Endüstriyel Uygulamalar için SystemC ile Döngü İçinde Donanım

Doğan Fennibay[1] Arda Yurdakul[2] Alper Şen[3]

[1] Bilgisayar Mühendisliği Bölümü, Boğaziçi Üniversitesi, İstanbul / Siemens San. ve Tic. AŞ, İstanbul
[2,3]Bilgisayar Mühendisliği Bölümü, Boğaziçi Üniversitesi, İstanbul

[1]e-posta: dogan.fennibay@boun.edu.tr [2] e-posta: yurdakul@boun.edu.tr
[3] e-posta: alper.sen@boun.edu.tr

Özetçe

Yaygın kullanıma sahip bir teknik olan döngü içinde donanım ile yeni ve geniş kabul gören bir sistem seviyesi modelleme dili olan SystemC'nin birleştirilmesi sistem doğrulaması alanında önemli fırsatlar doğurabilir ve endüstriyel sistemlerin geliştirilmesinde simülasyon yönteminin kullanımını genişletebilir. Bu çalışmada SystemC'de varolan kanal kavramı genişletilerek *melez kanal* kavramı ortaya atılmış, böylece gerçek altsistemle model arasındaki iletişimin net olarak belirtilmesi sağlanmıştır. Ayrıca SystemC simülasyon çekirdeği gerçek zaman davranışı göstermek üzere yamalanmış ve üzerinde çalıştığı işletim sistemi gerçek zamanlı işletim sistemi davranışı göstermek üzere iyileştirilmiştir. Ek olarak performans sınırlarının tespiti için matematiksel bir model geliştirilmiştir. Oluşturulan sistemin deneysel değerlendirilmesi sonunda kavramsal düzeyde bir sorunla karşılaşılmamış, 1 milisaniye çevrim zamanında kararlı, 100 mikrosaniye çevrim zamanında sınırlı oynaklık şeklinde gözlenen performans sınırları, özellikle endüstriyel uygulamalar için uygun bir yöntem geliştirildiğine işaret etmiştir. Çalışmamız sonucunda SystemC ile döngü içinde donanımın umut vaat eden bir yaklaşım olduğu sonucuna varılmıştır.

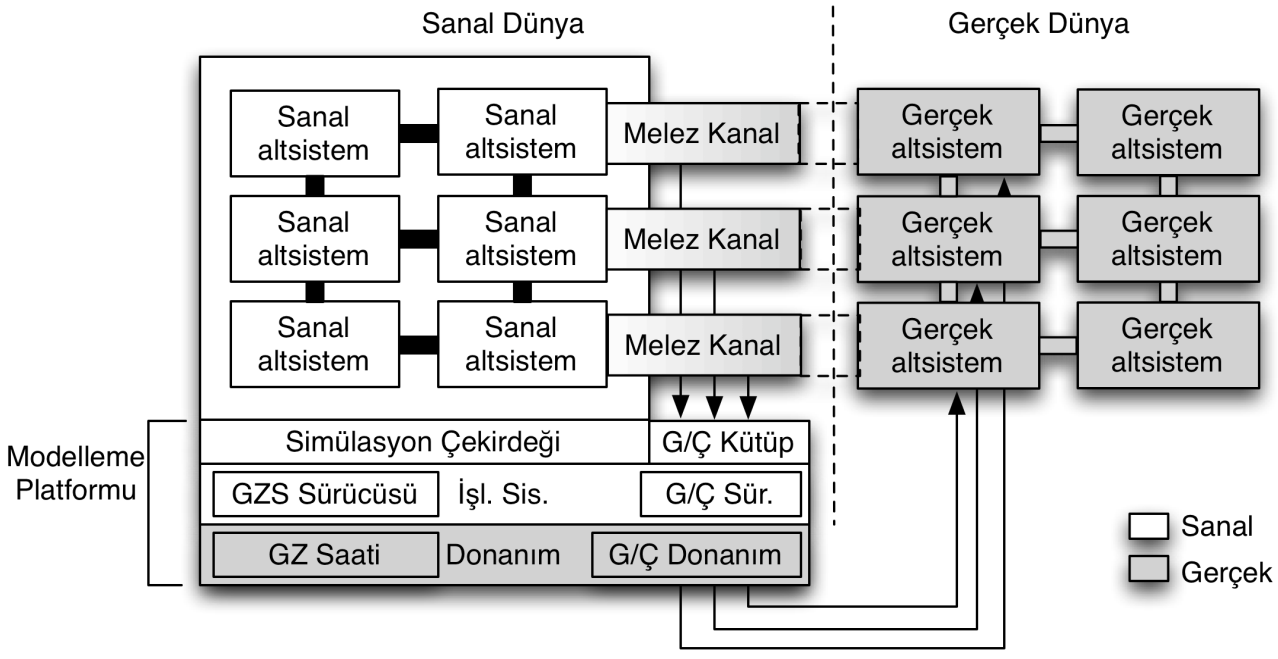
1. Giriş

Donanım/yazılım ortak-geliştirme süreci, analiz, tasarım, gerçekleme ve doğrulama süreçlerinden oluşmaktadır. Doğrulamanın temel yöntemi test iken artan sistem karmaşıklığıyla birlikte bu yöntemin yanı sıra model kontrolü, simülasyon vb. yöntemler kullanılmaya başlanmıştır. Test dışındaki yöntemlerin temelinde gerçek sistemin bir modelinin kurulması yatmaktadır.

Geliştirme sürecinin önemli bir kuralı, değişikliklerin ne kadar erken yapılırsa o kadar az maliyet ve efora neden olduğudur. Doğrulamanın erken yapılabilmesi, gerekli değişikliklerin erkenden saptanabilmesini mümkün kılar. Model kontrolü, simülasyon gibi yeni yöntemler gerçeklemeyi beklemek zorunda olmadığından, sadece modelin kurulmasına ihtiyaç duyduklarından erken doğrulamaya da izin verirler.

Bu yöntemlerin dezavantajı bir modele ihtiyaç duymaları, dolayısıyla modelle beraber gelen modelleme eforu ihtiyacı ve modellemedeki hataları beraberlerinde getirmeleridir.

Sıklıkla karşılaşılan özel bir durum, geliştirilen sistemin diğer bazı sistemlerle etkileşerek çalışacak bir sistem olması ve diğer sistemlerin gerçeklenmiş hallerinin hazır, ancak modellerinin mevcut olmadığı durumdur. Bu, diğer



Şekil 1: Gerçek ve sanal dünyalar.

sistemlerin hazır olarak satın alındığı veya başka bir geliştirme grubu tarafından tamamlanmış olduğu senaryolarda ortaya çıkar. Böyle durumlarda gerçek sistemleri modellerle (yani sanal sistemlerle) beraber çalıştırabilme imkanı bir dizi avantaj sağlar: bir yandan gerçek sistemler için modellemeden kaçınılarak daha yüksek bir modelleme doğruluğu sağlanılabilir ve toplam modelleme eforu azaltılabilir, öbür yandan henüz geliştirme aşamasındaki sanal sistemler daha model aşamasında doğrulamaya tabi tutulabilir. [1]

Döngü içinde donanım ya da döngü içinde yazılım, gerçek donanım veya yazılım altsistemlerinin sanal, yani modellenmiş, altsistemlerle entegre edilmesi yöntemine verilen isimdir. 40 yıl önce ilk uygulamasını uçuş simülasyonunda bulmuş, oturmuş bir yöntemdir. Yukarıda sayılan nedenlerle ve özellikle yüksek karmaşıklıktaki sistemlerin modellemesinden kaçınılabilmesi için bu yönteme başvurulmuştur. [1]

SystemC, C++ üzerine geliştirilmiş bir sistem seviyesi modelleme dilidir. SystemC'nin avantajlarından biri modüler tasarımıdır. Bu, her modül ve kanalın gereksinimlerinin sırasıyla port ve arayüzler aracılığıyla net olarak belirtilmesi sayesinde gerçekleşir. Modüler tasarım SystemC kanal ve modüllerinin kolayca tak-ve-çalıştır kullanımını mümkün kılar. SystemC'nin sunduğu bir diğer avantaj, modüller ve kanallar arasındaki ayrımdır. Ayrıca modellemedeki soyutlama seviyesinin değişkenliği modellemeye daha az zaman harcanmasını ve daha yüksek çalışma performansını mümkün kılar. Son olarak, SystemC'nin kazandığı popülerlik de önemli bir avantaj olarak belirtilmelidir.

Çalışmamız, SystemC kullanarak döngü içinde donanım sistemlerinin gerçeklenmesini sağlayacak altyapının geliştirilmesi üzerinedir. Mevcut teknolojiyle mümkün olan model çalışma hızları itibariyle hedef alanımız endüstriyel haberleşme sistemleridir. Böyle sistemlerde tipik iletişim çevrim zamanları 100 mikrosaniye mertebesi ve yukarısıdır (PROFINET IO, CAN). Sistem-üzerinde-kırmık[1] vb. daha hızlı sistemler kavramsal açıdan aynı yönteme tabi tutulabilir olsa dahi, böyle sistemlerde uygulanabilirlik açısından ciddi kısıtlarla karşılaşılacaktır.

İzleyen bölüm problem tanımını verecek, üçüncü bölüm konuyla ilgili önceden yapılmış çalışmaları özetleyecektir. Problemle ilgili ön bilgiler dördüncü bölümde verilmiştir. Bir sonraki bölüm, geliştirdiğimiz çözümü sunacak, altıncı bölüm sonucun deneysel değerlendirmesini detaylandıracaktır. Son bölümde de çalışma sonuçları değerlendirilecektir.

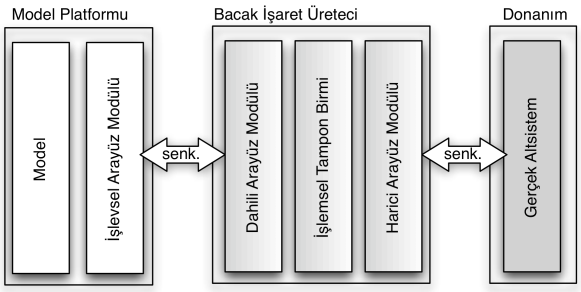
2. Problem Tanımı

Şekil 1 kısmen modellenmiş/sanal, kısmen gerçek tüm sistemin bir kavramsal görünümünü sunmaktadır. Şekilde görüldüğü üzere, sanal altsistemler bir simülasyon çekirdeği üzerinde koşmaktadırlar. Simülasyon çekirdeği bir genel amaçlı işletim sistemi üzerinde koşarken, genel amaçlı işletim sistemi de bilgisayar donanımı üzerinde koşmaktadır. Simülasyon çekirdeği, işletim sistemi ve bilgisayar donanımı üçlüsü *modelleme platformu*nu oluşturur.

Test altındaki sistem, gerçek ve sanal altsistemlerden oluşur. Bu altsistemler arasındaki iletişimi gerçekleme görevi *melez kanal*ın görevidir. Şekilde görüleceği üzere melez kanal, bir yanda simülasyon modelinin parçasıyken, aynı anda bilgisayarın G/Ç donanımıyla da haberleşmektedir. Bu haberleşme bacak[2] seviyesindeki, çevrim-hassas iletişimden işlem seviyesine kadar çeşitli soyutlama düzeylerinde gerçekleşebilir.

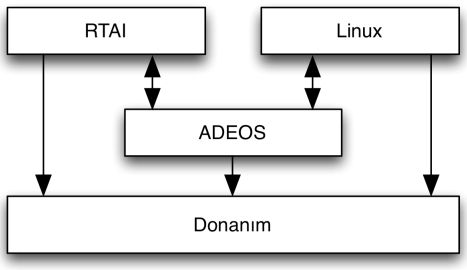
Altsistemler arasındaki çoğu etkileşim ortak bir zaman kavramının varolduğunu varsayar, örneğin bir protokoldeki zaman aşımı mekanizması haberleşen tarafların bu süreyi yaklaşık olarak eşit ölçebildiğini varsayar. Bu nedenle sanal sistemlerin *gerçek zamanlı* olarak çalışabilmelerini sağlamak gerekir. Dolayısıyla modelleme platformuna bir gerçek zaman saati dahil edilmesi gerekir. Ek olarak, sistemde oluşabilecek beklenmedik gecikmeleri önlemek üzere modelleme platformunun gerekirciliğinin iyileştirilmesi gerekir.

3. İlgili Çalışmalar



Şekil 2: Sanal Kırmık Mimarisi [3].

Sanal Kırmık C/C++ ile modellenmiş bir kırmığın gerçek donanıma bacak seviyesinde bağlanabilmesini sağlar. Bacak İşaret Üreteci adı verilen bir APKD[3] kartı modelleme platformu ile gerçek donanımın iletişimini sağlar. Mimari *Şekil 2*'de gösterilmiştir. Sanal Kırmık'ın özgün yaklaşımı İşlemsel Tampon Birimi'dir (İTB). Bu Modelleme Platformu ile Donanım'ın farklı hızlarda koşabilmesini sağlar. Her ikisi sırasıyla Dahili Arayüz Modülü ve Harici Arayüz Modülü ile senkronizedir. Bu sistemde İ TB tamponlama yöntemleriyle hız farkını yönetir. [3]



Şekil 3: RTAI, Linux, ADEOS ve donanım arasındaki etkileşim [6].

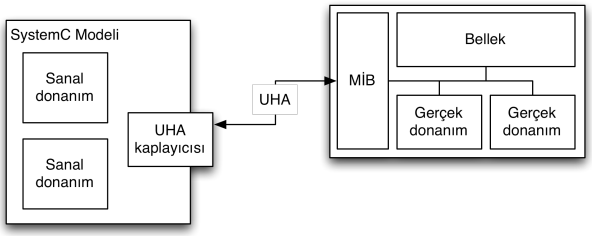
Referans [4]'teki çalışma modelleme platformunun gerekirciliği konusuna katkı yapmaktadır. Yazarlar genel amaçlı Linux'u, [5]'teki çalışmayla geliştirmişlerdir. Bu

[1] *İng.* System-on-Chip (SoC)

[2] *İng. pin*

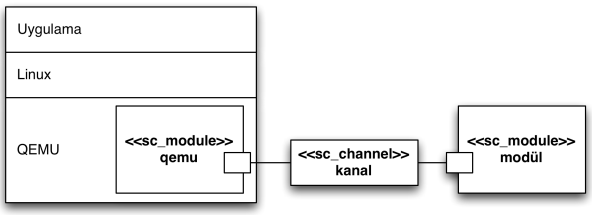
[3] Alan Programlamalı Kapı Dizileri (*İng. Field Programmable Gate Array, FPGA)*

yapıda (*Şekil 3*), RTAI [1] ve Linux ADEOS [2] üzerine yerleştirilmiştir. Donanıma erişimde RTAI'a Linux'un üzerinde öncelik verilir. ADEOS, klasik nanoçekirdek yaklaşımlarından işletim sistemlerine doğrudan donanım erişimi yetkisi vermesiyle ayrılır. RTAI Linux'a kıyasla daha az yetenekli ancak daha yüksek bir gerekircilik derecesine sahip bir işletim sistemidir. Böylece, gerçek zamanlı görevler RTAI'da, gerçek zamanlı olmayan görevler Linux'ta gerçekleştirilir. Bu şekilde [4] hem Linux'un yeteneklerinden hem de RTAI'ın gerekirciliğinden yararlanabilmiştir.



Şekil 4: Sanal Devreiçi Öykünücü.

Sanal Devreiçi Öykünücü çalışması gerçek ve sanal dünyalar arasındaki arayüzün basit bacak seviyesi yerine daha soyut bir *kanal seviyesi*nde yapılması bakımından önem taşımaktadır. Bunun için Uzaktan Hata Ayıklama Arayüzü (UHA) seçilmiştir. *Şekil 4*'te görüldüğü üzere gerçek altsistem sadece bir MİB (Merkezi İş lem Birimi) olabilir, yani bir döngü içi MİB sistemi oluşturulur. MİB bir bellek ve gerçek donanıma sahip olabilir, ancak bu bellek ve diğer çevre aygıtları doğrudan sanal altsistemlerle etkileşim kuramazlar. Son olarak belirtmek gerekir ki yazarların bu çalışmadaki amaçları sadece SystemC simülasyonunu hızlandırmaktır. [7]

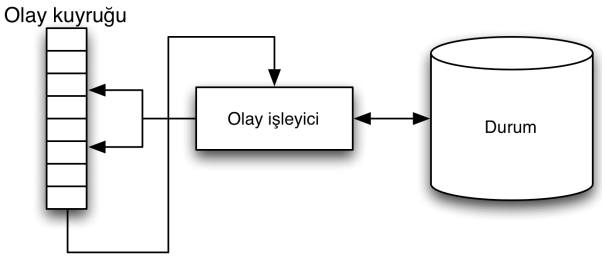


Şekil 5: QEMU ve SystemC'nin entegrasyonu [8].

Kanal seviyesinde etkileşim üzerine diğer bir örnek [8]'dir. Ancak, bu çalışma sanal altsistemleri gerçek altsistemlerle birleştirmek yerine, farklı sanal ortamlardaki sanal altsistemleri birleştirmektedir: SystemC ve QEMU emülasyon ortamı. İletiş im QEMU içine gömülmüş ve bir SystemC kanalıyla bağlantılı özel bir SystemC modülü aracılığıyla sağlanmaktadır (*Şekil 5*). Bu çalışmada ayrıca çeşitli soyutlama düzeyleri de değerlendirilmiştir.

Referans [9] Linux platformlarının gerekirciliğini arttırmayı amaçlayan bir çalışmadır. [5]'in aksine, [9] yeni bir işletim sistemi eklemeden bizzat Linux çekirdeğinin gerekirciliğini arttırmaktadır. Linux'un hali hazırda bir gerçek zamanlı zamanlayıcısı mevcuttur, dolayısıyla iş asıl olarak çekirdeğin kesilebilirliğini arttırmaktan ibarettir. Bu amaçla, döner kilitler kesilebilir hale getirilmiş, kesme hizmet yordamları çekirdek ipliklerine çevrilmiştir. Ayrıca öncelik terslenmesini önlemek üzere öncelik kalıtım protokolü semaforlar ve döner kilitler için gerçeklenmiş ve klasik Linux zamanlayıcıları yüksek çözünürlüklü yenileriyle değiştirilmişlerdir.[3]

4. Ön Bilgiler



Şekil 6: Bir ayrık olay simülatörünün temel elemanları.

SystemC simülasyon çekirdeği bir ayrık olay simülatörüdür, bir başka deyişle olaya dayalı simülasyon çekirdeğidir. *Şekil 6* bir ayrık olay simülatörünün temel elemanlarını göstermektedir. Simülasyon saati dahil simülasyonun tüm değişkenleri "durum" olarak gösterilebilir. Olay işleyici bir anda bir olayı işler, durum değişkenlerini uygun olarak günceller ve gerekirse olay kuyruğuna yeni olaylar ekler. Bir ayrık olay simülatörünün en tipik elemanı olay kuyruğudur. Tüm olaylar bu kuyrukta zamanlarının sırasına göre tutulur. Olay işleyici olayları bu kuyruğun başından okur ve yeni olayları zamanlarına göre bu kuyruğun çeşitli yerlerine ekler.

S*imülasyon saati* mevcut simülasyon saatinden daha yeni bir zamana sahip bir olay geldiği zaman ilerletilir. Bu nedenle simülasyon ayrıktır: zaman sürekli olarak modellenmez, sadece zamanda bir olayın mevcut olduğu noktalar modellenir. Gerçek zamanlı simülasyon; simülasyon saati T_S ile *duvar saati* T_D (yani gerçek zaman saati) arasında bir ilişki kurularak gerçeklenir. $T_{Sbaş}$ ve $T_{Dbaş}$ sırasıyla simülasyon saati ve duvar saatinin başlangıç değerleriyse, aralarındaki ilişki (1)'de gösterildiği ş ekilde verilebilir. Burada *ölçek* gerçek zaman simülasyonunun hızının duvar saatine göre ölçeklenmesi için kullanılabilir. [10] Çalışmamızda, ölçek daima 1 olacaktır.

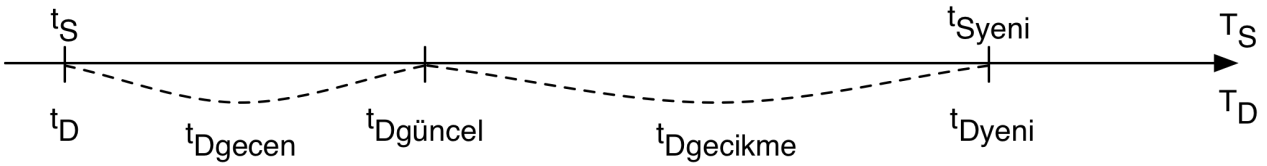
$$T_S = T_{Sbas} + \ddot{o}l\c{c}ek.(T_D - T_{Dbas}) \quad (1)$$

SystemC çekirdeğinin çalışması dört temel bölümden oluşmaktadır: başlangıç, işlem, güncelleme ve zamanı ilerletme (*Şekil 8*). İş lem ve güncelleme birlikte *delta çevrimi*ni oluşturur. Delta çevrimi zamanın çok ufak, delta miktarda ilerlediğinin varsayıldığı bir çevrimdir. Modül işlemleri işlem adımında yapılır, ancak sonuç değerleri çıkış kanallarına hemen yazılmaz, geçici bir tampon bellekte tutulur ve kanalın *request_update* (güncelleme iste) metodu çağrılır. Bu metot kanalı güncelleme listesine ekler. Güncelleme adımında güncelleme listesindeki kanalların *update* (güncelle) metodu çağrılır. Bu yaklaşım eş zamanlı

[1] Gerçek Zamanlı Uygulama Arayüzü (*İng. Real Time Application Interface, RTAI*)

[2] İşletim Sistemleri için İ ntibaklı Etki Alanı Ortamı (*İng. Adaptive Domain Environment for Operating Systems, ADEOS*)

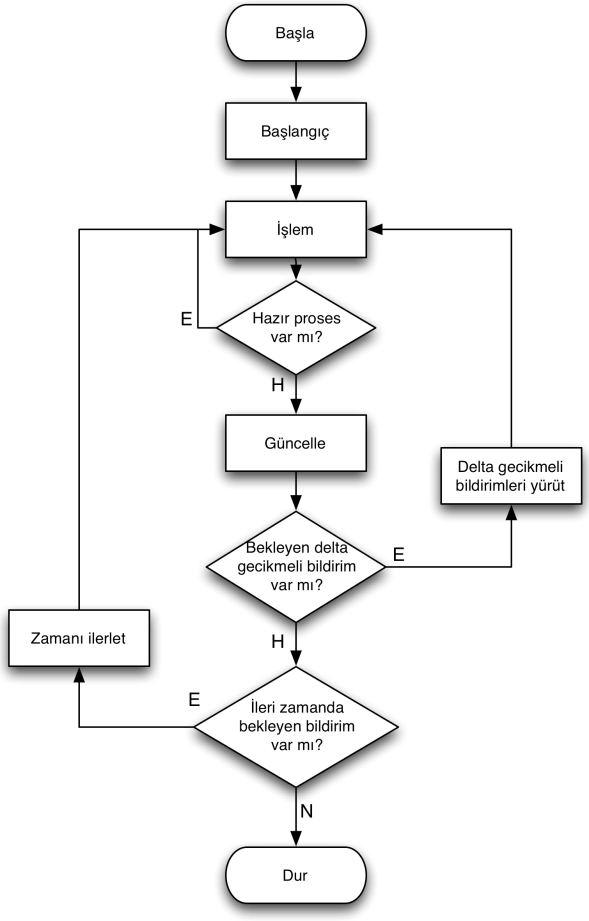
[3] Paragrafta kullanılan önemli terimler: zamanlayıcı (*İng. scheduler*), kesilebilirlik (*İng. preemptibility*), döner kilit (*İng. spinlock*), öncelik terslenmesi (*İng. priority inversion*), öncelik kalıtım (*İng. priority inheritance*)



Şekil 7: Simülasyon çekirdeği gerçek zaman yaması.

işlemlerin tek bir işletim ipliğinde modellenmesine imkan verir. [2] Ancak, bu aynı zamanda gerçek zamanlı simülasyon için bir zorluk oluşturur, zira gerçek zamanda sıfır zaman ilerlemesi diye bir şey yoktur.

Bir işletim sisteminin gerekirciliği temel olarak kritik işlemlerin son zamanlarından önce bitebilmesini garanti edebilmek için işlemlerin çalışma süresinin tahmin edilebilmesidir. Pratikte uygun bir gerçek zamanlı zamanlayıcı kullanılması ve zamanlayıcının müdahale edemediği kesilemez bölümlerin asgariye indirilmesi bu hedefe ulaşılmasını sağlar.



Şekil 8: SystemC simülasyonunun çalışması.

Adil zamanlayıcıların aksine gerçek zamanlı zamanlayıcılar görevler arasında adil dağıtımı veya sistemin çıktısının azamiye çıkarılmasını amaçlamazlar. Onun yerine basit bir görev önceliklendirilmesi modeli oluşturulur ve daha yüksek öncelikli olarak belirtilmiş görevlerin her zaman çalışma için öncelik hakkı olması garanti edilir. Günümüzde işletim sistemleri adil zamanlayıcıların yanı sıra gerçek zamanlı zamanlayıcı da sunmaktadır. Bu yapıda, gerçek zamanlı görevler, gerçek zamanlı olmayan görevlerin tümünden daha yüksek önceliğe sahiptir.

Gecikme, bir sistemin bir etkiye karşılık vereceği tepkinin süresi olarak tanımlanabilir. Gerçek zamanlı zamanlayıcılar ilgili görevin uygun önceliğe sahip olması durumunda çalışmasını garanti ederler. Ancak, kesmeleri kapatan kesme hizmet yordamları gibi kesilemeyen kod bölümleri zamanlayıcıların çalışmasını engeller. Dolayısıyla sistemin gecikmesini arttırırlar. Çözüm böyle bölümlerin asgariye indirilmesi ve daha küçük parçalara bölünmesidir. [9]'da kullanılan yaklaşım da budur.

5. Çözüm

Çözümümüz tanımlanan problemin dört boyutuna hitap etmektedir:

- Simülasyon çekirdeğinin yamalanarak gerçek zamanlı simülasyonun sağlanması
- Gerçek donanımla etkileşim için bir melez kanalın geliştirilmesi
- Çıktı zamanlamasının ayarlanarak delta çevrimlerinin yönetilmesi
- Platformun gerekirciliğinin iyileştirilerek simülasyonun gerçek zamandan sapmasının asgariye indirilmesi

Gerçek zaman simülasyon çekirdeği yaması simülasyon zamanının ilerletildiği noktaya bir zaman gecikmesi ekleyerek gerçeklenmiştir (*Şekil 7*). Simülasyon çekirdeği simülasyon zamanı t_S'i daha yeni bir zaman t_{Syeni}'ye sahip yeni bir olay gelmedikçe ilerletmez. t_{Syeni}'nin olay kuyruğundan geldiği anda halihazırda bir miktar zaman ($t_{Dgeçen}$) işlem ve güncelleme adımları için geçmiş durumdadır. Simülasyon bu noktada $t_{Dgecikme}$ kadar geciktirilirse t_{Dyeni} t_{Syeni}'ye denk gelir. Bu hesaplama için t_D ve $t_{Dgüncel}$ işletim sistemi çağrıları vasıtasıyla kaydedilir ve gecikme de yine bir işletim sistemi çağrısıyla sağlanır.

Çalışmamızda SystemC'nin kanal kavramı gerçek ve sanal altsistemler arasındaki iletişimi sağlamak üzere kullanılmıştır. Bu entegrasyon için bir modülün kullanıldığı [8]'den farklı bir yöntemdir. Anlayışımıza göre, ihtiyaç bir iletişimin gerçeklenmesidir, bu da SystemC'de kanalların görevidir.

Delta çevrimlerinin yönetilmesi ve çıktı zamanlamasının ayarlanması için simülasyon çekirdeğinin yöntemi izlenerek işaretin *güncellenmesinin istendiği* nokta yerine delta çevriminde çalışan ve işaretin model içinde *güncellendiği* noktada bu güncelleme donanıma aktarılmıştır.

Gerekirciliği sağlamak için gerçek zaman davranışı iyileştirilmiş bir işletim sistemi kullanılmıştır. MİB'e gerekirci erişim için simülasyon çekirdeği gerçek zaman zamanlayıcı ile çalışacak ş ekilde ayarlanmıştır. Belleğe gerekirci erişim için bellek yönetimi nedeniyle oluşacak

gecikmeler sayfaları önceden oluşturarak ve takas belleğe taşınmalarını engelleyerek önlenmiştir.

5.1. Çalışma hızını belirleyen etkenler

Çoğu simülasyon modeli salt simülasyon saatine bağlı çalıştığından olabildiği kadar hızlı çalışması yeterlidir, ancak çalışmamızda geliştirilen sanal sistem iletişim halinde olduğu gerçek sistemin hızını yakalayamazsa anlamlı bir sonuç alınamayacağından hızı belirleyen etkenlerin ayrıntılı bir analizi gereklidir. Çalışmamızda bu hızı belirleyen etkenlerle iki yönden ilgileneceğiz: (1) simülasyon saati ile duvar saati arasındaki ilişkinin korunması, (2) ardışık olarak gerçeklenen koşutzamanlı çıktıların gözleyen açısından koşutzamanlı olarak algılanması.

Şekil 7 incelendiğinde sistemde negatif bir gecikme gerçeklenemeyeceğinden anlamlı bir çalışma için $t_{Dgecikme}$'nin negatif olmaması gerektiği ortaya çıkar. Buradan da (2) eşitsizliği türetilebilir. Bu eşitsizliğin doğru olmadığı noktada sistem gereken hızı gösterememeye başlar.

$$t_{Dgecen} \le t_{Dyeni} - t_D = t_{Syeni} - t_S \quad (2)$$

t_{Dgecen}, t_S simülasyon anında çalışan tüm işlem (t_i) ve güncelleme/delta (t_g) çevrimlerinin (*Şekil 8*) sürelerinin toplamıdır. Bu değerin her t_S anı için tek tek hesaplanması oldukça zordur, ancak bir üst sınır bulmak daha kolay olabilir. Bunun için bir t_S anında çalışan işlem çevrimi (n_i) ve güncelleme çevrimi sayılarının (n_g) azami değeri ile bir işlem çevrimi (t_i) ve bir güncelleme çevriminin (t_g) azami sürelerinin bulunması yeterlidir. Buradan (3) ifadesine gidilebilir.

$$t_{Dgecen} = \sum t_i + \sum t_g \le \max n_i.\max t_i + \max n_g.\max t_g \quad (3)$$

n_i ve n_g'nin değerleri modelden analitik olarak (bir olayın veya zaman noktasının tetiklediği işlem ve güncelleme çevrimleri ve çalışan bu çevrimlerin aynı t_S anında çalışmak üzere dolaylı olarak tetiklediği tüm çevrimler) bulunabilir veya bu değerler çalışan bir sistemden ampirik olarak edinilebilir.

t_i ve t_g değerleri, modelden ve içinde bulunduğu ortamdan (işletim sistemi, g/ç sürücüsü ve donanımı, işletim sistemi) etkilenir. Azami t_i değeri modeldeki en uzun çalışma dizisi incelenerek bulunabilir, ancak buna işletim sisteminden kaynaklanan gecikmeleri eklemek gerekir. Azami t_g değeri ise – g/ç işlemleri delta çevrimlerinde yapıldığından ve modelin dahili güncelleme işlemine göre çok daha fazla zaman aldıklarından – modeldeki en karmaşık g/ç işlemi incelenerek bulunabilir. Dolayısıyla t_g, t_i'yi etkileyen etkenlere ek olarak g/ç donanımı, sürücüsü ve g/ç'nin yapıldığı ortamdan (veri yolu vs.) etkilenir.

(2) eşitsizliğinin diğer tarafı ($t_{Syeni} - t_S$) doğrudan modelden çıkarılabilir. Bu değer model içindeki zaman ilerlemeleri tarafından belirlenir. Model içindeki asgari zaman ilerlemesi belirlendiğinde bu değerin alt sınırı belirlenmiş olur.

Böylece (2) eşitsizliğinin taraflarının üst ve alt sınırları belirlenerek (4) eşitsizliği oluşturulur ve hesaplaması daha kolay olan bu eşitsizlik (2)'nin bir yaklaşımı olarak kullanılabilir.

$$\max n_i.\max t_i + \max n_g.\max t_g \le \min(t_{Syeni} - t_S) \quad (4)$$

Çalışma hızı üzerindeki diğer kısıtlama koşutzamanlı çıktıların – koşutzamanlı çalışma delta çevrimlerince modellendiğinden – aslında koşutzamanlı değil ardışık olarak yapılmasından kaynaklanır. Ardışık çıktılar arasındaki süre sanal sistemin iletişim halinde olduğu gerçek sistemin çıktıları koşutzamanlı olarak değil, ayrı zamanlardaki çıktılar olarak algılamaya başladığı noktaya çıktığında sistemin çalışması yine gerekli hızın altında kalmaya başlar.

Analize varılmak istenen hedeften başlamak gerekirse, sanal sistemin iletişimde olduğu gerçek sistemin n adet çıktıyı koşutzamanlı algılayabilmesi için bu n çıktının belli bir t_p zaman penceresi içinde gerçekleşmesi gerekir. t_p değeri kullanılan gerçek sistem tarafından belirlenir, sözgelimi periyodik veri transferi yapan bir sistemde t_p bir periyot olabilir.

Mevcut çözümümüzde çıktılar delta çevrimleri içinde yapıldığından aslında tüm çıktı zamanları t_{Dgecen} içinde dağılırlar. Dolayısıyla söz konusu kısıt (5) eşitsizliğiyle gösterilebilir. Ancak (5) eşitsizliğinin hesaplanması zor olduğundan (3)'ten yardım alarak (5) için (6) yaklaşımı geliştirilebilir.

$$t_{Dgecen} < t_p \quad (5)$$

$$\max n_i.\max t_i + \max n_g.\max t_g < t_p \quad (6)$$

6. Deney

Önceki bölümlerde aktarılan ve geliştirilen kavramsal yapıların gerçekleşebilir olduğunu sınamak üzere bir dürtü-genişliğinde-kipleme[1] motor sürme düzeneğinin modellemesi yapılmıştır. Deney basit olsa bile kavramların doğruluğunu göstermek açısından yeterlidir.

6.1. Gerçekleme ayrıntıları

Deneyi yapabilmek için çözümü gerçeklemek üzere SystemC simülasyon çekirdeği Linux işletim sistemi üzerinde çalıştırılmıştır.

Simülasyon çekirdeği gerçek zaman yaması, SystemC çekirdeği içinde *Şekil 8*'de gösterilen akışı gerçekleyen *sc_simcontext::simulate* fonksiyonunda, "Zamanı ilerlet" adımında yapılmıştır.

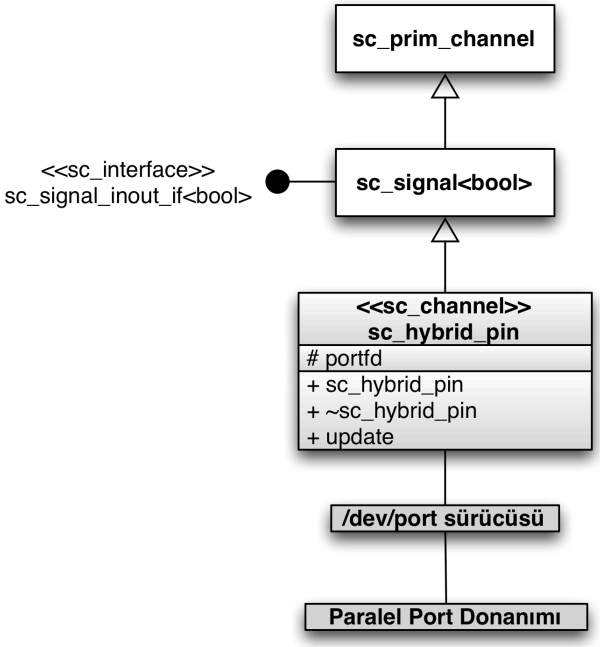
Şekil 9, önerdiğimiz melez kanal kavramının bir örneğini göstermektedir. *sc_hybrid_pin* bir tarafında bir SystemC modülünün portuna bağlanabilecek sc_signal_inout_if arayüzüne sahiptir ve diğer tarafında işletim sisteminin G/Ç sürücüsüne bağlıdır. Bu sürücü üzerinden G/Ç işlemleri yürütülür.

Delta çevrimlerinin yönetimi için SystemC'nin *sc_signal* sınıfı yeniden kullanılmıştır. sc_signal zaten request_update ve update metotlarının boş hallerine sahip sc_prim_channel sınıfından türetilmiştir. sc_signal bu metotları kendi amaçları için tümüyle gerçeklemektedir, dolayısıyla SystemC bakış açısından tüm gereksinimleri karşılamaktadır. Tek ek gereksinim gerçek donanıma çıktının yapılmasıdır. Çıkışın nihai değeri güncelleme adımından önce hazır olmadığı için, donanıma çıktının update metodunda yapılması yeterlidir. Bunun için sc_signal sınıfının update metodu türetilmiştir, bkz. *Şekil 9*.

Gerekirciliği sağlamak için Linux çekirdeğinin gerçek zaman davranışı RT_PREEMPT yamasıyla iyileştirilmiştir. SystemC ipliği gerçek zaman zamanlamaya ayarlanmış ve önceliği Linux çekirdek kesme hizmet ipliklerinin hemen altına alınmıştır. Takas bellekli bir bilgisayar kullanıldığından SystemC sürecine ait tüm sayfalar bellekte kilitlenerek sayfa hataları sebebiyle oluşacak gecikmelerin önüne geçilmiştir. Son olarak, iplik yığını simülasyon başlatılmadan önce çalışmada olabilecek azami noktasının ötesine kadar

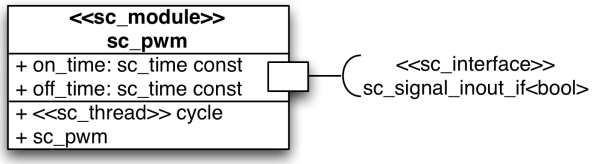
[1] *İng. Pulse Width Modulation, PWM*

büyütülerek çalışma esnasında yığın büyütülmesi sırasında oluşacak gecikmelerin önüne geçilmiştir.



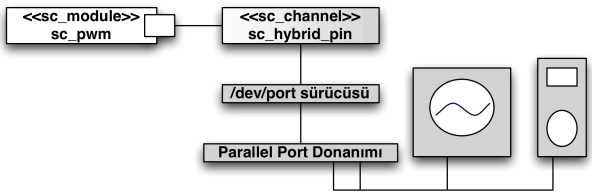
Şekil 9: Melez Kanal.

6.2. Deney düzeneği



Şekil 10: sc_pwm

Bu düzenekte bir dürtü-genişliği-kipleme modeli *sc_pwm*, işareti bilgisayarın paralel portundan dışarı aktaran bir melez kanal *sc_hybrid_pin*'e bağlıdır. Bu gerçek işaret bir osiloskop ve bir gerilim/akım/frekans-ölçer yardımıyla gözlenir. (*Şekil 10* ve *Şekil 11*)



Şekil 11: Deney düzeneği

Modelleme platformunun unsurları aşağıda listelenmiştir:

- Donanım platformu
  - Intel P4 3.2 GHz HT
  - 1 GB RAM
  - Standart paralel port
- Yazılım platformu
  - Linux 2.6.29.2-rt11
    - Simetrik çoklu işleme
    - Gerçek zaman kesilebilme (RT_PREEMPT)
  - openSUSE üzerinde KDE
  - SystemC 2.2.0
    - Gerçek zaman yamamızla birlikte
  - Eclipse CDT
  - İşlemci yükü oluşturma: basit bir kabuk betiği vasıtasıyla (sonsuz döngüde dön)

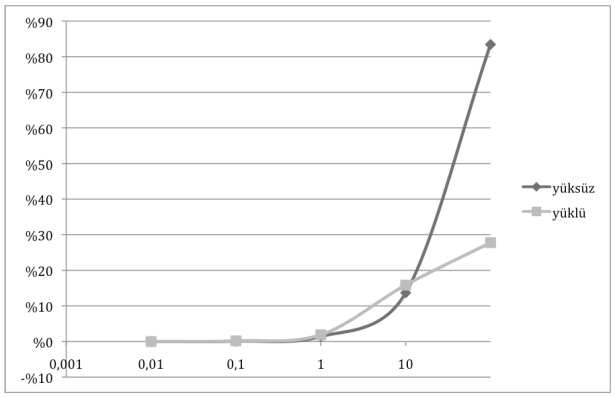
Tablo 1: Deney parametreleri

Parametre adı	Değerleri
Yük	Yok, var (%100 MİB kullanımı)
Dürtü oranı	%10, %50
İstenen dürtü frekansı	10 Hz, 100 Hz, 1 KHz, 10 KHz, 100 KHz

Deney parametreleri Tablo 1'de verilmiştir. Değerlendirme kriterleri gözlenen frekansın istenen frekanstan ortalama ve azami sapma oranları ve işaretin oynaklığıdır (*İng.* jitter).

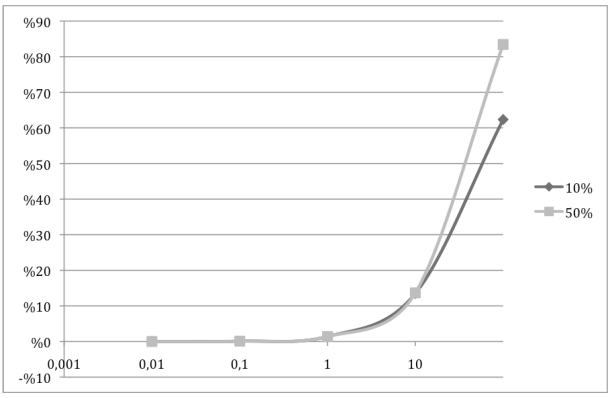
6.3. Deney Sonuçları, Değerlendirme

Şekil 12 ve *Şekil 13* 10 KHz istenen frekans değerinde dürtü-genişliğinde-kipleyicinin halen sınırlar dahilinde çalıştığını göstermektedir. Bu değerde sapma yaklaşık %17'ye kadar çıksa da sabit kalmaktadır. Ne var ki 100 KHz değerinde sistem herhangi kararlı bir frekans üretmemektedir. Dolayısıyla *Şekil 12*'deki bu değerde sisteme yük eklemenin daha iyi sonuç ürettiği gözlemi de yok sayılmalıdır.



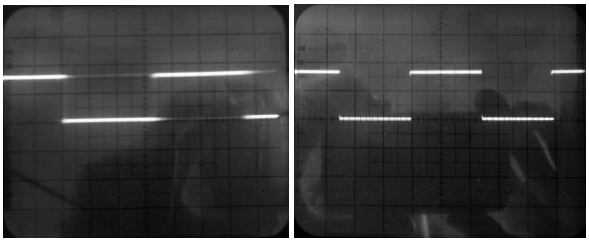
Şekil 12: İstenen frekanstan [KHz] ortalama sapma (%50 dürtü oranı)

100 KHz haricinde yük performansta sadece ufak bir kötüleşmeye yol açmıştır. MİB'in tümüyle kullanıldığı düşünülürse Linux zamanlayıcısının beklendiği gibi çalıştığı sonucuna varılabilir.

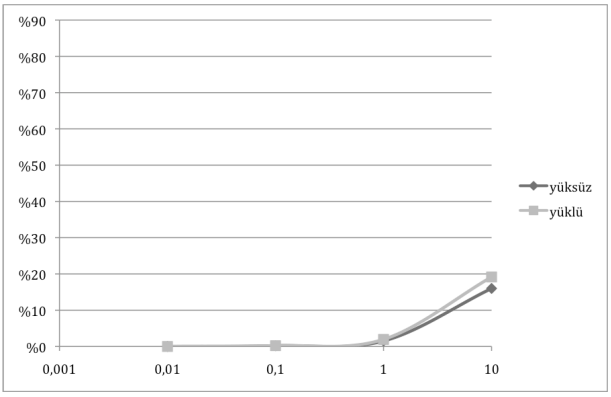


Şekil 13: İstenen frekanstan [KHz] ortalama sapma (yüksüz)

100 KHz değerindeki kararsızlığın nedeni yüksek oynaklıktır. Bu 100 KHz ve 1 KHz değerlerindeki dalga şekillerinin verildiği *Şekil 14*'te de görülebilir. Bu nedenle 100 KHz değerinde veri toplama bu noktada durdurulmuştur.



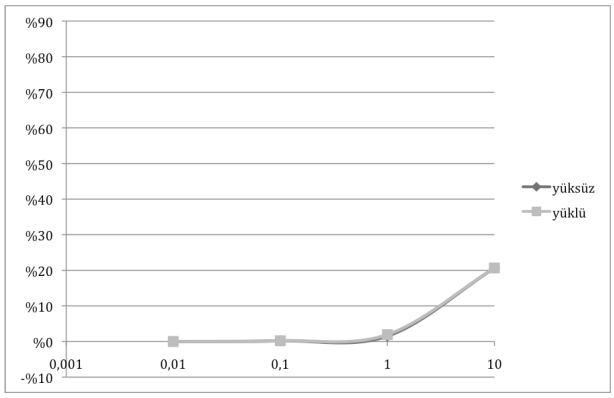
Şekil 14: Dalga şekillerinde görülen oynaklık (sol: 100 KHz, sağ: 10 KHz)



Şekil 15: İstenen frekanstan [KHz] azami sapma (%50 dürtü oranı)

Sonuçlar aynı zamanda azami sapmaların ortalama sapmalardan büyük oranda fark etmediğini (azami %5, 10 KHz'de) göstermektedir (*Şekil 15* ve *Şekil 16*'yı *Şekil 12* ve *Şekil 13* ile karşılaştırın). Bu platformun gerekirciliği yönünde olumlu bir gözlemdir.

Genel olarak dürtü oranının sistem üzerinde bir etkisi olmadığı gözlenmiştir. Ancak *Şekil 15* ve *Şekil 16*'nın karşılaştırılması yük ve dürtü oranının beraberce bir etki oluşturduğunu göstermektedir.



Şekil 16: İstenen frekanstan [KHz] azami sapma (%10 dürtü oranı)

7. Sonuç, Son Söz

Bu çalışmada SystemC ile döngü içinde donanım kavramını değerlendirmek üzere bir deney geliştirilmiştir. Ayrıca platform gerekirciliğiyle ilgili bir dizi iyileştirme yapılmış ve yapılan iyileştirmeler bir deneyde değerlendirilmiştir.

Kavramsal düzeyde herhangi bir sorunla karşılaşılmamıştır. SystemC tarafındaki geliştirmede bir değişiklik yoktur ve doğal yolunu izlemektedir. Ve G/Ç işlemlerinin gerçeklenmesi bir SystemC kanalı içinde toplanmıştır, bu da geliştirici için yine rahat bir ortam oluşturmaktadır. Ayrıca SystemC model geliştirme ile melez kanalın geliştirilmesi arasındaki işbölümü SystemC portları ve arayüzleri vasıtasıyla net bir biçimde tanımlanmıştır. Dolayısıyla yaklaşımımız aynı zamanda takım çalışmasına da uygundur.

Gerekircilik kapsamında yapılan iyileştirmelerle sistemin bir haneli milisaniye seviyesi dahil kabul edilebilir performans gösterdiği ve üç haneli mikrosaniye seviyesinde halen kontrol altında bir sonuç ürettiği tespit edilmiştir. Bu sonuçlar 100 mikrosaniye ve üzeri çevrim zamanlarının kullanıldığı endüstriyel uygulamalar için umut vaat etmektedir, üç haneli mikrosaniye mertebesi çalışmanın bu aşamasında yüksek oynaklık nedeniyle doğrudan ulaşılabilir konumda değildir, ancak iyileştirmeler yoluyla bu oynaklık azaltılabilir.

Ancak bu, aynı zamanda SystemC çekirdeğinin tasarımıyla da ilgilidir, zira SystemC çekirdeğinin tasarım hedefleri arasında gerekirci bir çekirdek geliştirme yoktur. Sistemde oluşturulan diğer yükler performansta asgari düzeyde bir düşüşe neden olmuştur. Bu problemin nedeni toplam gereken hesaplama gücüyle elde bulunan hesaplama gücü arasındaki farktan da kaynaklanabilir. Bu noktada performans sınırlarıyla ilgili 5.1 bölümünde geliştirdiğimiz matematiksel modeli bir deneye uygulayarak bu soruyu yanıtlamak yapmayı planladığımız çalışmalar arasındadır.

Yine gelecekte yapmayı planladığımız çalışmalar arasında sonuçların daha detaylı analizi ve iyileştirilmesi vardır. Bu başlık altında ayrıca daha gerekirci çalışma için önerilen Linux fonksiyonlarının kullanılması ve Linux için geliştirilmiş diğer gerçek zaman iyileştirmelerinin (örneğin RTAI) değerlendirilmesi de vardır.

İlgilenmeyi planladığımız bir diğer açık nokta çıktı zamanlamasının daha iyi yapılmasıdır. Halihazırda çıktı her güncelleme adımında yapılmaktayken bunun zamanın

ilerletildiği ana kadar bekletilmesi performans bakımından daha iyi sonuçlar verebilir; zira bu, çıktıları biraraya toplayarak hem toplam harcanan zamanı azaltabilir hem de koşutzamanlı çıktıların ardışık olarak yapıldığı zaman penceresini küçültüp modelden büyük oranda bağımsızlaştırarak bu alanda iyileşme sağlayabilir.

Yine gelecek çalışmalarımız arasında Ethernet gibi daha karmaşık melez kanalların gerçeklenmesi de var. Bu hem yeni sistemlerin oluşturulması için imkanları arttıracak hem de bacak seviyesi yerine kanal seviyesi modellerin etkisinin değerlendirilmesi imkanını verecektir.

Son olarak, yaklaşımımızın ölçeklenebilirliği, yani modeller üzerindeki boyut kısıtının ne olacağı, yine araştırılması gereken konular arasındadır.

8. Teşekkür

Siemens IT Çözümleri ve Hizmetleri SDE IEH Türkiye bölümüne destekleri için teşekkür ederiz.

9. Kaynakça

[1] M., Bacic, "On hardware-in-the-loop simulation", *Proceedings of IEEE Conference on Decision and Control, and the European Control Conference*, cilt 44, s. 3194-3198, 2005.

[2] T. Grötker, S. Liao, G. Martin, ve S. Swan, *System Design with SystemC*, Boston/Dordrecht/London: Kluwer Academic Publishers, 2002.

[3] N. Kim, H. Choi, S. Lee, I-C. Park, ve C-M. Kyung, "Virtual Chip: Making Functional Models Work On Real Target Systems", *DAC 98*, s. 170-173, 1998.

[4] B. Lu, X. Wu, H. Figueroa, ve A. Monti, "A Low-Cost Real-Time Hardware-in-the-Loop Testing Approach of Power Electronics Controls", *IEEE Transactions on Industrial Electronics, vol. 54, no. 2*, s. 919-931, 2007.

[5] P. Mantegazza, E. L. Dozzio, ve S. Papacharalambous, "RTAI: Real Time Application Interface", *Linux Journal, vol. 2000, no. 72es*, April 2000.

[6] K. Yaghmour, "Adaptive Domain Environment for Operating Systems", *Opersys*, 2001.

[7] L. Benini, D. Bruni, N. Drago, F. Fummi, ve M. Poncino, "Virtual in-circuit emulation for timing accurate system prototyping", *15th Annual IEEE International ASIC-SOC Conference*, s. 49-53, 2002.

[8] M. Montón, A. Portero, M. Moreno, B. Martínez, ve J. Carrabina, "Mixed SW/SystemC SoC Emulation Framework", *IEEE International Symposium on Industrial Electronics, vol. 2007*, s. 2338-2341, 2007.

[9] Real-Time Linux Wiki (2009, 05, 06) [Online], Available: http://rt.wiki.kernel.org/index.php/Main_Page

[10] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*, New York: Wiley-Interscience, 2000.