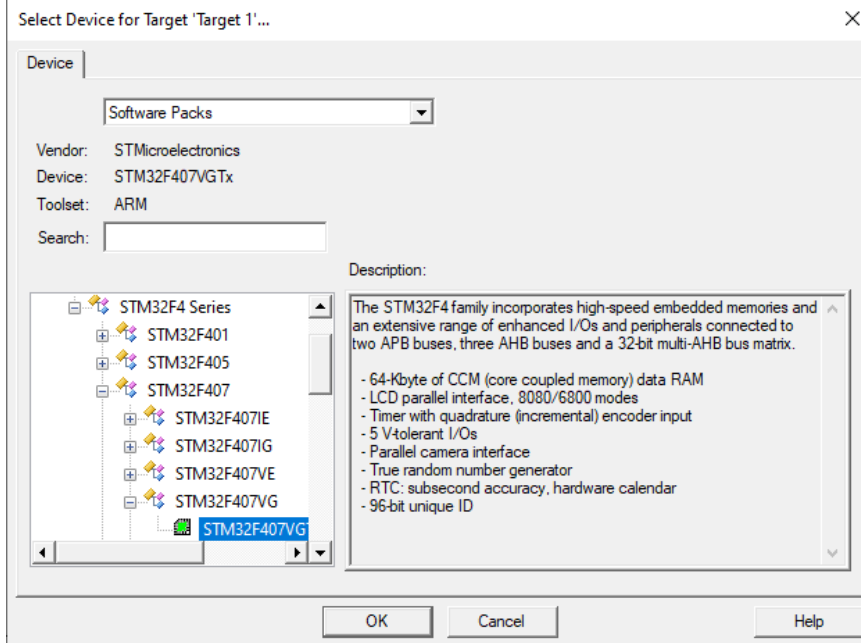
KEIL programını çalıştırdığınızda karşınıza çıkan pencerede “Project” kısmında “New μ Vision Project”e tıklayın.



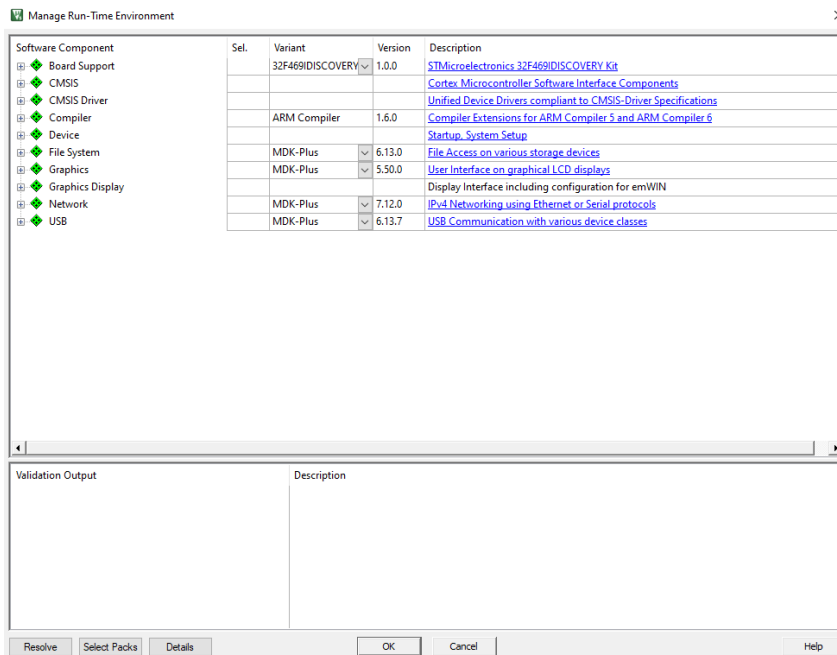
Daha sonra oluşturacağınız projenin yerini belirleyip ona isim verin. Bu örnekte biz projeye “test1” ismini verdik. Vereceğiniz herhangi bir isimden sonra “Save” butonuna tıklayıp bir sonraki aşamaya geçin.



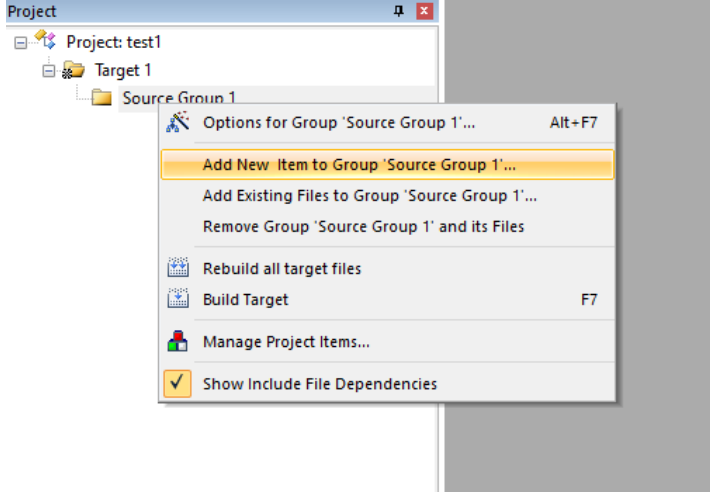
Daha sonra çıkan pencerede hangi karta atacağınızı seçin. Bu lab boyunca biz “STM32F407VGTx” kartını kullanacağız. O yüzden aşağıdaki pencerede “STMicroelectronics” sekmesini seçip oradan “STM32F407VG” sekmesini açın. Oradan “STM32F407VGTx” kartını seçin. Tüm bu adımları gerçekleştirdikten sonra oluşacak durum aşağıdaki gibi olmalıdır. Buradan “OK” butonuna basıp bir sonraki adıma geçin.



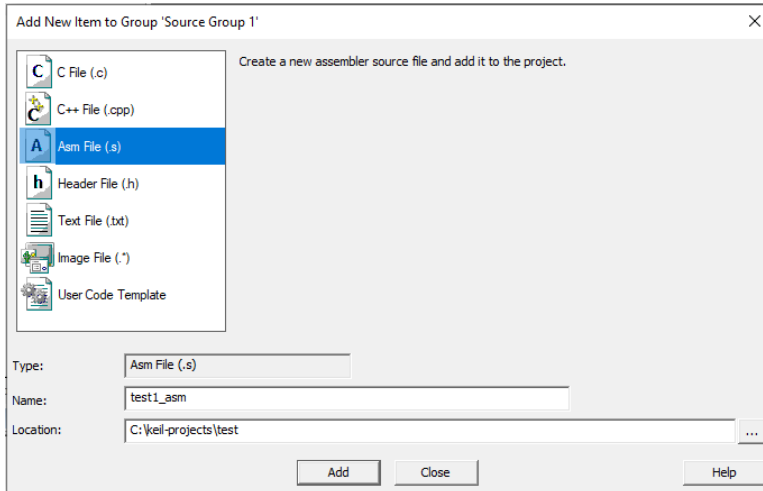
Açılan yeni pencerede, herhangi bir ayarı değiştirmeden “OK” butonuna basın.



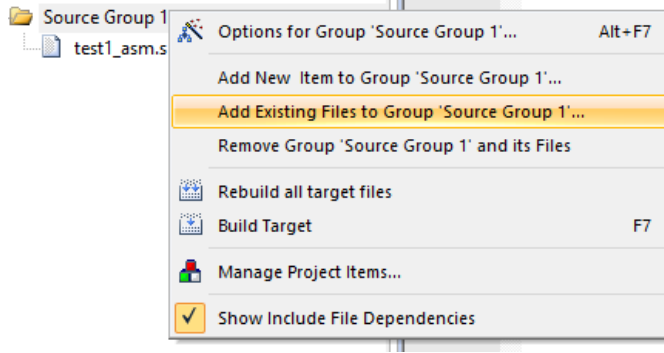
Proje açılması tamamlandıktan sonra artık kodumuzu ekleyeceğiz. Burada “Project” bölümünün altında bulunan “Source Group 1” kısmına sağ tıklayın. Daha sonrasında aşağıdaki gibi çıkan yeni pencerede “Add New Item to Group ‘Source Group 1’” kısmına tıklayın.



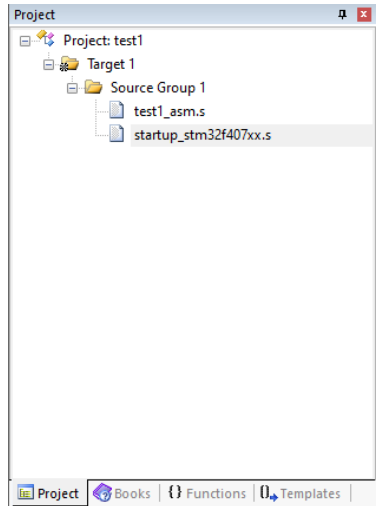
Açılacak yeni pencerede “Asm File(.s)” türünü seçin. Daha sonra aşağıda görülen “Name:” alanına oluşturacağınız dosyanın ismini girin. Bu örnekte aşağıda da görülebileceği üzere dosyaya “test1_asm” ismi verildi. Son olarak “Add” butonuna tıklayarak dosyayı projeye ekleyin.



Daha önceki sayfalarda bahsedildiği üzere kartı çalıştırmak için başlangıç dosyasına ihtiyacımız bulunuyor. Bu dosyayı eklemek için tekrar “Source Group 1” sekmesine sağ tıklayın. Açılan pencerede “Add Existing Files to Group ‘Source Group 1’” kısmına tıklayın. Bunu tamamladıktan sonra indirdiğiniz başlangıç dosyasını bulun ve projeye ekleyin.



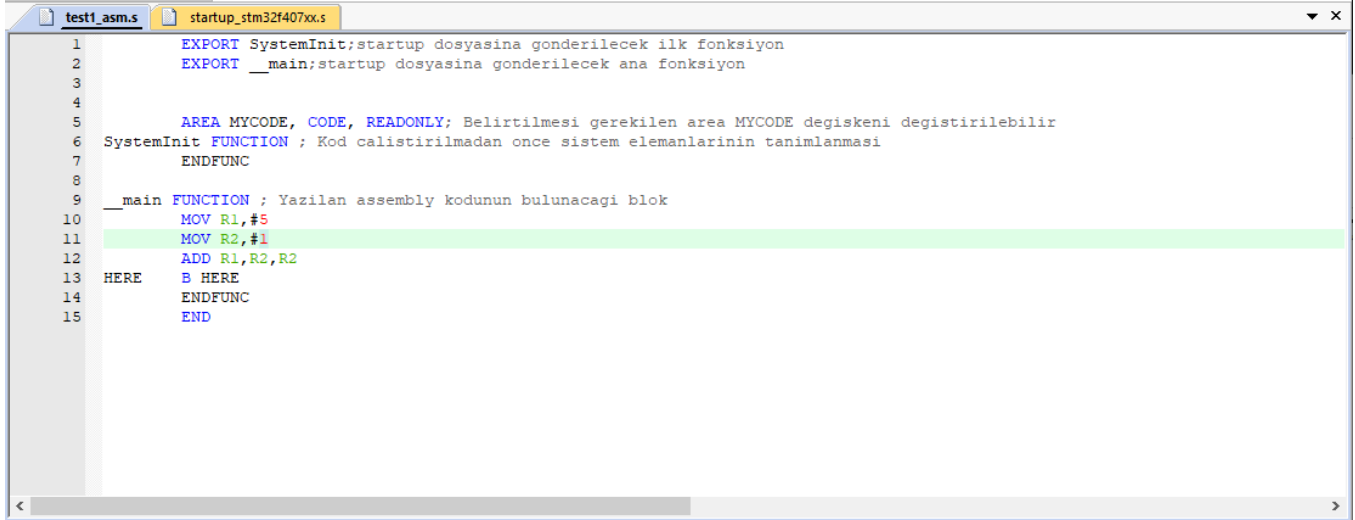
En sonunda oluşacak olan “Project” sekmesi aşağıdaki görsel gibi olmalıdır.



Bu adımlardan sonra artık kodunuzu “test1_asm.s” dosyasına yazabilirsiniz.

4.2 ASSEMBLY KODUNUN YAZILMASI

Daha önce yukarıda bahsedildiği gibi assembly kodunun haricinde başlangıç dosyasından dolayı kodu çalıştırmak için bazı kısıtlar bulunuyor. Aşağıda yazılmış örnek kodu inceleyelim.



```
1      EXPORT SystemInit;startup dosyasına gonderilecek ilk fonksiyon
2      EXPORT __main;startup dosyasına gonderilecek ana fonksiyon
3
4
5      AREA MYCODE, CODE, READONLY; Belirlilmesi gereken area MYCODE degiskeni degistirilebilir
6      SystemInit FUNCTION ; Kod calistirilmadan once sistem elemanlarinin tanimlanmasi
7          ENDFUNC
8
9      __main FUNCTION ; Yazilan assembly kodunun bulunacagi blok
10         MOV R1,#5
11         MOV R2,#1
12         ADD R1,R2,R2
13     HERE    B HERE
14         ENDFUNC
15         END
```

İlk iki satırda yer alan EXPORT buyruğu aslında startup dosyasına gönderilecek olan fonksiyonları belirtir.

Burada yer alan “SystemInit” adlı fonksiyon yazılan koddan bağımsız olarak eğer kartın çevre elemanları vs. gibi parçaları kullanılacaksa kod çalışmadan önce onların ayarlanması için kullanılır.

Diğer “__main” adlı fonksiyon ise yazdığınız kodun yer alacağı yerdir. Bu fonksiyonların kullanılması için gerekli syntax “<Fonksiyon İsmi> FUNCTION” olarak fonksiyon başlatılır. Daha sonra içine gerekli kod yazılır. Fonksiyon bittikten sonra “ENDFUNC” yazılarak fonksiyon sonlandırılır.

Fonksiyonlardan başka olarak uyulması gereken bir diğer syntax ise “AREA” ve “END” buyrukları.

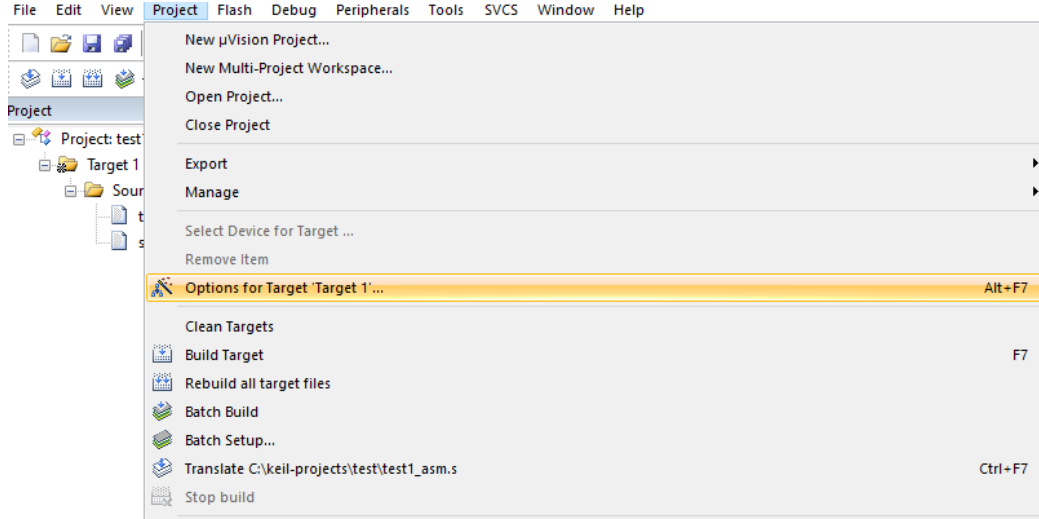
“AREA” buyruğu oluşturulurken koda verilecek izinlere karar verilir. Lablarımızda ise biz çoğunlukla “AREA <istediğiniz bir isim>, CODE, READONLY” yapısını kullanacağız.

“END” buyruğu ise kodun bittiğini gösteren bir buyruktur. Herhangi bir ek almaz kendinden sonra gelen buyrukları çalıştırmaz. Program sayacı o buyruğa geldiği anda yazılan program otomatik olarak sonlandırılır.

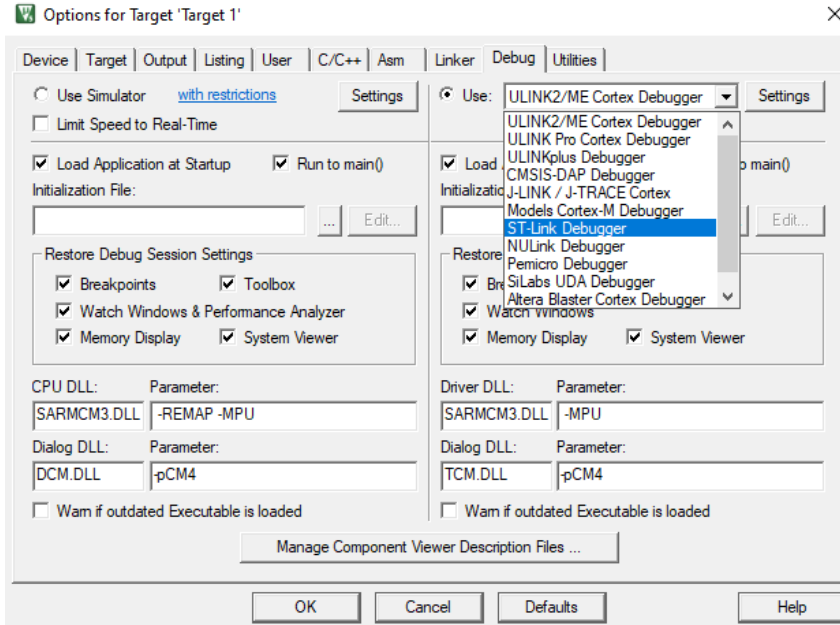
Yukarıda örnek olarak verilen koddaki, ana kodun herhangi bir çevre bileşenine ihtiyacı olmadığı için “SystemInit” fonksiyonu boş bırakılmıştır. Yazılan ana kod ise öncelikle R1 yazmacına 5 değerini, daha sonra R2 yazmacına 1 değerini yazar. En sonunda R1=R2+R2 işlemini gerçekleştirerek R1 değerini 2 olarak günceller.

4.3 KODUN DERLENMESİ VE KARTA ATILMASI

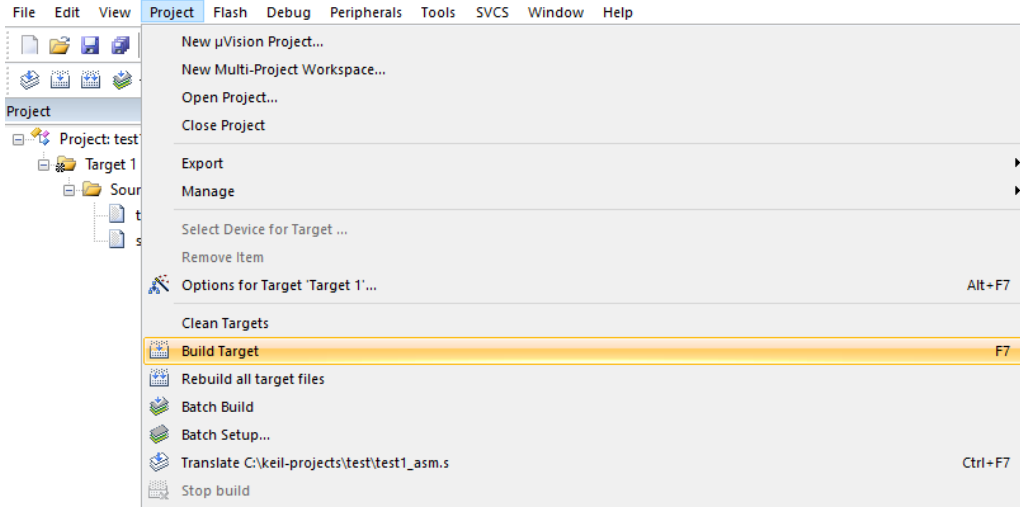
Kodu derlemek için öncelikle proje ayarlarını değiştirmemiz gerekiyor. Bunun için “Project” sekmesinden “Options for Target “Target 1”” kısmına tıklayın.



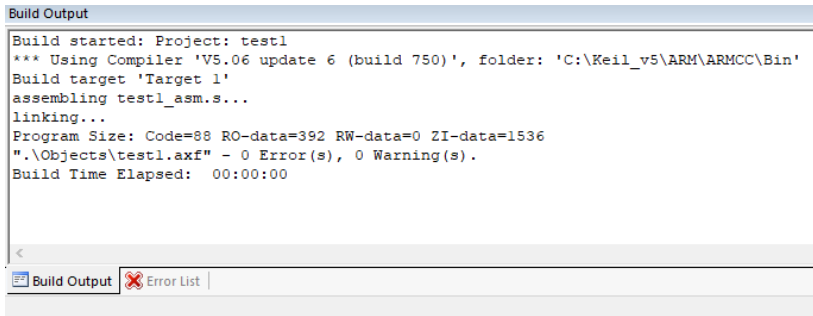
Açılan pencerede “Debug” sekmesine tıklayın. Daha sonra sağ bölmede “ULINK2/ME Cortex Debugger” adı verilen değişkeni “ST-Link Debugger” olarak değiştirin.



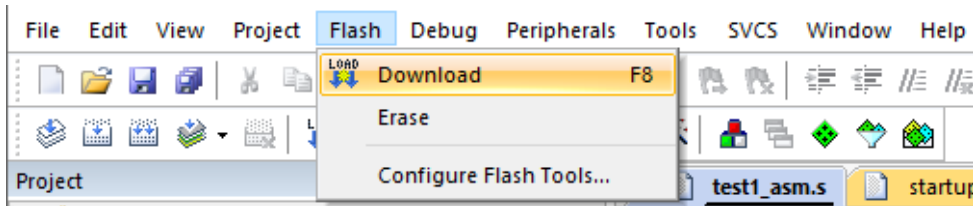
Bu ayarı düzelttikten sonra kodu derlemelisiniz. Bunu gerçekleştirmek için “Project” sekmesinden “Build Target”a tıklayın. Ya da kısayol olarak F7 tuşuna da basabilirsiniz.



Eğer konsolda aşağıdaki gibi bir çıktı elde ederseniz, yazdığınız kod herhangi bir hata almadan derlenmiş demektir.



Derlediğimiz kodu karta atmak için “Flash” sekmesine tıklayın. Daha sonra gelen pencerede “Download”a tıklayın. Ya da kısayol olarak direct F8 tuşunu kullanabilirsiniz.

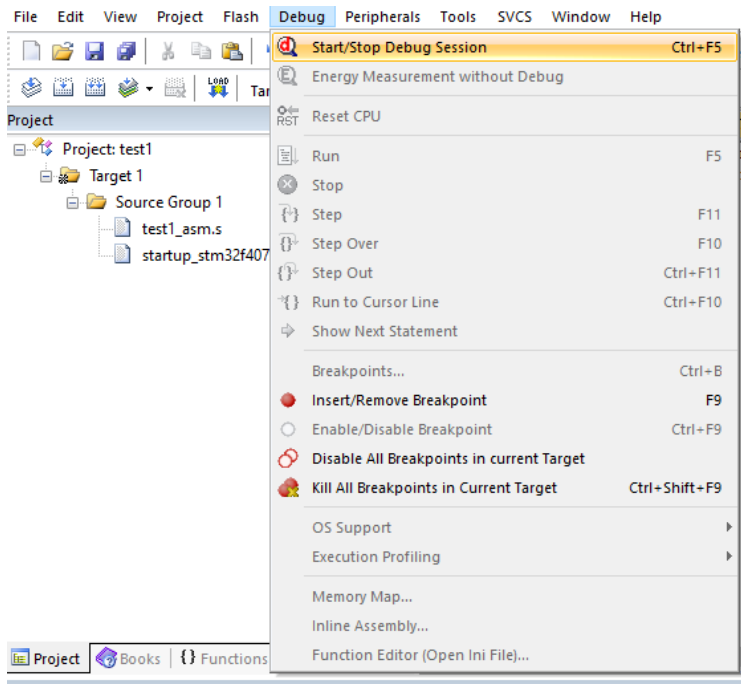


Eğer oluşan konsol çıktınız aşağıdaki gibiye başarılı bir şekilde yazdığınız kodla kartı programlamışsınız demektir.

```
Load "C:\\keil-projects\\test\\Objects\\test1.axf"  
Erase Done.  
Programming Done.  
Verify OK.  
Flash Load finished at 17:40:11
```

4.4 YAZILAN KODUN DEBUG EDİLMESİ

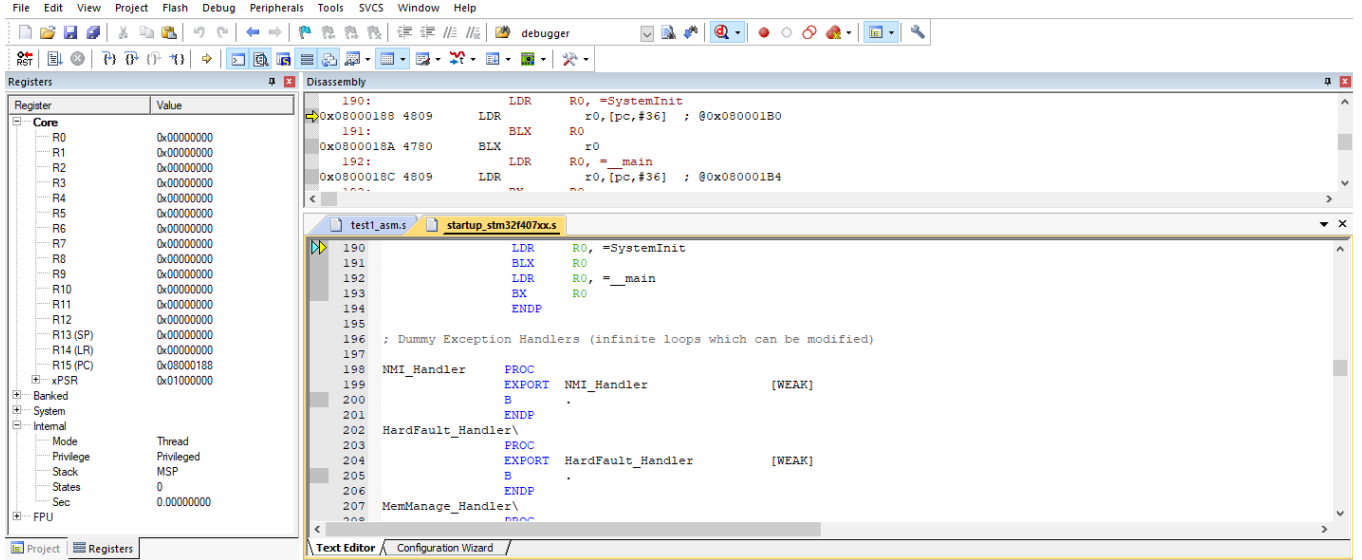
Yazılan kodların doğruluğunu ölçmek için led yakma veya bazı pinlere çıktı verme kullanılmaktadır. Fakat bu haftaki labda herhangi bir çevre elemanını kullanmadığımız için yazdığımız kodun doğruluğunu farklı bir şekilde ölçeceğiz. “Debug” sekmesine tıkladığınızda açılan pencerede “Start/Stop Debug Session” kısmına tıklayın.



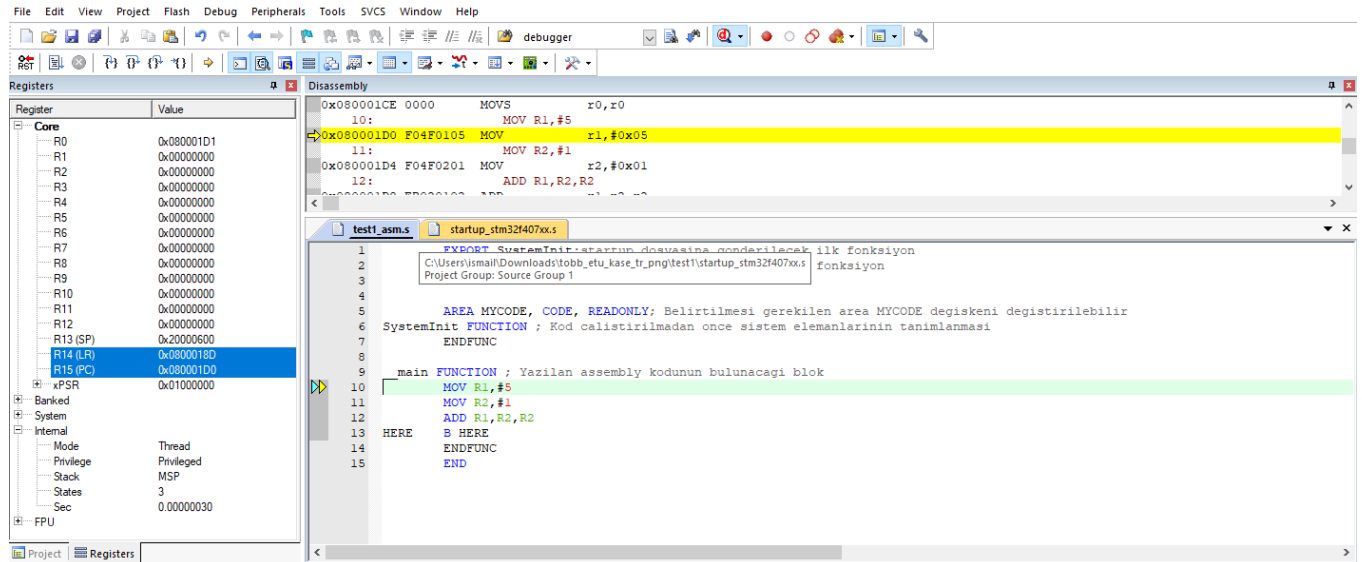
BİL362L - Mikroişlemciler Laboratuvarı

Lab 1 Öncesi

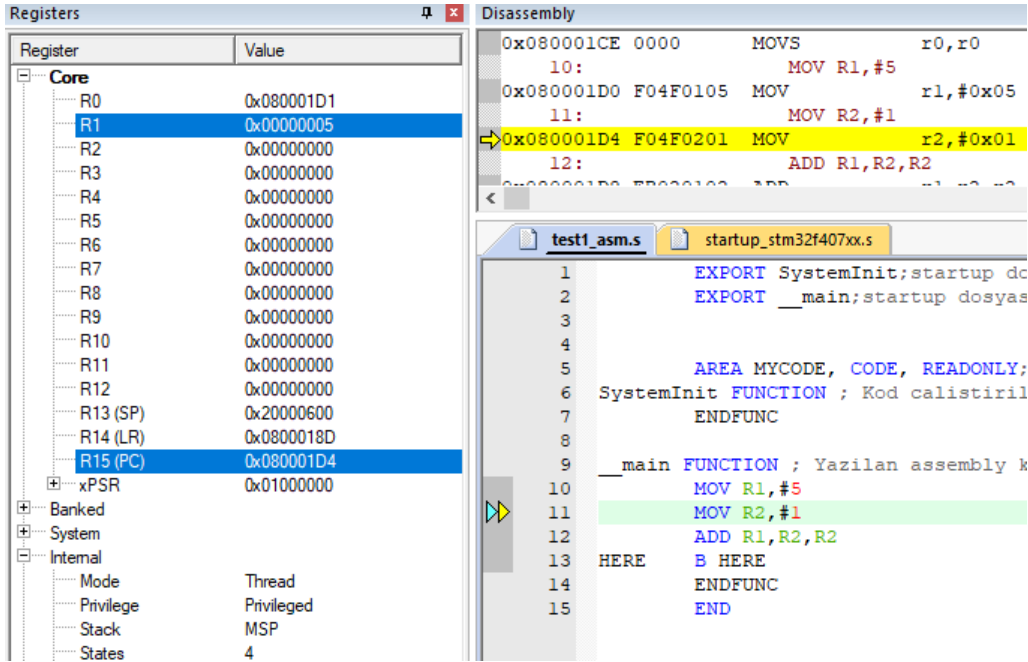
Açılacak pencere aşağıdaki gibi olacaktır. Aşağıdaki görselde gördüğünüz üzere solda yazmaç değerleri bulunmaktadır. Sağ tarafta “Disassembly” kısmında yazılan assembly kodun dönüştüğü daha düşük seviye dil ve kodun kaldığı yer bulunmaktadır. Onun aşağısında ise yazdığınız assembly kodu ve orada karşılık gelen kodun kaldığı yer bulunmaktadır.



Birkaç kez F11 tuşuna basarak aşağıdaki gibi kodun kaldığı yeri “__main” fonksiyonunun olduğu yere kadar getirin.



Bir adım daha ilerlettiğinizde aşağıdaki gibi R1 yazmacına 5 değerinin yazıldığını göreceksiniz.



Register	Value
R0	0x08001D1
R1	0x00000005
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000600
R14 (LR)	0x080018D
R15 (PC)	0x08001D4
xPSR	0x01000000

```
0x08001CE 0000 MOVS r0,r0
10:          MOV R1,#5
0x08001D0 F04F0105 MOV r1,#0x05
11:          MOV R2,#1
0x08001D4 F04F0201 MOV r2,#0x01
12:          ADD R1,R2,R2
0x08001D8 F04F0105 MOV r1,r2,r2
13:          B HERE
14:          ENDFUNC
15:          END

test1_asm.s startup_stm32f407xx.s
1          EXPORT SystemInit;startup dc
2          EXPORT __main;startup dosyas
3
4          AREA MYCODE, CODE, READONLY;
5          SystemInit FUNCTION ; Kod calistiril
6          ENDFUNC
7
8          __main FUNCTION ; Yazilan assembly k
9          MOV R1,#5
10         MOV R2,#1
11         ADD R1,R2,R2
12         HERE B HERE
13         ENDFUNC
14         END
```

Bir adım daha ilerlettiğinizde R2 yazmacına 1 yazıldığını göreceksiniz. Daha sonraki adımda ise R1 yazmacına yazılan sayı artık 5 değil 2 olacaktır. Artık kod 13.satıra yani “HERE B HERE” buyruğuna geldiğinde orada sonsuz döngü oluşacağından kodumuz ne kadar ilerletirsek ilerletelim o satırda kalacaktır.

“Debug” özelliğini kullanarak kodumuzun doğru çalışıp çalışmadığını görmüş olduk. Aslında sadece bunun için değil, kod doğru çalışmıyorsa nerede hatalı çalıştığını, buna neyin sebep olduğunu, bu arayüzü kullanarak bulabilirsiniz.