

**THE IMAGINATION UNIVERSITY PROGRAMME**

**RVfpga Deney 3**

**RISC-V Çevirici Dili**

# GİRİŞ

C, Java, Python gibi yüksek-düzeyli dillerde programlamak programlamacı için verimlidir. Bu yüksek-düzeyli diller yalın yönergeler grubu olan çevirici diline çevrilir. Yer yer performansı ya da zamanlaması önemli kod bölümleri belirli zamanlamanın garantisini vermek ya da işlem süresini azaltmak için çevirici dilinde yazılır. Bu deney sana PlatformIO’yu kullanarak RVfpga’de çalıştırabileceğin bir RISC-V çevirici dili programının nasıl oluşturulacağını gösterir. RISC-V çeviricisinin kısa bir tanıtımını yapıp ardından RVfpga’de nasıl bir çevirici programı oluşturulup çalıştırılır gösteriyoruz. Ardından kendi RISC-V çevirici programlarını yazmanın uygulamasını yapman için alıştırmalar veriyoruz.

# RISC-V Çevirici Dili Tanıtımı

RISC-V çevirici dili yüksek-düzeyli dilleri gerçekleştirmek için kullanılan yalın yönergeleri içerir. Örneğin, yaygın RISC-V yönergeleri iki işleneni toplayan, çıkaran ya da çarpan add, sub, mul yönergelerini içerir.

RISC-V yönergelerinin basit türleri: sayısal (arithmetic, logical, and shift) yönergeler, bellek işlemleri, dallandırmalar/sıçramalardır. Tablo 1’de en yaygın RISC-V yönergeleri verilmiştir. Yönergeler yazmaçlarda ya da bellekte yerleşmiş ya da sabit (diğer bir deyişle *anlık*) olarak kodlanmış işlenenleri kullanır. RISC-V 32 tane 32-bit yazmaç içerir. Tablo 2 32 RISC-V yazmacının adlarını verir. Adlarıyla (örneğin, zero, s0, t5, gibi gibi.) ya da yazmaç numaralarıyla (örneğin, x0, x8, x30) tanımlanabilirler. Programlamacılar genelde yazmacın genel amacı üzerine bilgi barındıran yazmaç adlarını kullanır. Örneğin, kaydedilmiş yazmaçlar, s0-s11, genelde program değişkenleri için kullanılırken, geçici yazmaçlar, t0-t6 geçici işlemler için kullanılır. zero yazmacı (x0), bu degree programlarda çok gerek duyulduğu için, hep 0 değerini barındırır. Tablo 2’de gösterildiği gibi diğer yazmaçların da belirli kullanımları var ancak bu deneyde yalnızca zero yazmacı, geçiciyle kaydedilmiş yazmaçları kullanman gerekecek.

Tablo 1. Yaygın RISC-V çevirici yönergeleri

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | **RISC-V Çeviricisi** | | **Tanım** | | **İşlem** |
| **Sayısal** | add s0, s1, s2 | | Ekle | | s0 = s1 + s2 | |
| sub s0, s1, s2 | | Çıkar | | s0 = s1 - s2 | |
| addi t3, t1, -10 | | Anlıkla ekle | | t3 = t1 – 10 | |
| mul t0, t2, t3 | | 32-bit çarp | | t0 = t2 \* t3 | |
| div s9, t5, t6 | | Bölüm | | t9 = t5 / t6 | |
| rem s4, s1, s2 | | Kalan | | s4 = s1 % s2 | |
| and t0, t1, t2 | | Bitsel AND yap | | t0 = t1 & t2 | |
| or t0, t1, t5 | | Bitsel OR yap | | t0 = t1 | t5 | |
| xor s3, s4, s5 | | Bitsel XOR yap | | s3 = s4 ^ s5 | |
| andi t1, t2, 0xFFB | | Anlıkla bitsel AND yap | | t1 = t2 & 0xFFFFFFFB | |
| ori t0, t1, 0x2C | | Anlıkla bitsel OR yap | | t0 = t1 | 0x2C | |
| xori s3, s4, 0xABC | | Anlıkla bitsel XOR yap | | s3 = s4 ^ 0xFFFFFABC | |
| sll t0, t1, t2 | | Sola mantıksal kaydır | | t0 = t1 << t2 | |
| srl t0, t1, t5 | | Sağa mantıksal kaydır | | t0 = t1 >> t5 | |
| sra s3, s4, s5 | | Sağa aritmetik kaydır | | s3 = s4 >>> s5 | |
| slli t1, t2, 30 | | Anlıkla sola mantıksal kaydır | | t1 = t2 << 30 | |
| srli t0, t1, 5 | | Anlıkla sağa mantıksal kaydır | | t0 = t1 >> 5 | |
| srai s3, s4, 31 | | Anlıkla sağa aritmetik kaydır | | s3 = s4 >>> 31 | |
| **Bellek** | lw s7, 0x2C(t1) | | Sözcüğü yükle | | s7 = memory[t1+0x2C] | |
| lh s5, 0x5A(s3) | | Yarım-sözcüğü yükle | | s5 = SignExt(memory[s3+0x5A]15:0) | |
| lb s1, -3(t4) | | Baytı yükle | | s1 = SignExt(memory[t4-3]7:0) | |
| sw t2, 0x7C(t1) | | Sözcüğü depola | | memory[t1+0x7C] = t2 | |
| sh t3, 22(s3) | | Yarım-sözcüğü depola | | memory[s3+22]15:0 = t315:0 | |
| sb t4, 5(s4) | | Baytı depola | | memory[s4+5]7:0 = t47:0 | |
| **Dallanma** | beq s1, s2, L1 | | Eşitse dallandır | | if (s1==s2), PC = L1 | |
| bne t3, t4, Loop | | Eşit değilse dallandır | | if (s1!=s2), PC = Loop | |
| blt t4, t5, L3 | | Azsa dallandır | | if (t4 < t5), PC = L3 | |
| bge s8, s9, Done | | Çok ya da eşitse dallandır | | if (s8>=s9), PC = Done | |
| **Sözde Yönergeler** | li s1, 0xABCDEF12 | | Anlığı yükle | | s1 = 0xABCDEF12 | |
| la s1, A | | Adresi yükle | | s1 = A değişkeninin depolandığı bellek adresi | |
| nop | | İşlem yok | | işlem yok | |
| mv s3, s7 | | Taşı | | s3 = s7 | |
| not t1, t2 | | Değil (Tersine Döndür) | | t1 = ~t2 | |
| neg s1, s3 | | Olumsuzla | | s1 = -s3 | |
| j Label | | Sıçra | | PC = Label etiketi | |
| jal L7 | | Sıçrayıp bağla | | PC = L7; ra = PC + 4 | |
| jr s1 | | Yazmaca sıçra | | PC = s1 | |

Gerçek RISC-V yönergelerinin yanı sıra RISC-V, gerçek RISC-V yönergesi olmayan ancak programlamacıların sıkça kullandığı, sözde yönergeler de içerir (Tablo 1’in aşağısında gösterildiği gibi). Sözde yönergeler bir ya da daha çok gerçek RISC-V yönergesi kullanılarak gerçekleştirilir. Örneğin, taşıma sözde yönergesi (mv s1, s2) s2’nin içeriğini kopyalayıp s1’e koyar. Şu gerçek RISC-V yönergesiyle gerçekleştirilir: addi s1, s2, 0.

**Tablo 2. RISC-V yazmaçları**

|  |  |  |
| --- | --- | --- |
| **Ad** | **Yazmaç Numarası** | **Kullanım** |
| **zero** | **x0** | Sabit değer 0 |
| **ra** | **x1** | Dönüş adresi |
| **sp** | **x2** | Yığın göstergesi |
| **gp** | **x3** | Genel gösterge |
| **tp** | **x4** | İş parçacığı göstergesi |
| **t0-2** | **x5-7** | Geçici değişkenler |
| **s0/fp** | **x8** | Kaydedilmiş değişken / Çerçeve göstergesi |
| **s1** | **x9** | Kaydedilmiş değişken |
| **a0-1** | **x10-11** | İşlev argümanları / Dönüş değerleri |
| **a2-7** | **x12-17** | İşlev argümanları |
| **s2-11** | **x18-27** | Kaydedilmiş değişkenler |
| **t3-6** | **x28-31** | Geçici değişkenler |

# Nokta ile başlayan komutlar çevirici program yönlendirmeleridir. Çevirici programca çevirilecek koddan çok çevirici programa komutlardır. Çevirici programa kodla verinin nereye konulacağını, programda kullanılmak üzere metin ile veri sabitlerini, gibi gibi bilgileri söylerler. Tablo 3 RISC-V çevirici programının ana yönlendirmelerini gösterir (*The RISC-V Reader: An Open Architecture Atlas, Patterson & Waterman*, © 2017).

**Tablo 3. RISC-V ana yönlendirmeleri**

|  |  |
| --- | --- |
| **Yönlendirme** | **Tanım** |
| **.text** | Arkasından gelen ögeler text bölümünde depolanır (makine kodu). |
| **.data** | Arkasından gelen ögeler data bölümünde depolanır (genel değişkenler). |
| **.bss** | Arkasından gelen ögeler bss bölümünde depolanır (0’la ilk değerlenen genel değişkenler). |
| **.section .foo** | Arkasından gelen ögeler .foo adlı bölümde depolanır. |
| **.align n** | Sonraki veriyi 2n-bayt sınırına hizalar. Örneğin, .align 2 sonraki değeri sözcük sınırına hizalar. |
| **.balign n** | Sonraki veriyi n-bayt sınırına hizalar. Örneğin, .balign 4 sonraki değeri sözcük sınırına hizalar. |
| **.globl sym** | sym etiketinin genel olup diğer dosyalardan referanslanabileceğini bildirir. |
| **.string “str”** | str karakter dizisini bellekte depolayıp boşla-sonlandır. |
| **.word w1,…,wn** | n 32-bit niceliği ardışık bellek sözcüklerinde depolar. |
| **.byte b1,…,bn** | n 8-bit niceliği belleğin ardışık baytlarında depolar. |
| **.space** | İlk değerleri verilmemiş değişken depolamak için bellek boşluğu ayırır. Yaygın olarak, girdi olarak da görev yapmadıklarında, çıktı değişkenlerini bildirmek için kullanılır. İstediğimiz boşluk hep byteların bir sayısı olarak ifade edilmelidir. Örneğin, RES: .space 4 yönlendirmesi ilk değeri verilmemiş dört bayt (bir diğer deyişle bir sözcük) ayırır. |
| **.equ name,constant** | name sembolünü constant değeriyle tanımla. Örneğin, .equ N,12, sembol N’yi 12 değeriyle tanımlar. |
| **.end** | .end yönlendirmesine ulaştığında çevirici program işini sonuçlandırır. Bu yönlendirmeden sonraki metinler yok sayılacaktır. |

# Aşağıdaki örnekler (Tablo 4 - Tablo 5’e göz at) RISC-V çeviricisindeki yaygın olarak kullanılan yüksek-düzeyli yapılan nasıl kodlanacağını gösterir. Önlemli olarak dallandırma yönergeleri (beq, bne, blt, and bge) koşullu olarak bir etikete sıçrarken; sıçrama yönergesi (j) koşulsuz bir etikete sıçrar. Bir-satırlı yorumlar C’de // ile gösterilir, RISC-V çeviricisinde # ile gösterilir.

İlk örnekte (bir if/else deyimini gerçekleştirme, Tablo 4’e göz at), önemli olarak C koduyla RISC-V çevirici kodu karşıt durumları denetler: C kodu azdırı denetler (<) çevirici eşdeğeri çok ya da eşitseyi denetler (>=).

**Tablo 4. RISC-V Çevirici Örneği 1: if/else deyimi**

|  |  |
| --- | --- |
| **// C Kodu**  int a, b, c;  if (a < b)  c = 5;  else  c = a + b; | **# RISC-V Çeviricisi**  # s0 = a, s1 = b, s2 = c  bge s0, s1, L1 # if (a >= b) goto L1  addi s2, zero, 5 # c = 5  j L2 # else bloğunu atla  L1: add s2, s0, s1 # c = a + b  L2: |

İkinci örnekte (bir tam sayı dizisini işleme, Tablo 5’e göz at), RISC-V çevirici kodu geçici değişkenleri tutmak için geçici yazmaçları (t0-t3) kullanır, sabit 100 ile veri dizisinin taban adresi gibi. İlk üç yönergede yazmaçlara ilk değerlerini verdiken sonra RISC-V çevirici kodu i >= 100’ü bge (çok ya da eşitse dallandır) yönergesi ile denetler; yine C kodunun karşıtı. Eğer koşul sağlanmışsa for döngüsü biter. Eğer dala **geçilmemişse**, i 100’den azdır dolayısıyla kalan kod yürütülür. Önemli olarak tam sayılar (32-bit ikiye tümler sayılar) bellekte 4 bayt kapladığından dolayı taban adrese eklenmeden önce i 4’le çarpılır (slli t2, s0, 2 yönergesini kullanarak. RISC-V’da bellek baytla-adreslenebilirdir (bir diğer deyişle bir bayt başına bir adres vardır). Dizi bir karakterler dizisi olsaydı (i.e., char data[100];), dizideki ögeler yalnızca birer bayt kaplar, i sayısı i dizi dizini (array[i]) oluşturmak için direkt olarak taban adresine eklenebilirdi. Dizin ögesi okunup, on azaltılıp, yazıldıktan sonra (lw, addi, sw yönergeleriyle), dizi dizini i (s0) arttırılıp program for döngüsünün başlangıcına sıçrar (j L5 yönergesini kullanarak).

**Tablo 5. RISC-V Çevirici Örneği 2: tam sayı dizisini işleme**

|  |  |
| --- | --- |
| **// C Kodu**  int i;  int data[100];  for (i=0; i<100; i++)  array[i] = array[i]-10; | **# RISC-V Çeviricisi**  # s0 = i, t1 = verinin taban adresi  # (0x300’da olduğu varsayılır)  addi s0, zero, 0 # i = 0  addi t0, zero, 100 # t0 = 100  li t1, 0x300 # dizinin taban adresi  L5: bge s0, t0, L7 # if (i>=100) döngüden çık  slli t2, s0, 2 # t2 = i\*4  add t2, t1, t2 # data[i]’nin adresi  lw t3, 0(t2) # t3 = array[i]  addi t3, t3, -10 # t3 = array[i]-10  sw t3, 0(t2) # array[i] = array[i]-10  addi s0, s0, 1 # i++  j L5 # döngü  L7: |

RISC-V çevirici dili üzerine daha çok detay için RISC-V Yönerge Kümesi El Kitabına (şuradan erişilebilir: <https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf>) ya da *Digital Design and Computer Architecture*, Harris & Harris, Elsevier, © 2021 (2021 yazında yayınlanacak) gibi bir ders kitabına ya da *The RISC-V Reader: An Open Architecture Atlas*, Patterson & Waterman, © 2017 bak.

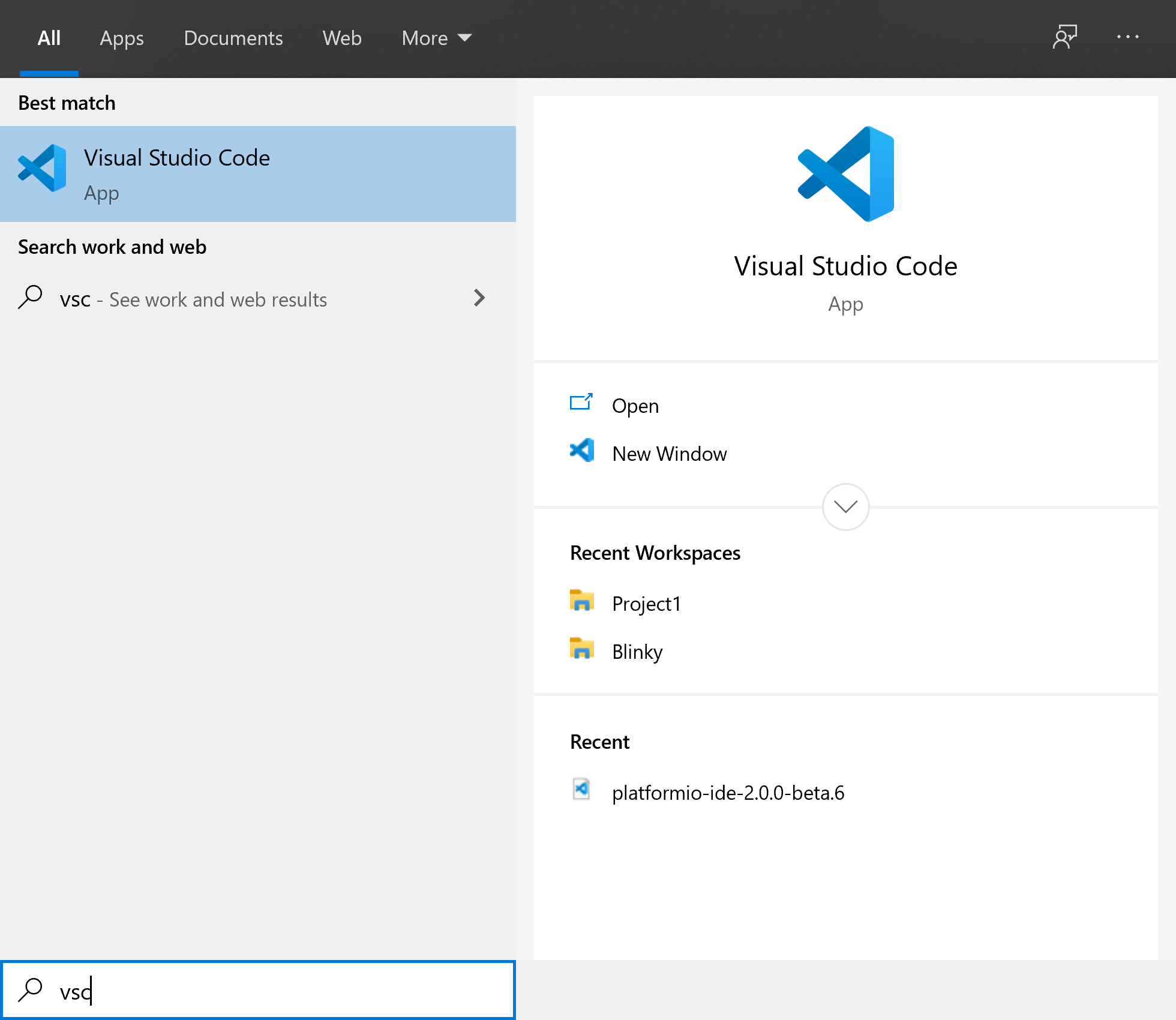
# RVfpga için bir RISC-V Çevirici Programı Yazma

Şimdi RISC-V çevirici programları yazmayı kendin keşfedip alıştırma olarak uygulayabilirsin. Kendi programlarını yazmadan önce bir PlatformIO projesi kurup RVfpga’de çevirici programları oluşturup çalıştırmak için şu adımları izle:

1. Bir RVfpga projesi oluştur
2. RISC-V çevirici dili programı yaz
3. RVfpga’i Nexys A7 FPGA kartına indir
4. C programını derle, indir, çalıştır

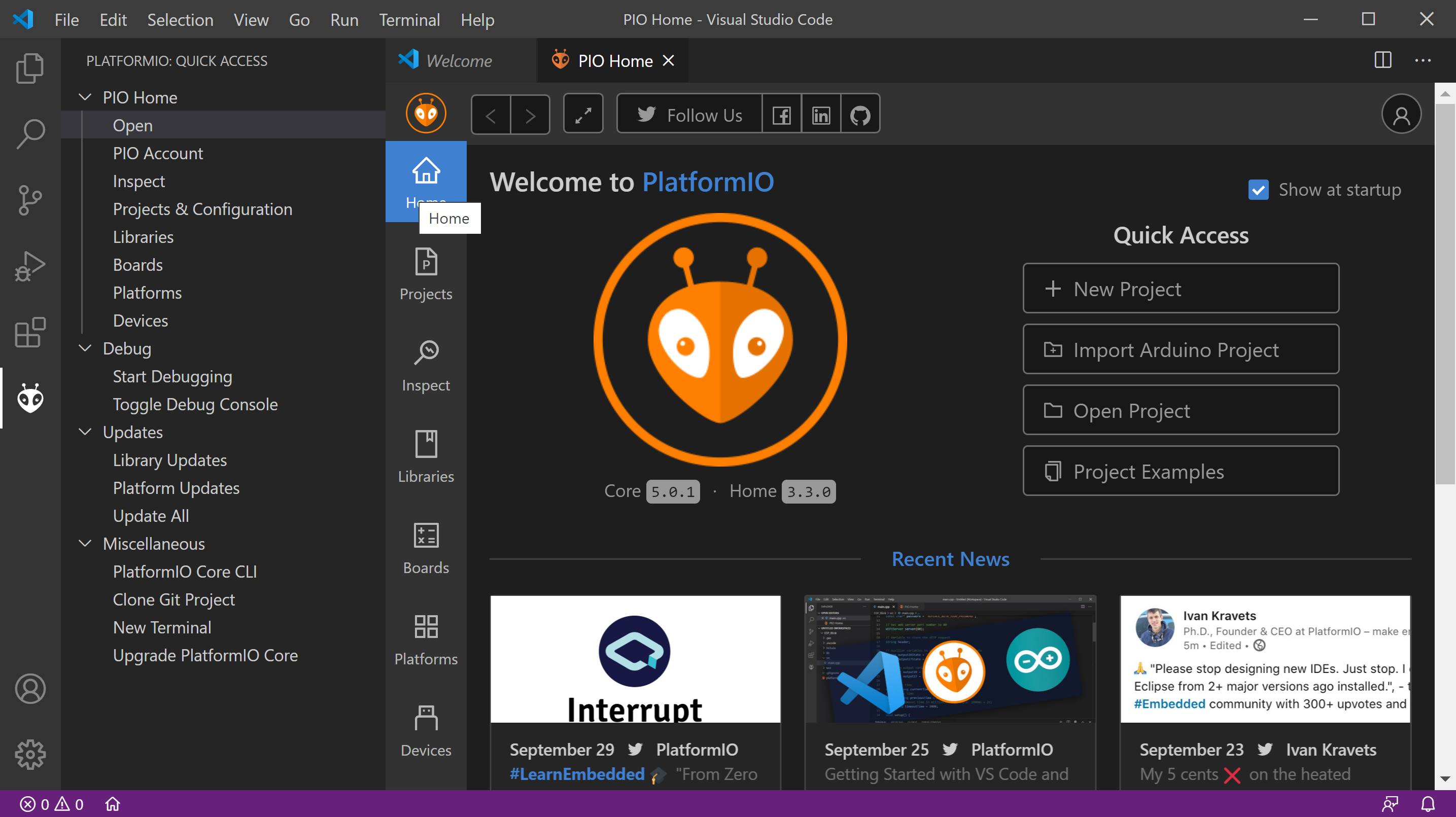
Adım 1. Bir RVfpga projesi oluştur

RVfpga Deney 2’nin Adım 1’ini izle – kolaylık olması için burada da yazılı. Start (Başlat) butonuna tıklayıp, VSCode yazıp ardından Visual Studio Code’a tıklayıp VSCode’u aç (Figür 1’e göz at).



Figür 1. VSCode’u aç

VSCode’u açtığında PlatformIO kendiliğinden açılmıyorsa sol menü şeridindeki PlatformIO ikonuna tıklayıp ardından PIO Home → Open (Aç) üzerine tıkla (Figür 2’ye göz at).

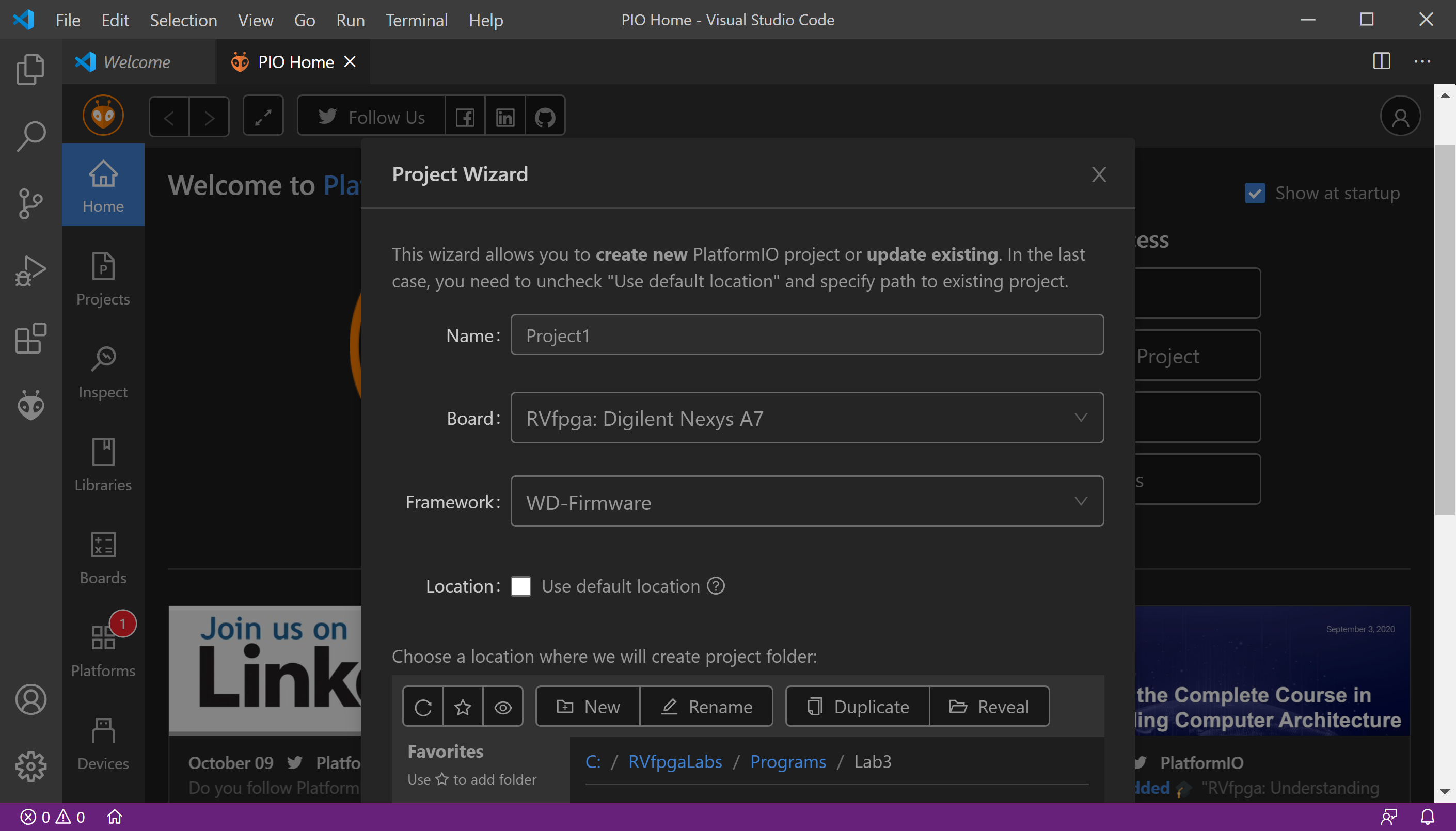


Figür 2. PlatformIO’yu açıp yeni proje oluştur

Şimdi, PIO Home hoşgeldin penceresinde New Project (Yeni Proje) üzerine tıkla (Figür 2’ye göz at).

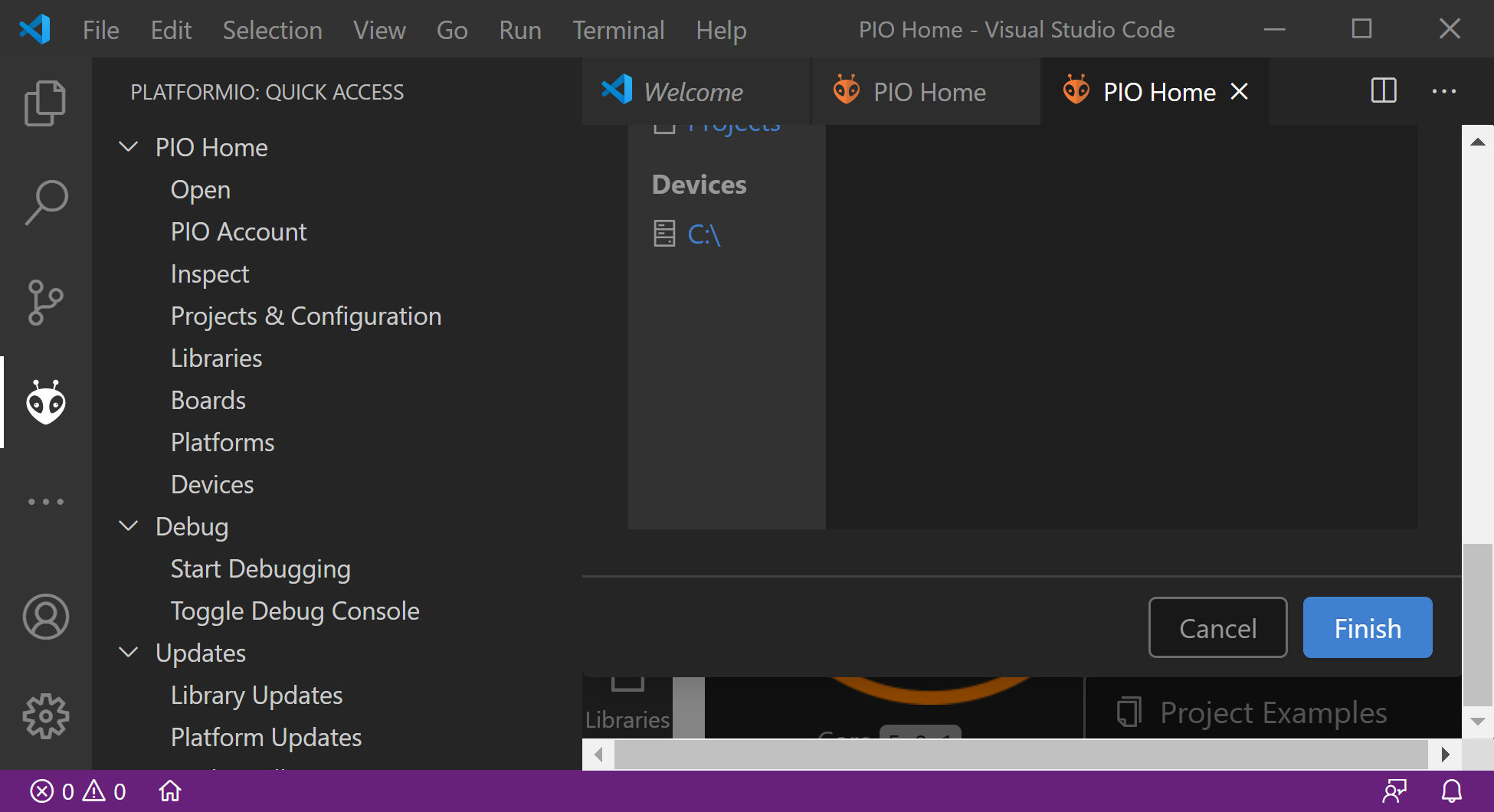
Figür 3’te gösterildiği gibi projeyi Project1 diye adlandırıp kartı RVfpga: Digilent Nexys A7 (RVfpga yazmaya başlarsan kart gözükecektir) olarak seç. Varsayılan çerçeveyi WD-framework (Western Digital framework –Freedom-E SDK gcc ile gdb içerir). Use default location (varsayılan konumu kullan) işaretini kaldırıp programını şuraya yerleştir:

[RVfpgaPath]/RVfpga/Labs/Lab3



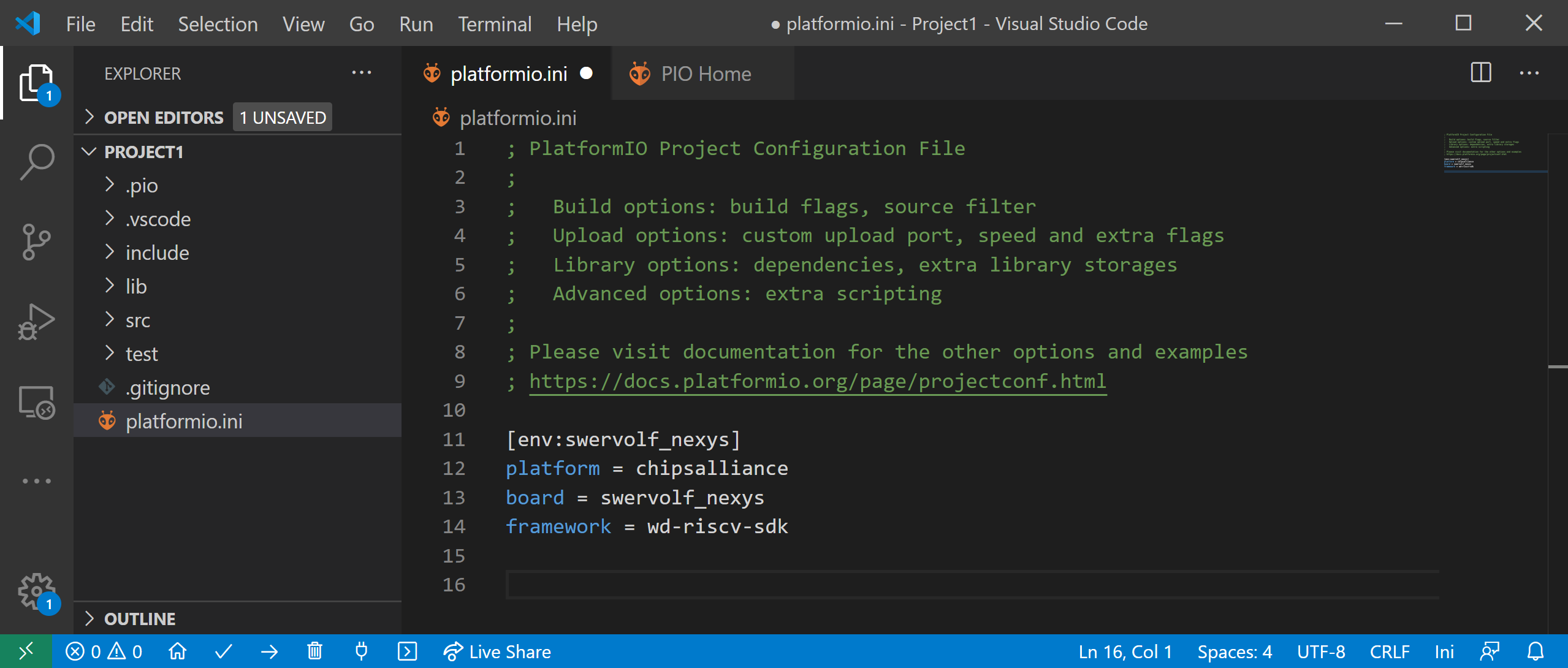
Figür 3. Projeni adlandırıp, kart ile proje klasörünü seç

Pencerenin aşağısında Finish’e (Bitir) tıkla (Figür 4’e göz at).



Figür 4. Proje oluşturmayı bitir

Soldaki Explorer (Gezgin) panelinde, PROJECT1 altında (bunu genişletmen gerekebilir), platformio.ini’yi açmak için üzerine çift-tıkla (Figür 5’e göz at). Bu PlatformIO ilk değerlendirme dosyasıdır.

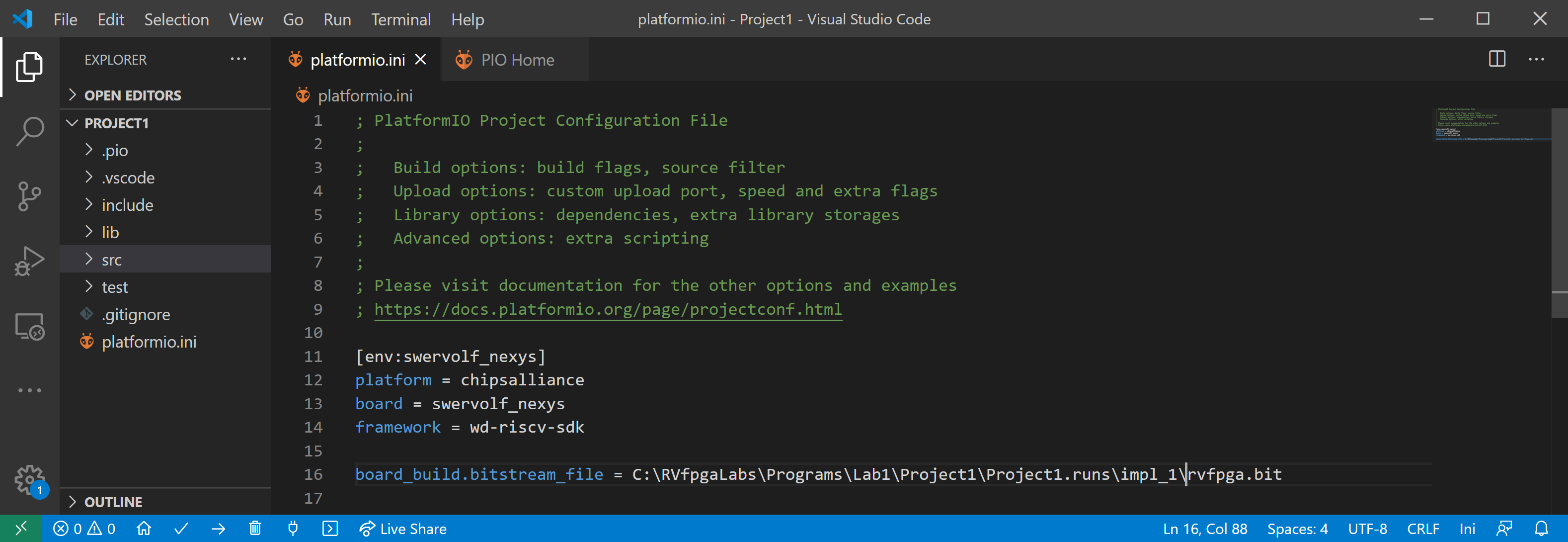


Figür 5. PlatformIO ilk değerlendirme dosyası: platformio.ini

Şu satırı platformio.ini dosyasına ekle, Figür 6’da gösterildiği gibi:

board\_build.bitstream\_file = [RVfpgaPath]/RVfpga/Labs/Lab1/Project1/Project1.runs/impl\_1/rvfpga.bit

Bu satır FPGA’e yüklemek için PlatformIO’nun veri akışı dosyasını nerede bulması gerektiğini gösterir. Yukarıdaki yol Deney 1’de oluşturduğun veri akışının yeri. (Eğer Deney 1’I bitirmediysen, İlk Kullanım Kılavuzu ile dağıtılan şuradaki RVfpga veri akışını kullanabilirsin: *[RVfpgaPath]/RVfpga/src/rvfpga.bit*.) platformio.ini dosyasını kaydetmek için Ctrl-s’e bas.

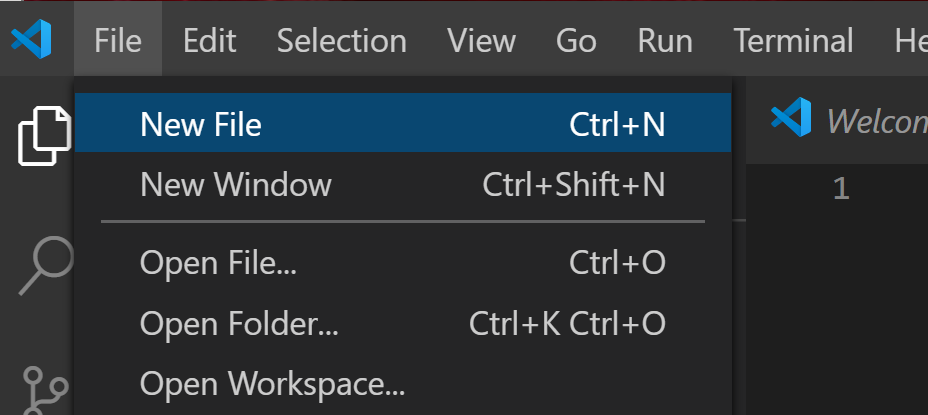


Figür 6. RVfpga veri akışı (bitstream) dosyasının yerini ekle (rvfpga.bit)

İlk Kullanım Kılavuzunda kullanılan örneklerde daha bütün bir *platformio.ini* dosyasının kullanıldığını unutma. Ek komutlar gerektiren bir işlevsellik kullanmak istersen (Verilator simülatörüne giden yol, dizisel konsolun yapılandırılması, whisper ayıklama aracı, gibi gibi.), o örneklerdeki *platformio.ini*’yi kullanabilirsin.

Adım 2. Bir RISC-V çevirici dili programı yaz

Şimdi bir RISC-V çevirici programı yazacaksın. File (Dosya) → New File (Yeni Dosya) üzerine tıkla (Figür 7’ye göz at).



Figür 7. Projeye dosya ekle

Boş bir pencere açılacaktır. Şu RISC-V çevirici programını o pencereye yaz (ya da kopyala/yapıştır) (Figür 8’e göz at). Bu programa şuradan da erişilebilir:

[RVfpgaPath]/RVfpga/Labs/Lab3/ReadSwitches.S

// memory-mapped I/O addresses

# GPIO\_SWs = 0x80001400

# GPIO\_LEDs = 0x80001404

# GPIO\_INOUT = 0x80001408

.globl main

main:

main:

li t0, 0x80001400 # base address of GPIO memory-mapped registers

li t1, 0xFFFF # set direction of GPIOs

# upper half = switches (inputs) (=0)

# lower half = outputs (LEDs) (=1)

sw t1, 8(t0) # GPIO\_INOUT = 0xFFFF

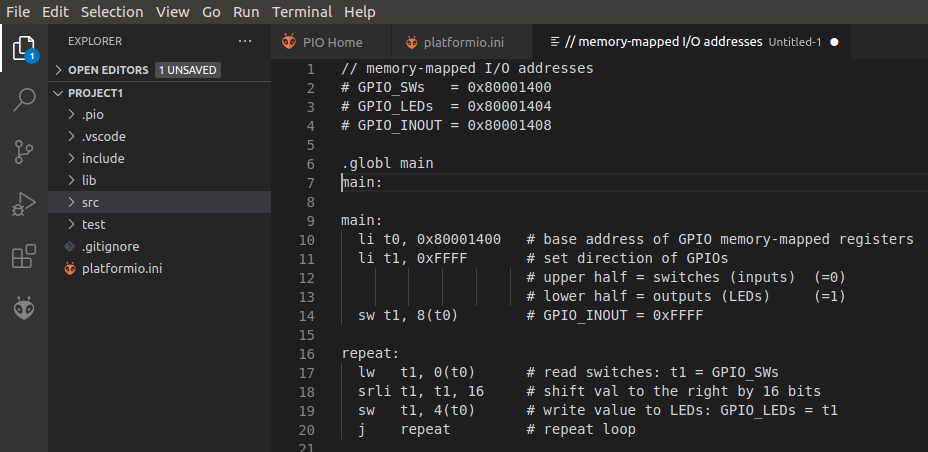
repeat:

lw t1, 0(t0) # read switches: t1 = GPIO\_SWs

srli t1, t1, 16 # shift val to the right by 16 bits

sw t1, 4(t0) # write value to LEDs: GPIO\_LEDs = t1

j repeat # repeat loop



Figür 8. RISC-V çevirici programını gir

Çevirici kodu, kod başlangıcında şu satırları barındırmalıdır:

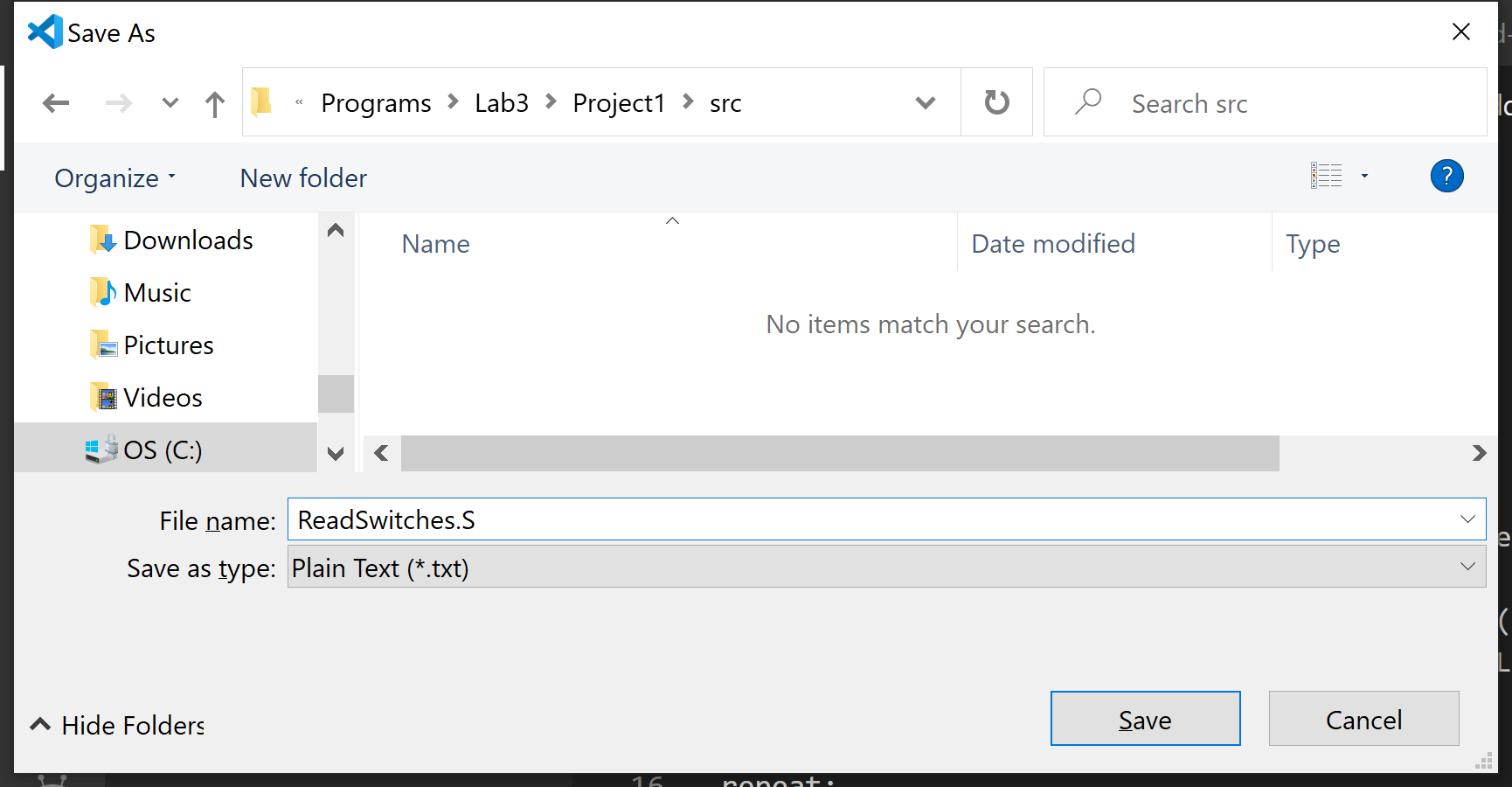
.globl main

main:

.globl çevirici yönlendirmesi etiketi bütün dosyalarda görünür kılar. Önyükleme kodu (*~/.platformio/packages/framework-wd-riscv-sdk/board/nexys\_a7\_eh1/startup.S*) sistemi yapılandırıp bu etikete sıçrayacaktır (*main*). Ayıklayıcı başladığında oraya geçici bir kesme noktası ayarlayacaktır.

Bu RISC-V çevirici programı Deney 2’deki örnekle aynıdır ancak RISC-V çeviricisiyle yazılmıştır. Genel-amaçlı I/O’nun (GPIO) yönünü ayarlayıp, anahtarların değerlerini durmaksızın okuyup LEDlere yazar..

Programı panele girdikten sonra dosyayı kaydetmek için Ctrl-s’e bas. ReadSwitches.S olarak adlandırıp Project1 dizininin src klasörüne kaydet (Figür 9’a göz at).



Figür 9. Dosyayı ReadSwitches.S olarak kaydet

Adım 3. RVfpga’i Nexys A7 FPGA kartına indir

Şimdi RVfpga’i Nexys A7 FPGA kartına indireceksin. RVfpga’i indirme için yönergeleri GSG ile Deney 2’de tanımlandığı gibi izle – burada kolaylık olması için tekrarlanmıştır.

Sol menü şeridindeki PlatformIO ikonuna  tıklayarak RVfpga’i Nexys A7 kartına indir, ardından Project Tasks → env:swervolf\_nexys → Platform genişletip Upload Bitstream tıkla.

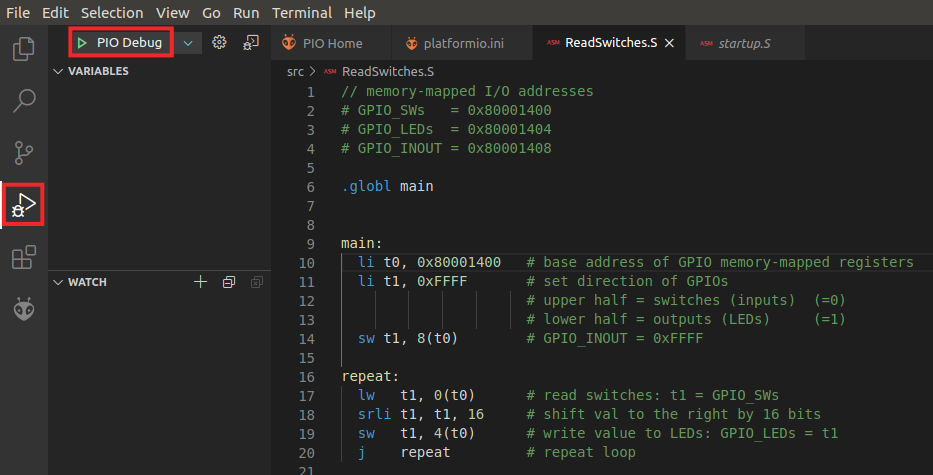
Alternatif olarak Figür 11’de gösterildiği gibi RVfpga’i bir PlatformIO terminal pencersinden de indirebilirsin. PlatformIO penceresinin aşağısındaki PlatformIO: New Terminal (PlatformIO: Yeni Terminal) butonuna () tıklayıp, ardından şunu PlatformIO terminaline yaz (ya da kopyala):

pio run -t program\_fpga

Adım 4. RISC-V çevirici programını derle, indir, çalıştır

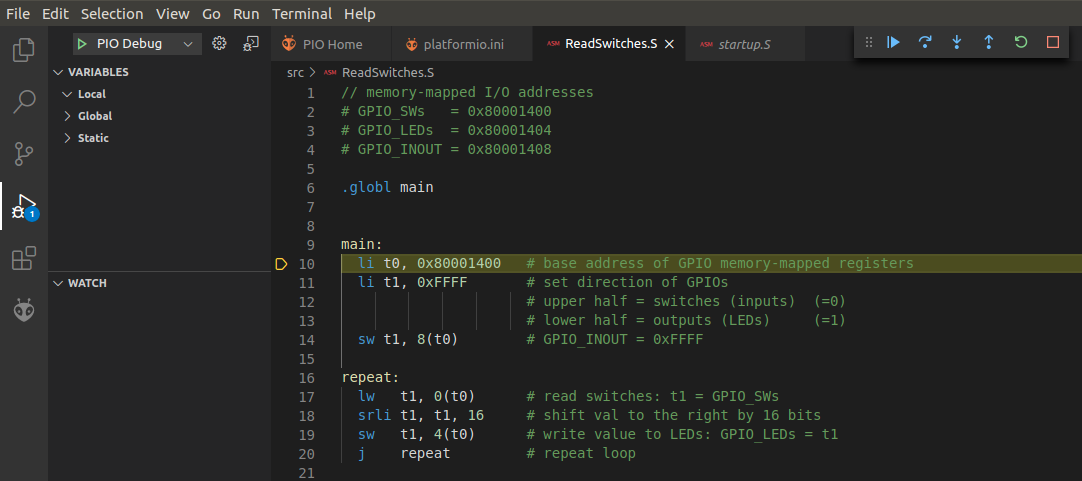
RVfpga kartta çalıştığına göre programını derleyip, RVfpga’e indirip, çalıştırıp/ayıklayacksın. VSCode açıkta değilse aç. Son projen, Project1, kendiliğinden açılmalı. Açılmazsa PlatformIO eklentisinin açık olduğunun sağlamasını yapıp File (Dosya) → Open Folder (Klasörü Aç) üzerine tıklayıp bu deneyde oluşturduğun Project1’i seç (ancak açma).

Sol menü şeridindeki Run (Çalıştır) butonuna tıklayıp ardından Start Debugging (Ayıklamayı Başlat) butonuna tıkla (Figür 10’a göz at).



**Figür 10. Programı RVfpga’de çalıştır**

Program Nexys A7 kartındaki FPGA’de çalışan RVfpga’e inecektir. Şimdi programı çalıştırıp ayıklamaya başlayabilirsin (Figür 11’e göz at).



**Figür 11. RVfpga’de program çalışıyor**

RVfpga İlk Kullanım Kılavuzu ile Deney 2’de tanımlandığı gibi ayıklama araç çubuğu ile Ayıklayıcı seçeneklerini programı çalıştırıp yönetmek için kullan. Örneğin, satır 17’ye bir kesme noktası ayarlayıp (satır numarasının soluna tıklayarak) ardından yazmaç t1’a anahtarların değerleri ona yüklendiği için bak. Stop (Durdur) butonuna  (ya da Shift - F5) basarak ayıklama oturumunu durdurduğunda ayıklama oturumu sona ermesine karşın program RVfpga’de çalışmayı sürdürür.

# Alıştırmalar

Şimdi Deney 2’deki alıştırmalarda C yerine RISC-V çeviricisi kullanarak kendi RISC-V çevirici programlarını oluştur. Kolaylık olmaslı için alıştırma tanımları aşağıda tekrarlanmıştır.

Unutma, RVfpga’i programlar arası karta yeniden yüklemeye gerek duymamak için Nexys A7 kartını bilgisayara bağlı bırakabilirsin. Ancak Nexys A7 kartını kapatırsan PlatformIO kullanrak RVfpga’i karta yeniden yüklemen gerekir.

Alıştırma 1. LEDlerin değerini anahtarlara yakıp söndüren bir RISC-V çeviricisi programı yaz. Darbe bir kişinin yanıp sönmeyi anlayabileceği yavaşlıkta olmalı. Programı FlashSwitchesToLEDs.S olarak adlandır.

Alıştırma 2. Anahtarların değerinin tersini LEDlerde gösteren bir RISC-V çeviricisi programı yaz. Örneğin anahtarlar şu ise (ikili sistemde): 0101010101010101, LEDler şunu göstermeli: 1010101010101010; anahtarlar şu ise: 1111000011110000, LEDler şunu göstermeli: 0000111100001111; gibi gibi. Programı DisplayInverse.S olarak adlandır.

Alıştırma 3. Bütün LEDler yakılana değin artan, kayan sayılar gösteren bir RISC-V çeviricisi programı yaz. Ardından örüntü kendini tekrarlamalı. Programı ScrollLEDs.S olarak adlandır.

Program şunları yapmalı:

1. İlk olarak bir yakılmış LED sağdan sola kayıp ardından soldan sağa kaymalı.
2. Ardından iki yakılmış LED sağdan sola kayıp ardından soldan sağa doğru kaymalı.
3. Ardından üç yakılmış LED sağdan sola kayıp ardından soldan sağa doğru kaymalı.
4. Gibi gibi, bütün LEDler yakılmış olana değin.
5. Ardından bu örüntü kendini tekrarlamalı.

Alıştırma 4. Anahtarların 4 düşük önemli bitiyle 4 yüksek önemli bitini pozitif 4-bit eklemeyle gösteren bir RISC-V çeviricisi programı yaz. Sonucu LEDlerin 4 düşük önemli (en sağ) bitinde göster. Programı 4bitAdd.S olarak adlandır. LEDlerin beşinci biti pozitif taşması (unsigned overflow) olunca yanmalı (yani taşınan (carry) 1 olunca).

**Alıştırma 5**. Öklit algoritmasıyla iki sayının, a ile b, en büyük ortal bölenini bulan birRISC-V çeviricisi programı yaz. a ile b’nin değerleri programda statik tanımlı değişken olmalıdır. Programı **GCD.S** olarak adlandır. Şurada Öklit algoritmasıyla ilgili ek bilgi vardır: <https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm>. Google’da şunu da aratabilirsin: “Euclidean algorithm”.

**Alıştırma 6**. Fibonacci dizisinin ilk 12 sayısını hesaplayıp, ardından bunu sonlu, 12 ögeli bir vektörde (bir diğer deyişle) depolayan bir RISC-V çeviricisi programı yaz. Fibonacci sayılarının sonsuz sekansı şöyle tanımlanır:

V(0)=0, V(1)=1, V(i)=V(i-1)+V(i-2) (i=0,1,2...)

Bir diğer deyişle i ögesinde denk gelen Fibonacci sayısı, dizinin önceki iki Fibonacci sayısının toplamına eşittir. Tablo 1 i = 0’dan 8’e Fibonacci sayılarını gösterir.

Tablo 1. Fibonacci dizisi

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***i*** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| ***V*** | 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 |

Vektör boyutu N programda bir sabit olarak tanımlanmalıdır. Programı **Fibonacci.S** olarak adlandır.

**Alıştırma 7**. N ögeli bir A vektöründen (dizisinden) yalnızca çift, sıfırdan büyük sayıları içeren bir başka B vektörü oluştur. Örneğin: *N*=12, *A* = [0,1,2,7,-8,4,5,12,11,-2,6,3], ise: *B* = [2,4,12,6]. Programı **EvenPositiveNumbers.S** olarak adlandır.

**Alıştırma 8**. İki N ögeli vektör (dizi) A ile B verilince şöyle bir C vektörü oluştur:

C(i) = |A[i] + B[N-i-1]|, i = 0,..,N-1.

Yeni vektörü hesaplayan programı RISC-V çeviricisinde yaz. Programında 12 ögeli diziler kullan. Programı **AddVectors.S** olarak adlandır.

**Alıştırma 9**. RISC-V çeviricisinde bubble sort algoritmasını gerçekleştir. Bu algoritma şu prosedürle bir vektörün bileşenlerini yükselen biçimde sıralar:

1. Vektörü bitene değin tekrar tekrar dolaş.
2. V(i) > V(i+1) ise komşu bileşenlere yer değiştirt.
3. Ardışık bileşenlerin hepsi sıralı olunca algoritma sonlanır.

Programını denemek için 12 ögeli diziler kullan. Programı **BubbleSort.S** olarak adlandır.

**Alıştırma 10**. Negatif olmayan bir sayı n’in faktöriyelini yinelemeli çarpımlarla hesaplayan bir RISC-V çeviricisi programı yaz. Programı değişik n değerleriyle dene ancak son değerin n= 7 olmalıdır. n program içerisinde statik tanımlanmış bir değişken olmalı. Programı **Factorial.S** olarak adlandır.