# RISC-V Mimarisi Yazılım Ekosisteminin Değerlendirilmesi

*Serdar Oğuz ATA*
*Aselsan*
*oguzata@aselsan.com.tr*

*Yusuf İbrahim ÖZKÖK*
*Aselsan*
*yozkok@aselsan.com.tr*

***Özetçe*— Günümüzde açık kaynak felsefesiyle geliştirilen projelerin sayısı sürekli artmaktadır. RISC-V de bu şekilde geliştirilen bir işlemci komut seti mimarisidir. Bu proje bize açık kaynaklı bir işlemci kullanma özgürlüğünün kapılarını aralamaktadır. Fakat RISC-V mimarisinin herkes tarafından yaygın olarak kullanılabilmesi için gelişmiş bir yazılım ekosistemi desteğine sahip olması gerekmektedir. Bu makalede, RISC-V mimarili işlemcileri destekleyen yazılım ekosistemi hem literatür olarak hem de fiili olarak değerlendirilecektir.**

***Anahtar Kelimeler — RISC-V, açık kaynak işlemci mimarisi, yazılım ekosistemi, benchmark, BooM, derleyiciler, yazılım geliştirme ortamı***

## I. GİRİŞ

Açık kaynaklı projeler, bilgi paylaştıkça büyür felsefesiyle yürütülen, herkesin özgürce erişebildiği, gelişimine katkı verebildiği ve kullanabildiği, geniş ve aktif topluluk desteğinin gücüyle kısa zamanda nitelikli ve büyük işler çıkarılabilen ve artık her alana yayılmış gittikçe de yaygınlaşan bir yaklaşımın eseridir. Genellikle yazılım dünyasında yaygın olan açık kaynaklı projeler, artık donanım dünyasında da yaygınlaşmaya başlamıştır.

Bunun bir örneği olarak RISC-V [1][2] komut seti mimarisi, Kaliforniya Berkeley Üniversitesi tarafından 2010 yılında başlatılan açık kaynaklı bir işlemci mimarisi geliştirme projesidir. Proje üniversite bünyesini aşarak 2015 yılında kurulan RISC-V vakfı ile dünyaya açılmış ve Google, NVIDIA, Qualcomm, Western Digital, Samsung, Alibaba, gibi büyük şirketlerin de katılımıyla oldukça geniş bir kabul görmüştür.

RISC-V mimarisi halen gelişmekte olan olgun bir mimari haline gelmiştir. Ancak bu mimarinin bir işlemci olarak herkes tarafından kullanılabilirliği gelişmiş bir yazılım ekosistemine sahip olmasına bağlıdır. Bu makalede RISC-V mimarisi üzerinde yazılım koşturulabilmesi ve geliştirme yapılabilmesi için gerekli yazılım ekosisteminin yeterliliği ve sorunsuz kullanılabilirliği incelenmiştir.

## II. YAZILIM EKOSİSTEMİ

Bir işlemci üzerinde herhangi bir uygulama geliştirebilmek ve bunu çalıştırabilmek için olgun bir yazılım ekosistemi elzemdir. Bunun en düşük kümesi; aşağıdaki başlıklar altında ele alınan derleyici/bağlayıcılar, ön yükleyiciler, temel kütüphaneler, işletim sistemi desteği, hata ayıklama araçları gibi elemanlardan oluşmaktadır.

### *A. Derleyici/Bağlayıcılar*

İşlemciler üzerinde çalıştırılmak üzere günümüz programlama dilleri ile geliştirilen kodların, işlemci komut seti mimarisi ile uyumlu makine diline dönüştürülmesi gerekir. Ayrıca günümüz karmaşık programları için birden fazla ve çok farklı yazılım kütüphaneleri ve işletim sistemi ile tüm kodun bellekte birbiri ile uyumlu çalışabilmesi için bağlayıcılar, gerektiğinde makine kodunu okunabilir yazılıma geri çevirecek çözücüler, programdaki hataları ayıklamaya yardımcı olacak ayıklayıcılar gibi araçlar da yine bu grupta ekosistemin olmazsa olmaz parçaları olarak düşünülmelidir. Bu temel araçlar olmadan yazılım geliştirilip bir donanım üzerinde çalıştırılması mümkün değildir. Bu araçlar derleyici ailesi olarak birlikte sağlanmaktadır. Günümüzde her işlemci mimarisi için birçok farklı derleyici seçeneği bulunmaktadır.

### *B. İşletim Sistemi Desteği*

Bir mimarinin yaygın olarak kullanılabilmesi için işletim sistemleri tarafından desteklenmesi gereklidir. İşletim sistemi olmadan da doğrudan donanım (ing; "bare metal") üzerinde yazılım çalıştırmak mümkün olsa da bu yazılımlar genelde daha basit ve sadece belirli bir amaca yönelik yazılımlardır. Daha büyük ölçekli, çok görevli, kapsamlı, karmaşık, farklı yazılımlarla birlikte çalışabilecek projelerin kullanılabilmesi için işletim sistemi desteği şarttır.

### *C. Önyükleme Yazılımı*

Önyükleme yazılımı, işlemciye güç verildikten sonra ilk olarak çalışan, donanımın özelliklerine bağlı olarak işlemci ve kart ilklendirmelerini yaparak, sonraki yazılımların veya işletim sisteminin yüklenmesini ve doğru şekilde çalışmasını sağlayan yazılımdır. Bu yazılım, donanıma ve kullanılan işlemcinin mimarisine uygun olarak özelleşmiş biçimde geliştirilen, tamamen işlemciye ve donanıma bağımlı bir yazılımdır. Bu şekilde donanımsal ilklendirmeler yapılmadan işletim sisteminin veya herhangi bir programın çalıştırılması mümkün değildir.

### *D. Hata Ayıklama*

Yazılım geliştirirken oluşan hataları, donanım üzerinde çalışırken ayıklayabilmek için hata ayıklama yazılımlarına gereksinim duyulur. Bu yazılımlar sayesinde, işlemci üzerinde

çalışan bir kod, istenilen komutta durdurulabilir, işlemci ve sistem üzerindeki yazmaçlara/belleklere erişilebilir ve bu sayede doğru çalışmayan noktalar tespit edilebilir. Bu şekilde bir yazılım aracı olmadan sistemde meydana gelen hataları noktasal olarak tespit etmek oldukça zor olacaktır.

### *E. Simülatör/Emülatör*

Yazılım geliştirme aşamalarında, yazılımın geliştirildiği donanıma erişim her zaman mümkün olmayabilir. Bu gibi durumlarda, işlemci ve donanım altyapısını taklit eden simülatör ve/veya emülatörler üzerinde geliştirme ve test yapılabilir. Simülatör ve emülatörler yazılım geliştirmenin donanıma ihtiyaç duymadan ve daha erken zamanlarda geliştirilmesine imkan veren gerekli ve faydalı araçlardır.

## III. RISC-V YAZILIM EKOSİSTEMİ

Önceki bölümde, bir işlemci mimarisinin üzerinde yazılım geliştirilip kullanılabilmesi için gerekli en küçük küme yazılım ekosistemi tanımlanmıştır. Bu bölümde RISC-V yazılım ekosisteminin bu gereksinimleri karşılayabilirliği değerlendirilmiştir.

### *A. Derleyiciler*

Günümüzde çoğu mimari için birçok farklı derleyici desteği bulunmaktadır. Benzer şekilde RISC-V mimarisi için de LLVM projesi ile geliştirilen Clang [3] ve GNU projesi ile geliştirilen GCC [4] derleyicileri mevcuttur.

#### *1) Clang*

Clang, LLVM projesinin bir parçası olarak geliştirilen çeşitli mimarileri destekleyen bir C/C++ derleyicisidir. Bu açık kaynak derleyici projesinde aktif olarak RISC-V mimari desteği de bulunmaktadır. Ancak bu derleyici sadece assembly kodunu oluşturmaktadır. Bu oluşan kodun tekrar farklı bir derleyici ile makine diline çevrilmesi gerekmektedir.

#### *2) GCC*

GCC (GNU Compiler Collection), en yaygın kullanılan ve çok geniş mimari desteği olan bir derleyicidir. GCC, RISC-V mimarisini de aktif olarak desteklemektedir. Clang'den farklı olarak sadece C tabanlı dilleri değil birçok farklı yazılım dilini de desteklemektedir.

GCC, kaynak kodunun yapılandırılması ile RISC-V mimarisinin seçilen özelliklerine (32-64 bit, kayar nokta desteği vb.) göre oluşturulabilir. Ayrıca "riscv-unknown-elf-" ve "riscv-unknown-linux-gnu-" şeklinde iki derleyici ailesi oluşturulabilir. "riscv-unknown-elf-" ailesi ile doğrudan donanım üzerinde işletim sistemsiz olarak ve "riscv-unknown-linux-gnu-" ile linux işletim sistemi üzerinde çalışacak şekilde yazılan kodlar derlenebilmektedir.

### *B. İşletim Sistemi Desteği*

RISC-V mimarisinde önceki bölümde anlatılan derleyiciler kullanılarak işletim sistemi olmadan doğrudan donanım üzerinde kod çalıştırmak mümkündür. Ancak daha önce bahsedildiği gibi, çok görevli, karmaşık ve birden fazla ve çok farklı yazılımları çalıştırabilmek için işletim sistemi desteği önemlidir.

RISC-V mimarisi, açık kaynaklı işletim sistemi olarak Linux işletim sistemi tarafından doğrudan desteklenmektedir. Bu mimariye özel tüm yapılandırma seçenekleri, RISC-V ile tasarlanmış örnek kartların sürücüleri, linux kaynak kodunda ana dalda mevcuttur. Bu sayede, kernel.org adresinden indirilen linux kernelinin kaynak kodu derlenerek RISC-V mimarili işlemciye sahip bir sistemde kendi yapılandırdığımız bir Linux işletim sistemini çalıştırmak mümkündür. Bu şekilde linux işletim sistemi üzerinde çalışmak üzere tasarlanmış programlar da RISC-V mimarili sistemlerde kullanılabilecektir.

### *C. Önyükleme Yazılımı*

RISC-V mimarisi açık kaynaklı bir proje olduğu için, işlemcilerin ve çevresindeki donanımların özellikleri erişime açıktır. Böylelikle mimariye özel donanım ilklendirmelerini de içeren önyükleme yazılımı herkes tarafından geliştirilebilir. Bununla birlikte mimarinin yaratıcıları tarafından BBL (Berkeley Boot Loader)[5] isimli açık kaynaklı bir önyükleme yazılımı da sunulmuştur. Bu yazılım Berkeley Üniversitesi tarafından tasarlanan BooM (Berkeley out-of-order Machine)[6] isimli RISC-V mimarili işlemciyi desteklemekle birlikte RISC-V türevi diğer işlemcilere de uyarlanabilir.

### *D. Hata Ayıklama*

GNU projesinin bir parçası olan GDB (GNU Debugger) [7] aracı RISC-V mimarisini desteklemektedir. Bu araç ile mimarinin sağladığı "Debugger Module" üzerinden işlemcinin durdurulup program sayacının adım adım ilerletilmesi, işlemci yazmaçlarının ve sistem belleğinin okunması mümkündür. GDB'nin sağladığı bütün özellikler RISC-V mimarisi için de mevcuttur.

### *E. Simülatör/Emülatörler*

RISC-V mimarili işlemciler ve donanımlar günümüzde henüz yaygın olmadığı için, donanımı taklit eden emülatörlere erişim önemli bir imkandır. Böylelikle yeni donanım tasarımlarında yazılım geliştirme çalışmalarına da erken safhalarda başlanabilir. Komut seti simülatörü olarak "Spike" [8] ve işlemci emülatörü olarak "QEMU" [9] açık kaynak projeleri tarafından RISC-V mimarisi desteklenmektedir.

#### *1) Spike*

Spike, RISC-V derleyicileri oluşturulurken ortaya çıkan araçlardan biridir. Derleyiciler oluşturulurken seçilen mimari özelliklerini destekleyen bir RISC-V komut seti simülatörüdür. Spike, RISC-V komut seti mimarisi için "altın referans" olarak görülmektedir. Spike'ın davranışı donanım ve yazılım için referans olarak belirlenmiştir. Bu simülatör, işlemcinin davranışını komut komut taklit ederek gerçeğe uygun bir şekilde işlemci komut işleme davranışının analiz edilebilmesini sağlar.

#### *2) QEMU*

QEMU, donanım davranışını sanal olarak taklit eden açık kaynaklı bir işlemci emülatördür. QEMU birçok mimari ile birlikte RISC-V mimarisini de desteklemektedir.

QEMU, taklit ettiği mimarinin komutlarını, üzerinde çalıştığı masaüstü bilgisayarın işlemcisinin komutlarına dönüştürerek çalışır. Böylece daha fazla komut daha kısa süre içinde çalıştırılabilir. Ancak QEMU, işlemcinin fonksiyonel davranışını taklit eder, spike gibi komut komut akışı taklit etmez.

Bu yüzden spike'tan daha hızlı çalışır ancak komut seviyesinde analiz imkanı sağlamaz.

## IV. DEĞERLENDİRME ORTAMI

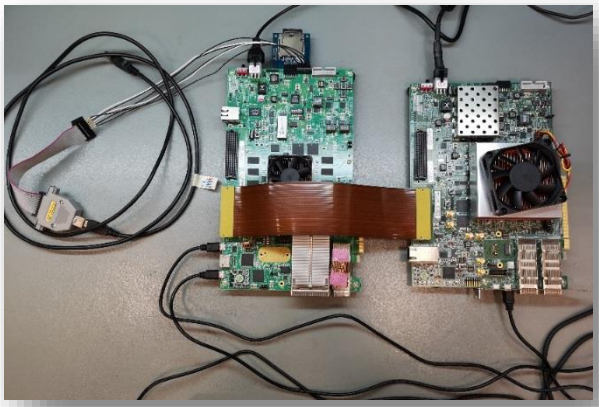
Bu bölümde buraya kadar bahsedilen ekosistem parçalarının gerçek bir ortamda denenebilmesi için kurulan geliştirme ortamından bahsedilecektir.

### A. *Test Donanımı*

RISC-V yazılım ekosistemini kullanmak, geliştirmek ve test etmek için bu mimari ile çalışan bir donanım veya donanımı taklit edecek bir emülatör gerekmektedir.

Maalesef elimizde gerçek bir RISC-V işlemcisi olmadığı için bu işlemciyi gerçeğe en yakın şekilde taklit edebilmek adına FPGA üzerinde RISC-V işlemcisi sentezlenerek çalışmalar yapılmıştır. FPGA üzerinde sentezlenen işlemci BooM (Berkeley out-of-order Machine) temel alınarak tasarlanmıştır. Bu şekilde FPGA bir RISC-V işlemci gibi davranarak, bu mimari için tasarlanan yazılımları çalıştırabilmektedir. Ayrıca FPGA üzerindeki çevresel birimleri de kullanarak gerçek bir işlemci kartı davranışı da test edilebilmektedir. Sentezlenen işlemci RISC-V mimarisinde, 64 bitlik komutları destekleyen, sırasız işlem yapan, tamsayı ve kayan nokta işlemlerini yapabilme özelliklere sahiptir. Bu mimari özellikler RISC-V IMAFD (Integer - Multiplication - Atomic - Floating - Double) olarak gösterilmektedir.

Bu makalede anlatılan RISC-V mimarisiyle yapılan tüm yazılım çalışmaları ve testleri FPGA üzerinde sentezlenen bu RISC-V işlemci üzerinde Şekil 1'de görülen donanım ortamında gerçekleştirilmiştir.



*Şekil 1-RISC-V işlemci IP çekirdeği içeren FPGA donanımı*

### B. *Yazılım Geliştirme Ortamı*

Açık kaynaklı RISC-V yazılım ekosisteminin kullanılabilmesi, derleyicilerin, emülatörlerin ve geliştirme ortamlarının kurulabilmesi için, tüm bu araçları yaygın olarak destekleyen linux tabanlı bir işletim sistemi gerekmektedir. Bu makalede anlatılan tüm yazılım geliştirme çalışmaları (yazılım araçlarının derlenmesi, RISC-V üzerinde koşacak kodun çapraz derlenmesi v.b.) Ubuntu 16.04 LTS işletim sistemine sahip bir bilgisayarda yapılmıştır.

## V. RISC-V YAZILIM EKOSİSTEMİNİN OLUŞTURULMASI

Geliştirilen bir yazılımı, donanım üzerinde çalıştırabilmek için yazılan kodunu derleyecek derleyiciler, kodun kullanabileceği fonksiyonları içeren kütüphaneler gibi çeşitli bileşenler gerekmektedir. Bu bileşenlerin kaynak kodlarına github üzerindeki "riscv-tools" [10] deposundan erişilebilir. Bu depoda ayrıca işletim sisteminin ihtiyaç duyduğu önyükleme yazılımı, işlemcinin doğruluğunu ve performansını ölçen test yazılımları gibi yazılımların da kaynak kodları bulunmaktadır. Bu makalede kullanılan hiçbir araç hazır olarak kullanılmamış tamamı kaynak kodlarından yeniden derlenmiştir. Bu bölümde, ihtiyaç duyulan bileşenlerin kaynak kodlarından oluşturulmasından bahsedilmektedir.

### A. *Derleyiciler*

Bahsedilen "riscv-tools" deposundaki kaynak kodlar indirildikten sonra, işlemcinin donanım mimarisine uygun şekilde ayarlamalar yapılmalı ve bu kaynak kodların içinde bulunan komut dosyaları çalıştırılarak derleyiciler oluşturulmalıdır.

İşlemci üzerinde kodu işletim sistemsiz olarak çalışabilecek şekilde derleyebilmek için "riscv64-unknown-elf-" derleyici ailesi, linux işletim sistemini ve bu işletim sisteminde çalışacak kodları derleyebilmek için "riscv64-unknown-linux-gnu-" derleyici ailesi oluşturulmalıdır. Bu derleyiciler 64 bitlik mimari için oluşturulmuştur. Benzer şekilde 32 bitlik mimari için "riscv32-unknown-elf-" ve "riscv32-unknown-linux-gnu-" derleyici aileleri de oluşturulabilir. Her bir derleyici ailesi kod derleme, bağlama, hata ayıklama gibi işlere yarayan uygulamaları içermektedir.

Bu makale kapsamındaki ihtiyaçlar için 64 bitlik RISC-V IMAFD mimarisini destekleyecek şekilde "riscv64-unknown-elf-" ve "riscv64-unknown-linux-gnu-" derleyici aileleri kaynak kodları kullanılarak oluşturulmuştur.

### B. *Kütüphaneler*

Programların çalışması ve verilen görevleri yapabilmesi için genellikle hazır fonksiyonlara ihtiyaç duyulur. Bu fonksiyonlar işlevlerine göre çeşitli kütüphaneler içinde bulunurlar.

Önceki bölümde anlatılan RISC-V derleyicilerinin kullandıkları kütüphaneler de farklıdır. "riscv64-unknown-elf-" derleyicileri "Newlib" [11] kütüphanesini, "riscv64-unknwon-linux-gnu-" derleyicileri ise "glibc" [12] kütüphanesini kullanır. Bu kütüphaneler de derleyiciler gibi kodun çalıştırılacağı ortama uygun olarak özelleşmişlerdir. "Newlib" kütüphanesi gömülü sistemler için geliştirilmiş bir C kütüphanesidir ve işletim sistemsiz şekilde çalışacak şekilde derlenir. "NewLib", "glibc"den daha sade ve küçüktür, daha az yer kaplar ancak "glibc"nin desteklediği bütün fonksiyonları da desteklemez. "glibc" ise linux işletim sisteminin ihtiyaç duyduğu bütün fonksiyonları içeren tam bir C kütüphanesidir.

RISC-V işlemci için kod derleyecek derleyiciler oluşturulurken, bu kütüphaneler de aynı mimarileri destekleyecek şekilde yeniden oluşturulup derlenirler.

### C. *Emülatörler*

RISC-V IMAFD mimarisini destekleyen derleyiciler oluşturulurken, spike isimli emülatör de oluşturulmuştur. Ancak bu makalede anlatılan çalışmalar FPGA üzerindeki donanımda çalıştırıldığı için spike veya QEMU kullanılmamıştır. Ancak donanımın bulunmadığı durumlarda bu araçlar da kullanılabilir.

## VI. RISC-V YAZILIM ELEMANLARI

Bu bölümde ise doğrudan RISC-V işlemcimiz üzerinde koşturacağımız ve yukarıda oluşturulan yazılım geliştirme araçları ile derlediğimiz uygulamalar anlatılmaktadır.

Her işlemcide olduğu gibi RISC-V işlemci üzerinde de, yazılan programlar işletim sistemsiz veya bir işletim sistemi kullanılarak çalıştırılabilmektedir. Bu yöntemlerden hangisinin kullanılacağına bağlı olarak kullanılacak derleyici, derleme yöntemi, yardımcı programlar, kütüphaneler gibi elemanlar farklılık göstermektedir

### A. *İşletim Sistemsiz Program Çalıştırma*

Bir programı işletim sistemsiz olarak çalıştırabilmek için "riscv64-unknown-elf-" derleyici ailesi ile derlemek gerekmektedir. Program derlenirken gerekli olan kütüphaneler de eklenerek bir imaj dosyası elde edilir. Bu dosya kendi başına çalışacak şekilde gerekli bütün kod parçalarını içermelidir. Bu dosya işlemciye bağlı bir RAM'e kopyalanarak çalıştırılabilir.

Bu şekilde genellikle fonksiyonel test ya da kıyaslama (benchmark) yapan kodlar çalıştırılmaktadır. Böylece işletim sisteminin getirdiği ekstra yük ve işlemler olmadan sistem test edilebilmekte veya performansı gerçeğe yakın olarak ölçülebilmektedir.

Bu çalışma kapsamında FPGA üzerinde kurulan ortamda, öncelikle donanımın doğrulanması için işletim sistemi olmadan fonksiyonel testler çalıştırılmıştır. Sonrasına FPGA üzerinde sentezlenen işlemcinin gerçek performansı, işletim sisteminin getirdiği ek yükler olmadan CoreMark [13], Dhrystone gibi kıyaslama programları ile gözlenmiştir.

### B. *Linux İşletim Sisteminin Çalıştırılması*

FPGA üzerinde sentezlenen RISC-V işlemci üzerinde tamamen açık kaynak araçlar kullanılarak kaynak kod'dan oluşturulan linux işletim sistemi imajı çalıştırılmıştır. Oluşturulan Linux işletim sistemi; Linux kerneli ve dosya sistemi, komut satırı gibi bu bölümde açıklanan gerekli en küçük yazılım kümesini içermektedir.

#### 1) *Araç Seti*

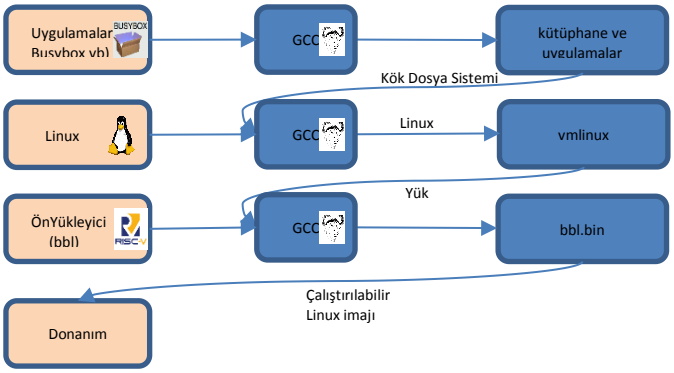
Gömülü sistemlerde linux kerneli üzerinde, sistemle etkileşim sağlayabilmek amacıyla, komut satırı, izleme araçları, dosya işlemleri gibi işlevler için bir araçlar topluluğuna ihtiyaç duyulmaktadır. Busybox[15] bu amaçla oluşturulmuş, UNIX komutlarının bir alt kümesini içeren ve linux işletim sisteminde bir komut satırı oluşturulmasına yarayan küçük bir araç setidir. Bu araç setinde, ihtiyaç duyulan UNIX komutları ihtiyaca göre seçilip yapılandırılabilir. Eklenen komutlar ve linux imajına eklenen diğer programlar, Busybox'ın sağladığı komut satırı arayüzü vasıtasıyla çağırılabilir.

#### 2) *Linux Kernelinin Derlenmesi*

Linux işletim sistemi imajını oluşturmak için öncelikle linux kernel kaynak kodu temin edilmiştir. Bu kodun güncel sürümüne kernel.org adresinden erişilebilir. Bu makale kapsamında yapılan çalışmalarda kernel'in 4.15 sürümü kullanılmıştır.

Linux işletim sisteminin çalışabilmesi için işletim sisteminin omurgasını oluşturan bir kök dosya sistemi eklenmesi gerekmektedir. Bu kök dosya sistemi[16] linux'un gerektirdiği standart dizin yapısına göre oluşturulmalı ve çalıştırılacak programların dosyaları, programlar ve işletim sistemi için gerekli olan kütüphaneler ile önceki bölümde anlatılan busybox araç seti, bu dosya sisteminin içindeki ilgili dizinlere kopyalanmalıdır.

Dosya sistemi oluşturulduktan sonra işletim sisteminin desteklediği formata sahip bir arşiv dosyası haline dönüştürülür. Linux kernel kaynak kodu "riscv64-unknown-linux-gnu-" derleyicileri ile derlenir ve arşiv dosyasıyla bağlanarak linux imajı oluşturulur. Linux İşletim Sistemi imajı oluşturma aşamaları Şekil 2'de resmedilmiştir.




*Şekil 2 Linux İşletim Sistemi imajı oluşturma akışı*

Oluşturulan linux imajı son adımda bir önyükleme yazılımı tarafından gerekli mimari ve donanımsal ilklemeler yapıldıktan sonra ilgili bellek alanına yüklenerek işlemci üzerinde çalıştırılır.

#### 3) *Önyükleme Yazılımı*

Önceki bölümlerde işlevine değinilen önyükleme yazılımı olarak, kullandığımız işlemci çekirdeği BooM için tasarlanmış olan BBL[5] kullanılmıştır. BBL, işlemcinin ilgili yazmaçlarına yazılması gereken değerleri yazarak işlemcinin ilklemelerini ve çevresel birimlerin ayarlamalarını yapar. Buna ek olarak ekran çıktıları aracılığıyla açılış işleminin ve linux'un açılış kayıtlarının izlenebilmesi için temel seviyede bir seri kanal sürücüsü içermektedir. Güç verilme anından itibaren devreye giren ve işlemciyi ilkendiren ön yükleyici Linux imajını da başlattıktan sonra görevini tamamlayarak işlemciyi Linux kerneline bırakır.

#### 4) *Uygulama Yazılımları*

İşletim sisteminin boot edilmesi aşamasından sonra artık herhangi bir Linux işletim sistemi üzerinde çalıştırılabilecek uygulama yazılımlarının denenmesi aşamasına geçilmiştir. Uygulama yazılımı olarak aynı zamanda sentezlediğimiz

işlemcinin performansına yönelik de bazı ölçütler alabilmek adına Coremark ve 7zip yazılımları seçilmiştir.

Coremark [13], gömülü işlemcilerin karşılaştırılmasında kullanılan bir kıyaslama programıdır. Değerlendirme yaptığımız donanımda, Linux işletim sisteminde Coremark kıyaslama programı çalıştırılmış ve Şekil-3'te gösterilen sonuçlar elde edilmiştir. Bu sonuçlar RISC-V mimarili BooM işlemcinin başarımı hakkında da bize kabaca bir bilgi vermektedir.

```
~ # ./bin/benchmarks/coremark.riscv
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 2978142356
Total time (secs): 59.562997
Iterations/Sec   : 167.889471
Iterations       : 10000
Compiler version : GCC7.2.0
Compiler flags   : No FLAG
Memory location  : STACK
seedcrc          : 0xe9f5
[0x0]crclist     : 0xe714
[0x0]crcmatrix   : 0x1fd7
[0x0]crcstate    : 0x8e3a
[0x0]crcfinal    : 0x988c
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 167.889471 / GCC7.2.0 No FLAG / STACK
CoreMark/MHz 3.357789
```

*Şekil 3-Coremark Kıyaslama Yazılımı Sonuçları*

Gömülü işlemciler için yazılan kıyaslama programlarının yanında, 7zip [18] sıkıştırma programı da test donanımı üzerinde çalıştırılmıştır. 7zip yazılımının kaynak kodu indirilip oluşturulan "riscv64-unknown-linux-gnu-" derleyicileriyle derlenmiştir. Gerekli yazılım kütüphaneleri ve derlenen program çıktısı, linux kök dosya sistemine eklenmiştir.

7zip programı masaüstü bilgisayarlarda da kullanılan yaygın ve birçok farklı kütüphane kullanan algoritmalar içeren karmaşık bir uygulama yazılımıdır. Bu yazılımın derlenerek RISC-V mimarisinde çalıştırılabilmesi, bu mimarinin yazılım ekosistemi ile standart bir uygulamanın da başarıyla derlenip çalıştırılabilmesine güzel bir örnek teşkil etmektedir.

7zip sıkıştırma programı, kullandığı algoritmalar sayesinde aynı zamanda bir performans kıyaslaması yan işlevine de sahiptir. Şekil 4'te 7zip'in kıyaslama özelliği kullanılarak elde edilen sonuçlar gösterilmektedir. Bu programa girdi olarak "21" sayısı verilerek $2^{21}$ byte = 2 MB'a kadar dosyalar oluşturup sıkıştırma ve açma işlemlerinin başarım sonuçları elde edilmiştir. Bu elde edilen sonuçlar da yine kullandığımız RISC-V Boom'un ve FPGA üzerinde oluşturduğumuz yapının bellek erişimi ve kullanımı konularında bize bir ölçüt sunmakta, masa üstünde farklı işlemciler üzerinde aynı işlemlerin sonuçlarını karşılaştırma imkanı vermektedir.

```
~ # 7za b -md21

7-Zip (a) 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=C,Utf16=off,HugeFiles=on,64 bits,1 CPU LE)
LE
CPU Freq:  48  48  48  48

RAM size:    935 MB,  # CPU hardware threads:  1
RAM usage:    30 MB,  # Benchmark threads:     1

                    Compressing |                  Decompressing
Dict     Speed Usage    R/U Rating |      Speed Usage    R/U Rating
         KiB/s     %   MIPS   MIPS |      KiB/s     %   MIPS   MIPS

18:         34   100     31     31 |        533   100     43     43
19:         33   100     30     30 |        533   100     44     44
20:         32   100     30     30 |        529   100     44     44
21:         32   100     30     30 |        523   100     44     44
---------------------------------- | ------------------------------
Avr:            100     30     30 |            100     44     44
Tot:            100     37     37
```

*Şekil 4-7zip Programı Başarım Sonuçları*

## VII. SONUÇ

Bu makalede RISC-V işlemci çekirdeği yazılım eko sistemi hem literatür olarak incelenmiş hem de deneysel olarak tecrübe edilmiştir. FPGA üzerinde sentezlenmiş bir RISC-V çekirdeği üzerinde sadece açık kaynak olarak erişilebilir araç ve yazılımlardan oluşan bileşenler ile oluşturulmuş uygulama ve işletim sistemi imajlarının bilfiil çalıştırılması aşamalarındaki gözlemler paylaşılmıştır. Bu çalışmada sentezlenen işlemcinin kendisinden, kullanılan tüm geliştirme araçlarına ve çalıştırdığımız işletim sistemi ve uygulama yazılımlarına kadar hiçbir aşamada hiçbir hazır ürün veya yazılım kullanılmamış, tüm araçlar ve yazılım bileşenleri kaynak kod olarak temin edilmiş ve baştan derlenerek kullanılmıştır. Kaynak koddan derlenen ön yükleyici, Linux çekirdeği ve uygulama yazılımlarının birbiri ile uyumlu bir şekilde yine kendi sentezlediğimiz RISC-V işlemci çekirdeği üzerinde sorunsuzca koşturulmuştur.

Sonuç olarak makalemizde çerçevesini çizdiğimiz en temel yazılım ekosistemi bileşenlerinin RISC-V için açık kaynak olarak mevcut olduğu ve herhangi bir ticari ürüne ihtiyaç duyulmadan bu mimarinin sorunsuz bir şekilde kullanılabilmesini sağlayacak olgunlukta olduğu ortaya konulmuştur. Böylelikle RISC-V açık kaynak işlemci mimarisinin isteyen herkes tarafından rahatlıkla kullanılabileceği herhangi bir ticari ürüne bağlı kalınmadan bir

işlemci ekosistemine sahip olunabileceği gösterilmiştir. Aynı zamanda yapılacak katkılar ile de bu ekosistemin daha da gelişmesine destek olunabileceği değerlendirilmektedir.

## KAYNAKLAR


*Not: Tüm internet sitesi erişimleri 10.08.2019'da kontrol edilmiştir.*

[1] *The RISC-V Instruction Set Manual Volume I: User-Level ISA*

[2] *The RISC-V Instruction Set Manual Volume II: Privileged Architecture*

[3] *https://llvm.org*

[4] *https://.gcc.gnu.org*

[5] *https://github.com/riscv/riscv-pk/tree/master/bbl*

[6] *https://boom-core.org*

[7] *https://www.gnu.org/software/gdb*

[8] *https://github.com/riscv/riscv-isa-sim*

[9] *https://www.qemu.org*

[10] *www.github.com/riscv/riscv-tools*

[11] *https://sourceware.org/newlib*

[12] *https://www.gnu.org/software/libc*

[13] *https://www.eembc.org/coremark*

[14] *www.github.com/eembc/coremark*

[15] *https://busybox.net*

[16] *https://www.tldp.org/LDP/sag/html/root-fs.html*

[17] *https://github.com/riscv/riscv-linux*

[18] *www.sourceforge.net/projects/sevenzip*

[19] *https://riscv.org/software-status*