# Donanım ve Haberleşme Soyutlama için Tasarlanmış Çekirdek Çerçeve Mimarisi

Aysun Ener, Hakan Akış, Tolga Uygunol, Zafer Çoban

HBT Yazılım Tasarım Müdürlüğü, ASELSAN A.Ş., Ankara, Türkiye

{abaykal, hakis, tuygunol, zcoban}@aselsan.com.tr

**Özet.** Gittikçe gelişen donanımlar üzerinde koşan ve gittikçe karmaşık görevler üstlenen gömülü yazılımların geliştirilmesini kolaylaştıracak metotlar, yazılımın kendisi kadar önem taşımaktadır. Bu çalışmada üst seviye yazılımların, üzerinde koştuğu donanımdan soyutlanarak tasarlanması ve gerçeklenmesi için oluşturulmuş Çekirdek Çerçeve yapısı anlatılmıştır. Böyle bir yapının sağlamış olduğu kazanımlar ortaya konulmuştur. Çekirdek Çerçeve, gömülü sistemlerde koşan yazılımların donanım bağımsızlığı, ölçeklenebilirlik ve taşınabilirlik gibi ihtiyaçlarından doğmuştur. Bu altyapıyla, üst seviyede kalan yazılımların donanım değişikliğinden minimum şekilde etkilenmesi, detaylarını bilmeden donanımdan girdi alabilmesi, fiziksel arayüz farkındalığı olmadan kendisiyle aynı veya kendisinden farklı bir donanımda bulunan yazılım birimleriyle haberleşebilmesi hedeflenmiştir. Çekirdek Çerçeve, donanım soyutlama katmanı görevine ek olarak yazılımların üzerinde koşacağı ve yapılandırmaya müsait bir taban yazılımı görevi de üstlenmektedir.

**Anahtar Kelimeler.** Çekirdek Çerçeve, Donanım Soyutlama, Taşınabilirlik, Taban Yazılımı, Gömülü Yazılım

**Abstract.** Methodologies that make easy to develop embedded software, which runs on gradually improving hardware, are as important as the software itself. In this work, Core Framework infrastructure has been introduced, which is made up to abstract design and implementation of high-level software from the hardware on which the software runs. Acquirements gained by this infrastructure are also mentioned. Core Framework is resulted from the needs of hardware independence of software in embedded systems, scalability and portability. Being influenced at minimum from hardware changes, getting input from hardware without dealing with hardware issues, letting software modules to communicate with other modules, which are either on the same or on different hardware platforms without physical interface awareness, are the goals of this architecture. In addition to the hardware abstraction task, Core Framework is also responsible of being a configurable base software that lets high level software run on itself.

**Keywords.** Core Framework, Hardware Abstraction, Portability, Base Software, Embedded Software

# 1 Giriş

Haberleşme sektörü, teknolojik gelişmelerden ve bu gelişim hızından en çok etkilenen sektörlerden biridir. Aselsan Haberleşme ve Bilgi Teknolojileri (HBT) bünyesinde de bu gelişmelere paralel olarak, artan haberleşme ihtiyacını ve çeşitliliğini karşılamak adına yazılım kontrollü dalga şekilleri geliştirilmektedir. Geliştirilen dalga şekli yazılımlarının el telsizi, araç telsizi, sırt telsizi, gemi telsizi, hava telsizi gibi farklı platformlar üzerinde çalışması beklenmektedir. Şimdiye kadar, projelerin başında farklı platformlara taşınabilirlik bir kriter olarak ele alınmadığından, zaman zaman benzer yazılımlar her platform için ayrı yönetilmiş, zaman zaman da derleme anahtarı kullanılarak farklılaşmalar yönetilmeye çalışılmıştır. Bu durum, yazılımların ve onların yönetiminin gittikçe karmaşık bir hale gelmesine sebep olmuş ve konfigürasyon yönetimini içinden çıkılamayacak bir hale getirmiştir.

Dalga şekli yazılımlarının, gün geçtikçe daha karmaşık ve kapsamlı hale gelmelerine rağmen, teslimat süreleri de gittikçe kısalmaktadır. Tüm bu şartlar, dalga şekli yazılımlarını donanımdan soyutlayacak ve bu yazılımların farklı platformlara kolayca taşınabilmesini sağlayacak bir Çekirdek Çerçeve yazılımı ihtiyacını doğurmuştur. Çekirdek Çerçeve yazılımının, üst seviye yazılımların donanım ve haberleşme soyutlama ihtiyacını karşılamasının yanında, temel donanım testi yapma, yazılım güncelleme, dalga şekli yazılımlarını çalıştırma gibi yeteneklere de sahip olması istenmiştir.

Askeri sistemlerde benzer amaçlarla yapılmış başka çalışmalar da bulunmaktadır [1, 2]. Yazılım tabanlı telsizler için hazırlanmış olan Yazılım Haberleşme Mimarisi'nde de yazılımların taşınabilirliğini artırma ve yazılım geliştirme süresini kısaltma amacıyla tanımlanmış bir çekirdek çerçeve mimarisinden bahsedilmektedir [3].

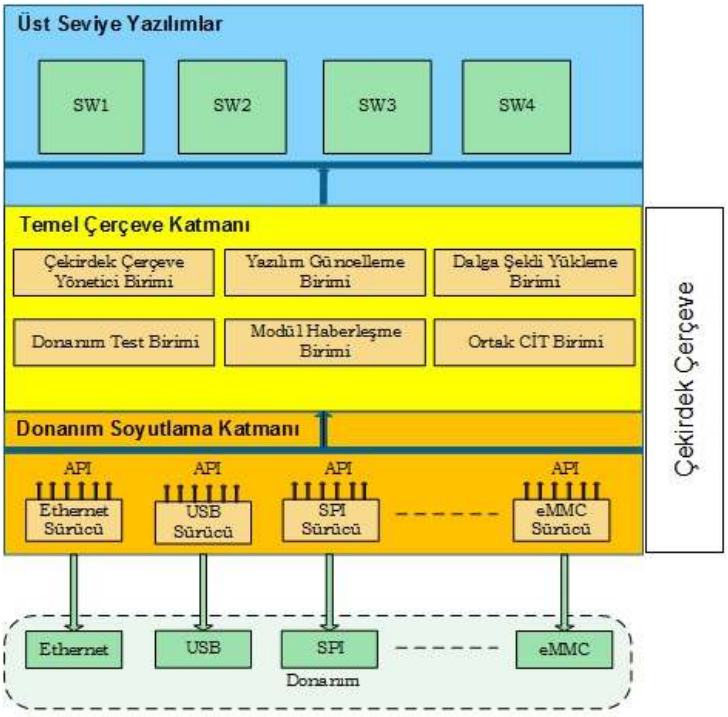
Bu makalede Aselsan HBT bünyesinde tasarlanan ve gerçeklenen Çekirdek Çerçeve yazılım mimarisi ve kazanımlarından bahsedilecektir. İlk olarak Çekirdek Çerçeve yazılımından önceki mimariye kısaca değinilecek, daha sonra Çekirdek Çerçeve mimarisi detaylandırılacaktır. En son bölümde de Çekirdek Çerçeve mimarisinin kazandırdıkları ve bu konuda gelecekte yapılabilecekler ele alınacaktır.

# 2 Çekirdek Çerçeve Öncesi Yazılım Mimarisi

Çekirdek Çerçeve öncesi yazılımlarda, her bir dalga şekli yazılımı için ayrı bir çekirdek (kernel) imajı oluşturulmakta ve bu imajlar kalıcı bellekte saklanmaktadır. Seçilen dalga şekline göre, ilgili yazılımları içeren çekirdek imajı ayağa kaldırılmaktadır. Bu durum, dalga şekli geçişlerinde cihazın kapatılıp açılması gerekliliğini doğurmuştur. Birimler arası haberleşme, yazılım güncelleme, yazılım günlüğü izleme gibi, dalga şekilleri için ortak olan servisleri sağlayan yazılım birimleri, ufak değişiklikler ile dalga şekli yazılımlarında tekrarlanmıştır. Bu yazılım birimlerinde yapılan bir değişiklik, tüm dalga şekli yazılımlarının yeniden derlenmesine sebep olmaktadır. Dalga şekli yazılımları donanımdan soyutlanmadığı için, farklı platformlara doğrudan taşınabilirliği ortadan kalkmış, aynı platformda koşacak farklı dalga şekilleri için de sürücü yazılımları yeniden kullanılamamıştır. Bu durum, her dalga şekli için yeni gerçeklemeler gerektirmiş, işçilik ve zaman maliyetlerini artırmıştır.

# 3 Çekirdek Çerçeve Yazılım Mimarisi

Çekirdek Çerçeve yazılım mimarisi temel olarak iki katmandan oluşmaktadır. En alt katman, donanım sürücülerinin bulunduğu, üst seviye yazılımlara donanımdan bağımsız arayüzler sağlayan Donanım Soyutlama Katmanı'dır. Onun üzerinde ise temel testleri yapabilen, yazılım birimleri arasındaki donanımdan soyutlanmış haberleşme ve yazılım güncelleme gibi servisleri veren Temel Çerçeve Katmanı bulunur. Yazılım mimarisi içinde Çekirdek Çerçeve yazılımının üst seviye yazılımlarla ve donanımla olan ilişkisi Şekil 1'de verilmiştir.




**Şekil 1.** Yazılım Mimarisi

## 3.1 Donanım Soyutlama Katmanı (DSK)

Donanım Soyutlama Katmanı, dalga şekli yazılımlarının, üzerinde koştuğu donanımdan bağımsız olması ve bu sayede farklı platformlara taşınabilmesi hedeflenerek tasarlanmıştır [4]. Bu katman, dahili ve harici arayüzlere bağlı çevre elemanları için geliştirilen sürücü yazılımlarından oluşmaktadır. Sürücüler, işletim sistemi I/O (Input/Output) servisine kayıtlanmaktadır. Yazılım birimleri, çekirdek ya da kullanıcı bölgesinde olmasından bağımsız olarak, `read()`, `write()`, `ioctl()` gibi birçok işletim sisteminde ortak olan I/O sistem arayüzleri ile istediği servise erişim sağlayabilmektedir.

## 3.2 Temel Çerçeve Katmanı

Çekirdek Çerçeve'yi yönetme ve cihaz üzerindeki diğer çekirdek çerçevelerle haberleşme, temel donanım doğrulama ve cihaz içi testlerin yapılması, yazılımların güncellenmesi, dalga şekli uygulamalarının çalıştırılması ve yazılım birimleri arasındaki haberleşmenin sağlanması Temel Çerçeve Katmanı'nda gerçekleştirilir.

**Çekirdek Çerçeve Yönetici Birimi.** Çekirdek Çerveve Yönetici Birimi işletim sistemi ayağa kalktıktan sonra koşan ilk yazılım birimidir. İlk olarak, kalıcı bellekten okunan donanım revizyon bilgisine uygun bir şekilde tüm sürücü ve Temel Çerçeve yazılım birimi ilklemeleri yapılmaktadır. Bu sayede, farklı donanım revizyonları için farklı ilkleme senaryoları uygulanabilmektedir ve üst seviye yazılımların donanım değişikliklerinden minimum şekilde etkilenmesi sağlanabilmektedir. İlkleme işlemlerinden sonra, cihaz içerisindeki diğer işlemcilerin ayakta olup olmadığı bilgisi Modül Haberleşme Birimi üzerinden gönderilecek mesajlar ile sorgulanmaktadır. Ayakta olan tüm yazılım konfigürasyon birimlerinden açılış cihaz içi test sonuçları toplanmaktadır ve toplanan test sonuçları Ortak Cihaz İçi Test yazılım birimine iletilmektedir.

**Donanım Test Birimi.** Donanım Test Birimi, kart üzerinde üretim sırasında veya sonradan oluşabilecek problemlerin tespit edilebilmesi için kullanılan Temel Çerçeve Katmanı yazılım birimidir. Kart doğrulama testleri, her kart için ayrı ayrı yapılmaktadır. Bu yazılım birimiyle aynı kart üzerindeki başka işlemcilerle Modül Haberleşme Birimi üzerinden haberleşilerek donanımsal olarak direkt kendisine bağlı olmayan çevre birimlerin de testi yapılabilmektedir. Testler cihaza bağlı seri port üzerinden komut satırı arayüzü ile yapılmaktadır. Kullanıcı tüm testlerin otomatik olarak peş peşe çalışmasını tetikleyebildiği gibi, listeden seçtiği bir testi de çalıştırabilmektedir. Testler, kendiliğinden çalışan testler ve etkileşimli testler olmak üzere ikiye ayrılmaktadır.

**Ortak Cihaz İçi Test (CİT) Birimi.** Cihaz içi testler (BIT: Built-in Test), askeri haberleşme gibi kritik uygulamalarda kullanılan gömülü sistemlerde, sistemin güvenilirlik seviyesinin arttırılmasını ve onarım süresinin minimuma indirilmesini sağlayan, cihazın kendi kendini test ettiği mekanizmalardır. James A. Butler [5], cihaz içi testleri Açılış CİT, Sürekli CİT ve Operatör Kontrollü CİT olmak üzere üç grup altında listelemektedir. Açılış CİT, yazılım ve donanım modüllerinin işlevselliğinin kapsamlı olarak ele alındığı testlerden oluşmaktadır. Operatör kontrollü CİT, genellikle hata sonrasında hatanın yerini belirlemek amaçlı kullanılmaktadır. Sürekli CİT'ler ise arka planda cihazın çalışmasına engel olmayan, periyodik olarak yapılan testleri kapsamaktadır. Bu yazılım biriminin Temel Çerçeve'ye eklenmesindeki amaç, dalga şekli uygulamalarından bağımsız olarak cihaza özel testlerin yapılmasının sağlanmasıdır.

**Modül Haberleşme Birimi.** Modül Haberleşme Birimi'nin temel prensibi, yazılım birimlerinin haberleşmede sadece hedef yazılım birimlerini muhatap almasıdır. Hedef yazılım birimine giden fiziksel yollar soyutlanmıştır. Yazılım birimleri mesaj almak ve

göndermek için her zaman haberleşme birimi tarafından sunulmuş yazılım arayüz fonksiyonlarını kullanır. Bu birim sayesinde aynı donanım üzerinde aynı işlemcide çalışan iki yazılım birimi de, farklı kartlar üzerindeki yazılım birimleri de aynı arayüzleri kullanarak haberleşebilmektedir. Hedef yazılım biriminin konumuna göre ilgili fiziksel ya da mantıksal arayüzden paket gönderilir.

Bir haberleşme birimi diğer bir donanımda bulunan Modül Haberleşme Birimi ile konuşur. Yazılım birimlerinin arayüz fonksiyonlarını kullanarak sağlamış olduğu bilgilere dayanarak, Modül Haberleşme Birimi, hedef yazılım biriminin hangi donanım parçası üzerinde olduğunu tespit eder. Arayüzlerdeki tekillik sayesinde, hedef yazılım biriminin işlemci çekirdeği değiştiğinde, ya da farklı bir işlemciye geçtiğinde; arayüzlerde herhangi bir değişiklik yapmadan, kaynağa ve hedefe bu değişikliği bildirmeden sadece yazılım birimi - donanım eşleme bilgisini güncelleyerek haberleşmenin devam etmesi sağlanabilir.

**Dalga Şekli Yükleme Birimi.** Telsiz donanımları; üzerinde çok çekirdekli bir işlemci bulunan kontrol kartı, üzerinde FPGA bulunan bir kripto kartı ve üzerinde birden fazla işlemci çekirdeği ve FPGA bulunan SoC (System On Chip) yapısındaki bir işlemciye sahip Modem kartı gibi birçok kartın bulunduğu bir mimariden oluşmaktadır. Dalga Şekli Yükleme Birimi, bu kartlarda koşacak dalga şekli yazılım konfigürasyon birimlerinin tümünü çalıştırmakla yükümlüdür.

Cihazda donanımsal bir sorun bulunmadığı ve cihazın dalga şekli yüklenmesine hazır olduğu bilgisi edinildikten sonra, parametre alanından okunan açılış dalga şekli moduna göre ilgili dalga şekli yazılımlarını çalıştırma senaryosu başlatılmaktadır. Dalga Şekli Yükleme Birimi, kendisinin koştuğu işlemci çekirdeğinde, ilgili işletim sistemi arayüzlerini kullanarak, üst seviye dalga şekli yazılım konfigürasyon birimini çalıştırmaktadır. Diğer karttaki SoC yapısı üzerindeki ikinci çekirdekte ise, AMP (Asymmetric MultiProcessing) [6] çalışma mantığını kullanarak ilgili yazılım konfigürasyon birimi imajını koşturmaktadır. SoC üzerindeki FPGA'in konfigürasyonu için de, SoC mimarisindeki FPGA konfigürasyon arayüzü kullanılmaktadır.

**Yazılım Güncelleme Birimi.** Donanım üzerine yazılım atma veya mevcut yazılımları güncelleme işi Çekirdek Çerçeve içinde bulunan Yazılım Güncelleme Birimi tarafından yürütülür. Kişisel bilgisayar üzerindeki bir uygulama Yazılım Güncelleme Birimi ile bağlantı kurar. Yazılım Yükleme Birimi yükleyeceği yazılımlar hakkında sabit bir bilgiye sahip değildir. Bu da, bu katmana esneklik kazandırmaktadır. Söz gelimi, yüklenecek yazılımın ismi, cihaz içerisindeki donanım parçalarından hangisine yazılım yükleneceği, yazılımın türü gibi bilgiler uygulama tarafından sağlanır.

## 4 Çekirdek Çerçeve Gerçekleştirimi

Çekirdek Çerçeve gerçekleştirimi vxWorks 6.9 işletim sistemi üzerinde yapılmıştır. Bu işletim sisteminin kullanılma sebebi, var olan dalga şekli yazılımlarının da bu işletim sistemini kullanmasıdır. Çekirdek Çerçeve gerçekleştirimi için "C" programlama dili kullanılmıştır. Dalga şekli yazılımlarında ise hem "C", hem de "C++" programlama

dilleri kullanılmaktadır. Çekirdek çerçeve, işletim sisteminin çekirdek bölgesinde çalışmaktadır. Dalga şekillerinin ise hem kullanıcı bölgesinde, hem de çekirdek bölgesinde çalışması desteklenmektedir. Çekirdek çerçeve, dalga şekillerine kullanımı kolay, donanımdan bağımsız arayüzler sağlar. Bu arayüz fonksiyonları ise kullanıcı modundan çekirdek moduna erişim için `ioctl()` sistem çağrısı kullanır. Gerçeklenen `ioctl()` metodu içinde çeşitli sürücü işlevleri sağlanır. Üst seviye yazılımlara sağlanan bir sürücü arayüzünde aşağıdakine benzer fonksiyonlar bulunabilir:

```
ConfigSensor()
{
   ioctl( CONFIG_SENSOR_COMMAND )
}

ReadSensor()
{
   ioctl( READ_SENSOR_COMMAND )
}
```

Bu işlevler, sürücü içerisindeki `ioctl()` metodu içerisinde aşağıdaki örnekte olduğu gibi gerçeklenir:

```
SensorDrvIoctl()
{
   switch(SENSOR_COMMAND)
   {
      case CONFIG_SENSOR_COMMAND:
         SensorDrvConfig();

      case READ_SENSOR_COMMAND:
         SensorDrvRead();

   }
}
```

# 5 Sonuç

Bu çalışmada, üst seviye yazılımların donanımdan soyutlanarak daha taşınabilir hale getirilebilmesi amacıyla tasarlanan Çekirdek Çerçeve mimarisinden bahsedilmiştir. Bu mimariye uygun olarak gerçeklenen Çekirdek Çerçeve, yeni telsiz projelerinde kullanılmaya başlanmış ve projelere çeşitli kazanımlar sağlamıştır. Mesela, geliştirilen donanım soyutlama katmanı sayesinde yeni dalga şekillerinde sürücü geliştirmek için harcanması gereken 12 adam*ay büyüklüğündeki işgücü gereksinimi ortadan kalkmıştır. Yeni dalga şekillerinin tak-çalıştır mantığında sisteme kolayca eklenebildiği görülmüştür. Donanım değişikliklerinin Çekirdek Çerçeve'de ele alınması sayesinde, üst seviye

yazılımların, donanımda değişiklik yapıldığında değiştirilmesi gerekmemektedir. Ayrıca yazılım birimleri arasındaki haberleşmenin soyutlanması sayesinde, yazılım birimleri, haberleşecekleri yazılım biriminin kendisiyle aynı kartta ya da farklı kartta olmasından bağımsız olarak aynı arayüzü kullanabilmektedirler. Bu sayede yazılımların taşınabilirliği artmıştır. Temel donanım testlerinin Çekirdek Çerçeve'de yapılabiliyor olması sayesinde, ayrıca bir test yazılımı yükleme ihtiyacı olmadan cihaz donanımı hızlı bir şekilde test edilebilmektedir. Bu da, üretim ve bakım süreçlerinin hızlanmasını ve hataların erken aşamalarda tespit edilebilmesini sağlamıştır. Çekirdek çerçeve yazılımı, farklı işlemciler kullanan farklı kartlara da kolayca taşınabilmiştir. Üst seviye yazılımları daha taşınabilir hale getirmesinin yanında kendisi de kolay taşınabilir olduğundan, projelerin kısalan tasarım sürelerine uygun bir altyapı sağlamıştır.

Mevcut yapısıyla Çekirdek Çerçeve, üst seviye yazılımların donanım ve haberleşme arayüzlerini soyutlamakta, ancak işletim sistemi arayüzlerini soyutlamamaktadır. Gelecekte, üst seviye yazılımların işletim sisteminden de soyutlanması sağlanarak taşınabilirlik artırılabilir. Üst seviye yazılımların taşınabilirliğinin artırılması için kullanıcı arayüzü yazılımları da fonksiyonel yazılımdan ayrıştırılabilir. Ayrıca, üst seviye yazılımlarda yazılım ürün hattı yaklaşımının uygulanarak yazılımlar arası farklılıkların derleme esnasında değil, çalışma anında konfigürasyon dosyaları kullanılarak yönetilmesinin de, yazılımların taşınabilirliğini artırmasının yanında konfigürasyon yönetimini de kolaylaştıracağı düşünülmektedir [7].

# 6 Kaynaklar


1. Kim, J., Lee, J., Bae, D.H., Ryu, D., Lee, S.: Developing a Common Operating Environment for Military Application. In: FTDCS'03, pp. 367–373. IEEE (2003)
2. Pritchett, W.W., Bartkiewicz, J.D.: A Real-Time Operating Environment for Army Weapon Systems. In: Proceedings of 18th Digital Avionics Systems Conference, pp. 8.A.3-1–8.A.3-7 (1999)
3. Software Communications Architecture Specification Version 4.1, http://www.wireless-innovation.org/sca-based-standards-library (2015)
4. Yoo, S., Jerraya, A. A.: Introduction to Hardware Abstraction Layers for SoC. Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'03). IEEE Computer Society (2003)
5. Butler, J. A.: Application and Evaluation of Built-In-Test (BIT) Techniques in Building Safe Systems. The Journal of Defense Software Engineering (2006)
6. McDougall, J.: Simple AMP Running Linux and Bare-Metal System on Both Zynq SoC Processors. Application Note: Zynq-7000 AP SoC. (2013)
7. Kalay, A.: Atış Kontrol Yazılımlarında Ürün Hattı Yaklaşımının Uygulanması. UYMS'15, (2015)