SAVTEK 2018, 9. SAVUNMA TEKNOLOJİLERİ KONGRESİ
27-29 Haziran 2018, ODTÜ, Ankara

BİR AÇIK KAYNAK KODLU GERÇEK ZAMANLI İŞLETİM SİSTEMİ (FreeRTOS) İLE GÖMÜLÜ YAZILIM GELIŞTİRME ÇALIŞMALARI

Alper YAZAR

Uzman Elektronik Donanım Tasarım Mühendisi, Sayısal ve Gömülü Sistemler Elektronik Tasarım Müdürlüğü Savunma ve Sistem Teknolojileri (SST) Sektör Başkanlığı ASELSAN, 06172, Ankara,
ayazar@aselsan.com.tr

ÖZET

Bir gömülü yazılım geliştirme sürecinin başlangıcında ihtiyaca göre karmaşık veya basit bir işletim sisteminin kullanılması, işletim sistemi kullanılmaması gibi kararlar alınır. Sistemin gerçek zamanlılık gereği var ise kullanılacak işletim sisteminin de bu özellikte olması sistem gereklerini sağlamak adına anlamlı olmaktadır. Gerçek zamanlı bir işletim sisteminin kullanımı kararlaştırıldığı zaman ise kullanılabilecek ücretli veya ücretsiz çeşitli işletim sistemi alternatifleri ortaya çıkmaktadır. Bu bildiride popüler, ücretsiz, açık kaynak kodlu bir gerçek zamanlı işletim sistemi olan FreeRTOS ile Xilinx Zynq SoC üzerinde yapılan çalışmalarda elde edilen tecrübeler aktarılmaktadır.

Anahtar Kelimeler: Gerçek zamanlı işletim sistemi, FreeRTOS, Xilinx Zynq SoC, açık kaynak

EMBEDDED SOFTWARE DEVELOPMENT WITH AN OPEN SOURCE REAL TIME OPERATING SYSTEM (FreeRTOS)

ABSTRACT

At the beginning of an embedded software development process, decisions such as the need to use a complex or simple operating system, no operating system, etc., are taken. If the system has to operate in real time, using a real time operating system makes sense to satisfy requirements easier. When a real time operating system is decided to be used, various alternatives, either paid or free, arise. This paper conveys experience gained from works on Xilinx Zynq SoC with FreeRTOS, a popular, free, open source real-time operating system.

Keywords: Real time operating system, FreeRTOS, Xilinx Zynq SOC, open source

1. GİRİŞ

Bir gömülü yazılım geliştirileceği zaman verilmesi gereken önemli kararlardan biri yazılımın koşacağı platformda bir işletim sistemi olup olmayacağıdır. İşletim sistemi kullanılacak ise özelliğinin (genel amaçlı, gerçek zamanlı gibi), kaynak koduna erişim ve değiştirme olanaklarının, lisanslama ve ücret seçeneklerinin, kod büyüklüğünün, alınabilecek teknik destek hizmetinin ne olacağı düşünülmelidir. İşlemci kabiliyetlerinin artması, uygulamaların karmaşıklaşması ve ürün geliştirme sürelerinin her sektörde olduğu gibi savunma sanayii sektöründe de kısalması sebebiyle savunma sanayii ürünlerinde bulunan gömülü sistemlerde de işletim sistemi kullanımının yaygın olduğu görülmektedir. Bu bildiride, bu alanda kazanılan tecrübeler FreeRTOS özelinde aktarılmaktadır.

2. FreeRTOS HAKKINDA

Açık kaynak kodlu ve ücretsiz bir gerçek zamanlı işletim sistemi olan FreeRTOS, farklı sektörlerden katılımla yapılan bir araştırmada 2017'de gerçek zamanlı gömülü işletim sistemleri arasında en çok kullanılan ve 2018'de tüm gömülü işletim sistemleri arasında kullanılması en çok planlanan işletim sistemi seçilmiştir [1]. Amazon firması AWS isimli bulut hizmetine bağlı olacak, temelde nesnelerin interneti ("IoT") amaçlı çalışacak mikrokontrolcü tabanlı cihazlar için FreeRTOS tabanlı bir işletim sistemi yayınlamıştır [2]. Firmanın bu girişimi, FreeRTOS'un sektörde kullanımının devam edeceğinin bir göstergesidir. İşletim sisteminin geliştirilmesine 2003 yılından itibaren devam edilmektedir ve bu bildirinin yazıldığı an itibariyle güncel sürümü 10.0'dır.

2.1. Lisans

Bir yazılım bileşeni bir üründe kullanılacak ise kontrol edilmesi gereken ilk noktalardan biri lisans konusudur. FreeRTOS ticari ürünlerde kullanılabilir, ürünün kaynak kodu kapalı tutulabilir, üründe FreeRTOS olduğunun belirtilmesine gerek yoktur. Sürüm 9'a kadar (9 dahil) FreeRTOS'un kaynak kodunda değişiklik yapılırsa bu değişikliklerin açık kaynak olarak yayınlanması gerekmekteyken güncel sürüm 10'dan itibaren işletim sisteminin kaynak kodunda kapalı değişiklikler yapılabilmektedir [3, 4]. Tecrübe ettiğimiz gömülü yazılım geliştirme çalışmalarında, işletim sisteminin kodunda değişiklik yapma ihtiyacı duyulmamıştır. Genel anlamda lisans konusu ile ilgili bir problem olmasa da konunun bu işletim sistemini kullanmayı düşünen geliştiriciler tarafından detaylı bir şekilde araştırılması tavsiye edilmektedir.

2.2. İşlemci Desteği

FreeRTOS resmi olarak Zynq, Zynq MPSoC gibi SoC'lerde bulunan uygulama seviyesi işlemcilerden ARM Cortex-M, Microchip PIC gibi mikrokontrolcülere kadar 35'ten fazla farklı işlemci ailesini desteklemektedir [5]. Bu destek özellikle geniş ürün çeşitliliğinde çalışan ve bundan dolayı farklı ürün aileleri için yazılım geliştiren tasarım ekiplerinin yazdığı kodların taşınabilir olması için önemlidir.

2.3. Dokümantasyon

FreeRTOS'un resmi web sitesinden indirilebilen yeterli sayıda hem eğitim kaynağı hem de referans kaynak bulunmaktadır. Ayrıca resmi olarak desteklenen her işlemci için örnek projeler de indirilebilmektedir. Tecrübe edilen diğer yazılım kütüphaneleri ve işletim sistemleri göz önünde bulundurulduğunda sağlanan dokümanların yeterli olduğu görülmektedir. Sağlanan kaynaklar ürünün geliştirme sürecinin kısıtlı olduğu durumlarda bu işletim sistemi ile çalışmamış bir gömülü yazılımcının gerekiyorsa 1-2 hafta gibi kısa bir sürede FreeRTOS'u öğrenebilmesine imkan tanımaktadır.

2.4. Destek

Geliştirici firma, desteği kendi forumları üzerinden sağlamaktadır. Tecrübeler sonucunda, sağlanan desteğin birçok konuda yeterli ve hızlı olduğu görülmüştür. Ücretli destek istenildiği durumlarda OpenRTOS işletim sisteminin değerlendirilmesi uygun olacaktır.

3. TEMEL İŞLETİM SİSTEMİ ve FreeRTOS BİLEŞENLERİ

Düşük kaynaklı mikrokontrolcülerde çalışabilen işletim sistemleri geliştiriciye kapsamlı işletim sistemlerinin (gömülü Linux gibi) sağladığı olanakları sunamaz. FreeRTOS ve eşdeğer işletim sistemlerinin temel olarak sağladığı iki adet bileşen vardır: "Scheduler" (Zamanlayıcı) ve IPC ("Inter Process Communication").

FreeRTOS'ta yapılacak işe göre "pre-emption" ve "time slicing" özellikleri kullanılarak 3 farklı "scheduler" konfigürasyonu yapılabilir. Proje geliştirmeye başlarken "pre-emption" ve "time slicing" özelliklerinin açık olmasının faydalı olduğu gözlemlenmiştir. Zamanlayıcının nasıl çalıştığını anlamak FreeRTOS'u etkili ve doğru kullanmak için önemlidir ve detayları bu bildirinin konusu dışındadır.

Diğer temel bileşen ise sağlanan IPC mekanizmalarıdır. Bunlar arasında "queue", "mutex", "semaphore" gibi mekanizmalar yer almaktadır. "Queue", iki "task" arasında veya ISR ("Interrupt Service Routine") ile "task" arasında veri taşınması için kullanılabilir. "Mutex" ve "semaphore" ise daha çok kaynak paylaşımı, sinyalizasyon ve senkronizasyon için tercih edilen mekanizmalardandır.

Bu temel bileşenlere ek olarak "Heap Management", "Software Timers" gibi yardımcı bileşenler de bulunmaktadır. FreeRTOS çalışılan işlemciye göre destek sayısı değişmekle beraber 5 farklı "Heap Management" yöntemi sunmaktadır. Bu sayede kodun çalışması sırasında dinamik hafıza gerektiren uygulamalar daha kolay yazılabilmektedir. Fakat gömülü ve gerçek zamanlı bir sistemde dinamik hafıza kullanımı tasarımcıların dikkat etmesi gereken bir konudur ve bu bildiride detaylandırılmayacaktır. FreeRTOS bileşenleri dilenirse derleme esnasında statik olarak yaratılabilir ve uygulama yazılımı da bu şekilde hazırlanırsa hiç dinamik hafıza alanı kullanılmadan yazılım tamamlanabilir.

“Software Timers” ise yazılımcının periyodik veya belirli bir zaman sonra yapılması gereken işleri tanımlamasını kolaylaştırır.

FreeRTOS’un sunduğu diğer imkanlara dokümanlarından ulaşılabilir.

4. FreeRTOS KULLANARAK YAZILIM GELİŞTİRME

FreeRTOS kullanmanın çeşitli avantaj ve dezavantajları bulunmaktadır. Geliştirici, çalıştığı proje özelinde bu noktaları değerlendirerek karar vermelidir.

4.1. Avantajlar

4.1.1. Birçok Geliştiricinin Beraber Çalıştırmasını Kolaylaştırması

FreeRTOS’un getirdiği ilk kazanım, birçok geliştiricinin yazdığı kod parçalarının entegrasyonunu hızlandırmasıdır. Zaman sınırı olan çalışmalarımızda ekip olarak bu avantaj gözlemlenmiştir. Bu kazanım sadece FreeRTOS’a özgü değildir ve başka işletim sistemlerinin kullanımı ile de kazanılabilir. Donanıma çok yakın çalışan geliştiricilerin tercih ettiği “bare metal” yaklaşımı ile yazılan bir yazılım parça parça, birden fazla geliştirici tarafından yazıldıysa bu parçaların entegrasyonu sırasında uzun süren hata ayıklama seansları gerekebilmektedir. FreeRTOS’un sağladığı IPC ve “scheduler” mekanizmaları sayesinde parçaların daha kolay ve hızlı bir şekilde birleştirilebildiği tecrübe edilmiştir. FreeRTOS’un sağladığı bu mekanizmaların kararlı olduğu görülmüş ve FreeRTOS kaynaklı bir hata ile karşılaşılmamıştır.

4.1.2. Kısıtlı Donanım ile Geliştirme Yapılmanın Kolaylaşması

Savunma sanayii projelerinin bir özel durumu ise yapılan elektronik kartların yapıldığı proje ihtiyaçlarına göre birkaç adet ile sınırlı kalabilmesidir. Projelerin sıkı takvimleri düşünüldüğü zaman üretilen birkaç adet karta aynı anda donanım tasarım, donanım doğrulama, yazılım geliştirme, üretim altyapısı geliştirme ekipleri gibi var olan kart sayısından daha çok birimin ihtiyacı olmaktadır. Böyle durumlarda yazılım geliştiricinin benzer işlemci içeren var olan başka bir kartta (“demo board” gibi) çalışması gerekebilir. FreeRTOS kullanımının getirdiği bir avantaj da geliştirilen yazılımların kartlar arasında taşınmasının daha kolaylaşmasıdır. Örneğin geliştirdiğimiz bir projenin kendi kartının kısıtlı sayıda olmasından dolayı yazılım geliştirme çalışmaları Xilinx ZC-702 kartında başlamış, daha sonra ürünün kendi kartında bulunan başka tip numaralı bir Xilinx Zynq SOC ürününe sorunsuzca taşınmıştır. FreeRTOS bir miktar soyutlama sunarak yazılımcının işini kolaylaştırmaktadır. Hiç donanım olmadığı durumlarda bile FreeRTOS’un sunduğu Windows’ta çalışan emülatorü kullanılarak da kısıtlı testler yapılabilir.

4.1.3. Donanımdan Soyutlanabilme

Üst seviye (uygulama seviyesi) yazılım geliştiricileri donanımdan mümkün olduğunca uzaklaşmak isterken sürücü geliştiricileri gibi donanıma çok yakın

çalışan geliştiriciler ise tüm donanım kabiliyetlerine direkt olarak erişmek ister. Gömülü yazılım geliştiricisinin ilk defa çalışılan bir işlemcide öğrenmesi gereken temel donanım bileşenleri vardır. Bunlardan ve muhtemelen hata ayıklama süresi en uzun olabileceklerden biri de “interrupt” (kesme) mekanizmalarıdır. FreeRTOS kullanıldığı zamanlarda ise temel “interrupt” kodları işletim sistemi tarafınca sağlanır. Özellikle bu bildiride bahsedilen Xilinx Zynq SoC’lerdeki ARM çekirdeklerinde bulunan “nested interrupt” yönetimi geliştiriciyi biraz daha zorlayabilir. “Nested interrupt”, FreeRTOS tarafından desteklenmekte ve geliştiricinin işini kolaylaştırmaktadır. FreeRTOS bunun gibi özelliklerle donanımın zor olabilecek kısımlarını soyutlarken geliştiricinin “bare metal” yöntemindeki gibi donanıma tümüyle erişmesine imkân tanır. Böylece soyutlama-direkt erişim gücü arasında iyi bir denge de sağlanmış olur.

4.1.4. Yazılımın Alt Parçalara Daha Kolay Bölünebilmesi

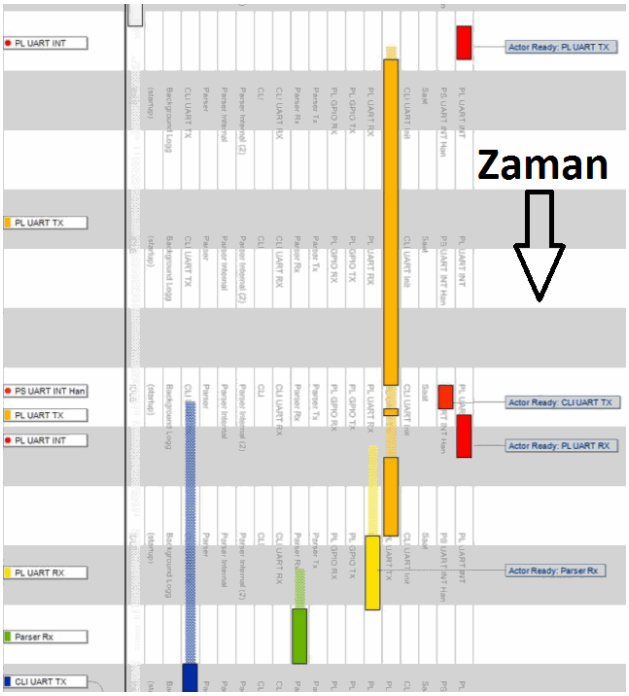
“Bare metal” programlamadan farklı olarak FreeRTOS gibi bir işletim sisteminin sunduğu zamanlayıcı ve IPC olanakları yazılımın gereklerinin alt parçalara (“task”) bölünmesini ve bu bölümlerin ayrı ayrı doğrulanabilmesini sağlamaktadır.

4.2. Dezavantajlar

4.2.1. Hata Ayıklamanın Zorlaşması

“Bare metal” tekniği ile karşılaştırıldığı zaman ortaya çıkabilecek en önemli problem hata ayıklamanın, işletim sistemli bir gömülü sistemde daha zor olabileceğidir. Bu durum özellikle yazılım tasarımlarında tek bir büyük döngü (“Super Loop”) barındıran tasarımcılar için daha zor olabilmektedir. FreeRTOS gibi işletim sistemlerinde bulunan zamanlayıcı çeşitli olaylar oldukça “task switch” kararı vermektedir. Böyle bir durumda klasik “step-by-step” hata ayıklama sürecinde işlemcinin “program counter” (PC) yazmacı bir sonraki satırdaki kodun adresini değil, zamanlayıcının yeni başlattığı kodun adresini alır. Bu da hata ayıklayan kişinin kullandığı yazılımın kaynak kodunda kontrolü dışında atlamalar gözlemlemesine sebep olur. Hata ayıklama sırasında yazılımın mimarisi izin veriyor ve zamanlayıcının “time slicing” özelliği açıksa FreeRTOS çekirdeğinin kullandığı zamanlayıcı donanımın durdurulması bu problemlerin yaşanmasını seyrekleştirebilir.

Alternatif bir çözüm ise “FreeRTOS Aware Debugger” yazılım ve donanımlarının kullanılmasıdır. Bu çözüm, kodun ve FreeRTOS bileşenlerinin davranışlarını daha iyi gözlemleme şansı sunar. Piyasada ücretli/ücretsiz çeşitli çözümler bulunmaktadır. Örnek bir görüntü olması açısından Şekil 1’de bu tarz bir hata ayıklama yazılımının sunduğu kayıt gözükmektedir. Bu kayıtta koyu olan çubuklar, o esnada hangi kodun (“task”ın) çalıştığını göstermektedir.



Şekil 1 – Bir “FreeRTOS Aware Debugger” Yazılımı ile Kaydı Alınmış, Kayıt Esnasında 15 Adet “Task” ve 2 Adet “Nested Interrupt” Çalışan Yazılımın Zaman İçerisindeki Akışı

4.2.2. FreeRTOS ile Çalışmanın Öğrenilmesi

Dezavantaj olarak görülebilecek bir diğer konu da FreeRTOS ile çalışmaya başlayacak kişinin işletim sisteminin temellerini öğrenmesi gerekliliğidir. Önceki kısımlarda belirtildiği gibi FreeRTOS dokümantasyonu oldukça iyi olan bir yazılımdır ve bu gereğin zorluğu elde edilebilecek kazanımlar düşünülerek proje özelinde geliştirici tarafından değerlendirilmelidir.

Diğer bir olası tehlike veya zorluk ise geliştiricinin işletim sisteminin çalışma şeklini göz ardı ederek hatalı kurgulanmış kodlar oluşturması ve bu kodlardaki hata ayıklama ile uğraşmasıdır. Dikkat edilmesi gereken nokta işletim sisteminin zamanlayıcısının “task”lar arasında geçiş yapabileceğidir. Bundan dolayı “task” içerisinde çağırılacak fonksiyonların düzenlemesine, kaynakların korunması ve paylaşımına, kodlama tekniklerinin doğru uygulanmasına özen gösterilmedir. Burada yapılacak hatalar çalışma esnasında, her zaman gözlemlenemeyen problemlerin (örneğin “deadlock”, “stack overflow/underflow”) ortaya çıkmasına sebep olabilir.

5. ÇALIŞMALAR SONUCU ELDE EDİLEN TECRÜBELER

FreeRTOS kullanarak yapılan çalışmalarda elde edilen tecrübeler aşağıda belirtilmiştir. Tecrübelerin özellikle ilk defa FreeRTOS ile çalışma yapacak geliştiricilere faydalı olacağına inanılmaktadır.

1. Geliştirmeye başlamadan önce FreeRTOS web sitesinden eğitici dokümanları okunmalıdır. İşletim sisteminin tasarım mantığı anlaşılmadan yapılacak kod geliştirme çalışmaları verimli olmayacaktır.
2. Daha önce proje yapılmamış bir işlemcide çalışılacaksa FreeRTOS'un web sitesinden ilgili işlemciye ait örnek kodlar temin edilmesi ve projenin bu kodlar temel alınarak geliştirilmesi birçok problemin oluşmasını engelleyecektir.
3. Yazılımın gereklere göre alt parçalara bölünebilmesi, bunlara uygun "task"ların belirlenmesi, bu "task"lar arasındaki iletişimin ve yazılımın genel akışının kurgulanması tasarımın en önemli adımıdır. Bu adımda herhangi bir kodlamanın yapılmasına gerek yoktur. Bu aşamada harcanan zaman kayıp olarak düşünülmemelidir.
4. Xilinx ürünleri için SDK yazılımında proje oluştururken FreeRTOS projesi oluşturulabilmektedir. Bunun yerine "standalone" seçeneği ile "bare metal" proje oluşturulması ve FreeRTOS'un kaynak kodlarının projeye eklenerek devam edilmesinin daha iyi bir olacağı düşünülmektedir. Bunun sebebi ise 1) SDK tarafından sunulan FreeRTOS sürümünün en güncel sürüm olmaması ve en güncel sürümün geliştiricinin ihtiyaç duyabileceği teknik veya lisans konusunda ek özellikler sunması 2) Otomatik oluşturulan FreeRTOS projesindeki konfigürasyon isimlerinin FreeRTOS'un kendi dokümantasyonu ile birebir olmamasından dolayı oluşacak karışıklıkların önlenmesidir.
5. Eğer bir işletim sistemi kullanılacaksa ve özellikle kapalı kaynak kodlu olan belirli bir işletim sistemine ihtiyaç duyulmuyorsa FreeRTOS gibi açık kaynak kodlu, anlaşılır bir şekilde dokümante edilmiş ve kodlanmış bir işletim sisteminin seçilmesi tavsiye edilmektedir. Kullanılan yazılım bileşenlerinin açık kaynak kodlu olmasının getirdiği bir avantaj o bileşende olan hatanın geliştirici firma veya topluluğu beklemeden düzeltilebilmesidir. Örneğin çalıştığımız bir projede karşılaştığımız hatanın yazdığımız koddan değil kullanılan bir Xilinx kütüphanesinden kaynaklandığı tespit edildiğinde hata, açık kaynak kodun getirdiği avantajla hızlıca düzeltilebilmiş ve yazılım, takvime uygun şekilde tamamlanmıştır [6].

6. SONUÇ

Açık kaynak kodlu, gerçek zamanlı ve ücretsiz bir işletim sistemi olan FreeRTOS ile gömülü yazılım geliştirme çalışmaları sırasında elde edilen bilgiler ve tecrübeler diğer geliştiricilere fayda sağlayabilmek için bu bildiride paylaşılmıştır. Geniş bir işlemci ailesini destekleyen FreeRTOS, gömülü yazılım geliştiricilerinin işini kolaylaştırabilecek, yazılım olgunluğu görece iyi ve gelecek

vadeden bir işletim sistemidir. Genel olarak kullanımı önerilen bu işletim sistemini her ekibin ve geliştiricinin, proje özelinde avantaj ve dezavantajlarını göz önünde bulundurarak değerlendirmesi uygun olacaktır.

KAYNAKÇA

[1] ASPENCORE. (2017). 2017 Embedded Markets Study. https://m.eet.com/media/1246048/2017-embedded-market-study.pdf internet adresinden 31 Ocak 2018 tarihinde edinilmiştir.

[2] Walker, T. (29 Kasım 2017). Announcing Amazon FreeRTOS – Enabling Billions of Devices to Securely Benefit from the Cloud. https://aws.amazon.com/blogs/aws/announcing-amazon-freertos/ internet adresinden 31 Ocak 2018 tarihinde edinilmiştir.

[3] Straughan, D. (29 Kasım 2017). Announcing FreeRTOS Kernel Version 10. https://aws.amazon.com/blogs/opensource/announcing-freertos-kernel-v10/ internet adresinden 31 Ocak 2018 tarihinde edinilmiştir.

[4] FreeRTOS. License Details. https://www.freertos.org/a00114.html internet adresinden 31 Ocak 2018 tarihinde edinilmiştir.

[5] FreeRTOS. Official FreeRTOS Ports. https://www.freertos.org/RTOS_ports.html internet adresinden 31 Ocak 2018 tarihinde edinilmiştir.

[6] Github. (8 Eylül 2017). FIX: XuartPS_ReceiveBuffer() reads extra 1 character from FIFO. https://github.com/Xilinx/embeddedsw/pull/27 internet adresinden 31 Ocak 2018 tarihinde edinilmiştir.