Küçük Ölçekli Gömülü Sistemler İçin Bir Yazılım Geliştirme Sistemi

İsmail Savaşkan[1], *Feza Buzluca*[2]

[1]{isavaskan}@neta.com.tr

[2]Bilgisayar Mühendisliği Bölümü
İstanbul Teknik Üniversitesi, İstanbul.
{buzluca}@itu.edu.tr

Özetçe

Modern yazılım mühendisliği tekniklerinin "küçük ölçekli gömülü sistemler" (KÖGS) üzerinde uygulanması sık rastlanan bir çalışma şekli değildir. Bu durum, fiziksel kaynak kısıtlılığı, donanımlar arası bir standardın olmaması ve yazılım mühendisliği yaklaşımlarının KÖGS'e yönelik oluşturulmamasından ileri gelmektedir. Bu çalışmada ise yeni bir geliştirme süreci tasarlanarak bilinen yazılım mühendisliği teknikleri KÖGS'e uyarlanmıştır. Oluşturulan yazılım geliştirme sistemi ile özel kavramlar cinsinden modellenen sistem, sınırlandırılmış nesneye dayalı bir dille jenerik olarak gerçeklenmekte, daha sonra uygun araçlarla istenen donanıma yönelik derlenebilir kaynak kod elde edilmektedir. Böylece donanımlar arası taşınabilir, yazılım mühendisliği kriterlerini yerine getiren, aynı zamanda etkin kaynak kullanımlı ve KÖGS doğasına uygun yazılımlar geliştirilebilmektedir. Metinde sistemin kuramsal altyapısı ve pratikte nasıl gerçeklenebileceği üzerinde durulmuştur.

1. Giriş

"Küçük ölçekli gömülü sistem" tanımlaması genellikle 8 ya da 16 bit mikrodeneteleyici tabanlı, donanım karmaşıklığı düşük elektronik dizgeler için kullanılmaktadır.[1] Kullanım alanları incelendiğinde ise dış dünyayla etkileşimleri yoğun, reaktif sistemler olarak değerlendirilebilirler. KÖGS yazılımları da aynı bakışla işlem performansından çok çevre birimleri denetimine ağırlık verilerek "olaya dayalı" olarak tasarlanırlar.

Mevcut yöntemlerle bu tür sistemler için yazılım geliştirmede bir dizi sorunla karşılaşılır. Geliştirmenin donanım üreticileri tarafından sağlanan geliştirme ortamları üzerinde yapılmasından ötürü; tüm yazılım ekibinin öncelikle kullanılacak donanımın teknik özelliklerine ayrıntılarıyla hakim olmaları gereklidir. Geliştirilen yazılım platforma özgüdür ve hiçbir düzeyde başka bir platforma taşınamaz. Kaliteli bir yazılımda olması beklenen tekrar kullanabilirlik, okunabilirlik, hataların kolay bulunabilmesi gibi kriterleri sağlamak için gerekli araçlar ve materyal genellikle sağlanmamıştır. Eşzamanlı çalışmayı ve kaynak paylaşımını kotarabilmek oldukça güçtür. Basit işletim sistemleri veya geliştirme ortamlarıyla birlikte verilen SDK ve API gibi hazır yazılımlar belli çözümler getirse de modellemeden kodlama aşamasına tüm süreci kuşatan, standart bir geliştirme sistemi mevcut değildir.

Bu çalışmada bahsedilen problem irdelenmiş ve bir çözüm önerilmiştir. Temel yaklaşım, geliştirmenin kaynak kod seviyesinden bir üst katmanda, jenerik (donanımdan bağımsız) olarak, hem modern yazılım kriterlerini hem de gömülü bilgisayar sistemlerinin özel yönlerini dikkate alınarak yapılmasıdır. Geliştirilen yazılım istenen hedef donanım için derlenebilir kaynak koda dönüştürülebilecektir.

Bu çalışma İTÜ Bilgisayar Mühendisliğinde hazırlanan "Gömülü Sistemler İçin Bir Geliştirme Ortamı "*An Advanced Software Development System for Small Scale Embedded Systems"* (ADSES1) bitirme çalışmasına dayanmaktadır. Bitirme çalışmasında kuramsal yapının oluşturulmasının yanı sıra geliştirilen bir masaüstü uygulaması ile basit yazılımlar hazırlanmış ve bir mikrodenetçi üzerinde test edilmiştir. Bu konudaki çalışmalar sürdürülerek yazılımın özellikle kuramsal altyapısı iyileştirilmiş ve bu metnin kapsamını oluşturan ADSES 2.0 geliştirilmiştir.

Sonraki bölümlerde konuyla ilgili önceki çalışmalar ele alınmış, geliştirilen sistemin temel unsurları birer bölümle açıklanmış, ADSES2.0 ile örnek bir proje tasarımı yapılmış ve sonuçlar ortaya konulmuştur.

2. İlgili Çalışmalar

Gömülü sistemlere yönelik yazılım geliştirme konusunda literatürde birçok yayın bulunmaktadır. Ancak bu çalışmaların geneli karmaşık sistemlere yöneliktir ve uygulanmaları için bir işletim sistemine ihtiyaç duyulur. Gömülü sistem geliştiricilerine model üzerinde tasarım, ve benzetim olanağı veren, tasarlanan modelden Java kodu oluşturabilen Ptolemy projesi bu çalışmalara örnek verilebilir. [2]

Küçük ölçekli gömülü sistemlere yönelik öne çıkan çalışmalardan biri Processor Expert projesidir. Processor Expert bileşen tabanlı ve nesneye dayalı yazılımlar geliştirmeye olanak verir. Ancak geliştirilen yazılımlar donanım bağımlıdır ve yalnızca Freescale donanımları desteklenmektedir. [3]

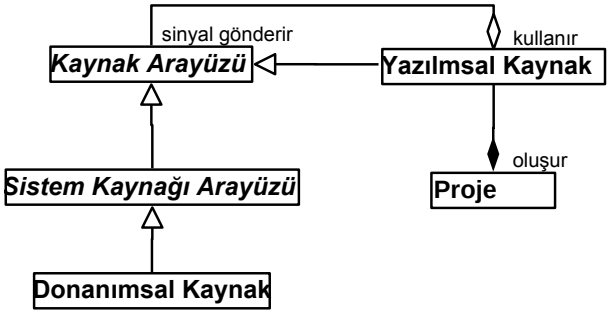
Mathworks firmasının Real-Time Workshop, Stateflow ve Simulink araçları ile model tabanlı mimari kullanarak platformlar arası taşınabilir, tekrar kullanılabilir yazılımlar geliştirilebilmekte, bilgisayar ortamında test ve benzetim

yapılabilmektedir.[4] Oldukça yüksek lisans ücretlerine sahip bu araçlar, genellikle belirli endüstrilerde işaret işleme projelerinde kullanılmaktadır.

3. Yazılım Modelleme Yöntemi

İyi bir modelleme ve tasarım için bazı kısıtlamalarla nesneye dayalı yaklaşım temel alınmıştır. Geliştirilecek yazılımların bileşenleri "kaynak" adı verilen aktif nesneler olacaktır. Genel amaçlı nesneye dayalı yazılımlardaki "içerme" ilişkisinin yerine (bu ilişki de semantik olarak bir nesnenin başka bir nesneye sahip olması üzerinde durulur: kitap ve yazarı gibi) "kullanma" ilişkisi konulmuştur (bu ilişki de ise semantik olarak bir nesnenin başka bir nesneyi kullanması vurgulanır: klima ve termometre gibi).

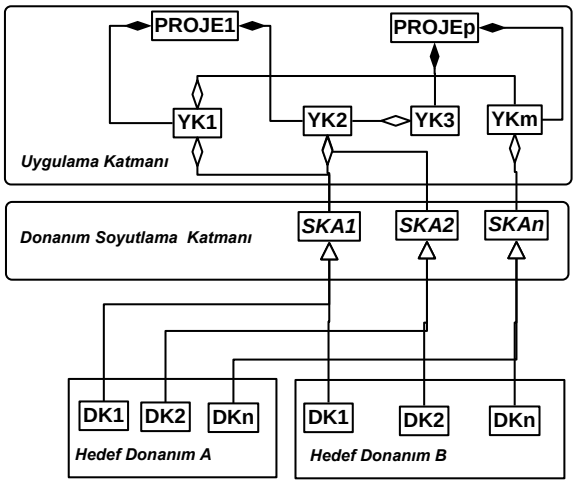
3.1. Modelleme Elemanları



Şekil 1: Modelleme elemanları arası ilişkiler.

Şekil 1'de modelleme elemanları arası ilişkiler UML diyagramı şeklinde gösterilmiştir. Kaynak arayüzü, dışarıdan çağrılan metotlardan ve kendisini kullanan kaynağa gönderilen sinyallerden oluşur. Yazılımsal kaynaklar, başka kaynakları kullanabilir, böylece sınırsız sayıda kaynak oluşturmak mümkün olur. Sistem kaynağı arayüzleri donanımsal modüllere (seri iletişim birimi, ADC gibi) karşı düşer. Bir donanımsal kaynak ise bir sistem kaynağı arayüzünün belli bir hedef donanım için gerçeklenmesidir. Yazılım kaynaklarının birleşiminden oluşan projeler çalıştırılabilir yazılım birimleridir.

Geliştirilen jenerik yazılımların belli bir donamınla kullanılması için sistem kaynaklarının arayüzü kullanılabilecek donanımların benzerliklerinden yararlanılarak standartlaştırılmalıdır. Bu kaynakların gerçeklenmesi ise kullanılacak her donanım için farklı olacaktır. Bu yapı işletim sistemi terminolojisindeki "donanım soyutlama katmanı"na karşı düşer.



Şekil 2: Yazılım Katmanları

Yazılımsal kaynaklar ve geliştirilen projeler uygulama katmanını oluşturur. Şekil 2'de yazılım/donanım katman yapısı temsili olarak gösterilmiştir. *n* adet sistem kaynağı arayüzü (SKA) donanım soyutlama katmanını oluşturmaktadır. A ve B hedef donanımları için donanım kaynakları (DK) ile sistem kaynakları gerçeklenmiştir. Uygulama katmanındaki ilişkiler ise örnek mahiyetindedir. Burada belirtilen yazılım kaynağı (YK) sayısı *m* ve proje sayısı *p*, gerçekte sonsuzdur.

3.2. Kaynaklar Arası Etkileşim

Bir kaynak nesnesine metotları üzerinden erişilebileceği gibi bu nesne de kendisini kullanan kaynağa sinyal gönderebilir. Bu etkileşim yöntemi "Qt C++ Application Framework" sinyal/slot mekanizmasının daha basit bir uyarlamasıdır. [5]

```
class Kaynak1{
public:
     Kaynak1();
     void method1(int m){
        //gerekli işlemler
        emit sthHappened;
     }
signals:
     void sthHappened();
 private:
     int m_value;
 };

class Kaynak2{
     Kaynak1 myKaynak1;
public slots:
     void handler1();//@{myKaynak1.sthHappened}
public:
     Kaynak2();
};
```

Yukarıdaki temsili bildirime göre, `myKaynak1` nesnesinin `sthHappened` sinyali yayınlandığında (`emit sthHappened` ile) bu nesneyi kullanan `Kaynak2` nesnesinin `handler1` metodu tetiklenir. Sinyal-slot bağlantısından dördüncü bölümde açıklandığı üzere çekirdek sorumludur.

3.3. Analiz ve Tasarım

Yeni bir projeye başlanıldığında genel amaçlı nesneye dayalı yazılımlarda olduğu gibi öncelikle istekler belirlenir ve problem domeninin analizi yapılır. Uygulamayı

(gerçeklenecek olan sistemi) oluşturan kavramlar, bunların özellikleri ve aralarındaki ilişkiler tespit edilir. [6]

Tasarım aşamasında ise analiz aşamasında belirlenen kavramların yazılım domeni karşılıkları bulunur. Problem domeni kavramlarına mümkünse eldeki yazılımsal kaynaklar karşı düşürülür. Aksi halde mevcut yazılımsal kaynaklar ve/veya sistem kaynaklarından yenilileri oluşturulur.

Tasarımda, bir yazılımsal kaynağa yalnızca onu kullanan nesne tarafından erişileceği göz önüne alınmalıdır. Yazılımsal kaynağın arayüzü (kamusal metotlar ve sinyaller) "bu kaynağı kullanan neleri değiştirmeye ve nelerden haberdar olmaya ihtiyaç duyar" sorusuna göre tasarlanmalıdır. Kaynakları hiyerarşik bir yapıda modelleyerek projeyi büyükten küçüğe parçalara ayırmak (*decompositon*) tasarımı kolaylaştıracaktır.

Tasarlanan yazılımsal kaynaklar ileriki projelerde kullanılmak üzere saklanabilirler. arayüzleri kesişen kaynaklar birbirlerin yerini alabilirler.

Bu şekilde hedef donanımdan bağımsız, tekrar kullanılabilen parçalardan oluşan projeler geliştirilebilir.

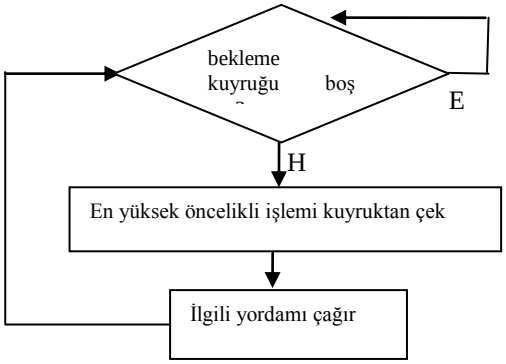
4. Çekirdek

ADSES2.0 ile geliştirilen yazılımlarda yazılım kaynakları arasındaki mesajlaşmayı yürütmek, kesmeleri kotarmak, açılış işlemlerini gerçekleştirmek için bir çekirdeğe (*kernel*) ihtiyaç duyulur.

4.1. Çekirdek Olay Döngüsü

Problemin doğası gereği yazılımın akışı, fonksiyon çağrıları ve koşullu dallanmalardan çok gelen asenkron sinyaller ile tetiklenen yordamların çalışması şeklinde olacaktır. Çekirdek bu iş için çok düzeyli FIFO (ilk giren ilk çıkar) yapısında bir kuyruk içerir.

Çekirdek içinde sinyallerle bunları kotaracak yordamların adreslerini tutan bir tablo yer alır. Sinyaller yazılım kaynaklarından yayınlanabileceği gibi kesme altyordamlarından da yayınlanabilirler. Bir olay döngüsü çekirdeğe gelen bir sinyalle başlar. Çekirdek sinyale karşı düşen yordamın adresini tablodan bulup varsa parametrelerle birlikte bekleme kuyruğuna ekler. Sırası geldiğinde yordam yürütülür ve kuyruktan çıkarılır. Böylece bir çevrim sona erer.



Şekil 3: Olay döngüsü akış şeması

4.2. İş Sıralama

Yukarıdaki akış şemasında görüldüğü üzere kesintisiz iş sıralama yöntemi tercih edilmiştir. Kesintili iş sıralama yöntemine göre çekirdeği basitleştirdiği ve işlem yükünü hafiflettiği için bu yöntem seçilmiştir.[7] Ayrıca semafor ve mutex gibi işletim sistemi yapılarını kullanmaya gerek kalmaz.

Kesintisiz iş sıralamanın en büyük riski çalışmaya başlayan bir yordamın çekirdek tarafından durdurulamamasıdır Ancak KÖGS'de işlemci yoğunluğunun düşük, G/Ç yoğunluğunun yüksek olduğu göz önüne alındığında tetiklenen bir yordamın çalışma zamanının kısa olacağı varsayılabilir. Bu tezi doğrulamak için donanım kaynaklarının tüm G/Ç işlemleri blokesiz (*non-blocking*) gerçeklenmelidir. Ayrıca, yoklama yöntemi yerine her zaman kesmeli çalışma tercih edilmelidir.

Eğer geliştirilecek sistemin işlemci yoğun görevleri varsa işlemler zamana yayılarak sorun giderilebilir. Bunun için görev alt yordamlara ayrılır, bu yordamlar doğrudan çağrılmadan sinyaller ile tetiklenir. Böylece daha yüksek öncelikli görevler araya girebilir.

5. Hedef Donanım Betimleme Yöntemi

Belirli bir donanıma özgü kaynak kod oluşturabilmek için hedef donanımı belli bir standarda göre betimleyen veriye sahip olunması gerekir.

Bir hedef donanım betimlemesi; sistem kaynaklarının gerçeklenmesi (donanımsal kaynaklar), donanımsal modül yapısı ve birbirleriyle ilişkileri ve kesme hizmet programı tanımlamalarından oluşur.

Geliştirilen yazılımın performansı yapılan jenerik tasarım kadar hedef donanım için yapılan gerçeklemelerin ve konfigürasyonların kalitesine de bağlı olacaktır.

5.1. Donanımsal Kaynak Gerçeklemesi

Donanım soyutlama katmanındaki sistem kaynağı arayüzlerinin her biri gerçeklenmelidir. Eğer bir sistem kaynağının işlevini doğrudan yerine getirecek donanım modülü yoksa diğer modüller tarafından işlev öykünür.

5.2. Donanım Modülleri Konfigürasyonu

Her bir sistem kaynağının hedef donanımın bir donanım modülüne karşı düşürülmesi gerekir. Bunun için hedef donanım konfigürasyonu hedef cihazın sahip olduğu donanım modüllerini ve bunların özellikleri içerir. Bir donanım modülü başka bir donanım modülünü gerektiriyorsa ya da bir modül birden fazla kaynak tarafından kullanılabiliyorsa bunlar konfigürasyonda belirtilir.

ADSES1'de donanım modülü konfigürasyonları fiziksel olarak, belli arayüzleri gerçekleyen sınıflardan oluşan Java kütüphaneleri şeklinde temsil edilmişlerdir. Konfigürasyonları XML dosyaları şeklinde ifade etmek de uygun bir yöntem olacaktır.

5.3. Kesme Konfigürasyonu

KÖGS'de kesme rutinleri farklılık göstermektedir. Bazı donanım üreticileri kesme vektörü kullanırken bazıları kesme kaynağını yoklama yöntemi ile tespit ederler. İlgili kesme bayrağı bazılarında yazılım tarafından temizlenmeli iken bazılarında bu işlem bazı saklayıcıların okunması ile yapılır. Bunun için donanım betimlemesine bir kesme rutini şablonu eklenmesi gerekir.

6. Kaynak Kod Üreteci

ADSES2 sisteminin çıktısı hedef donanım derleyicileri tarafından derlenebilen kaynak koddur. Kaynak kod üreteci aracı, bir projeyi çekirdek ve hedef donanım betimlemesi ile birleştirerek kaynak kod üretir.

6.1. Yazılım-Donanım Eşleşmesi

Projenin yazılımsal kaynakları teorik olarak sınırsızdır ancak bunları karşılayacak donanım modülleri ise kısıtlıdır. Ayrıca geliştirici bazı yazılımsal kaynaklara atanacak fiziksel donanım modüllerini kendi belirlemek isteyebilir. Bu sebeple, kaynak kod oluşturulurken projenin yazılımsal kaynaklarına donanım modülleri atanması için bir kaynak paylaştırma algoritması yürütülür. Algoritmanın temel prensibi bir istemci (yazılımsal kaynak ya da donanım modülü) için birden fazla modül uygun ise toplamda en az istenen modülün bu istemciye verilmesidir. Eğer kullanılan sistem kaynakları donanımsal kaynaklarla karşılanamıyorsa kod oluşturulamaz.

6.2. Programlama Dili Seçimi

Tasarlanan yazılım nesneye dayalı olduğu için C++ dilinde kod oluşturmak daha kolaydır. Bu şekilde proje geliştirme aşamasında yapılan kodlama büyük ölçüde son koda taşınabilir. Öte yandan C++ desteği veren mikrodenetçi sayısı artmaktadır. ADSES1 projesinde MC9S12C32 mikrodenetleyicisi için belirtilen kısıtlamalarla C++ dilinde kaynak kod oluşturulmuş, kod derlenmiş ve çalıştırılmıştır.

Ancak son kodu C dilinde oluşturmak hem taşınabilirlik hem de durağanlık açısından daha iyi sonuçlar verecektir. Bunun için nesneye dayalı yapıdan yordamsal yapıya kodu taşıyan araçlara ihtiyaç duyulur. Bu işlem genel olarak çok zor olsa da bu çalışmada nesneye dayalı programlamanın çok şekillilik (polymorphism), şablonlar (templates) gibi özellikleri kullanılmadığı için teorik olarak mümkündür.

7. Örnek Proje

Örnek proje olarak "derslikler için aydınlık kontrol sistemi" tasarlanmıştır.

7.1. İstekler ve Kullanım Senaryoları

Dersliklerde elektronik kontrol edilebilir aydınlatma elemanları olduğu varsayılmıştır. Sistem, derslikteki kişileri giriş ve çıkışlarda sayacak içeride kimse kalmadığında aydınlatmayı kapatacaktır. İçeride birileri varsa aydınlığı istenen düzeyde tutacaktır. Derslik içine doğal ışık da gireceğinden aydınlatma elemanlarının çalışma seviyesi aydınlık seviyesi ölçülerek hesaplanmalıdır. İstenen aydınlık seviyesinin ayarı, içerideki kişi sayısının sorgulanması gibi kullanıcı işlemleri I2C hattı üzerinden yapılan iletişimle gerçekleştirilecektir.

7.2. Analiz

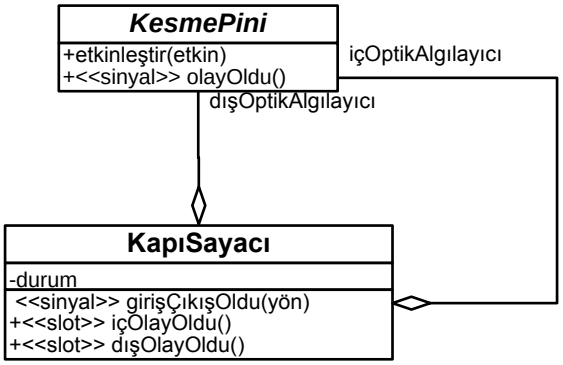
Problemin tanımından sistemin varlıklarının elemanlarının kapı sayacı, aydınlık ölçer, aydınlatma elemanı sürücü, kullanıcı arabirimi olduğu anlaşılmaktadır. Ayrıca bunları denetleyecek bir ana denetleyiciye ihtiyaç duyulur.

Şekil4:Problem domeni UML diyagramıı

7.3. Tasarım

Analiz bölümündeki her bir varlık yazılımsal kaynak olarak ifade edilmelidir. Bunun için gerekli sistem kaynakları kullanılmalı ve ilgili sinyallerle metotlar eşleştirilmelidir.

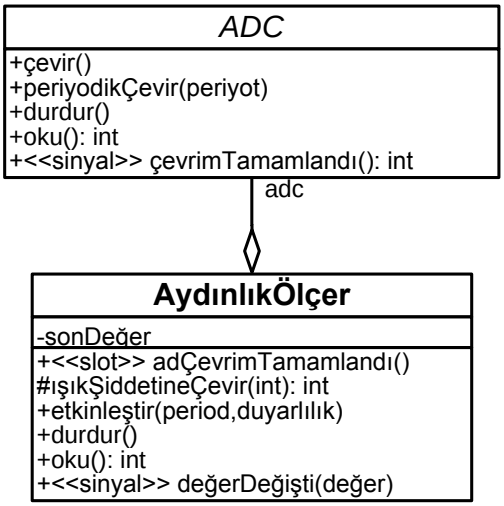
Kapıdaki hareketleri fark etmek ve hareket yönünü anlamak için kapılara bir bit lojik çıkışlı ikişer optik algılayıcı konulmuştur. Algılayıcı önüne bir engel geldiğinde çıkışı değişmektedir. Bu çıkışlar sistemin kesme girişlerine bağlanmalıdır.



Şekil 5:KapıSayacı UML diyagramı

KapıSayacı, KesmePini sistem kaynağını kullanarak optik algılayıcıdan gelen sinyalleri alır. Bir giriş/çıkış tespit ettiğinde ise bunu girişÇıkış oldu sinyali ile bunu bildirir.

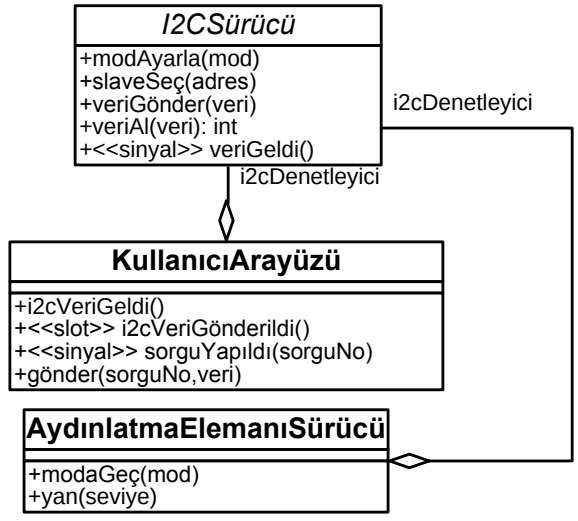
Aydınlık seviyesi bir fototransistorun iki ucu arasımdaki gerilimden ölçülecektir. Bunun için bir ADC kanalına ihtiyaç duyulur.



Şekil 8: AydınlıkÖlçer UML diyagramı

`AydınlıkÖlçer` yazılımsal kaynağı bir ADC sistem kaynağı kullanır. `AydınlıkÖlçer` etkinleştirdiğinde ADC'yi periyodik ölçüm yapacak şekilde koşullar. Ölçümün bittiğini `adÇevrimTamamlandı` metodunun tetiklenmesi ile anlar. Eğer son değerle yeni değer arasında anlamlı bir fark varsa `değerDeğişti` sinyalini yayınlar.

Kullanıcı işlemleri ve aydınlatma elemanıyla iletişim için I2C protokolü kullanılmıştır.



Şekil 7:KullancıArayüzü ve AydınlatmaElemanıSürücü UML diyagramı

Bu sebeple `KullancıArayüzü` ve `AydınlatmaElemanıSürücü` yazılımsal kaynakları `I2CSürücü` sistem kaynağını kullanır. `AydınlatmaElemanıSürücü` yalnızca verilen komutu I2C üzerinden aydınlatma elemanına iletir. `KullanıcıArayüzü` ise gelen veriyi ayrıştırıp istenen sorguyu `sorguYapıldı` sinyali ile iletir. `gönder` metodu ile de istenen veriyi I2C üzerinden iletir.

Şekil 8: AnaDenetleyici UML Diyagramı

`AnaDenetleyici` yazılımsal kaynağı bir `KapaSayacı` dizisi, birer `AydınlıkÖlçer`, `AydınlatmaElemanıSürücü` ve `KullanıcıArayüzü` kullanır. Bir giriş çıkış algılandığında kişi sayısını günceller, ilk kişi girişi ve son kişi çıkışında aydınlatma elemanın açar/kapatır. Aydınlık değeri değiştiğinde durumu değerlendirerek istenen seviyeye göre aydınlatıcıya yeni değerini bildirir.

Şekil 5'den 8'e kadar olan diyagramlarda sinyaller ve bunları kotaracak metotlar (slotlar) stereotip şeklinde gösterilmiştir.

7.4. Örnek Kodlama

Örnek olarak `KapıSayacı::içOlayOldu` ve `AnaDenetleyici::aydınlıkDeğişti` metodlarının kodlaması yapılmıştır.

```
void KapiSayaci::icOlayOdu(){
   //dış algılayıcıdan geçildiyse:
   if(durum == ICERI_GIRILIYOR){
      // g/ç tamamlandı:
      durum = IDLE;
      //içeri girildiğini bildir:
      emit girisCikisOldu(ICE);
   }
   //muhtemel dışarı çıkış:
   else if(durum == IDLE)
      durum = DISARI_CIKILIYOR;
}

void AnaDenetleyici::aydinlikDegisti
                      (int yeniAydinlik){
   //ölçülen aydınlık yeterli ise:
   if(yeniAydinlik >=
        istenenAydinlikSeviyesi)
      aydinlatici.son();
   //değilse yeteri kadar yak:
   else{
      int fark = istenenAydinlikSeviyesi -
                 yeniAydinlik;
      aydinlatici.yan(fark);
   }
}
```

8. Sonuçlar ve Yorumlar

Küçük ölçekli gömülü sistemler, ihtiyaç duyulmasına rağmen yazılım mühendisliği tekniklerinin fazlaca kullanılamadığı bir alandır. ADSES ile bu ihtiyacı karşılamak için jenerik ve donanıma özgü işlemleri ayıran bir geliştirme süreci ileri sürülmüş, hem modelleme hem de gerçekleme aşamaları için özel yöntemler ortaya konulmuştur. Bu ileri geliştirme sistemi KÖGS yazılımlarına donanımlar arası taşınabilirlik özelliği

katmakta ve geliştiricilere yazılım modüllerinin tekrar kullanılması, yazılımların daha anlaşılabilir olması, hataların daha kolay bulunması gibi bir dizi kolaylık sunmaktadır. İleriki çalışmalarda ADSES'in tam olarak standarda oturtulması ve gerekli geliştirme araçlarının sağlanmasıyla kapsamlı deneysel sonuçlar da elde edilebilir.

9. Kaynakça

[1] R. Kamal, *Embedded Systems : Architecture, Programming and Design,* McGraw-Hill, 2003

[2] H. Lig ve diğerleri, "Overview of The Ptolemy Project", Memorandum UCB/ERL, 2003, http://ptolemy.eecs.berkeley.edu/

[3] Processor Expert, http://www.processorexpert.com/

[4] R. Colgren, Basic MATLAB®, Simulink® and Stateflow®, AIAA, 2007

[5] J. Blanchette, M. Summerfield, *C++ GUI Programming with Qt4, 2nd Edition,* Prentice Hall, 2008

[6] F. Buzluca, "Yazılım Modelleme ve Tasarımı Ders Notları", 2010, http://www.buzluca.info/ymt

[7] M. Short, "The Case For Non-preemptive, Deadline-driven Scheduling In Real-time Embedded System", Proceedings of the World Congress on Engineering, London, UK, 2010, s:399