

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ
(YÜKSEK LİSANS TEZİ)

GÖMÜLÜ SİSTEMLERLE BİR KONTROL
SİSTEMİ TASARIMI

Tülin OKYAY

Uluslararası Bilgisayar Enstitüsü

Bilim Dalı Kodu : 619.02.04

Sunuş Tarihi: 25.08.06

Tez Danışmanı : Prof. Dr. Bahar Karaoğlu

Bornova-İZMİR

Sayın **Tülin Okyay** tarafından YÜKSEK LİSANS TEZİ olarak sunulan “**Gömülü Sistemlerle Bir Kontrol Sistemi Tasarımı**” başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi’nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 25.08.2006 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

-

Jüri Üyeleri:

İmza

Jüri Başkanı : Prof. Dr. Bahar Karaoğlu

Raportör Üye: Prof. Dr. Turhan Tunalı

Üye : Yrd. Doç. Dr. Osman Ünalır

ÖZET

GÖMÜLÜ SİSTEMLERLE BİR KONTROL SİSTEMİ TASARIMI

OKYAY, Tülin

Yüksek Lisans Tezi, Uluslararası Bilgisayar Anabilim Dalı

Tez Yöneticisi: Prof. Dr. Bahar KARAOĞLAN

Ağustos 2006, 103 sayfa

Günümüzde televizyonlardan cep telefonlarına kadar neredeyse bütün elektronik cihazlar gömülü sistemlerle kontrol edilmektedir. Gömülü sistemlerin bir çoğunun içinde binlerce ve hatta bazı durumlarda milyonlarca satırdan oluşan karmaşık bilgisayar programları çalışır. Ancak bu programların geliştirilmesi için kullanılan teknikler ve araçlar, genel amaçlı bilgisayarlar için kullanılanlara göre bazı noktalarda farklılıklar gösterir. Bu tezde, bu farklılıklar vurgulanarak, gömülü sistemler ile bir denetim mekanizması tasarlanmış ve yazılımı gerçekleştirilmiştir. Gerçeklenen tasarım, kalorifer sisteminin otomatik denetlenmesinde sınanmıştır. Tasarımda gerçekleştirilen yazılımın daha sonra farklı cihazlara da kolaylıkla uygulanabilmesi amacıyla, modüler, esnek ve anlaşılabilir olmasına dikkat edilmiştir.

Anahtar sözcükler: Gömülü sistemler, gömülü yazılım, 8051, ısıtıcı kontrolü.

ABSTRACT**A CONTROL SYSTEM DESIGN BY USING EMBEDDED SYSTEMS**

OKYAY, Tülin

MSc, International Computer Institute

Supervisor: Asoc. Prof. Dr. Bahar KARAOĞLAN

August 2006, 103 pages

In today's world, from TV sets to cellular phones, almost every electronic device is controlled by an embedded system. Most of embedded systems contain thousands or in some cases millions of lines of code. However the techniques used to develop those software, have differences from generic programming at some points. Emphasizing these differences, a control system is designed and implemented by using embedded systems. The implemented design is used in the control of a heating system. To be able to port the designed system to different platforms easily, modularity, readability and flexibility are considered in the development of the source code.

Keywords: Embedded systems, embedded software, 8051, heating control

TEŞEKKÜR

Bu çalışma süresince özellikle kıymetli görüşlerinden yararlandığım, tezin biçimlenmesinde değerli katkılarını aldığım ve yakın ilgisini esirgemeyen sayın Prof. Dr. Bahar Karaoğlu' a, tezin donanım geliştirme aşamasında büyük katkı sağlayan ve desteğini esirgemeyen eşim Mehmet Onur Okay'a teşekkürü bir borç bilirim.

İÇİNDEKİLER

Sayfa

ÖZET.....	V
ABSTRACT.....	VII
TEŞEKKÜR.....	IX
ŞEKİLLER DİZİNİ.....	XV
ÇİZELGELER DİZİNİ	XV
1. GİRİŞ	1
2. GÖMÜLÜ SİSTEMLER	3
2.1 Gömülü Sistem Örnekleri.....	3
2.2 Genel Sistem Parçaları	5
2.3 Mikrodenetleyici Kavramı	6
2.3.1 Giriş ve Çıkışlar	7
2.3.2 Bellek	7
2.3.3 Haberleşme.....	8
2.3.4 Mikrodenetleyici seçimi.....	8
2.4 8051 Mikrodenetleyici Ailesi	9
2.5 Gömülü Sistem Geliştirme Parametreleri	11
2.6 Gömülü Sistemlerde Yazılım Kavramları	12
2.6.1 Gerçek Zamanlılık.....	13
2.6.2 Çokluişlem	14
2.6.3 Çokluişlem ile programlama alışkanlıkları	18
2.6.4 Bellek içeriğini doğrulama.....	20
2.7 Gömülü Sistemler Üzerinde Yazılım Geliştirme Süreci.....	23
2.7.1 Derleme, Bağlama, ve Yerleştirme	25

İÇİNDEKİLER (devam)

	<u>Sayfa</u>
2.7.2 Yükleme ve hata ayıklama	31
2.7.3 PC’ de çalışmaktan farkları	37
3. ISITICI DENETİM SİSTEMİ (IDS)	39
3.1 Sistemin Özellikleri	39
3.2 Kullanıcı Arabirimi	41
3.3 IDS Donanım Modülleri	44
3.4 IDS Yazılım Ayrıntıları	53
3.4.1 Modüllerin kısa açıklamaları.....	54
3.5 Yazılım Ayrıntıları.....	56
3.5.1 Program akışı	56
3.5.2 İleti sistemi (Message system)	59
3.5.3 Durum makineleri	60
3.5.4 Olay işleyiciler (Event Handlers).....	66
3.5.5 Menü sistemi	66
3.5.6 Ekran durum makinesi	69
3.5.7 Ekran motoru (Display Engine)	71
3.4.7 Bir kablolu yol (1Wire Bus) ve sürücüsü.....	73
3.5.8 İki kablolu yol (2Wire Bus – i2C) ve sürücüsü.....	77
3.5.9 Sistemin başlatılması.....	80
3.5.10 Platform Yalıtım Katmanı - PYK	81
3.5.11 Hata denetimi	83
3.5.12 Yapılandırma ve Kalibrasyon	85
3.5.13 Hata ayıklama (Debugging)	88

İÇİNDEKİLER (devam)

	<u>Sayfa</u>
3.5.14 Uzaktan telefon bağlantısı.....	90
4. UYGULAMA SİMULATÖRÜ	92
4.1 Genel Yapı.....	93
4.2 İz Ekranı	95
5. SONUÇ	96
Pazardaki Çözümler	97
KAYNAKLAR DİZİNİ	101
ÖZGEÇMİŞ	103
EKLER.....	

ŞEKİLLER DİZİNİ

<u>ŞEKİL</u>	<u>Sayfa</u>
2.1 Genel gömülü sistem blokları	5
2.2 Genel mikrodenetleyici yapısı	6
2.3 Örnek bir 8051 iç yapısı	10
2.4. Sonsuz döngünün kesmelerle kullanımı	15
2.5 İşletim sistemi yalıtım katmanı kullanımı	17
2.6. Gömülü yazılım geliştirme süreci	24
2.7 Gömülü yazılım inşa süreci	26
2.8 Bir uzaktan hata ayıklama oturumu	33
2.9 En ideal ortam: bir genel hata ayıklayıcı arayüzü	36
3.1 Sistemin genel özellikleri	39
3.2. Sistem Kurulumu	41
3.3 Ana ekran görüntüsü	42
3.4 Örnek sıcaklık programlama tablosu	43
3.5 IDS donanım blokları	45
3.6 Ana kart ön yüz	46
3.7 Kontrol kartı	47
3.8 Anakart arka yüz	48
3.9 Ana ekran görüntüsü	49
3.10 IDS deprem sensörü	51
3.11 IDS bilgisayar bağlantısı	52
3.12 IDS yazılım blokları	54
3.13 Program akışı	58

ŞEKİLLER DİZİNİ (devam)

<u>ŞEKİL</u>	<u>Sayfa</u>
3.14 IDS Durum Makinası Manuel Konum Durumları	63
3.15 IDS durum makinesi	63
3.16 Sistem sıcaklık aralığı	64
3.17 İleti akışı ve olay işleyici	67
3.18 Menü ağaç yapısı	69
3.19 Ekran güncelleme senaryosu	71
3.20 Ekran tamponu sağlama toplamı	72
3.21 Bir kablolu yol sıfırlama sinyali	73
3.22 Bir kablolu yol veri sinyalleri	75
3.23 DS1820 sürücü yapısı	76
3.24 İki kablolu yol senkronizasyon sinyalleri	78
3.25 İki kablolu yol veri aktarımı	79
3.26 İki kablolu yol paket yapısı	79
3.27 EEPROM sürücü yapısı	80
3.28 PYK kullanıcı ayarları örneği	82
3.29 Eeprom veri yapısı	86
3.30 Uzaktan telefon bağlantısı	91
4.1 HCS Simulatörü	92
4.2 Simulatör ve cihaz farkı	94
4.3 İz ekranı	95
5.1 Dataterm IHC görüntüsü	98
5.2 Honeywell görüntüsü	99

ÇİZELGELER DİZİNİ

<u>ÇİZELGE</u>	<u>Sayfa</u>
2.1 Uluslararası Jeneratör Polinomları Standardı	22
2.2 PC ve Gömülü Sistem arasındaki farklar	38
3.1 IDS durum tablosu	65
5.1 Yazılım boyu	97
5.2 IDS ve pazardaki diğer çözümlerin karşılaştırma tablosu	100

1. GİRİŞ

Bu tezde gömülü sistemler ile bir denetim mekanizmasının gerçekleştirilmesi ve bu sistemin bir kalorifer sistemi üzerinde sınanması hedeflenmiştir.

Tasarlanan sistem mikrodenetleyici tabanlıdır ve standart bir kalorifer sisteminin tüm elemanlarını (kazanlar, su pompaları, yakıt ön ısıtıcısı vs.) kontrol eder. Sistem ekonomik, modüler, bakımı kolay, değişik platformlarda çalışmaya uygun, dış dünya (hava sıcaklığı, sarsıntı, güç kesilmesi) ile etkileşimlidir.

Sistem mimarisi, donanım basitliği açısından 8 bitlik bir mikrodenetleyici üzerine kurulmuştur. Yazılım, esneklik, taşınabilirlik ve genişletilebilirlik öğeleri dikkate alınarak C programlama dili ile geliştirilmiştir. Ayrıca uygulama geliştirmeyi hızlandırmak amacı ile PC üzerinde sistemin bir simulatörü tasarlanıp uygulanmıştır.

Sınama platformu bir kalorifer kontrol sistemi olsa da, bu tezin amacı gömülü sistemler üzerinde yazılım geliştirmenin temel kavramlarını ortaya koymak, genel programlama tekniklerinden farklı yönlerini göstermek ve tartışmaktır.

Günümüzde kalorifer sistemlerinin kontrolü yaygın olarak elle yapılmaktadır. Değişen hava sıcaklığına bağlı olarak kalorifer sisteminin uygun sıcaklıkta çalıştırılması için bir kişinin sürekli bu işle ilgilenmesi gerekmektedir.

Sistem, günün değişik zaman dilimlerinde ve değişen koşullar altında programlanabildiği için insan müdahalesi en düşük seviyeye indirilmiş, bunun yanında maksimum konfor ve enerji tasarrufu sağlanması hedeflenmiştir.

Programlama yapılabilmesi özelliği ile sistem, bir defa programlandıktan sonra kendiliğinden çalışmasını istenilen koşullara göre sürdürmektedir. Böylece hava sıcaklığı değişimlerinde sistemin sıcaklık ayarını elle yapma ihtiyacı ortadan kalkmaktadır.

Sistem, ayrıca uzaktan telefonla erişilebilir ve mevcut kalorifer sistemlerine kolay entegre edilebilir özelliktedir.

Sistem depreme karşı korumalıdır. Deprem anında sarsıntıyı algılayan cihaz güç ünitesini tamamen kapatarak kalorifer sisteminin çalışmasını durdurmaktadır.

Geliştirdiğimiz sistem, sadece elektrikle kontrol edilebilen kalorifer sistemlerinde kullanılabilir.

Gömülü sistemler ve gerçek zamanlı sistemler hakkında genel bilgi 2. bölümde verilmektedir. Bu bölümde gömülü sistemlerin tanımı, özellikleri, nasıl kullanıldıkları, ve nasıl programlandıkları üzerinde durulmuştur. Isıtıcı Denetim Sisteminin donanım ve yazılım yapısı hakkında ayrıntılar 3. bölümde; Isıtıcı Denetim Sistemi için geliştirilen simülâtörün yapısı ve çalışma mantığı 4. bölümde; Sonuçlar ve IDS' nin pazarda bulunan bazı ürünlerle karşılaştırılması 5. bölümde yer almaktadır.

2. GÖMÜLÜ SİSTEMLER

Gömülü sistemler, özel bir işi gerçekleştirmek için tasarlanmış, bilgisayar donanımı, yazılımı, ve diğer mekanik parçaların birleşimidir.

Kişisel bilgisayarlar da donanım, yazılım ve mekanik parçalardan oluşur. Fakat, özel bir işi gerçekleştirmek için tasarlanmamışlardır, bir çok farklı işi yapabilirler. Bu ayrımı belirginleştirmek için genel-amaçlı bilgisayar terimi kullanılır. Üretici, müşterinin aldığı bilgisayarı ne amaçla kullanacağını bilmez. Bir müşteri bilgisayarı ağ üzerinde dosya sunucusu olarak kullanabileceği gibi, bir diğeri bunu oyun oynamak için kullanabilir.

Bir gömülü sistem çoğunlukla, daha büyük bir sistemin parçasıdır. Bir genel-amaçlı bilgisayar birçok gömülü sistemden oluşur. Örneğin, klavye, fare, modem, video kartı, harddisk, ve ses kartının her biri bir gömülü sistemdir. Bu cihazların herbiri ayrı bir işlemci ve yazılıma sahiptir ve özel bir işi gerçekleştirmek için tasarlanmıştır.

Intel'in dünyada ilk mikroişlemciyi geliştirdiği yıl olan 1971'den önce böyle bir sistem tasarımı ortada yoktu. İlk yonga, Japon firması Busicom tarafından hesap makinelerinde kullanılmıştır. 1969 yılında, Busicom yeni hesap makineleri modellerinde kullanılmak üzere Intel'den bir dizi bütünleşmiş devre tasarlamasını istemiş ve Intel bu amaçla 4004 mikroişlemcisini tasarlamıştır. Her bir hesap makinesi için ayrı bir donanım tasarlamak yerine, Intel tüm hesap makinelerinde kullanılabilecek genel-amaçlı bir işlemci geliştirmiştir. Bu genel-amaçlı işlemci, bir bellek yongasında saklanan bir dizi komutu okumak ve işlemek için tasarlanmıştır. Intel'in amacı, her bir hesap makinesinin kendisine ait özelliklerini, sadece yazılım değişikliği ile gerçekleştirebilmesidir (Kanellos 2001).

2.1 Gömülü Sistem Örnekleri

Gömülü sistemler arasındaki çeşitliliği ve tasarım ihtiyaçlarının donanım üzerinde etkilerini göstermek için, aşağıdaki üç gömülü sistem tasarımı örnek verilebilir.

Sayısal saatler, basit ve ucuz 8-bitlik işlemciler içerir. Bu tür işlemciler çok fazla bellek adresleyemeyeceğinden, yonga üzerinde kendi ROM'ları bulunur. Eğer yeterli kayıtcı varsa, bu uygulama hiç RAM istemeyecektir. İşlemci, bellek, sayaçlar ve gerçek-zamanlı saatler gibi tüm elektronik parçalar genellikle tek bir yonga üzerinde yerleşmiştir. Girişler (tuşlar) ve çıkışlar (LCD ve hoparlör) saatin diğer donanım parçalarıdır.

Saat tasarımcılarının gerçek amacı alışlagelmişin dışında düşük üretim maliyeti olan güvenilir bir ürün yaratmaktır. Eğer, üretim sonrası, bazı saatler diğerlerinden daha güvenilir zaman tutuyorsa, bu saatler bir fiyat artışı ile farklı bir marka ismi ile satılabilir. Daha düşük maliyetli sürümler için, saat durdurma düğmeleri ya da hoparlör çıkarılabilir. Bu, saatin fonksiyonelliğini azaltır fakat herhangi bir yazılım değişikliği gerektirmez (Barr, 1999).

Bu cihazlar karşılaştırılabilir seviyede olan kişisel bilgisayarlardan daha güçlüdür. Tüketici pazarı için video oyun oynatıcılar, kişisel bilgisayarlara kıyasla daha ucuzdur. Yüksek işlemci gücü ve düşük üretim maliyeti bu cihazın üretiminde önemli rol oynamaktadır.

Oyun konsolu üreticileri, üretim maliyeti düşük olan bu ürünün geliştirme maliyetini önemsemezler. Video oyun konsolundaki işlemci 64-bit olsa bile, 64-bitlik kişisel bir bilgisayarda aynı tip işlemci olması gerekmez. İşlemci, tasarlanan video oyununun gereksinimlerine göre özelleştirilmiştir (Barr, 1999).

1976'da insansız iki uzay mekiği Mars'a gönderilmiştir. Bunların görevi, Mars yüzeyinden bilgi toplayıp bu bilgilerin kimyasal analizlerini yaparak Dünya'ya göndermektir. Sistem yapısı kişisel bilgisayarlardan oluşmuştur.

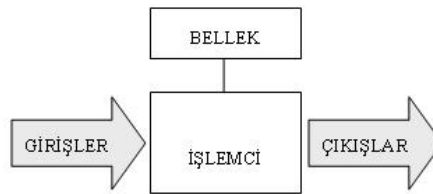
NASA, daha sonra Mars Kaşif'i yaratmıştır. Mars Gezgin'i'ndeki fazlalıklar azaltılmıştır fakat daha fazla işlemci gücü ve belleğe sahip olmuştur. Mars Kaşif'i iniş ünitesi ve gezgin olmak üzere iki gömülü sisteme sahiptir. İniş ünitesi 32-bit işlemci ve 128MB RAM, gezgin ise 8-bit işlemci ve 512KB RAM içermektedir. Bu seçimler bu iki sistemin farklı fonksiyonel ihtiyaçları olduğunu göstermektedir (Barr, 1999).

2.2 Genel Sistem Parçaları

Genel-amaçlı bilgisayarların tersine, gömülü sistemler donanım ve yazılım olarak eşsizdirler (unique). Her gömülü sistemin kendine has bir donanım-yazılım kombinasyonu vardır. Özel bir donanım için tasarlanmış gömülü yazılımlar genellikle önemli bir değişiklik yapılmadan diğer gömülü sistemler üzerinde çalışmazlar. Her bir gömülü sistemdeki donanım, sistem maliyetini düşük tutmak için uygulamaya özel olarak geliştirilir. Bunun sonucunda, gereksiz devreler çıkarılır ve donanım kaynakları mümkün olduğunca paylaşılır.

Bütün bilgisayar sistemleri gibi gömülü sistemler de, bir işlemci ve bellek yongaları kullanırlar. Bellek, hem yazılımın saklanması (ROM), hem de çalışma sırasında kullanılan değişken değerlerin tutulması (RAM) için kullanılır. Sistem ihtiyaçları doğrultusunda, eğer sadece çok küçük bir bellek gerekiyorsa, bu bellek işlemci yongasının içinde gömülü olarak da bulunabilir. Bütün sistem tek bir yongaya sığmıyorsa, belleklerden biri, ya da ikisi birden, sistem üzerinde ayrı yongalar halinde bulunurlar.

Ayrıca, tüm gömülü sistemler giriş ve çıkışlara sahiptir. Örneğin, bir mikrodalga fırında paneldeki düğmeler ve sıcaklık ayarı girişlerdir, okunabilir ekran ve ısıtma gücü miktarı ise çıkışlardır. Gömülü sistemlerin çıkışları, girişlerinin bir fonksiyonudur. Sistem girişleri sensörler, iletişim sinyalleri, ya da kontrol düğmeleri şeklinde olur. Çıkışlar genellikle ekranlar, iletişim sinyalleri, ya da dış dünyada kontrol edilen bir cihazdır. Şekil 2.1’ de genel gömülü sistem blokları görülmektedir.

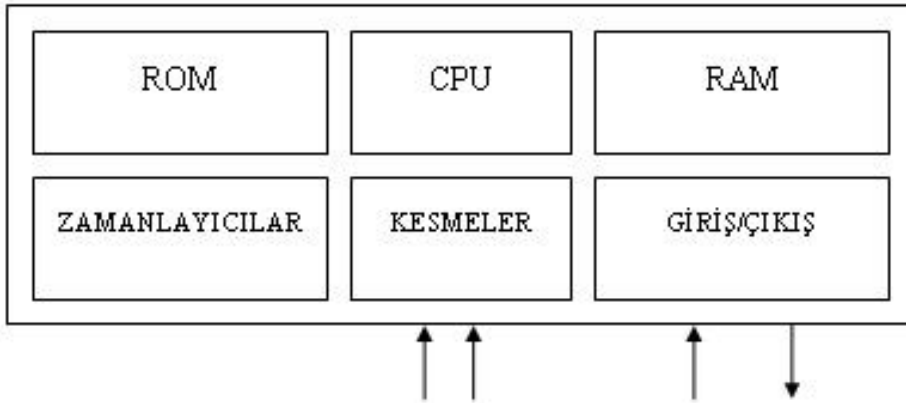


Şekil 2.1 Genel gömülü sistem blokları

Bu birkaç genel özelliğin dışında, her gömülü donanımın geri kalan yapısı genellikle kendisine özeldir. Bu çeşitlilik farklı tasarım kriterlerinin sonucudur. Her sistem tamamen farklı gereksinimlerle karşı karşıyadır, ürünün gelişim süresi boyunca yapılan donanım değişiklikleri ve maliyet düşürmeleri sonucu, temelde aynı ürünün iki farklı versiyonu için bile farklı yazılımlar gerekebilir.

2.3 Mikrodenetleyici Kavramı

Mikrodenetleyiciler gömülü sistem tasarımlarının merkezidir. Bir mikrodenetleyici ve genel amaçlı bir bilgisayar arasında keskin bir fark yoktur. Hepsinin programı çalıştırmak için sistemin bir parçası olarak işlemcisi vardır. Bir işlemcinin mikrodenetleyici olarak kullanılabilmesi için bellek ve giriş/çıkış yeteneğinin olması gerekir. Eğer bunların hepsi tek bir yonga üzerindeyse, buna mikrodenetleyici denir. Eğer yol arayüzü, bellek, ve giriş/çıkış için birden fazla yonga kullanmak gerekiyorsa, tüm bu sisteme mikrobilgisayar denir. Şekil 2.2 bir mikrodenetleyicinin genel birimlerini göstermektedir.



Şekil 2.2 Genel mikrodenetleyici yapısı

2.3.1 Giriş ve Çıkışlar

Mikrodenetleyicilerin üzerinde dış dünya ile etkileşim kurabilmek için giriş ve çıkışlar bulunur. Bunlar basitçe analog ve sayısal olmak üzere iki bölüme ayrılabilirler.

Sayısal girişlerin her biri tek bacaklık bir bağlantı noktasıdır. Giriş ve çıkış değerleri mantıksal olarak '1' ya da '0' olabilir. Bazı durumlarda bir bacak hem giriş, hem de çıkış olabilecek şekilde programlanabilir. Bacakların fonksiyonlarını belirleyen özel yazmaçlar vardır ve bunlar program akışı içerisinde ihtiyaca uygun olarak programlanarak sistemin davranışı değiştirilebilirler.

Genel amaçlı sayısal giriş-çıkışlar çok kullanışlıdır. Bunlar açma/kapama, tuş okuma gibi fonksiyonlar için kullanılabilecekleri gibi, özel sürücü yazılımlarıyla I2C, SPI gibi iletişim protokolü emulasyonu ya da darbe genişlik modülasyonu (PWM) gibi özel sinyallerin üretilmesinde de kullanılabilirler. Bir mikrodenetleyicinin üzerinde ne kadar çok genel amaçlı giriş-çıkış varsa sistemin donanımsal ve yazılımsal tasarımı da o derece kolaylaşır.

Bazı mikrodenetleyiciler analog karşılaştırıcılar ya da analog-sayısal dönüştürücüler içerir. Bu gibi durumlarda mikrodenetleyicinin bu devrelere bağlı bacakları sadece '1' ve '0' değil, minimum ve maksimum gerilim seviyeleri arasındaki herhangi bir değeri algılayabilir. Bu şekilde yazılımdan dış dünyadan bu bacaklara bağlanmış sinyallerin seviyeleri doğrudan okunabilir.

2.3.2 Bellek

Mikrodenetleyicilerin içinde genelde flash ROM ve durağan RAM olmak üzere bellekler bulunur. Bunların boyu mikrodenetleyicinin türüne göre ve tasarlanan kullanım alanına göre farklılık gösterir.

Bazı mikrodenetleyicilerin içinde, kullanıcı ayarlarının saklanabilmesi için EEPROM gibi yardımcı birimler de bulunmaktadır. Bu mikrodenetleyiciler kullanılarak, dışarıdan bir EEPROM bağlama ihtiyacı ortadan kalktığı gibi, bu yapıların genelde mikroişlemcinin adres alanına konulmuş olmaları sayesinde, dışarıdan bağlı bir EEPROM için

geliştirilmesi gereken yazılım sürücülerine de ihtiyaç olmayacak ve kod boyu küçülecektir.

2.3.3 Haberleşme

Bazı mikrodnetleyiciler, içlerinde donanımsal olarak birleştirilmiş haberleşme birimleri taşırlar. Bir çok mikrodnetleyici standart olarak UART seri haberleşme birimi içerir. Bunun dışında, örneğin, I2C, ve SPI gibi yongalar arası haberleşme protokollerini donanımsal düzeyde gerçekleştirebilen mikrodnetleyiciler vardır. Bu sayede, örneğin, I2C haberleşmesi için genel amaçlı giriş-çıkışları kullanan ayrı bir sürücü yazılmasına gerek kalmayacak ve aynı zamanda bu sürücülerin yazılımın çalışması sırasında getirecekleri işlemci yükü de ortadan kalkacaktır.

2.3.4 Mikrodnetleyici seçimi

Mikrodnetleyiciler çok çeşitlidirler. Her amaca uygun mikrodnetleyici bulmak mümkündür.

Günümüzde bazı gelişmiş mikrodnetleyici aileleri, içlerinde donanımsal olarak gerçekleştirilmiş TCP/IP ağ yığınları, USB arabirimleri, MPEG çözücüler veya doğrudan ethernet bağlantıları gibi birçok gelişmiş özellik barındırmaktadırlar.

Doğal olarak mikrodnetleyicinin içinde bulunan yardımcı arabirimlerin sayısı arttıkça, birim maliyeti de artmaktadır. Ancak gereksinim duyulan özelliklerin ne kadarı mikrodnetleyici içine gömülü olursa sistemin tasarım ve bakım maliyeti de o derece azalır; ve çalışma güvenilirliği artar.

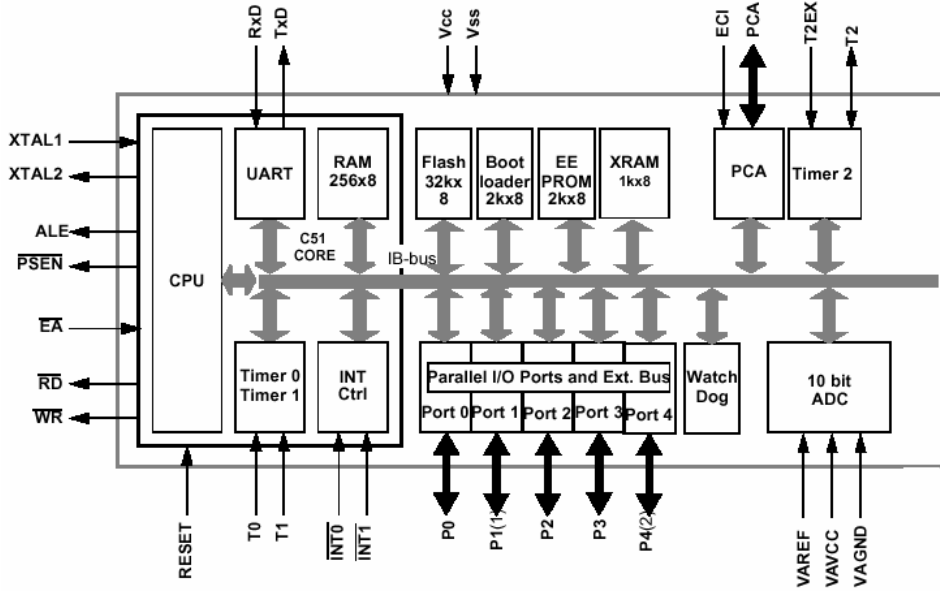
Mikrodnetleyici seçimi yapılmadan önce sistemin tam bir analizi yapılmalıdır. Bu analizde dikkat edilmesi gereken noktalar ve sorulması gereken sorular şu şekilde sıralanabilir:

- Kod boyu tahminen ne kadar olacak?

- Kodun ihtiyaç duyacağı RAM miktarı ne kadar olacak?
- Kaç adet giriş-çıkış ihtiyacı var?
- İşlemcinin çalışma hızı ne olmalı?
- Tüketeceği güç önemli mi? (Örneğin pille beslenen cihazlarda çok önemlidir)
- Uygulamaya özel yardımcı arabirimlere sahip mi? (Örneğin uygulama bir MP3 oynatıcısı ise, mikrodnetleyicinin içinde gömülü gelen bir MP3 format çözücüsü tercih nedeni olabilir)

2.4 8051 Mikrodnetleyici Ailesi

İlk tek-yonga mikrodnetleyici olan 8048 den sonra gelen mikrodnetleyicilerin genel adıdır. Intel tarafından geliştirilmiştir. Yapısı ve komut seti açısından daha çok kod alanı, daha fazla zamanlayıcı, kesme, ve giriş-çıkışlar içeren geniş bir mikrodnetleyici ailesidir (Shultz, 1996). Şekil 2.3'te örnek bir 8051 mimarisinin iç yapısı bloklar halinde görölmektedir.



Şekil 2.3 Örnek bir 8051 iç yapısı (Atmel, 2000, s.1)

8051 ailesinde yonga üzerindeki ROM miktarı üyeler arasında değişebilir. Birisi sadece 1K kod belleğine sahipken, bir kısmı 2K, bazıları da 64Kbayta kadar belleği üzerinde bulundurabilir. Bazıları kodu yonga üzerinde tutmak yerine harici kod saklama birimi kullanır.

8051 ailesi içinde, veriyi tutmak için 64 den 256 bayt a kadar dahili RAM olabilir. Bu durağan bellekler hızlıdır ve çeşitli adresleme modlarını desteklerler.

8051 ailesi standart olarak 64kbayta kadar harici RAM desteğine sahiptir. Ancak RAM ve ROM yongalarının harici olarak kullanılması, işlemci üzerinde bu amaçla kullanılan bacakların genel amaçlı olarak kullanılamamasına neden olur. Bazı 8051 türevleri, yukarıdaki blok diyagramdaki gibi, kendi içinde harici RAM bulundurlar. Bu durumda bu belleğe erişim yazılımsal olarak harici RAM gibi yapılsa da, donanımsal olarak yonganın içindedir ve dışarıdaki genel amaçlı giriş/çıkışlar da boş kalmış olur.

Standart bir 8051 üzerinde, 4 adet 8 bitlik kapı olmak üzere, toplam 32 adet genel amaçlı giriş çıkış bulunur. Bazı giriş çıkışların birden fazla fonksiyonu vardır. Örneğin bazıları kesme, sayma ya da haberleşme arabirimi olarak kullanılabilirler.

Standart bir 8051 iki adet donanımsal zamanlayıcı içerir. Bu zamanlayıcılardan bir tanesi seri haberleşmeyi çalıştırmak için kullanılırken, diğeri genel amaçlı olarak, gerçek zaman saati gibi, belirli aralıklarla düzenli kesmeler üreten bir zamanlayıcı olarak kullanılabilir. Bazı 8051 türevleri üzerinde üç ya da daha çok zamanlayıcı bulunabilmektedir.

C, BASIC, ve assembly 8051 ailesi ile en yaygın kullanılan programlama dilleridir. Bunun yanında, çok az da olsa Forth ve PL/M dilleri de programlama için kullanılmıştır. Son zamanlarda eğilim büyük oranda C üzerine olmuştur. C gibi yüksek seviye dillerin getirdiği faydalar ile ilk mikrodenetleyiciler için gerekli olan zor ve ayrıntılı programlama teknikleri aşılmıştır.

Assembly dilini biraz anlamak 8051'in limitlerinin anlaşılması için faydalıdır. Örneğin, 8051 assembly dil komutlarının bellek alanına nasıl eriştiğini bilmek yonga üzerindeki RAM in değişkenler için verimli kullanılmasını sağlayacaktır.

C, assembly diline göre daha kolay yazılır çünkü derleyici sistem açılışından bellek düzenlenmesine kadar bir çok ayrıntıyı programcı adına gerçekleştirir. Yaklaşık 1985 yılından beri 8051 mikrodenetleyiciler için kullanılmaktadır, ve 8051 C derleyicileri satan birden fazla firma vardır. Fakat bunların hepsi 8051'in farklı mimarisi için çok verimli kodlar üretemezler. Bu projede 8051 için optimize edilmiş Keil C derleyicisi kullanılmıştır. Bu derleyicinin deneme sürümü internetten elde edilebilir (Shultz, 1996).

2.5 Gömülü Sistem Geliştirme Parametreleri

Üretim maliyeti, gömülü donanım tasarımcılarının karşılaştığı olası kısıtlardan sadece biridir. Diğer tasarım parametreleri, işlemci gücü, bellek, geliştirme maliyeti, birim sayısı, beklenen ömür ve güvenilirlik olmak üzere sıralanabilir.

İşlemci gücü, işin yapılması için gerekli olan işlemci gücü miktarıdır. İşlemci gücünü kıyaslamanın genel yolu MIPS (millions of instructions per second) birimidir. Eğer iki farklı işlemci 25 MIPS ve 40 MIPS işlem gücüne sahipse, ikincisi birincisinden daha güçlüdür denilebilir. Ancak, buna rağmen, işlemcinin diğer önemli özellikleri de dikkate alınmalıdır. Bunlardan birisi 8 ve 64 bit arasında değişen yazmaç genişliğidir. Günümüzün genel-amaçlı bilgisayarları sadece 32 ve 64 bit işlemciler kullanır, fakat gömülü sistemler 8 ve 16 bit işlemciler kullanılarak da tasarlanabilir.

Sistemin işletilebilir yazılımı ve kullandığı veriyi tutmak için ihtiyacı olan bellek (ROM ve RAM) miktarı, işlemci seçiminde etkin rol oynar. Genellikle, işlemcinin kayıtçı genişliği, erişebileceği en yüksek bellek kapasitesini belirler. (örneğin, 8-bit lik kayıtçı sadece 256 bellek yerinden birini seçebilir.) Bellek ihtiyacını ise sistem yazılımının boyu ve yazılımın ihtiyaçları belirler.

Geliştirme maliyeti, donanım ve yazılım tasarımının maliyetidir. Yüksek miktarda üretimlerde bu maliyet önemsizken, düşük miktarda üretimlerde tasarım maliyeti sistem maliyetinin gerçek ölçüsüdür.

Üretim maliyeti ile geliştirme maliyeti arasındaki dengeyi, beklenen üretim hacmi etkiler. Yüksek hacimde üretilecek projelerde, birim maliyetini düşürmek için tasarım maliyeti yüksek tutulurken, düşük miktarda üretilecek projelerde birim maliyeti belirleyici faktör olmaktan çıkar.

Sistemin ortalama ne kadar süre işleyeceği, beklenen ömrü, donanım komponentlerinin seçiminden, sistemin geliştirme ve üretim maliyetine kadar tüm tasarım kararlarını etkiler.

Ürünün cinsi, beklenen güvenilirlik ölçüsünü belirler. Eğer ürün bir çocuk oyuncağı ise her zaman doğru çalışması gerekmez, fakat eğer bir otomobil parçası ise her zaman beklenen görevi yapması gerekir.

2.6 Gömülü Sistemlerde Yazılım Kavramları

Gömülü sistemlerde kullanılan yazılım kavramları büyük oranda, genel amaçlı yazılım kavramları ile aynıdır. Ancak bir çok durumda, gömülü sistemlerin sınırlı kaynakları ya da gerçekleştirilen sistemin

doğasından kaynaklanan ihtiyaçlar ve/veya kısıtlar nedeniyle, kullanılan yazılım kavramlarının kapsamları ve kullanım şekilleri bir miktar değişiklik gösterebilir. Bu bölümde kısaca, gerçek zamanlılık ve çokluışlem üzerinde durulmuştur.

Çokluışlem yapısını gerçeklemek için kullanılan yöntemler, gömülü sistemlerde de kullanılabilir. Ancak bu yöntemleri ayrıntıları ile incelemek bu tezin kapsamında olmadığından üzerinde durulmamıştır.

2.6.1 Gerçek Zamanlılık

Bir çok gömülü sistem gerçek zamanlı olarak düşünülebilir. Gerçek zamanlı sistemler, hesaplamaların doğruluğunun yalnızca mantıksal doğruluğa değil, aynı zamanda sonucun üretildiği zamana bağlı olduğu sistemlerdir. Başka bir deyişle, geç bir cevap yanlış cevaptır.

Bir sistemin gerçek-zamanlı olması donanım ve kullanıcı ihtiyaçlarını yeterince hızlı karşılayacak şekilde girişlere cevap vererek çıkışlar üretmesi demektir. Örneğin, sistemin kullanıcı tarafından basılan bir tuşa 100 milisaniye içinde tepki vermesi gerçek zamanlı bir davranış sayılırken, aynı sistemin A/D çeviriciden gelen verileri değerlendirirken gerçek zamanlı davrandığını söyleyebilmek için 100mikrosaniye içinde tepki vermesi gerekebilir (Burns, 2001).

Gerçek zamanlı donanım gereksinimleri

Gerçek zamanlı sistemlerde donanım olarak, zamanlayıcı (timer), kesme (interrupt) ve gerçek-zaman saatine (real-time clock) ihtiyaç vardır.

Bir zamanlayıcı devresi, programlanan zaman aralıklarında kesmeler üretir. Bu kesmeler tek ya da sürekli olacak şekilde programlanabilir. Gerçekte bir zamanlayıcı mikrodnetleyicinin saatini sayan bir sayaçtır. Sayma işlemi, yazılımdan bağımsız olarak devam ettiğinden, gecikme döngülerine bağımlı olmadan zaman gecikmeleri düzenlenebilir.

Bir gerçek-zamanlı sistem en az bir kesmeye sahip olmalıdır. En önemli kesme, düzenli aralıklarla zamanlayıcı tarafından üretilen kesmedir. Bu kesme işlemciyi o an çalıştırdığı işten çıkarır ve böylece işlemci başka bir işin ihtiyacına cevap verebilir. Bu kesme ile her bir işin işlemcinin zamanını paylaşarak kullandığından emin olduğumuzdan işlerin içinde çokluişlemli sistemin sürekliliğini sağlamak için özel bir kod yazmaya gerek yoktur.

Gerçek zaman saati bir zamanlayıcı tarafından sürülen dijital bir saattir. Gerçek zamanı yıl, ay, gün, vb şeklinde saklar. Genelde pille de beslenen gerçek zaman saatleri sistem kapalıyken bile saatin unutulmamasını sağlarlar.

2.6.2 Çokluişlem

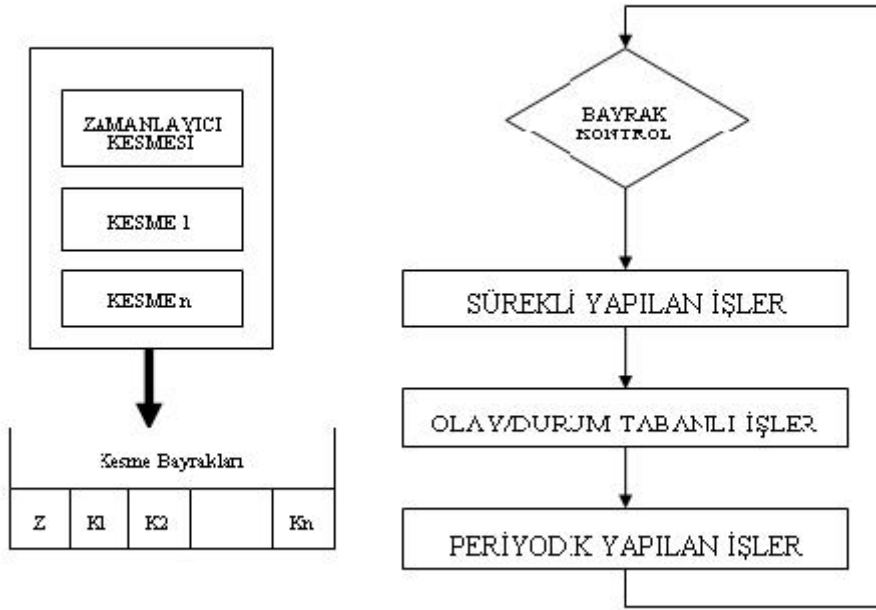
Genel amaçlı bilgisayar sistemlerinde olduğu gibi, gömülü sistemler de çoğu zaman birden fazla işi aynı anda yapmak zorundadırlar. Kullanıcı arabirimi ile kontrol sisteminin aynı anda çalışması bunun için verilebilecek basit bir örnektir. Kullanıcı, kontrol düğmeleri ve ekran aracılığı ile sistem hakkında bilgi edinirken ya da sistem ayarlarını yaparken, sistem arka planda kontrol işlerini gerçekleştirmeye devam edebilir.

Çokluişlem etkisini yaratmak için bir işletim sistemi kullanılabilir. Ancak sistem kaynaklarının çok sınırlı olduğu durumlarda ise, işletim sistemi yerine donanım kesmeleriyle desteklenmiş, sonsuz döngü tabanlı bir sistem kullanılabilir. Bu sistem *round-robin* mantığında bütün işleri sırasıyla yaparak, yine de aynı anda çalışıyorlarmış gibi etki yaratabilir. Kesmeler genelde işi doğrudan yapmak yerine bayrak kaldırırılar. Bu şekilde, işin daha sonra zaman olduğunda yapılması sağlanarak program akışının uzun süre kesilmesi engellenir. Ancak, bu durumda kesmeye verilen tepki gecikir ve tepki süresinin maksimum değeri döngü süresine eşittir.

İşlerin sonsuz döngüde nasıl yapıldığı Şekil 2.4' de gösterilmiştir. Sisteme zamanlayıcılardan veya dışarıdan gelen kesmeler bayraklarda tutulur. Sonsuz döngü içinde çalışan program, bayrakların değerlerine bakarak olaylardan haberdar olur ve buna bağlı olarak, olay tabanlı işler

ile zaman tabanlı işlerin, döngünün bu çevriminde yapıp yapılmayacağına karar verir.

Bu tür sistemlerde dikkat edilmesi gereken, işlerin durum güncellemelerini en kısa sürede yaparak işlemci zamanını diğer işlere bırakmalarıdır. Gerçek-zaman etkisini kaybetmemek için toplam döngü süresi de çok uzun olmayacak şekilde programlama yapılmalıdır. Bu nedenle güncellemeye gerek duyulmayan işler güncellenmemeli, bazı işler sadece bir olay gerçekleştiğinde güncellenmeli ya da tuşların taranması gibi bir takım işler de belli zaman aralıklarıyla yapılmalıdır. Böylece, sistem tasarımı sırasında belirlenmiş olan kabul edilebilir en büyük döngü süresi korunmalıdır. Kabul edilebilir en büyük döngü süresi aynı zamanda kabul edilebilir en büyük kesme tepki gecikmesidir.



Şekil 2.4 Sonsuz döngünün kesmelerle kullanımı

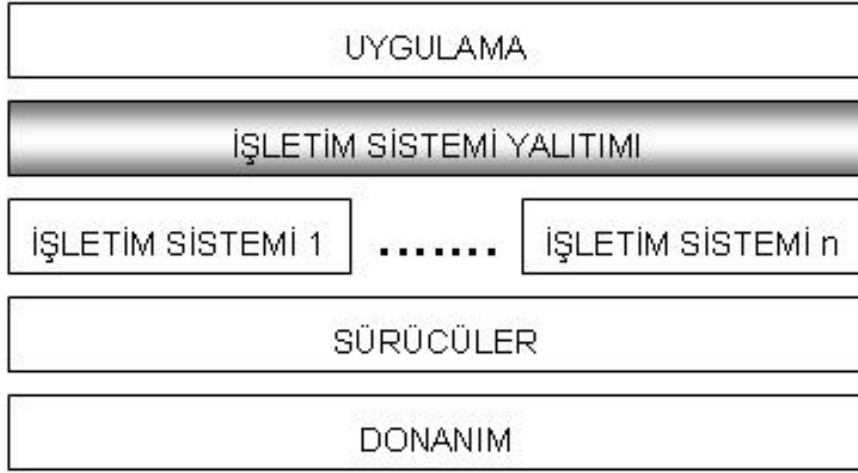
Gömülü sistemlerde kullanılan işletim sistemleri de genel amaçlı işletim sistemlerinden farklıdır. Bu işletim sistemleri, genelde çok küçük ve çok hızlı olacak şekilde tasarlanmışlardır. Büyük oranda sadece çoklu işlem sağlayan bir çekirdekten oluştukları için genel amaçlı işletim sistemlerine göre çok daha az kaynağa ihtiyaç duyarlar.

Gömülü işletim sistemlerinin ana amaçları çoklu işlem desteği vermektir. Ancak bunun yanında çoklu işlemli programlamanın temel ihtiyaçları olan işlerin senkronizasyonu ve birbiri ile haberleşmesini de sağlarlar. Ayrıca bir çok gömülü işletim sistemi bellek yönetimi ve donanım sürücülerini de kendilerinin bir parçası olarak gerçeklerler. Bu durumda yazılım geliştiricisinin tek ihtiyacı olan, işletim sisteminin arayüzünü kullanmayı bilmektir.

Genel amaçlı işletim sistemlerinin tersine, gömülü işletim sistemleri esnektirler. Örneğin kullanılmayan özellikleri derleme sırasında kullanılan yapılandırma seçenekleriyle iptal edilebilir. Bu durumda kod boyu küçülür ve sistemin gereksinim duyduğu bellek miktarı azalır. Örneğin bir sistem semafor desteğine ihtiyaç duymuyorsa, bu destek derleme sırasında sökülebilir.

QNX, VxWorks, RTX, ucLinux bilinen gömülü işletim sistemlerinden bazılarıdır. Her işletim sisteminin kendine has yapısı, özellikleri ve kullanım arabirimi vardır. Bu nedenle çoklu işlem ve işletim sistemi kullanan uygulamalar, gelecekte işletim sisteminin değişebileceği düşünülerek, bir işletim sistemi yalıtım katmanı üzerinde geliştirilmelidir. Bu şekilde yazılan kodların taşınabilirliği ve esnekliği önemli ölçüde artmış olur.

Bu yapı Şekil 2.5’ de görülebilir. İşletim sistemi yalıtım katmanı, bütün işletim sistemleri tarafından sağlanması gereken temel fonksiyonları içerir. Uygulama sadece bu fonksiyonlar kullanılarak geliştirilirse, yalıtım katmanı altında kalan işletim sistemi değişimlerinden etkilenmez.



Şekil 2.5 İşletim sistemi yalıtım katmanı kullanımı

Çoklu işlemlerde, iş (job), görev (task) ve öncelik (priority) kavramları önemlidir.

İş, bir mikrodalga fırını kontrol etmek olabilir. Bir iş, kullanıcı dokunma panelinin taranması, ekranın tazelenmesi, zamanın izlenmesi, güvenlik kilidi cihazlarının görüntülenmesi, ve magnetronun sürülmesi gibi birden fazla görevden oluşabilir.

Bir iş mikrodenetleyiciyi en verimli kulanacak şekilde mantıksal parçalara bölünür, ve bu parçalardan her biri bir görev olarak düşünülebilir. Bir mikrodenetleyici için, görevler, klavyeden gelen girişi beklemek ve bunu işlemek ya da, ayarlanmış bir hızda bir motorun çalışmasını sürdürmek olabilir. Bir görev, az sayıda kod içeren tek bir fonksiyon ya da, birbiriyle ilişkili bir den fazla fonksiyondan oluşabilir.

Yüksek-öncelikli bir görev daha önemli ya da acildir, ilk önce yapılması gerekir. Bir sistemde farklı öncelikli görevlerin bulunması; bazı aktivitelerin acil ve diğerlerinden daha önemli olduğunu belirtir. Yüksek öncelikli bir görev sırada bekleyen bütün görevlerin önüne geçebilir.

Öncelik için örnek vermek gerekirse, bir aracın ABS sisteminden gelen sensör verilerini okumak, yol bilgisayarı bilgilerini güncellemekten

daha önceliklidir. Sensör okuma zamanı geldiğinde mutlaka bütün görevler bırakılıp sensör bilgileri okunmalıdır.

2.6.3 Çokluişlem ile programlama alışkanlıkları

Çokluışlemi destekleyen bir ortamda uygulanması gereken programlama alışkanlıkları, tek iş üzerinde çalışan programlara göre farklılıklar gösterir. Ve dikkat edilmesi gereken noktalar vardır.

Asla Bekleme

Program parçaları aktivitenin hemen yapılmasını ya da bir sonraki zamana kadar ertelenmesini sağlayacak şekilde yazılmalıdır. Örneğin, bir iş kullanıcının tuşa basmasını beklerse ya da uzun bir döngüde gecikme yaratırsa, diğer herşey beklemek zorunda kalacaktır.

Bayrak Kaldır (Flag setting)

Gerçekleşen bir olay için anında cevap veremediğimiz durumlarda, olayın kaydedilmesi için bir sayacın artırılması ya da bir bayrağın kurulması gerekir. Genelde olaylar, eğer kesmeler gibi acil değillerse, asenkron olarak işletilebilirler. Bayrakların ayarlanması olayların gecikmesiz kaydını sağlar, ve böylece arka planda çalışan işler, olayların sonucunda yapılması gereken uzun işlemleri yeterli zaman olduğunda gerçekleştirirler. Örneğin bir tuşa basıldığında oluşan kesme asenkron olarak işlenebilir. Bunun için aşağıdaki gibi bir yapı kurulabilir (Hansen, 1973).

```

BOOL keyPressed; // Bayrak
void interrupt KeyInterrupt()
{
    keyPressed = TRUE; // Bu noktada kesme fonksiyonunun işi biter
}

```

```
// Ana program
main(){
    while(1){
        if(keyPressed){
            // Tuşa basıldığında yapılan işi yap
            keyPressed = FALSE;
        }
        // diğer işlere devam et
    }
}
```

Geciktirme Döngülerinden Kaçınma

Geciktirme döngüleri tarafından üretilen yazılım-kaynaklı gecikmeler yerine mikrodenetleyicinin zamanlayıcı özelliği kullanılmalıdır. Zamanlayıcıları doğrudan kullanmak yerine, tüm işler için gereken zamanlama tek bir zamanlayıcı kesmesi kullanılarak yapılabilir. 1 ms kabul edilebilir bir zaman artışı olduğu sürece, tüm zaman-bağımlı aktiviteler gerçek-zamanlı saat ile yapılabilir. 1 ms de, her 1000. kesmede yapılan işlem saniyede bir olmaktadır.

Örneğin aşağıdaki gibi bir geciktirme döngüsünün kullanımından kaçınılmalıdır. Bu döngü çalıştığı sürece sistemdeki diğer işlerin devam etmesini engeller.

```
void WaitOneSecond()
{
    int i = 0xffff;
    while(i--);
}
```

2.6.4 Bellek içeriğini doğrulama

Gömülü sistemler çevresel faktörlerden etkilenmeye her zaman açıktırlar. Bunun sonucu olarak bellek içeriğinin bozulma riski vardır. Örneğin, bir flash yongaya yazma sırasında enerjinin kesilmesi bu yongayı kullanılamaz duruma getirecektir. Ya da ani gelen bir eletromanyetik darbe RAM içeriğinin bozulmasına yol açabilir.

Gömülü sistemler üzerinde çalışırken bellekteki bilgilerin doğruluğunu test etmek gerekir. Çünkü bellek verilerindeki bozulma sistemin hatalı davranmasına yol açabilir. Bellek içeriğini doğrulamada kullanılan iki temel yöntem vardır: sağlama toplamı (checksum) ve çevrimseş artıklık kodları (cyclic redundancy code –CRC).

Sağlama Toplamı (Checksum)

Kalıcı bellek üzerinde tutulan program ya da verinin geçerli olup olmadığını anlamanın en kolay yolu verinin sağlam olduğu düşünüldüğünde bir sağlama toplamını hesaplamaktır. Böylece, her zaman verinin geçerliliğini doğrulamak için, sağlama toplamını yeniden hesaplayarak sonucu bir önceki hesaplanan değerle karşılaştırmak gerekmektedir. Eğer bu iki sağlama toplamı eşit ise verinin sağlam olduğu kabul edilir. Sağlama toplamı algoritmasını dikkatli seçerek bazı tip hataların yakalanma olasılığını artırabiliriz.

En kolay sağlama toplamı algoritması tüm veri baytlarını toplamaktır. Bu algoritmanın en zayıf yönü, sağlama toplamını da içeren tüm verinin yanlışlıkla 0'lar ile üstüne yazılmasıdır. Bu yüzden bu veri bozulması saptanamayacaktır. Ayrıca, büyük bir sıfırlar bloğunun da toplamı sıfırdır. Bu zayıflıkla başa çıkmanın en kolay yolu sonucu tersine çevirmektir. Bu da sağlama toplamı algoritmasına son adım olarak eklenir. Bu yöntemle, eğer veri ya da sağlama toplamı bir şekilde 0'lar ile üstüne yazılırsa, doğru sağlama toplamı FFh olacağından test başarısız olacaktır.

Bunun gibi basitçe veriyi toplayarak yapılan sağlama toplamı bir çok genel veri hatasını tespit edemez. Eğer verinin tek bir biti bozulduysa (örneğin 1 den 0 a değiştiyse), bu hata tespit edilecektir. Ancak benzer

bir dizide iki bit bozulduysa (birincisi 1 den 0 a, diğeri 0 dan 1 e), sağlama toplamı değişmeyeceğinden hata tespit edilemeyecektir. Bu tür bit hatalarının tespiti için daha iyi sağlama toplamı algoritmaları kullanılmalıdır.

Beklenen sağlama toplamı hesaplandıktan sonra, bunu saklayacak bir yere ihtiyaç duyulacaktır. Bir seçenek, sağlama toplamını daha önceden hesaplamak ve bunu veriyi doğrulama yönteminde bir sabit olarak tanımlamaktır. Bu yöntemin de kusurları vardır. Bunların arasında en önde geleni, ürünün ömrü süresince verinin ve beklenen sağlama toplamının değişebilme olasılığıdır. Bu yöntem, test edilen verinin yeni özellikler eklendiğinde ya da hatalar onarıldığında periyodik olarak güncellenen bir gömülü yazılım olması durumunda uygundur.

En iyi fikir sağlama toplamını bellekte sabit bir yere koymaktır. Örneğin, bellekteki en son yer kullanılabilir. Bu, sağlama toplamının yerleştirilmesini kolaylaştırır. Sadece sağlama toplamı hesaplanır ve bu toplam belleği programlama öncesinde belleğe yerleştirilir. Sağlama toplamı yeniden hesaplandığında, beklenen sonucu içeren bellek yeri atlanır, ve yeni sonuç orada tutulan değer ile karşılaştırılır. Sağlama toplamını başka bir kalıcı bellekte saklamak diğeri bir yöntemdir. Bu çözümlerin her ikisi de pratikte iyi çalışır.

Çevrimsel Artıklık Kodları (Cyclic Redundancy Codes)

Bir çevrimsel artıklık kodu (CRC) bir çok genel veri hatalarını yakalamak için tasarlanmış özel bir sağlama toplamı algoritmasıdır. CRC çok büyük veri bloklarının aktarılmasını ya da saklanmasını isteyen gömülü uygulamalarda oldukça faydalıdır.

Veri topluluğu uzun bir 1'ler ve 0'lar dizisinden oluşur. Bu ikili dizi, jeneratör polinomu olarak adlandırılan daha küçük sabit bir ikili dizi ile bölünür. Bu uzun ikili bölme işleminin kalanı CRC sağlama toplamını verir. Jeneratör polinomunun istenilen belirli matematiksel özellikleri sağlamasına göre seçilmesiyle, sağlama toplamı mesaj içerisindeki bir çok hatayı yakalamak için kullanılabilir. Jeneratör polinomlarının en güçlüsü tüm tek ve çift bit hataları ile ardışık hata bitlerinin tüm tek-uzunluktaki dizilerini yakalayabilir.

En iyi hata-yakalama yetenekleri ile jeneratör polinomları uluslararası standartlar olarak benimsenmiştir. Bu standartlar Çizelge 2.1’ de parametrelendirilmiştir. Jeneratör polinomunun genişliği, bu polinomun bölen olarak adlandırılan ikili gösterimi, kalanın başlangıç değeri, ve en son kalan ile XOR (exclusive-or) değeri her bir standartla ilişkilendirilmiştir.

Çizelge 2.1 Uluslararası Jeneratör Polinomları Standardı (McDonnell, 2004)

	CCITT	CRC16	CRC32
Sağlama toplamı genişliği	16 bits	16 bits	32 bits
Jeneratör polinomu	$x^{16} + x^{12} + x^5 + 1$	$x^{16} + x^{15} + x^2 + 1$	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
Bölen	0x1021	0x8005	0x04C11DB7
Başlangıç kalanı	0xFFFF	0x0000	0xFFFFFFFF
En son XOR değeri	0x0000	0x0000	0xFFFFFFFF

Verimli bir CRC kodu temel olarak iki bölümden oluşur.

```

crcInit() {
    // Bakma tablosunu hazırlar
}
crcCompute() {

```

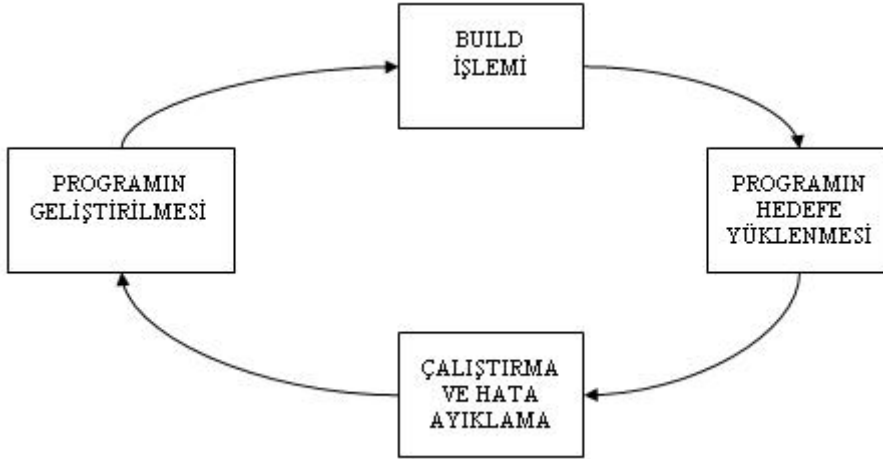
```
//İstenilen CRC değerini hesapla
}
```

İlk olarak `crcInit` fonksiyonu çağırılır. Bu fonksiyon CRC algoritması tarafından istenen alışılmamış ikili bölmeleri hesaplar. Mesaj verisinin bir baytının her 256 olası değeri için kalanı önceden hesaplar. Bu ara sonuçlar bir global bakma tablosunda tutulur ve `crcCompute` fonksiyonu tarafından kullanılır. Bu yöntemle, büyük bir mesajın CRC değeri bit bit yerine, bir anda bayt olarak hesaplanabilir. Bu da CRC hesaplama süresini belirgin bir şekilde azaltır (Barr, 1999).

`crcCompute()`, CRC yi hesaplamak ve doğrulamak için uygulama üzerinden çağırılabilen bir rutindir. Hesaplamanın `crcInit` ve `crcCompute` arasında ikiye ayrılmasının faydası `crcInit` fonksiyonunun gömülü sistem üzerinde çalıştırılmasına gerek olmamasıdır. Bunun yerine, bu fonksiyon bakma tablosunun içeriğini oluşturmak için önceden çalıştırılabilir. Tablo içindeki değerler daha sonra ROM içinde tutulur ve `crcCompute` tarafından defalarca kullanılır.

2.7 Gömülü Sistemler Üzerinde Yazılım Geliştirme Süreci

Gömülü sistemlerin programlanması, genel programlamadan çok farklı değildir. Değişen tek şey, her bir gömülü sistem programının hedef seçtiği donanımın farklı olmasıdır. Bu fark da yazılımda ek bir karmaşıklığa yol açar, ve yazılım build işleminin daha dikkatli yapılması gerekir. Yazılım geliştirme sürecinde kullanılan derleyici, hata ayıklayıcı gibi araçlar, gömülü sistemler için farklıdır ve farklılığı sistem tasarımında kullanılan mikrodenetleyici belirler. Her mikrodenetleyici üreticisi kendi yazılım geliştirme ortamını sağlamaktadır. PC’de olduğu gibi, yazılım geliştirme araçları sadece yazılım değil, aynı zamanda PC ile gömülü sistemi birbirine bağlayan donanım parçalarından da oluşur. Bu yazılım- donanım kümesi ayrılmaz bir bütündür ve kullanılan mikrodenetleyiciye özeldir. Şekil 2.6 yazılım geliştirme sürecini basitçe gösterir.



Şekil 2.6 Gömülü yazılım geliştirme süreci

C programlama dilinin gömülü sistem tasarımlarında kullanımı oldukça yaygındır. Assembly ve C++, gömülü sistem programlayıcıları tarafından kullanılan diğer dillerdir.

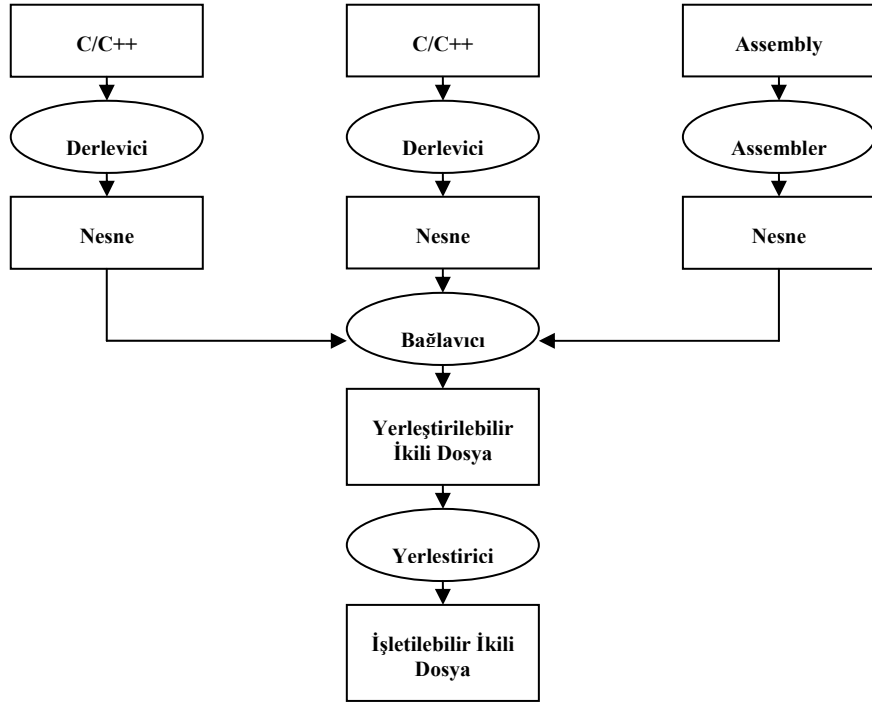
Eskiden, gömülü yazılım assembly dilinde yazılmaktaydı. Bu, programcılara işlemcinin ve diğer donanımın tüm kontrolünü vermekteydi fakat bunun bedeli vardı. Kod taşınabilirliğinden yoksun olması ve geliştirme maliyeti assembly dilinin dezavantajlarıdır. Günümüzde, assembly dili üst-seviye dillere ek olarak, kodun az yer kaplaması ve etkili olması gereken küçük parçalarında kullanılır.

C++ gömülü programlayıcılar arasında yaygınlaşan, C'nin nesneye-yönelik bir üst kümesidir. C++, programlamanın daha nesneye-yönelik olması ve daha iyi veri soyutlama için yeni özellikler eklemektedir. Bu yeni özellikler yazılım geliştiriciler için çok faydalıdır, fakat işletilebilir programın etkinliğini azaltmaktadır.

2.7.1 Derleme, Bağlama, ve Yerleştirme

Gömülü yazılımın kaynak kodunun işletilebilir hale gelmesi üç ayrı adımda gerçekleşir. Birincisi, her bir kaynak dosyası derlenerek bir nesne dosyasına dönüştürülmelidir. İkincisi, birinci adımdan sonra oluşan tüm nesne dosyaları tek bir nesne dosyası oluşturmak için birbirine bağlanmalıdır. Üçüncüsü, fiziksel bellek adresleri birbirine bağlı ofsetlere çevirilir, ve bu işlemin adına yeniden yerleştirme (relocation) denir. Üçüncü adımın sonunda, gömülü sistem üzerinde çalışmaya hazır işletilebilir kod içeren bir dosya oluşur. Bu sürece inşa (build) süreci denir.

Gömülü yazılım inşa işlemi Şekil 2.7' de gösterilmiştir. İnşa süreci, yukarıdan aşağıya üç adımda anlatılmıştır. Bun adımları gerçekleştiren araçlar, yuvarlak kutuların içinde gösterilmiştir. Bu geliştirme araçlarının herbiri bir ya da daha fazla dosyayı girdi alarak tek bir çıkış dosyası üretir.



Şekil 2.7 Gömülü yazılım inşaa süreci

Gömülü yazılım inşaa sürecinin her adımı, genel-amaçlı bir bilgisayar üzerinde çalışan yazılım araçları tarafından yapılır. Geliştirme yapılan bu bilgisayar, hedef gömülü sistemden ayırt edilebilmesi için, geliştirme bilgisayarı olarak adlandırılır. İnşaa yazılımının parçaları derleyici (compiler), assembler, bağlayıcı (linker), ve yerleştiricidir (locator) ve bunlar geliştirme bilgisayarı üzerinde çalışır. Bu araçlar, sadece hedef gömülü sistemde çalışabilecek çalıştırılabilir ikili dosyayı üretirler.

Her işlemci kendisine özel makine diline sahiptir, bu nedenle belirli bir hedef işlemci için program üretebilecek bir derleyici seçmek gerekir. Derleyici genellikle geliştirme bilgisayarı üzerinde çalışır. Gömülü sistem üzerinde derleyicinin çalıştırılmasının bir mantığı yoktur. Bir bilgisayar üzerinde çalışan ve başka bir bilgisayar için kod üreten bu derleyicilere çapraz-derleyici (cross-compiler) denir. Çapraz-

derleyicinin kullanımı gömülü yazılım geliştirmenin belirleyici bir özelliğidir. Örneğin GNU geliştirme platformlarının parçası olan C/C++ derleyici (gcc) ve assembler (as), çapraz-derleyici olarak yapılandırılabilir.

C/C++, assembly, ya da başka bir dilin kullanımından bağımsız olarak, çapraz-derleyicinin çıktısı, genellikle Common Object File Format (COFF) ya da Extended Linker Format (ELF) biçiminde bir nesne dosyası olacaktır (Simon, 1999). Bu, dil dönüştürme işleminin sonucunda üretilen bir dizi komut ve veri içeren özel biçimlendirilmiş bir ikili dosyadır. Bu dosyanın bir kısmı çalıştırılabilir kod içermesine rağmen, nesne dosyası doğrudan çalıştırılmak için tasarlanmamıştır.

Bir çok nesne dosyası takip eden bölümleri tanımlayan bir başlık (header) ile başlar. Her bir bölüm orjinal kaynak dosyasından türetilmiş veri yada kod bloğu içerir. Ancak, bu bloklar birbirleri ile ilişkili olacak şekilde derleyici tarafından gruplandırılır. Örneğin, tüm kod blokları *text* adı verilen bölümlerde toplanır, global değişkenler *data* adı verilen bölümlerde ilklendirilir, ve ilklendirilmeyen global değişkenler *bss* adı verilen bir bölümde toplanır.

Ayrıca, nesne dosyası içinde tüm değişkenlerin ve fonksiyonların isimlerini ve yerlerini içeren bir sembol tablosu bulunur. Ancak, tüm değişkenler ve fonksiyonlar her zaman aynı dosya içinde tanımlı olmayacağından bu tablonun bazı parçaları tamamlanmamış olabilir. Bu semboller diğer kaynak dosyalarda tanımlanmış değişkenlere ve fonksiyonlara karşılık gelir, ve bunun gibi çözümlenmemiş karşılıkları çözmek bağlayıcıya kalmıştır.

Programın çalışabilmesi için birinci adım sonunda oluşan tüm nesne dosyalarının bir şekilde birleştirilmesi gerekir. Nesne dosyaları içinde bazı değişkenler ve fonksiyon referansları tam çözümlenmemiştir. Bağlayıcının işi bu nesne dosyalarını biraraya getirmek ve, tüm çözümlenmemiş sembolleri çözümlenmektir.

Bağlayıcının çıktısı, giren nesne dosyalarının tüm kodunu ve verisini içeren yeni bir nesne dosyasıdır ve aynı nesne dosya biçimindedir. Bunu *text*, *data*, ve *bss* bölümlerini birleştirerek gerçekleştirir. Böylece, bağlayıcı çalışmasını bitirdiğinde, tüm makine dili kodu yeni dosyanın *text* bölümünde, ve tüm ilklendirilmiş ve ilklendirilmemiş değişkenler karşılıklı olarak yeni *data* ve *bss* bölümlerinde kalacaktır.

Yazılım geliştirme araçlarının otomatik olarak yaptığı şeylerden biri de başlangıç kodu koymaktır. Başlangıç kodu üst-seviye dilde yazılmış yazılımın çalıştırılmasını hazırlayan küçük bir assembly kod bloğudur. Her üst-seviye dilin kendi özel beklediği çalışma koşulları vardır. Örneğin, C ve C++ da yazılımın çalıştırılabilmesi için bir yığın (stack) alanı ve bu alanın ilklendirilmesi gerekir. C/C++ programlar için bu, başlangıç kodunun sorumluluklarından biridir.

Gömülü sistemlerin kullandığı bir çok çapraz-derleyici startup.asm, crt0.s, ya da daha farklı bir şekilde adlandırılan assembly dosyası içerir. Bu dosyanın yeri ve içeriği genellikle derleyici ile verilen dökümantasyonda tanımlanır.

C/C++ programları için başlangıç kodunun faaliyetleri genellikle aşağıdaki gibi sırasıyla tanımlanır:

1. Tüm kesmeleri etkisizleştir.
2. ROM ve RAM deki tüm ilklendirilmiş veriyi kopyala.
3. İlklendirilmemiş veri alanını sıfırla.
4. Yığın için alan tahsis et ve ilklendir.
5. İşlemcinin yığın işaretçisi (pointer) ni ilklendir.
6. Heap i yarat ve ilklendir.
7. Tüm global değişkenler için yapılandırıcıları ve ilklendiricileri çalıştır.
8. Kesmeleri etkinleştir.
9. main' i çağır.

Genellikle, başlangıç kodu main'i çağırdıktan sonra da bir kaç komut daha içerir. Bu komutlar sadece üst-seviye dilde yazılmış program sonlandığında kullanılır. Örneğin, main fonksiyonu bittiği zaman. Gömülü sistemin yapısına göre bu komutlarla işlemci durdurulabilir, sıfırlanabilir, ya da kontrol bir debugging aracına transfer edilir.

Başlangıç kodu otomatik olarak üretilmeyeceğinden, programcı kendisi bir başlangıç kodu yazmalı ve buna ait nesne dosyasını bağlama işlemine eklemelidir.

Eğer aynı sembol birden fazla nesne dosyasında deklare edilmişse, bağlayıcı devam edemez. Hata mesajı verir ve çıkar. Eğer bağlayıcı nesne dosyaları içinde sembolü bulamazsa, kütüphanelerin içinden bulmaya çalışır.

Tüm kodu ve veri bölümlerini birleştirdikten ve tüm sembol referanslarını çözümledikten sonra, bağlayıcı programın özel bir “yeniden yerleştirilebilir” kopyasını üretir. Başka bir deyişle, koda ve veri bölümüne bellek adresi tahsis edilmesi dışında program tamamlanmış sayılır.

Fakat genellikle gömülü yazılım inşa işlemi bu noktada bitmez. Gömülü sistem bir işletim sistemi içerse bile, hala tam yerleştirilmiş bir ikili (binary) dosyaya ihtiyaç vardır. Eğer bir işletim sistemi varsa, genelde onun da kod ve verisi yeniden yerleştirilebilir programın içinde bulunur. İşletim sistemi de dahil olmak üzere bütün gömülü programlama, genellikle birbirine statik olarak bağlanır ve tek bir ikili dosya olarak işletilir.

Yeniden yerleştirilebilir programdan çalıştırılabilir ikili dosyaya dönüştürme işlemini gerçekleştirir. Gömülü yazılım geliştirme adımlarının üçünün arasından en kolayıdır. Hedef platformun bellek bilgisi yerleştiriciye verilmelidir. Yerleştirici, yeniden yerleştirilebilir program içinde kod ve veri bölümlerine fiziksel bellek adresleri tahsis etmek için bu bilgiyi kullanır. Daha sonra hedef ROM a yüklenebilecek bir binary dosyayı çıktı olarak üretir.

Yerleştirici tarafından istenen bellek bilgisi bir bağlayıcı script i biçiminde ona iletilebilir. Bazen bu scriptler yeniden yerleştirilebilir program içinde kodun ve veri bölümlerinin gerçek sırasını kontrol etmek için kullanılırlar. Fakat, burada istenilen sadece sıralama kontrolü değil, ayrıca bellekda her bölümün yerinin tesis edilmesidir.

Örnek olarak, aşağıda 512K RAM ve ROM a sahip olan hayali bir gömülü sistem için bir bağlayıcı scripti görülebilir (Barr, 1999):

```
MEMORY
{
    ram: ORIGIN = 0x00000, LENGTH = 512K
    rom: ORIGIN = 0x80000, LENGTH = 512K
```

```

    }
    SECTIONS
    {
        data ram:                                /* Initialized data */
        {
            _DataStart = . ;
            *(.data)
            _DataEnd = . ;
        } >rom
        bss :                                    /* Uninitialized data */
        {
            _BssStart = . ;
            *(.bss)
            _BssEnd = . ;
        }
        _BottomOfHeap = . ;                      /* The heap starts here */
        _TopOfStack = 0x80000;                  /* The stack ends here */
        text rom :                              /* The actual instructions
*/
        {
            *(.text)
        }
    }

```

Bu skript hedef platform üzerindeki bellek bilgisini verir ve data ve bss bölümlerini RAM’de ve text bölümünü ROM da nasıl yerleştireceğini öğretir. Buna rağmen, data bölümü içindeki değişkenlerin başlangıç değeri o bölümün tanımının en sonuna >rom eklenmesi ile ROM un bir parçasını oluşturacaktır.

Alt çizgi ile başlayan tüm isimler (örneğin, _TopOfStack) kaynak kod içinde referans edilebilen değişkenlerdir. Bağlayıcı bu sembolleri girdi nesne dosyaları içinde referansları çözmek için kullanacaktır.

Örneğin, iklenendirilmiş değışkenlerin başlangıç değerlerini ROM' dan RAM içindeki data bölümüne kopyalayan bir gömülü yazılım parçası olabilir. Bu işlem için başlangıç ve bitiş adresleri, tam sayı değışkenleri _DataStart ve _DataEnd e refer edilerek sembolik olarak kurulabilir.

Build işleminin bu son adımında oluşan dosya tamamıyla yerleştirilmiş bir ikili dosyadır. Bu dosya gömülü sisteme yüklenebilir ya da salt-okunur bellek aracı içine programlanabilir. Bir önceki örnekte, bellek dosyasının boyutu 1 MB olurdu. Fakat, iklenendirilmiş veri bölümleri için başlangıç değeri ROM'da tutulur, bu dosyanın alt yarısındaki 512 KB bölümü sadece sıfırlardan oluşmuştur, dolayısıyla bu dosyanın sadece üstte kalan yarısı değerlidir.

2.7.2 Yükleme ve hata ayıklama

Geliştirme bilgisayar üzerinde tutulan çalıştırılabilir ikili dosyanın gömülü sisteme yüklenmesi ve çalıştırılması gerekir. Çalıştırılabilir iki dosya genellikle hedef platform üzerinde bellek cihazı üzerine yüklenir ve buradan çalıştırılır. Doğru araçlar kullanılarak program içinde durma noktaları kurulabilir.

Gömülü yazılımı aktarmanın en açık yolu ikili dosyayı bir salt-okunur belleğe (ROM) yüklemek ve bu yongayı hedef platform üzerine yerleştirmektir.

İdeal geliştirme senaryosunda, ROM programlayıcısı geliştirme bilgisayar ile aynı ağı bağlanır. Böylece, ROM programlama için çalıştırılabilir ikili dosyalar kolaylıkla taşınabilir. İkili dosyalar programlayıcıya taşındıktan sonra, bellek yongası uygun boyutta ve şekilde olan sokete yerleştirilir ve ekran menüsü üzerinde cihaz türü seçilir. Cihaz programlama işlemi dosyanın büyüklüğüne bağlı olarak değışir.

ROM programlandıktan sonra, platform üzerindeki sokete yerleştirilebilir. Bu işlem gömülü sistemde elektrik varken yapılmamalıdır. Enerji kesilmeli ve yonga dikkatlice yerleştirildikten sonra yeniden verilmelidir.

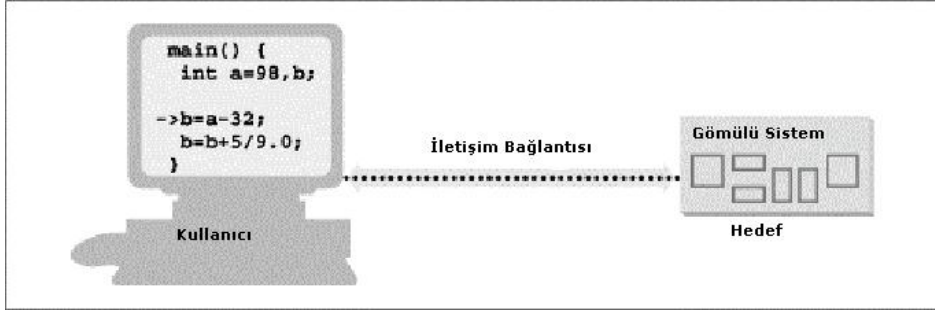
Enerji verildikten sonra, işlemci ROM’da saklanan kodu yakalamaya ve çalıştırmaya başlar. Fakat, her tip işlemcinin ilk komutunun yeri hakkında kendi kuralları olduğu unutulmamalıdır. Örneğin, Intel 80188EB işlemcisi açıldığında, fiziksel adres FFFF0h üzerinde ne saklanmışsa, onu yakalamaya ve çalıştırmaya başlar. Buna sıfırlama adresi denir, ve bu adreste tutulan komutların tümü reset kodu olarak bilinir (Barr, 1999).

Program çalışıyor gibi görünmüyorsa, sıfırlama koduyla ilgili bir problem var demektir. ROM’ a yüklenen ikili dosyanın hedef işlemcinin sıfırlama kurallarını karşıladığından emin olunmalıdır.

Uzaktan Hata Ayıklayıcı (Remote Debuggers)

Bir uzaktan hata ayıklayıcı ana bilgisayar ve hedef arasında seri port ya da ağ bağlantısı üzerinden gömülü yazılımı aktarmak, çalıştırmak ve hata ayıklamak için kullanılabilir. Uzaktan hata ayıklayıcı genellikle bir metin ya da GUI-tabanlı ana pencere ve kaynak kod, yazmaç içeriği, ve çalıştırılan programla ilgili diğer bilgiler için bir çok küçük pencereye sahiptir. Gömülü sistemlerde hata ayıklayıcı ve hata ayıklanan yazılım iki farklı bilgisayar sistemi üzerinde çalışır.

Bir uzaktan hata ayıklayıcı gerçekte iki parça yazılımdan oluşur. Birinci kısım ana bilgisayar üzerinde çalışır ve kullanıcı arabirimi sağlar. Fakat hedef işlemci üzerinde çalışan gizli bir ikinci kısım vardır ve bu bir bağlantı üzerinden birinci kısım ile iletişim yapar. İkinci kısım hedef işlemcinin alt-seviye kontrolünü sağlar ve genellikle hata ayıklama ekranı olarak adlandırılır. Şekil 2.8 bu iki parçanın nasıl çalıştığını gösterir.



Şekil 2.8 Bir uzaktan hata ayıklama oturumu (Barr, 1999, s.32)

Hata ayıklama ekranı ROM üzerinde bulunur ve hedef işlemci sıfırlandığında otomatik olarak başlatılır. Ana bilgisyara iletişim bağlantısını görüntüler ve uzaktan hata ayıklayıcısından gelen isteklere cevap verir. Bu istekler ve görüntülenen cevaplar önceden tanımlanmış bazı iletişim protokollerine uymalıdır ve genellikle alt-seviye bir yapısı vardır. Hata ayıklayıcı örnek olarak “read register x,” “modify register y,” “read n bytes of memory starting at address,” ve “modify the data at address” isteklerini verebilir.

Uzaktan hata ayıklayıcı bunun gibi alt-seviye komut dizilerini bir araya getirir ve böylece bir program aktarma, onun içinden tek-adımlarla geçme, ve breakpoint ler koyma gibi üst-seviye hata ayıklama görevlerini gerçekleştirir.

GNU buna benzer bir hata ayıklayıcısıdır (gdb). Diğer GNU araçlarına benzer şekilde, GNU’nun kendisi için tasarlanmıştır ve daha sonra çapraz-platform hata ayıklamayı gerçekleştirme özelliği verilmiştir. Böylece, desteklenen her bilgisayarda çalışan ve desteklenen her hedefin opcode ve yazmaç isimlerini anlayan bir GDB yapılandırılabilir.

Hata ayıklayıcı ekran ile hedef sistem arasındaki iletişim bayt-tabanlıdır ve bir seri bağlantı üzerinden taşıma için tasarlanmıştır. Gömülü yazılımın gelişimi süresince uzaktan hata ayıklayıcıların çoğunlukla kullanılmasının temel nedeni düşük maliyetli olmalarıdır. Gömülü yazılım geliştiricilerinin geliştirme bilgisayarı kullanmaları gereklidir. Ayrıca, bir uzaktan hata ayıklayıcının fiyatı çapraz-geliştirme araçları (derleyici, bağlayıcı, yerleştirici, gibi.) takımının maliyetine önemli bir ek getirmes. Son olarak, uzaktan hata ayıklayıcı sağlayıcıları

kullanıcı tabanında kurulmuş programlarının sayısını artırmak için genellikle hata ayıklayıcı ekranlarının kaynak kodunu vermek isterler.

Emulatörler

Uzaktan hata ayıklayıcıları gömülü yazılımın durumunu kontrol etmek ve görüntülemek için faydalıdır. Fakat, sadece bir devre-içi emulatör (ICE) üzerinde programın çalıştığı işlemcinin durumunu incelemeye olanak verir. Gerçekte, ICE hedef platform üzerinde işlemcinin yerini alır. Hedef işlemcisinin, RAM' in, ROM'un kopyası ve kendi gömülü yazılımı ile kendisi bir gömülü sistemdir. Bunun sonucunda, devre-içi emulatörler genellikle hedef donanımdan daha pahalıdır. Fakat, çok güçlü araçlardır, ve zor bir hata ayıklama yerinde işi daha iyi yapabilecek bir araç yoktur.

Bir hata ayıklayıcı ekranı gibi, bir emulatör kullanıcı arayüzü için uzaktan hata ayıklayıcı kullanır. Bazı durumlarda her ikisi için de aynı hata ayıklayıcı kullanmak mümkündür. Fakat emulatör hedef işlemcinin kopyasına sahip olduğundan gerçek zamanlı olarak işlemcinin durumu kontrol edilebilir ve görüntülenebilir. Bu emulatörün donanım breakpointleri ve gerçek-zamanlı izleme gibi güçlü hata ayıklama özelliklerini desteklemesini sağlar.

Hata ayıklama ekranı ile program içinde kesme noktaları kurulabilir. Yazılım kesme noktaları, sadece belli adreslerdeki komutları yakalar ve program akışının o noktada durmasını sağlar. Emulatörler ayrıca donanım kesme noktalarını da destekler. Donanım kesme noktaları çok çeşitli olaylara cevap vermek için çalışmayı durdurabilir. Bu olaylar sadece komut yakalamayı içermez, ayrıca bellek ve giriş/çıkış okuma/yazma ve kesmeleri içerir.

Devre-içi emulatörlerin bir diğer faydalı özelliği gerçek-zamanlı izlemedir. Genellikle, bir emulatör, çalıştırılan her işlemci çevrimi ile ilgili bilgi saklamak için, genel-amaçlı RAM'in büyük bir bölümünü kendisine ayırır. Bu özellik, olayların işleniş sırasının izlenmesine olanak sağlar.

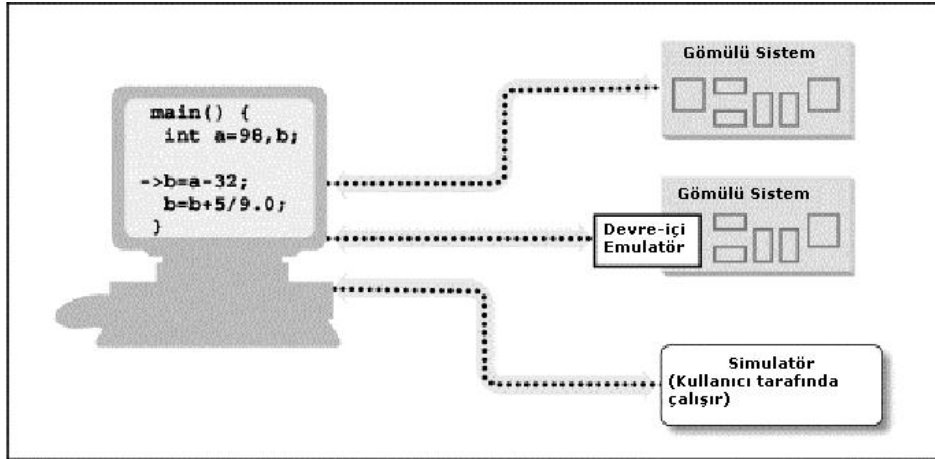
ROM Emulatörleri

ROM emulatörleri salt-okunur bellek yongalarını taklit eder. ICE gibi, hedefe bağlanan ve ana bilgisayar ile iletişim kuran bir gömülü sistemdir. Fakat, hedef bağlantısı bir ROM soketi ile olur. Gömülü işlemciye göre bu, diğer salt-okunur bellekler gibi görünür. Fakat uzaktan hata ayıklayıcıya göre bu, bir hata ayıklama ekranı olarak görünür.

ROM emulatörleri hata ayıklama ekranları üzerinde bir çok avantaja sahiptir. Birincisi, hata ayıklama ekranı kodu özel hedef donanımına taşınmak zorunda değildir. İkincisi, ROM emulatörleri ana bilgisayara kendi seri yada ağ bağlantıları yolu ile bağlanır, böylece hedefin genellikle sınırlı olan kaynaklarını kullanmaya gerek kalmaz. Son olarak ROM emulatörü, doğrudan orijinal ROM'un yerine takılır. Bu nedenle hata ayıklama kodu tarafından hedefin belleği hiç kullanılmaz.

Simulatörler ve Diğer Araçlar

Bir simulatör, hedef işlemcinin komut setini ve fonksiyonelliğini taklit eden, tamamen ana bilgisayarda çalışan bir programdır. Kullanıcı arabirimi uzaktan hata ayıklayıcınıninki ile aynı ya da benzerdir. Şekil 2.9'da görüldüğü gibi, bir hata ayıklayıcı arayüzü simulator bağlantısı için de kullanılabilir.



Şekil 2.9 En ideal ortam: bir genel hata ayıklayıcı arayüzü (Barr, 1999, s.37)

Simulatörler, bir çok dezavantaja sahip olmalarına rağmen, bir projenin erken gelişim aşamalarında, henüz çalışılabilecek gerçek bir donanım yokken oldukça fayda sağlarlar. En büyük dezavantajı sadece işlemciyi simule etmesidir. Fakat gömülü sistemler çoğu zaman bir ya da daha fazla yardımcı cihaz içerirler. Bu cihazlarla etkileşimi işlemci simulatörü ile taklit etmek zordur. Gerçek gömülü sistem donanımı elimizde olmadan işlemci simulatörü ile pak fazla bir şey yapılamaz.

Hedef donanım hazır olduğu zaman, özellikle donanım hata ayıklaması sırasında *logic analyzer* ve osiloskoplar kullanılır. Kart üzerindeki işlemci ve diğer yongaların etkileşimi bu şekilde incelenebilir. Bu cihazlar sadece işlemcinin dışındaki sinyalleri gösterdiği için, bir hata ayıklayıcı ya da emulatör gibi program akışını gösteremezler. Bu durum bu araçların tek başına kullanımını anlamsız hale getirir. Ancak bir hata ayıklama yazılımı ile birlikte kullanılırlarsa bir anlam ifade ederler.

Logic analyzer, sayısal donanım sorunlarını çözmek için tasarlanmış özel bir laboratuvar gerecidir. Onlarca hatta bazı durumlarda yüzlerce girişi olabilir. Bu girişler, bağlı olan elektrik sinyallerinin mantıksal seviyesini 1 ya da 0 olarak gösterir.

Osiloskoplar da yazılım hata ayıklama aracı olarak kullanılabilirler. Bazı durumlarda herhangi bir noktadaki gerilimi göstermek için logic analyzerlardan daha kullanışlıdır. Çünkü, ayarlanmaları kolaydır ve

daha abuk sonu verirler. Ancak giriř sayıları ok daha azdır (en fazla drt).

2.7.3 PC' de alıřmaktan farkları

Temelde, bellek ve iřlemci kullanımı aısından PC ve gml sistemler arasındaki fark olduka belirgindir. Bunun yanında, program geliřtirme, program saklama, ve hata ayıklama iřlemleri de farklılık gsterir. Gml sistem ortamında yazılım geliřtirmenin gzle grlr farkları izelge 2.2'de listelenmiřtir.

Programcı, kodu kısıtlı sistem kaynaklarını en iyi kullanacak řekilde yazmalıdır. Gml sistemlerin kısıtlı kaynakları, oėu zaman yeni geliřtirilen yazılım mhendisliėi kavramlarının gml sistemler zerinde uygulanmasını imkansız hale getirir ya da kavramlar byk oranda biim deėiřikliėine uėratılarak kullanılabilir.

rneėin tasarım desenleri (Design Patterns), bir ok noktada genel programlamayı kolaylařtıran teknikler sunar. Ancak bunların bir oėunun olduėu gibi gml sistemlere uygulanması, sonuta kullanılan RAM miktarının artarak sistem maliyetinin ykselmesine neden olabilir.

Çizelge 2.2 PC ve Gömülü Sistem arasındaki farklar

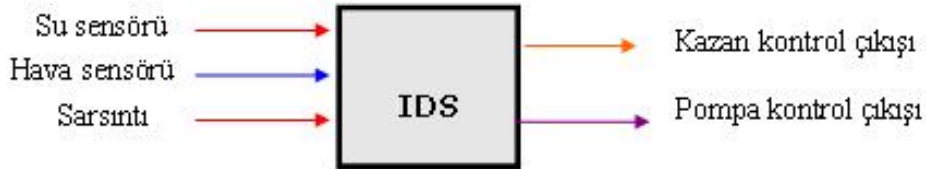
	PC	Gömülü sistem
Program alanı	Hard disk (~GB)	Flash (~MB) (IDS'de 20K)
Bellek	İstenildiği kadar bellek ayrılabilir.örnek: Malloc(100MB)	Bellek alanı sınırlıdır (IDS'de 256bayt)
Hız	İşlemciye göre değişir. Örnek: 3.5GHz	Hızı daha düşüktür. (IDS'de 12MHz)
Program geliştirme	PC üzerinde yazılır, derlenir, çalıştırılır.	Çapraz platform çalışılır. PC üzerinde yazılır, derlenir, ROM'a yazılır ve çalıştırılır.
Hata ayıklama	Yazılım içine durma noktaları konularak adım adım izlenebilir.	Seri kapı üzerinden cihaza bağlanarak aynı işlem yapılabilir fakat, daha yavaş çalışır. Ya da sadece <i>printf</i> çıktıları alınabilir.

3. ISITICI DENETİM SİSTEMİ (IDS)

Bu tez kapsamında, gömülü sistem yaklaşımı ile bir denetleme mekanizması geliştirilmiş ve tasarım, kalorifer denetleme sistemine uyarlanmıştır. Kalorifer sistemi denetiminin seçilmesindeki temel nedenler; sistemin kolay uygulanabilir olması, gerçekten ihtiyaç duyulan bir sistem olması ve tez bitiminde ortaya çıkan cihazın ürüne dönüştürülebilir olmasıdır. Bu amaçla, 8-bit/8051 tabanlı mikrodenetleyici mimarisi üzerinde bir donanım geliştirilmiştir. Yazılım C programlama dilinde geliştirilip derlendikten sonra mikrodenetleyici üzerindeki Flash belleğe yüklenmiştir.

3.1 Sistemin Özellikleri

Isıtıcı Denetim Sistemi (IDS), bir kalorifer sisteminin bütün modüllerini tek başına kesintisiz kontrol edebilen bir sistemdir. Basitçe tanımlamak gerekirse, Şekil 3.1’de de gösterildiği üzere, sistem girdileri hava sıcaklığı, su sıcaklığı ve sarsıntı sistem çıktıları ise, kazan, pompalar ve güç kaynağının denetlenmesidir.



Şekil 3.1 Sistemin genel özellikleri

Sistem otomatik programlama konumunda çalıştırıldığında, hava sensöründen aldığı dış sıcaklık değerini belleğinde bulunan sıcaklık tablosundaki değerle eşleştirir. Buna denk gelen su sıcaklığını okur ve

sistemi çalıştırır. Çalışma sırasında her 1 dakikada hava ve su sıcaklığı değerlerini okuyarak kazanlar ve pompanın çalışıp çalışmayacağına karar verir. Eğer cihazlar çalışacaksa çalışma kapasitesini (kazanların tek mi çift mi çalışacağı, pompa çalışma durumu, vs...) ayarlar.

Sıcaklık tablosuna göre hava sıcaklığı 18 derecenin üzerinde olursa ısıtma işlemi gerçekleşmez. Bu, sistemin çalışmaya başlama derecesidir. Bu sıcaklık değeri isteğe bağlı olarak 15 – 25 derece arasında bir değere ayarlanabilir.

Sistemin, kurulu olduğu binanın ve tesisatın özelliklerine uygun hale getirilebilmesi için, sadece yetkili kullanıcıların girebileceği bir kalibrasyon menüsü tasarlanmıştır. Bu menüdeki ayarlar sayesinde ortam farkından kaynaklanabilecek hataların dengelenmesi hedeflenmiştir.

Sistem çalışmaya başladığında pompaları devreye sokmadan önce kazanı belirlenen süre boyunca çalıştırarak ön ısıtma sağlar. Ön ısıtma sistemin sağlığı ve toplam yakıt harcaması açısından gereklidir. Suyun sıcaklığı arttıkça verimi artar (Dixon, 1998). Bu sayede tam ısınmamış suyun sistemde dolaşmasının yaratacağı kayıplar azaltılmaktadır.

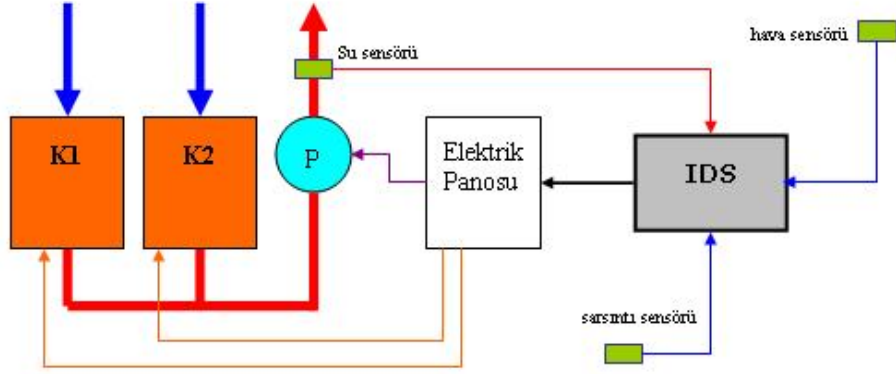
Sistem ısıtma işlemi durduktan sonra, içerisindeki sıcak su 30 derecenin altına düşünceye kadar bu suyu dolaştırmaya devam eder. Su sıcaklığı 30 derecenin altına düştüğünde sistem pompaları da kapatır. Bu sıcaklık seviyesi, kalibrasyon menüsünde ayarlanabilmektedir. Böylece sistemdeki sıcak suyun en verimli şekilde kullanılması hedeflenmiştir.

IDS, kalorifer yakma mevsimi dışında da haftada bir defa yarım saat, sadece pompaları çalıştırarak, pompalarda uzun süre hareketsiz kalmaktan kaynaklanabilecek mekanik arızaları en alt seviyeye indirmeyi amaçlamaktadır.

Elektrik kesildikten sonra tekrar geldiğinde sistem kaldığı yerden devam eder; saat pil ile beslendiği için, sistem, zaman bilgisini kaybetmez.

IDS, elektrikle kontrol edilebilen kalorifer sistemleri (sıvı yakıt, doğal gaz ya da elektrikli) için tasarlanmıştır. IDS doğrudan kalorifer sistemine giden gücü kontrol eder. Su sıcaklığını okuması için, su sıcaklık algılayıcısının kazan çıkış borularının üzerine yerleştirilmesi yeterlidir. Kurulu sistemler üzerinde zaten var olan termostatlar sistem güvenliğini artırır. IDS'nin takıldığı sistemlerde kazanların kendi

termostatları en yüksek seviyeye ayarlanır ve geri kalan kontrol IDS tarafından yapılır.



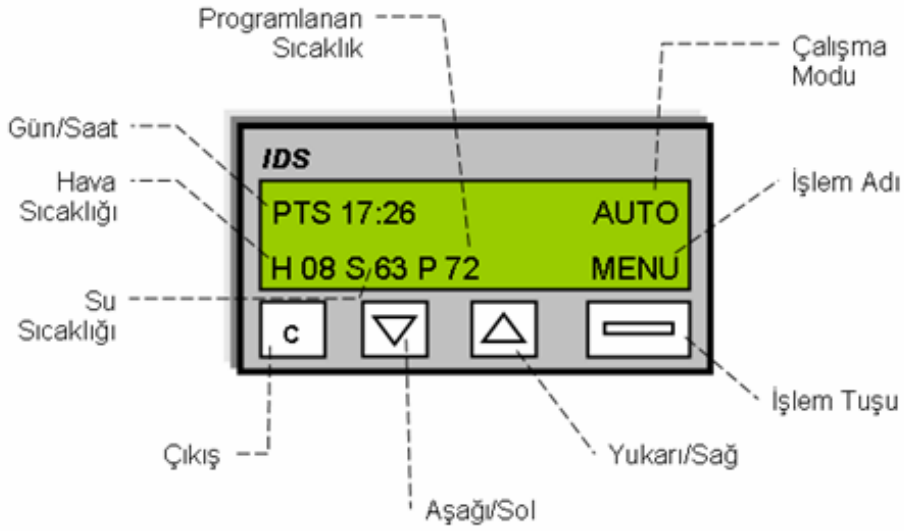
Şekil 3.2 Sistem Kurulumu

Şekil 3.2'de IDS'nin var olan bir sisteme nasıl entegre edildiği anlatılmaktadır. Şekilde gösterilen sistem iki kazan (K1, K2) ve bir pompadan (P) oluşmaktadır. Yeşil kutucuklar su ve hava sıcaklığı ile sarsıntı sensörlerini göstermektedir. Su sıcaklık sensörü kazan çıkışına en yakın sıcak su borusu üzerine yerleştirilir. Bu yer muhtemelen pompa çıkışı olabilir. IDS'den alınan kontrol çıkışları ile, elektrik panosu içinde bulunan ve sistemin açılıp kapanmasını sağlayan kontaktörlerin kontrolü sağlanır.

3.2 Kullanıcı Arabirimi

Kullanım arabirimi bir LCD ekran ve 4 adet tuştan oluşmaktadır. Sistemin kontrolü ve programlanması küçük bir ekran ve bunun altında konumlandırılmış dört adet tuş ile gerçekleştirilmektedir. Ekran üzerinde zaman (saat/gün), sensörlerden okunan hava sıcaklığı, su sıcaklığı, programlanan sıcaklık, seçili olan çalışma durumu, ve menu bilgisi bulunmaktadır. Şekil 3.3, IDS' nin ana ekran görüntüsünü

göstermektedir. Ekranda kullanıcının sistem hakkında anlık ihtiyacı olabilecek bütün bilgiler bulunmaktadır.



Şekil 3.3 Ana ekran görüntüsü

PTS : haftanın gününü gösteren bilgi

17:26 : sistem saati

AUTO : sistemin çalışma durumu (otomatik, program, elle, kapalı)

H 08 : sistemin hava sensöründen okuduğu sıcaklık değeri

S 63 : sistemin kazana çıkışına en yakın boruya bağlı su sensöründen okuduğu su sıcaklığı değeri

P 72 : sistemin programlandığı hedef sıcaklık değeri

MENU: Onay tuşu fonksiyonu, menü sistemine giriş.

Menü sistemi oldukça sade ve anlaşılabilir tasarlanmıştır. İşlem tuşu ile menünün alt kısımlarında ilerlenebilmektedir. Menü içerisindeki

değişiklikler Aşağı/Sol, Yukarı/Sağ tuşları ile yapılabilmektedir, Çıkış tuşu ile bir alt menüden bir üst menüye geçiş yapılabilmektedir.

Şekil 3.4' deki örnekten görülebileceği gibi günün 24 saati 4 farklı zaman dilimine ayrılabilir, ve bunların her biri ayrı ayrı programlanabilir. Haftalık programlama 3 ayrı grupta yapılabilmektedir: hafta içi, Cumartesi, ve Pazar.

Zaman dilimi	Hafta içi	cumartesi	pazar
00:00 - 06:00	kapalı	düşük	düşük
06:00 - 11:00	normal	normal	normal
11:00 - 17:00	düşük	düşük	normal
17:00 - 00:00	yüksek	yüksek	yüksek

Şekil 3.4 Örnek sıcaklık programlama tablosu

Her bir zaman dilimi düşük, normal, yüksek, ve kapalı olmak üzere 4 ayrı şekilde programlanabilir. Düşük sıcaklık seçeneği, normal sıcaklık seçeneğinden bir fark değeri çıkarılarak sağlanır. Yüksek sıcaklık değeri de normal sıcaklık değerine bir fark değeri eklenerek sağlanır. Bu fark değerleri aynıdır ve kalibrasyon menüsünde ayarlanır.

Sistem 4 durumdan birisinde bulunmaktadır: Otomatik, elle, programlanmış ve kapalı.

Otomatik mod: Sistemin otomatik çalışabilmesi için termostat en yüksek seviyeye ayarlanır. Böylece su sıcaklığı IDS tarafından kontrol edilir. Müdahale etmek gerekmeksizin sistem kendi kendine çalışır.

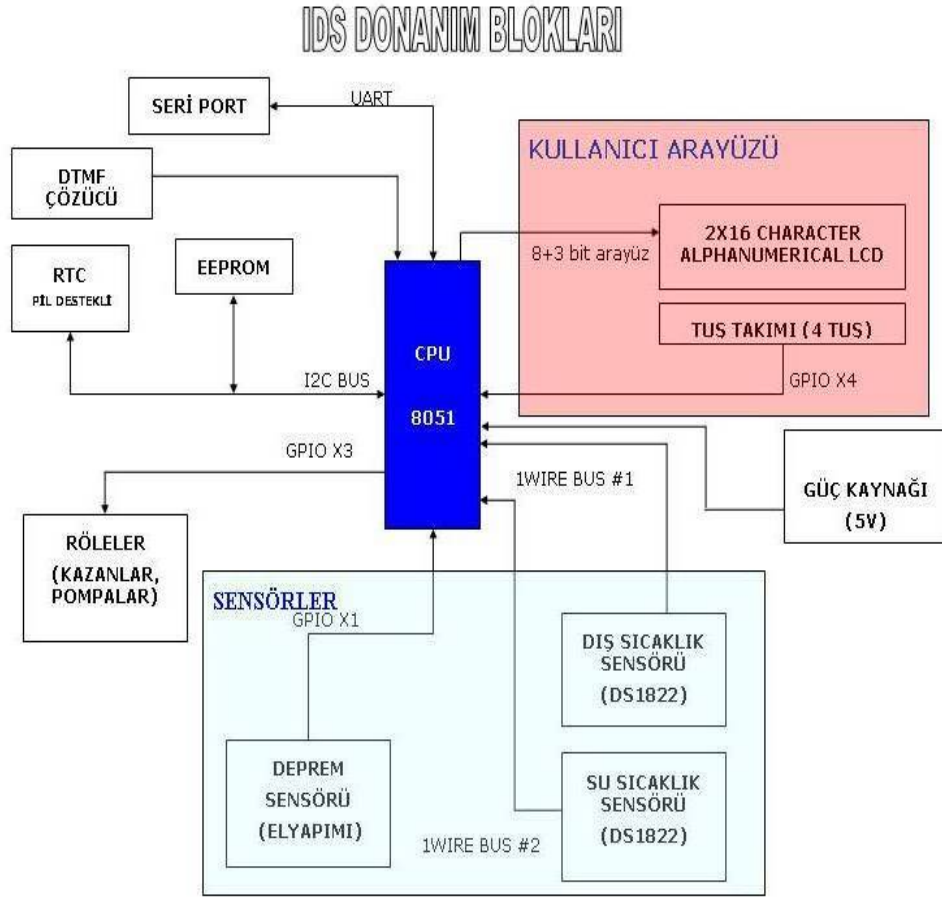
Elle: Bu mod seçili iken sistemin kontrolü cihaz üzerindeki termostat ve saat tarafından yapılır. Böylece sistem tamamen devre dışı bırakılır.

Zamana bağlı programlayıcı: Sistem, her günün 4 ayrı zaman diliminde haftalık programlanması ile çalışır. Çalışma düzeni tamamen programlanmış değerlere bağlıdır.

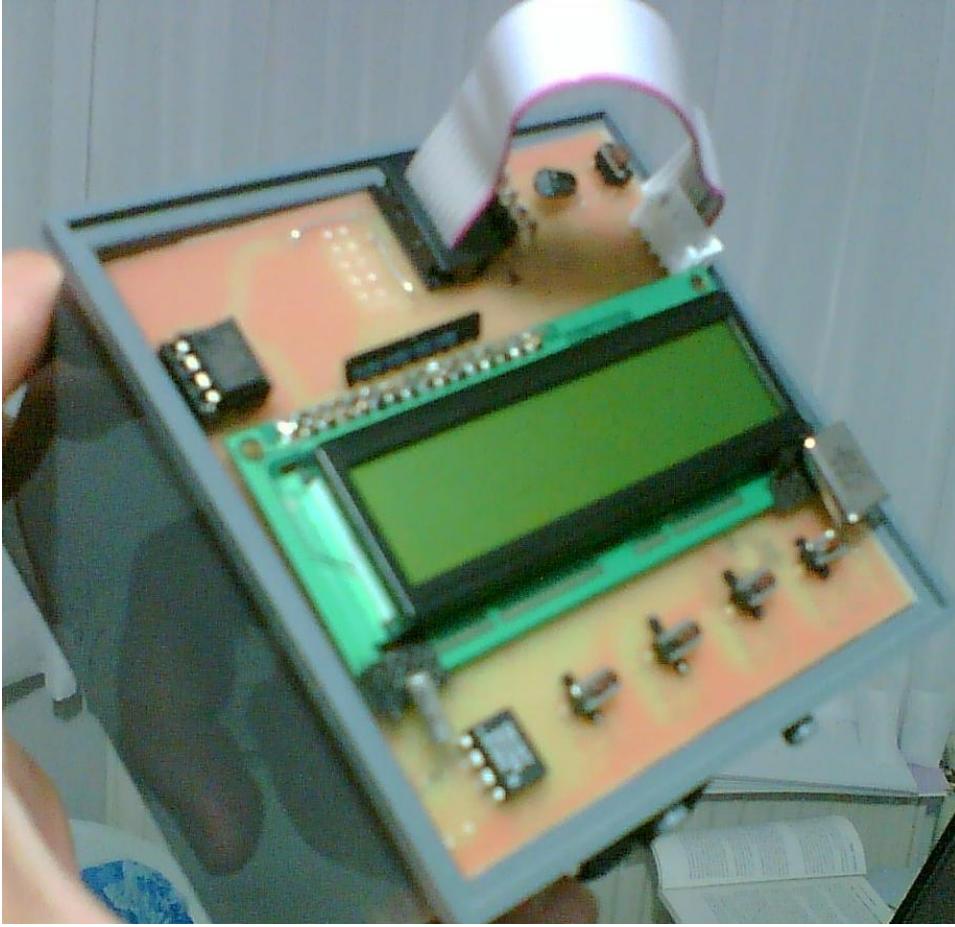
Kapalı: Kalorifer sistemi kapalı konumdadır. Kapalı konumuna geçmek için sistem güç ünitesini kapatır.

3.3 IDS Donanım Modülleri

IDS, sistem maliyeti ve içine konulacağı kutunun hacmi düşünülerek 8 bitlik bir mimari ile üç adet kart üzerinde tasarlanmıştır. Kartlar anakart, güç devresi ve rölelerin bulunduğu giriş-çıkış kartı ve telefon devresinin bulunduğu telefon kartıdır. Telefonla erişim biriminin kart üzerinde tasarlanmış olması sistemin modülerliğini artırmaktadır. Donanım şeması Şekil 3.5’ de verilmiş ve öğeleri anlatılmıştır. Anakartın ön görünüşü ve kutunun içine oturmuş hali Şekil 3.6’da gösterilmiştir. Şekil 3.7’de görülen ve kutu içinde dik biçimde yerleştirilmiş olan kart, kontrol kartıdır ve üzerinde röleler, güç kaynağı ve bağlantı noktalarını bulundurmaktadır. IDS’nin anakartının küçük olması nedeniyle mikrodenetleyici kartın arka yüzüne Şekil 3.8’de görüldüğü gibi yerleştirilmiştir. Çalışır durumdaki IDS’nin ana ekran görüntüsü Şekil 3.9’da gösterilmiştir.



Şekil 3.5 IDS donanım blokları



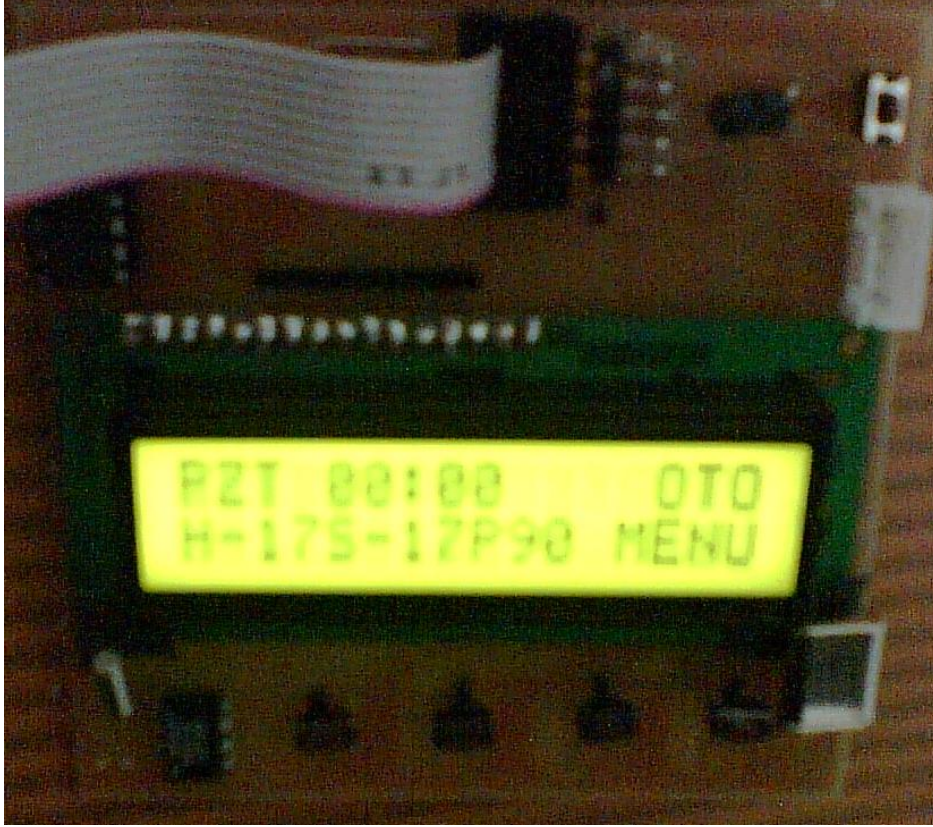
Şekil 3.6 Ana kart ön yüz



Şekil 3.7 Kontrol kartı



Şekil 3.8 Ana kart arka yüz



Şekil 3.9 Ana ekran görüntüsü

İşlemci olarak 8051 tabanlı Atmel 89C55WD mikrodenetleyicisi kullanılmıştır. Yonganın içinde 20KB Flash bellek ve 256 byte RAM bulunmaktadır. Sistem saati 11.0592 MHz 'lık bir kristal tarafından sağlanmaktadır. Ayrıca içinde bulunan *watchdog* zamanlayıcısı sayesinde yazılımın çökmesi durumlarında sistemi yeniden başlatabilir. 8051 bir çok firma tarafından desteklenmesi ve güçlü bir C derleyicisinin olması nedeniyle de tercih nedeni olmuştur.

Sıcaklık Sensörleri: Sistemde su ve hava sıcaklıklarını ölçen iki adet sıcaklık sensörü bulunmaktadır. Sıcaklık ölçümü için Dallas DS1822 bir-telli sayısal sıcaklık sensörleri seçilmiştir. Cihazla iletişim için tek bir kapı bacağı gerektiğinden bir-telli yol kullanımı oldukça pratiktir. İletişim protokolü darbe genişliklerine dayanır. Sıcaklık yarım derece

hassasiyetle ölçülebilmektedir. Bu yonga -55 ile +125 derece arasındaki sıcaklıklarda kullanılabilir. Bir-telli yol protokolü, her ne kadar aynı hat üzerine birden fazla cihaz konulmasına olanak sağlasa da, yazılım karmaşıklığını ve boyutunu azaltmak adına her sensör kendisine ait bir yolda çalışmaktadır.

Röleler: Pompalar ve kazanlar gibi sıcaklık sistemini kontrol etmek için kullanılırlar. Sistem üzerinde 3 adet röle vardır. 10A kapasiteli, 12V kumandalı Omron G5LE tipi röleler seçilmiştir. Rölelerin çıkış kapasitesi ve şekli, bağlandığı sistemi destekliyorsa, sistem elemanları doğrudan IDS'nin çıkışına bağlanabilir. Ancak çıkışların doğrudan bağlantıya uygun olmadığı durumlarda, araya kontaktörler eklenerek uygun hale getirilmelidir.

Ekran: Kullanıcı arayüzü ekranı için 2x16 karakter boyunda alfanümerik LCD kullanılır. Bu, ayrıca, yazılım tarafından kontrol edilebilir bir arka ışığa sahiptir. LCD'nin denetimi 8 bitlik veri yolu üzerinden yapılmaktadır.

Tuşlar: Cihazın üzerinde kullanıcı etkileşimi için 4 adet tuş bulunmaktadır. Bu tuşlar doğrudan mikrodenetleyicinin girişlerine bağlıdır. Tuşların taranması işlemi her 100ms'de bir gerçekleşir.

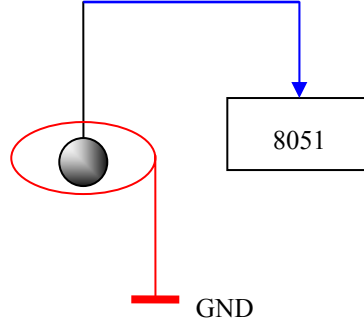
EEPROM: Kullanıcı ayarlarını, konfigürasyon ve kalibrasyon bilgilerini tutan, 256 bayt boyunda bir 24C02 EEPROM kullanılmıştır. EEPROM, I2C yolu üzerindedir ve seri olarak denetlenir.

Saat (RTC): Gerçek zamanlı saat olarak Dallas DS1307 yongası seçilmiştir. Bu yonga I2C yolu üzerindedir ve seri olarak denetlenmektedir. Bir pille beslenerek, sistem kapalıyken de çalışmaya devam eder. Bu şekilde sistem saati elektrik kesintilerinde kaybolmaz. Ayrıca üzerinde bulunan durağan bellek de genel amaçlı bellek olarak kullanılabilir.

DTMF Çözücü: Telefon tuşları ile bir telefon hattı üzerinden sistemin kontrolünün yapılabilmesi için kullanılır. Telefon hattından gelen DTMF seslerini çözerek 4 bitlik veriler halinde işlemciye bildirir. Bu iş için Mitel 8870 yongası seçilmiştir. Piyasada kolay bulunur ve fiyatı uygundur.

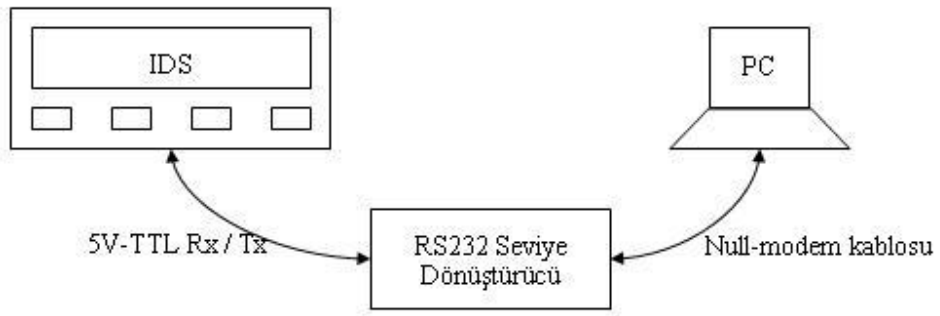
Deprem Sensörü: Sistemde el yapımı bir deprem sensörü kullanılmıştır. Böylece deprem sırasında sistem kendisini kapatır. Deprem sensörü basit olarak, sallandığında birbirine çarpan bir metal

ubuk ve bir halkadan oluřur. Bunlar birbirine arptıka oluřan darbeler mikrodeneleyici tarafından sayılır ve sınır deęer ařıldığında IDS deprem olduęuna karar vererek sistemi gvenlik amacıyla kapatır. Deprem sensr řekil 3.10’da gsterilmiřtir.



řekil 3.10 IDS deprem sensr

Seri Kapı (Port): Sistemle haberleřmemizi ve geliřtirme sırasında yazılımda kullanılan izleri (trace) grntlememizi saęlar. IDS’nin seri portunu bilgisayara baęlamak iin apraz baęlanmış bir sıfır-modem (null-modem) kablosu kullanılabilir. Haberleřme hızı 9600 baud olarak belirlenmiřtir ve akıř denetimi yoktur. IDS’nin seri port ıkıř seviyesi TTLdir ve bilgisayara baęlanmadan nce TTL-RS232 seviye dnřm yapılmalıdır. řekil 3.11 IDS’nin bilgisayarla baęlantısını gstermektedir.



Şekil 3.11 IDS bilgisayar bağlantısı

3.4 IDS Yazılım Ayrıntıları

IDS yazılımı yeniden kullanılabilirlik, yapılandırılabilirlik, esneklik ve taşınabilirlik ilkeleri göz önünde bulundurularak C programlama dilinde geliştirilmiştir. Bu ilkelerden en çok taşınabilirlik üzerinde durulmuş ve yazılım bir çok platforma, uygulamada herhangi bir değişiklik yapmaksızın kolaylıkla taşınabilecek şekilde geliştirilmiştir.

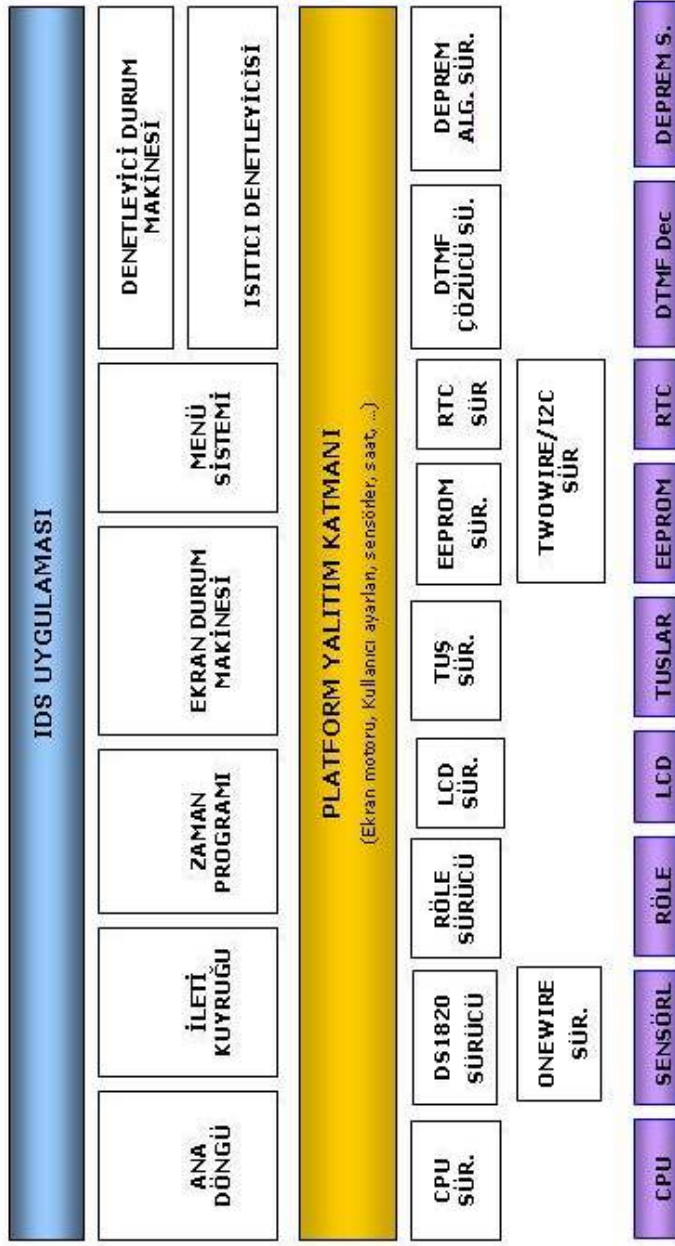
Sistem tasarımında, hedef platform olarak seçilen 8051 tabanlı mikrodenetleyicinin kaynak sıkıntıları göz önünde bulundurularak işletim sistemi kullanılmamıştır. Bunun yerine, işlerin paralel çalışması anlamında işletim sistemi etkisini veren, *round robin* modeliyle çalışan sonsuz bir döngü, kesmeler ve ileti kuyruğundan oluşan bir model tasarlanmıştır. Bu model sayesinde işler büyük oranda paralelde çalışıyormuş gibi görünür ve birbirinden ayrı olan modüller diğerlerinin çalışmasını engellemez.

Oluşan olaylar modüllere, senkron fonksiyon çağrılarını yerine ileti kuyruğu üzerinden gönderilen iletiler yoluyla bildirilirler. Bu şekilde bütün modüllerin olaylardan haberdar olması sağlanırken, her modül oluşan olay karşısında nasıl davranacağına kendisi karar vermektedir. Ayrıca modüllerin bir olayın oluşumunu beklemeleri için sonsuz döngüye girmelerine gerek yoktur.

Ancak *round robin* gibi çevrimsel çokluişlem modellerinde programcının sorumluluğu, işletim sistemi olması durumuna göre daha fazladır. Döngü çevrim süresini en aza indirmek için arka arkaya çağırılan işlerin hızlı bir şekilde bitmesi gerekir. Döngü çevriminin gereğinden fazla uzaması durumunda ileti kuyruğunda taşmalar yaşanabilir. Bu da sistemde çalışan işlerin, oluşan bazı olayları kaçırmaları anlamına gelir. IDS için kabul edilebilecek en uzun döngü çevrim süresi, sistem saati olarak çalışan zamanlayıcı kesmelerini kaçırmaması için, 50 milisaniyedir.

Sistem temel olarak iki ana modül üzerine kurulmuştur. Bunlar, Isıtıcı Denetim Sistemi (HCS) ve Menü Alt Sistemi (MSS) dir.

IDS Yazılım Blokları



Şekil 3.12 IDS yazılım blokları

3.4.1 Modüllerin kısa açıklamaları

Şekil 3.12’de görülen IDS yazılım bloklarının açıklamaları aşağıda verilmiştir:

IDS Uygulaması (HCS): Bu blok uygulamanın ana giriş noktasıdır. Diğer bütün modüllerin beraber çalışmasından sorumludur. Sistem durum makinesi bu blok içinde yer alır.

Ana Döngü: Bu, cihaz açık olduğu süre boyunca çalışan sonsuz döngüdür. Tuşların okunması, ekranın tazelenmesi gibi periyodik işlerden sorumludur. Ayrıca sürücülerden gelen olayları toplarlar ve bunları iletilere dönüştürdükten sonra ileti alıcılarına gönderir.

İleti Kuyruğu: İleti kuyruğu ilk-giren-ilk-çıkart mantığıyla çalışan bir kuyruktur. Boyu sabittir ve derleme sırasında belirlenir. SendMessage() ve GetMessage() adlarında iki temel fonksiyona sahiptir. Olay kaynakları ileti gönderir ve alıcılar iletileri alır. İletiler bir tanımlayıcı ve bir de parametre olmak üzere iki bölümden oluşurlar. İletilerin hedefi yoktur. Sistemdeki bütün mesaj alıcılarına yayınlanırlar.

Zaman Programı: Isıtıcı sisteminin haftalık programlanmasından sorumludur. Günleri, isteğe uygun şekilde ayarlanabilen zaman dilimlerine böler. Sistemin kolay kullanılabilirliği açısından gün sayısı hafta içi, cumartesi ve pazar olmak üzere üçle sınırlandırılmıştır. Ayrıca her zaman dilimi kapalı, normal, yüksek, ve düşük olacak şekilde programlanabilmektedir.

Ekran Durum Makinesi: Ekran durum makinesi sistem durumuna bağlı olarak ekranın doğru şekilde tazelenmesinden sorumludur.

Menü Sistemi: Menü sistemi kullanıcının sistem ayarlarını değiştirebildiği görsel bir arabirim sağlar. Aşağı, yukarı, iptal ve onay olmak üzere 4 adet panel tuşu tarafından kontrol edilir. Onay tuşu ekranın sağ alt köşesine yerleştirilmiş olup, fonksiyonu ekranda gösterilir.

Denetleyici Durum Makinesi: DDM ısıtıcı sisteminin kalbidir. Dış sıcaklığa, sistem durumuna ve programa bağlı olarak kazanların ve pompaların uygun şekilde çalıştırılmasından sorumludur.

Isıtıcı Denetleyicisi: Bu modül alttaki ısıtıcı sistemini DDM’den yalıtım için tasarlanmıştır. Kısacası DDM altında olan fiziksel ısıtıcı

sistemi hakkında bir bilgiye sahip değildir. Sadece ısıtıcı denetleyicisine ısıtma gücünü azaltması ya da artırması gerektiğini söyler. Gerçek ısıtıcı sistemi bir ya da daha fazla kazandan oluşabilir. Kazanların kumanda edilmesinden ısıtıcı denetleyicisi sorumludur.

3.5 Yazılım Ayrıntıları

Bu bölümde IDS'nin yazılım ayrıntıları modüller ve işleyiş açısından ele alınmıştır.

3.5.1 Program akışı

Sistem bir sonsuz döngü, ileti kuyruğu ve zamanlayıcı kesmelerinden oluşan bir model üzerine kurulmuştur. Sistem çalışmaya ilk başladığında donanım ve yazılıma ait iklendirmeler yapıldıktan sonra sonsuz döngüye girer.

Sonsuz döngünün üç ana işlevi vardır. Bunlar;

- Periyodik işleri gerçekleştirmek.
- Sürücülerden gelen olay bilgilerini ileti kuyruğuna eklemek
- İleti kuyruğundan gelen iletileri ilgili yerlere göndermek.

Sistemin zamanlaması 50ms'lik kesmeler üreten bir zamanlayıcı tarafından gerçekleştirilmektedir. Her 50 ms'lik kesme sonucunda ileti kuyruğuna bir adet MSG_SYSTEM_TICK iletisi eklenir. Bunun yanında sistem saatinin güncellenmesi ile her zaman olayına ait iletiler de benzer şekilde ileti kuyruğuna gönderilir. Örneğin bir yazılım parçasının bir sonraki saniyenin geldiğini anlaması için, bir önceki saniye bilgisini durağan olarak saklamasına gerek yoktur. Sadece MSG_NEW_SECOND iletisini beklemesi yeterli olacaktır. Bu da sistemin bellek kullanımını azaltan bir faktördür. Zaman olaylarıyla ilgili iletiler aşağıdaki gibidir.

```

#define MSG_SYSTEM_TICK      0x20
#define MSG_TICK_100         0x21
#define MSG_TICK_200         0x22
#define MSG_NEW_SECOND       0x23
#define MSG_NEW_MINUTE       0x24
#define MSG_NEW_HOUR         0x25
#define MSG_NEW_DAY          0x26

```

Sistemde saatin güncellenmesi, ekranın tazelenmesi ve tuşların taranması her zamanlayıcı tikiinde periyodik olarak yapılır. Bu olayların sonucunda ortaya çıkan iletiler de yine ileti kuyruğuna gönderilir.

Kesmelerin uzun sürüp sistemin geri kalanını durdurumaması için asenkron işlenmeleri sağlanmıştır. Bu bağlamda her kesme fonksiyonu, o kesmenin geldiğini ifade eden bir bayrak kaldırır. Ana döngü bu bayrakları gördüğü zaman ilgili işlemleri yapar ve ileti kuyruğuna ilgili iletiyi ekler. Bu sayede sistemdeki modüllerin geri kalanı da gelişen olaylardan haberdar edilirler. Kendisine gelen iletilerin nasıl işleneceği, ya da işlenip işlenmeyeceği, modülün sorumluluğundadır ve içinde bulunduğu duruma göre farklılıklar gösterir. Örneğin menü sistemini kontrol eden modül, menü kapalı iken gelen bütün iletileri göz ardı eder. Bu şekilde modüllere serbestlik getirilirken, aynı anda oluşan sistemin bir yerinde oluşan bilginin, bütün sisteme ulaşması da sağlanmış olur.

Ana döngünün bir diğer işi de ileti kuyruğundan aldığı bir iletiyi ileti alıcılarına göndermektir. Sistem tasarımında iki adet ileti alıcısı bulunmaktadır. Bunlar IDS ana modülü ve menü kontrol sistemidir. İletiyi alan modül, o iletiyi kullanıp kullanmadığını geri dönüş değerinde belirtir. Bu bilgi doğrultusunda ana döngü ekranın hangi modüle geçmesi gerektiğine karar verir.

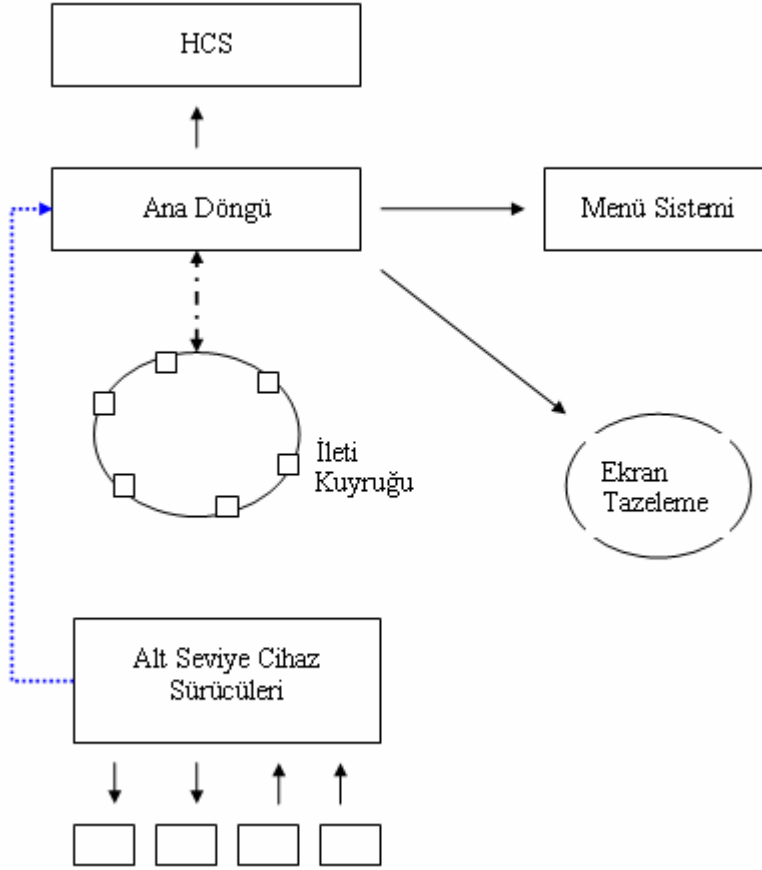
Şekil 3.13’ de gösterilen program akışından görülebileceği gibi, ana döngü tarafından alınan ileti önce HCS modülüne gönderilir ve daha sonra aynı ileti menü sistemine de ulaşır. Ana döngü aynı zamanda, ekranın periyodik olarak tazelenmesinden de sorumludur.

```

if( HCS_ProcessSystemMessage( sysMsg.msg, sysMsg.param ) )
    curDisplay = D_HCS;

```

```
// Daha sonra MSS modülüne gönderiliyor.
if( MSS_ProcessSystemMessage( sysMsg.msg, sysMsg.param ) )
curDisplay = D_MSS;
//Eğer menü açıksa ekran denetimi MSS modülüne aktarılıyor.
DSM_SetActiveDisplay( curDisplay );
```



Şekil 3.13 Program akışı

3.5.2 İleti sistemi (Message system)

İleti sistemi, sistemde oluşan olayların hepsinin kaydedilmesinden ve gerekli yerlere iletilmesinde sorumludur. İleti sistemi basitçe bir halka tampondan oluşur ve ilk-giren-ilk-çıkart mantığı ile çalışır. Tamponun boyu sabittir ve derleme aşamasında belirlenir.

İletilerin her birinin bir belirleyicisi ve bir adet parametresi vardır. İleti belirleyicisi iletinin ne olduğu belirtir. Parametre ise o iletiyle ilgili bir veriyi taşır. Bütün iletiler parametre bilgisi taşımak zorunda değildirler. Ancak tuş iletileri gibi bazı özel iletilerde bu fazladan bilgiye ihtiyaç duyulmaktadır.

```
typedef struct
{
    BYTE msg;
    BYTE param;
}SYSMSG;
```

Örneğin tuş olayları ile ilgili 3 adet ileti belirlenmiştir. Bunlar beraberinde o iletinin hangi tuşa ait olduğu bilgisini de taşırlar.

```
#define MSG_KEY_PRESSED    0x04
#define MSG_KEY_HOLD      0x05
#define MSG_KEY_RELEASED   0x06
```

MSG_KEY_PRESSED iletisi, kullanıcı bir tuşa bastığında gönderilir. Kullanıcı tuşa bir saniyeden uzun basılı tutarsa, her 500ms’de bir MSG_KEY_HOLD iletisi gelir, ve kullanıcı tuşu bıraktığında ise MSG_KEY_RELEASED iletisi ile bu durum uygulama yazılımına bildirilir.

Sistem basitliği açısından, iletiler hedefsizdir. İleti kuyruğuna gönderilen iletiler bütün ileti alıcılarına iletilir. Yani iletiler bütün sisteme yayınlanırlar. Eğer iletiyi gönderenin kendisi de aynı zamanda bir ileti alıcısı ise, bu durumda gönderdiği ileti bir süre sonra kendisine de gelecektir.

İleti kuyruğuna ileti gönderip almak için iki temel fonksiyon vardır. Bunlar `SendMessage()` ve `GetMessage()` fonksiyonlarıdır, ve hata durumunda hata kodu döndürürler. Eğer sistemin işlemesi bir yerde gereğinden uzun duraklamışsa ileti kuyruğu taşabilir. Bu durumda yeni gönderilen iletileri kabul etmeyerek bunu bir hata koduyla geriye bildirir.

`GetMessage()` fonksiyonu ileti kuyruğundan bir adet ileti alır ve alınan ileti kuyruktan silinir. Bu nedenle iletileri kuyruktan alıp, alıcılara gönderme işlemi merkezi olarak yapılmalıdır ve IDS yazılımında bu işin sorumlusu ana döngüdür. Ana döngü ileti kuyruğunu sürekli kontrol eder ve yeni gelen bir ileti varsa bunu ileti alıcılarına aktarır.

3.5.3 Durum makineleri

IDS uygulaması üç adet durum makinesi içerir. Bunlar;

- Sistem Durum Makinesi (SDM)
- Denetleyici Durum Makinesi (DDM)
- Ekran Durum Makinesi (EDM)

Sistem Durum Makinesi (SDM)

SDM, sistemin en üst seviye durumlarının denetiminden sorumludur. Sistem durumları ve açıklamaları aşağıdadır.

SS_AUTO: Sistemin otomatik durumda olduğunu belirtir. Sistem bu durumda kazan ve pompaları, zamanlayıcıdan bağımsız olarak kontrol eder.

SS_SCHEDULER: Bu durum otomatik durum ile zamanlayıcının birleşimidir. Sistem otomatik durumdadır, ancak zamanlayıcı tarafından istenilen zamanlarda bu davranışı gösterir.

SS_MANUAL: Sistemin kontrolü elle yapılmak istendiği zaman sistem bu duruma alınır. Sistem bu duruma girdiğinde bütün çıkışlarını aktif hale getirerek denetim işini operatöre bırakır.

SS_OFF: Bu durumda sistem bütün çıkışlarını kapatır.

SS_INIT: Sistemin açılması tamamlanana kadar bu durumdadır. Bu durum sırasında kalibrasyon menüsüne girilebilir ya da PC'den seri kapı ile bağlantı gerçekleştirilebilir.

SS_POWERFAIL: Güç kesildiğinde sistem bu duruma girer. Sistemin pille desteklendiği durumda geçerlidir. Bu duruma giren sistem bütün çıkışlarını kapatır. Gücün tekrar geldiğini algıladığı zaman son kaldığı durumdan devam edecek şekilde sistemi aktif hale getirir.

SS_ERROR: Sistemde herhangi bir hata oluştuğunda sistem bu duruma girer. Bu duruma girerken bütün çıkışlar kapatılır. Ekranı hata numarası ve mesajı basılır. Sistem kapatılıp açılmadığı sürece hata konumundan çıkmaz.

SS_INIT, SS_POWERFAIL ve SS_ERROR sistemin otomatik olarak girdiği durumlardır. Kullanıcı isteğiyle bu durumlardan herhangi birine girilemez.

SS_OFF, SS_AUTO, SS_SCHEDULER, SS_MANUAL durumlarından herhangi biri çalışma sırasında kullanıcı tarafından seçilebilir. Bunun için kullanıcının, sistemin ana ekranındayken sağ veya sol tuşlarından birine basarak sistem durumu menüsünü açması yeterlidir. SDM, ana uygulama kodu tarafından kontrol edilir.

Denetleyici Durum Makinesi (DDM)

DDM ısıtıcı sisteminin kalbidir. Dış sıcaklığa, sistem durumuna ve programa bağlı olarak kazanların ve pompaların uygun şekilde çalıştırılmasından sorumludur.

DDM durumları periyodik olarak saniyede bir güncellenir. DDM her güncellemede sıcaklık sensörlerini okur. Kalibrasyon değerlerini de okuduğu değerlere ekleyerek kazan ve pompaların bir sonraki durumuna karar verir.

DDM durumları ve açıklamaları aşağıdaki gibidir:

CS_IDLE (Boşta): Sistemin kapalı olduğu durumdur.

CS_PREHEAT (Ön Isıtma): Ayarlanmış ön ısıtma süresi boyunca, pompalar çalışmadan yapılan ısıtmadır. Ön ısıtma süresi dolduğunda bu durumdan çıkılır.

CS_FULLHEAT (Tam Isıtma): İki kazanın da aynı anda çalıştığı durumdur.

CS_HALF_HEAT (Yarım Isıtma): Tek kazanın çalıştığı durumdur. Sıcaklık, normal sıcaklık aralığına ulaştığında sistem yarım ısıtma durumunda çalışır, ısıtma gücü düşer. Sıcaklık değeri normal aralığı aşıp yüksek aralığa girdiği zaman sadece pompalar çalışır.

CS_PUMPSONLY (Sadece Pompa durumu): Kazanların kapalı olduğu, ve sadece pompanın çalıştığı durumdur. Sistem kapatıldıktan sonra kazandaki su ayarlanan sıcaklığa düşünceye kadar pompa çalışmaya devam eder.

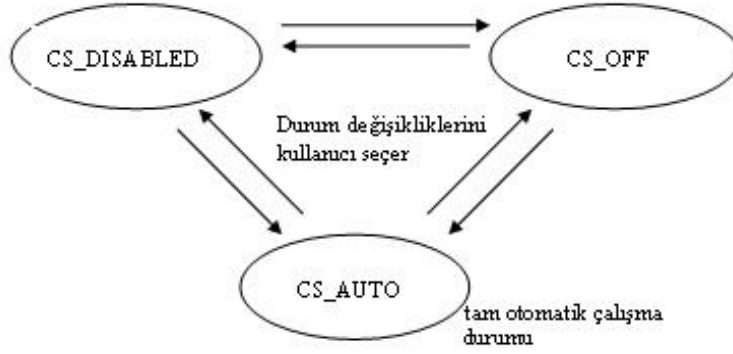
DDM'nin durumları sistem durumları ile bağlantılıdır.

Sistem durumu SS_AUTO iken, denetleyici durum makinesinde CS_AUTO alt durum makinesi işletilir. Yukarıda sözü edilen durumlar CS_AUTO alt durum makinesinin durumlarıdır ve sistem bu durumlar arasındaki geçişine otomatik olarak kendisi karar verir.

CS_DISABLE ve CS_OFF ise sırasıyla SS_MANUAL ve SS_OFF ile ilişkilidir.

Isıtıcı Kontrol Sistemi Durum Makinası Manuel konum durumları:

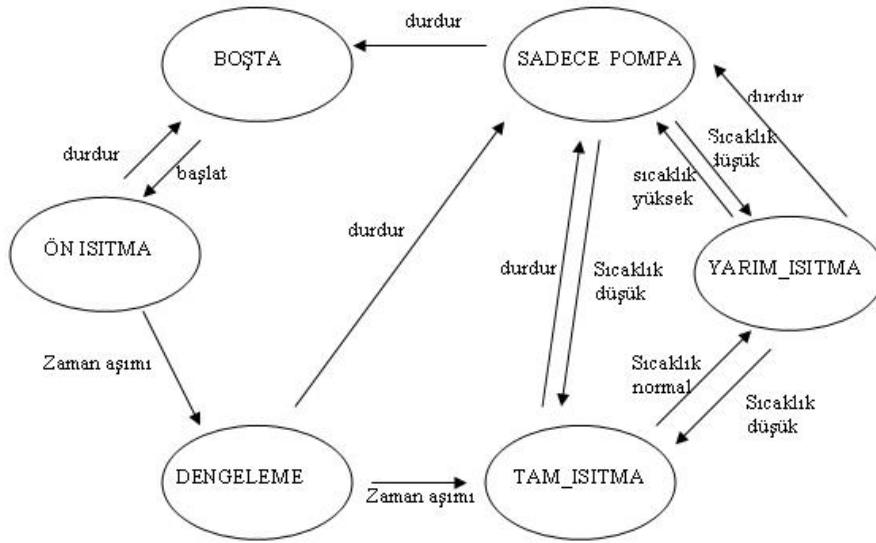
Sistemin manuel konum durumları Şekil 3.14' de görülmektedir.



Şekil 3.14 Isıtıcı Kontrol Sistemi Durum Makinası Manuel konum durumları

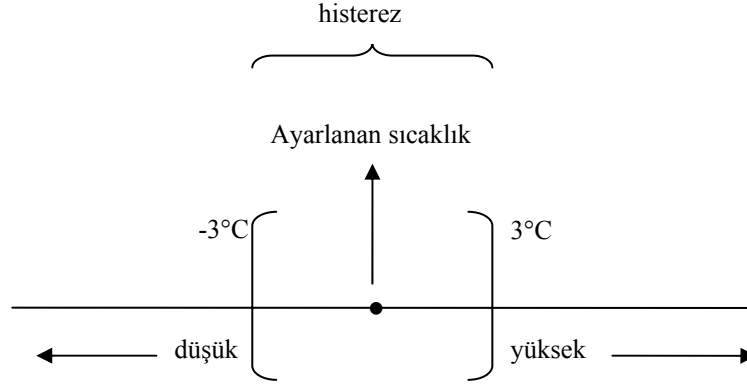
Otomatik konum durumları (SS_AUTO alt durum makinesi):

Sistem durum makinesi Şekil 3.15’ de gösterilmiştir. Sistem durumlarının açıklaması Çizelge 3.1’de yapılmıştır.



Şekil 3.15 IDS durum makinesi

Sıcaklık Aralığı



Şekil 3.16 Sistem sıcaklık aralığı

Şekil 3.16’ da sistem sıcaklık aralığı gösterilmiştir. İki kazanla su ısıtılmaya başlanır. Su, normal sıcaklık aralığının içine girdiğinde sistem tek kazanla çalışmaya devam eder. Böylece, su sıcaklığı sabit tutulmaya çalışılır. Eğer tek kazanın gücü sıcaklığını artırmaya yetiyorsa, su normal sıcaklık seviyesinin üzerine çıkar ve daha sonra kazanların her ikisi birden kapanır, sadece pompa çalışır.

Olaylar (Events):

Başlat: Zamanlayıcı ısıtmaya karar verirse, ve dış sıcaklık ayarlanmış değerin altına düşerse meydana gelir.

Zaman aşımı: Ayarlanan zaman dolduğunda meydana gelir.

Durdur: Kullanıcı veya zamanlayıcının sistemi kapattığı, dış sıcaklığın ayarlanan değerin üzerine çıktığı ya da su sıcaklığının ayarlanan değerin (30°C) altına düştüğü durumlarda oluşur.

Sıcaklık_normal: Su sıcaklığı normal bölgededir.

Sıcaklık_düşük: Su sıcaklığı normal bölgenin altındadır.

Sıcaklık_yüksek: Su sıcaklığı normal bölgenin üzerindedir.

DURUM TABLOSU:

Çizelge 3.1 IDS durum tablosu

DURUM	KAZAN 1	KAZAN 2	POMPA	AÇIKLAMA
BOŞTA	KAPALI	KAPALI	KAPALI	Sistem bekleme durumundadır. Bu durumdan başlat olayı ile çıkar.
ÖN ISITMA	AÇIK	AÇIK	KAPALI	İki kazan da çalışır ve pompa açılmadan önce su sıcaklığı belirli bir seviyeye getirilir. Sistem, bu durumdan bir zaman aşımı olayı ile çıkar.
DENGELEME	AÇIK	AÇIK	AÇIK	Bu duruma ön ısıtma durumundan sonra girilir. Sensör okumalarının doğru olması için sistem istikrarlı hale gelene kadar burada bekler. Bu durumda zaman aşımı olayı ile çıkılır.
TAM ISITMA	AÇIK	AÇIK	AÇIK	Su sıcaklığını hızlı bir şekilde istenilen seviyeye çıkarmak için bu duruma girilir. Su sıcaklığı istenilen sıcaklık aralığına ulaştığında bu durumdan çıkılır.
YARIM ISITMA	AÇIK	KAPALI	AÇIK	Sistem, su sıcaklığı istenilen sıcaklık aralığına girdikten sonra, bir kazanla çalışır. Tek kazanla sıcaklık sabit tutulmaya çalışılır. Sıcaklık_düşük, sıcaklık_yüksek olayları ile bu durumdan çıkılır.
SADECE POMPA	KAPALI	KAPALI	AÇIK	Sistemin daha verimli olması için kazanlar kapalı olduğunda kalan enerjinin pompa ile sisteme aktarılması durumudur. Ayrıca, kazanlar için sıcaklık_yüksek bölgesine girildiğinde stand-by durumudur.

3.5.4 Olay işleyiciler (Event Handlers)

Olay işleyiciler bir modül adına gelen iletileri işleyen fonksiyonlardır. Normal ileti işleyen fonksiyonlardan farklı olarak olay işleyiciler, dinamik olarak kurulup kaldırılabilirler. İleti işleme dışında bir özellikleri de kurulup kaldırılmaları sırasında bilgilendirilmeleridir (Buschmann, 1996).

Olay işleyici EventHandler.h dosyasında tanımlanmış basit bir fonksiyon prototipidir.

```
typedef void (*EV_HANDLER) (HANDLER_OP op, BYTE msg, BYTE  
param);
```

Bir olay işleyici fonksiyonun üç adet işlemi desteklemesi gerekir. Bunlar,

- HDLR_INIT: Olay işleyici fonksiyon kurulurken çağrılır ve olay işleyicinin kendisini ilklendirmesi sağlanır.
- HDLR_EXIT: Olay işleyici fonksiyon kaldırılırken çağrılır. İş bittikten sonra yapması gerekenler varsa, örneğin bellek temizliği gibi, bu noktada yapabilir.
- HDLR_PROC: Olay işleyicinin gönderilen iletiyi işlemesi sağlanır. Normal çalışma durumudur.

3.5.5 Menü sistemi

Menü sistemi kullanıcı ile etkileşimli olarak sistem ayarlarının yapılmasını sağlayan, hiyerarşik yapıda, karakter tabanlı bir arayüzdür. Menü sistemine ait bütün kod MenuSubSystem (MSS) modülünün içindedir.

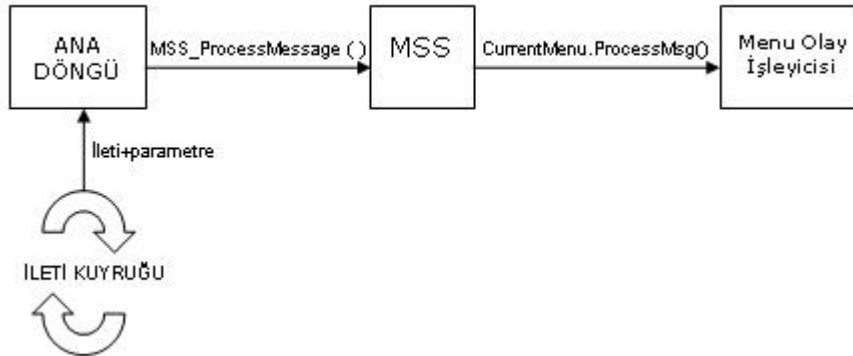
MSS de sistemin geri kalanında olduğu gibi ileti tabanlı çalışır. Ana döngüden iletiler geldikçe işlemlerini gerçekleştirir. MSS'nin aktif hale gelmesini HCS modülü, 'IDM_OPENMENU' iletisini göndererek

gerçekleştirir. Menü aktif hale gelip, MSS iletileri işlemeye başladıktan sonra ekranın kontrolü MSS modülüne bırakılır.

Menülerin birbiri ile ilişkisi hiyerarşiktir. Her menünün üst, alt, sonraki ve önceki olmak üzere dört adet komşusu ve bir adet 'olay işleyici' fonksiyonu vardır. MSS modülüne gelen iletiler o an açık olan menüye ait olay işleyicisine yönlendirilir. Olay işleyicilerinin kurulup kaldırılmaları, MSS'nin sorumluluğunda olup, menü değişimleri sırasında gerçekleşir.

Herhangi bir menü olay işleyicisi, iklendirme sırasında üzerinde çalışacağı değişkenlerin o anki değerleri ile birer geçici kopyasını oluşturur. Olay işleyicisi bu geçici değişkenler üzerinde işlerini tamamlar. Bu şekilde değişkenler üzerinde oluşan değişiklikler, menünün iptal edilmesi durumunda sistemdeki değerleri bozmaz. Benzer şekilde menü iptal edilmeden son adıma kadar gelip, işlem onaylanırsa, geçici değişken değerleri sistem değerlerine kopyalanır.

Menülerin ekranda gösterilmesinden Ekran Durum Makinesi (Display State Machine - DSM) sorumludur. MSS'nin içinde bulunduğu durumu DSM'ye bildirir ve DSM de MSS'ye ait duruma ve onun iç bilgilerine göre ekranın yenilenme işlemini gerçekleştirir. Şekil 3.17'de ileti kuyruğundan alınan bir iletinin menü olay işleyicisine kadar nasıl geldiği gösterilmektedir.



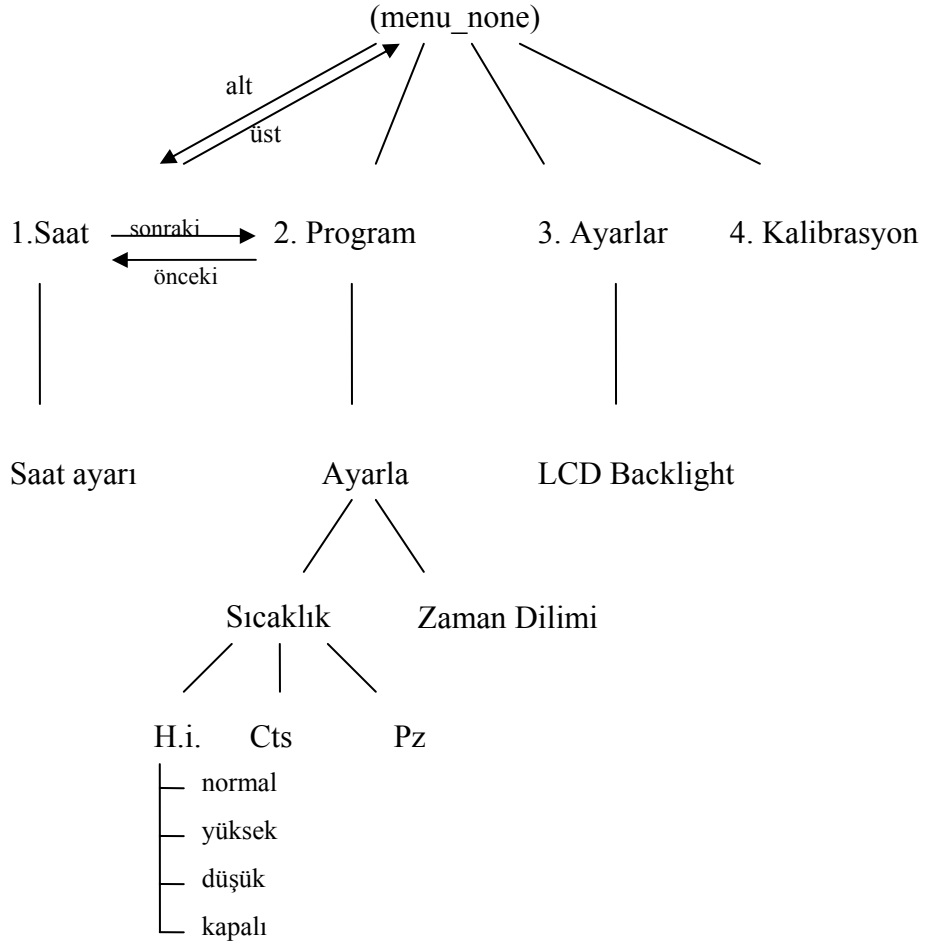
Şekil 3.17 İleti akışı ve olay işleyici

Menüler bir tabloda durağan olarak kodlanmışlardır. Her menüye ait bir tanımlama numarası vardır ve kodun içinde IDM_XXX olarak anılırlar. Menüler birbirleri ile ilişkilerinde bu tanımlama numaralarını kullanırlar. Şekil 3.18’de menü ağaç yapısı gösterilmiştir.

```
typedef struct
{
    BYTE CODE * text;

    MenuID prev;
    MenuID next;
    MenuID sub;
    MenuID super;
    //void (*ProcessMessage) (BYTE msg); // eski hali
    EV_HANDLER ProcessMessage;
} MenuItem;

CODE MenuItem menuList[] =
{
    /*IDM_SET_TIME*/
    {
        IDS_SET_TIME, // Başlık
        IDM_SET_TIME, // Önceki
        IDM_SCHED, // Sonraki
        IDM_SET_TIME_EXEC, // Alt
        IDM_NONE, // Üst
        MSS_MenuCommon // Olay işleyici
    }, ...
}
```



Şekil 3.18 Menü ağaç yapısı

3.5.6 Ekran durum makinesi

Ekran durum makinesi (Display State Machine- DSM), sistem durumuna ve değişkenlerine bağlı olarak ekranın güncellenmesinden sorumludur. Ekranın düzenli aralıklarla tazelenmesi işini, her 50ms'de

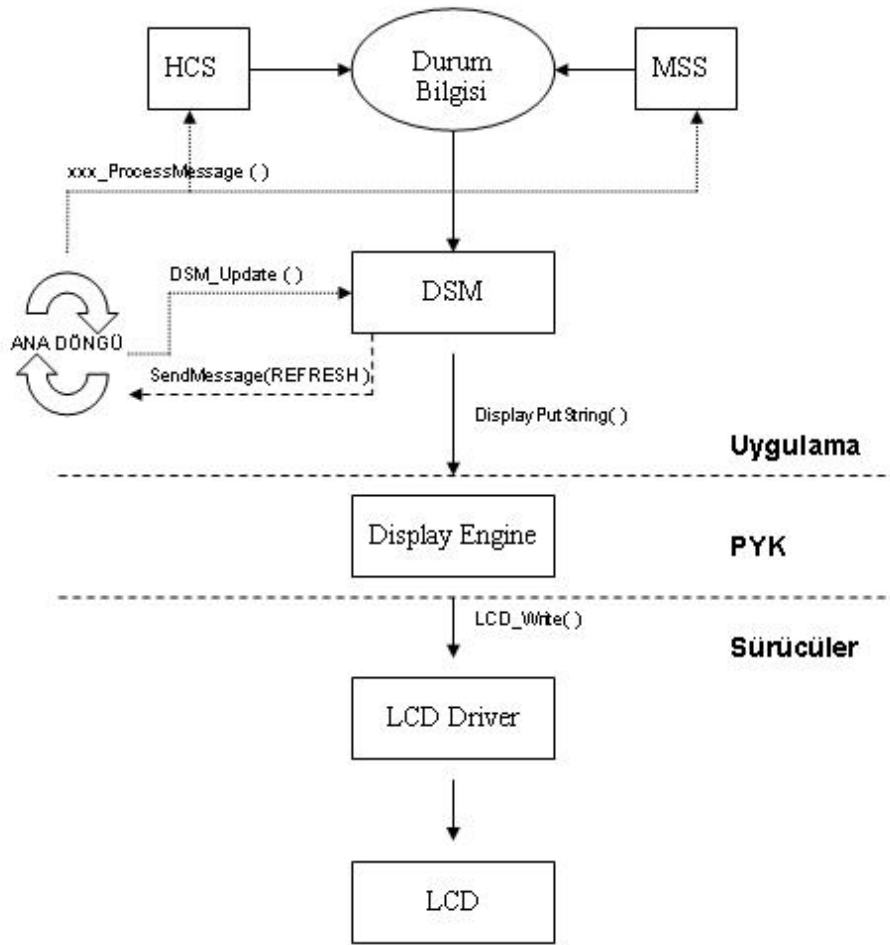
bir ana döngü tarafından çağırılan `DSM_Update()` fonksiyonu gerçekleştirir.

DSM ekranı güncellerken hangi modülün aktif olduğunu ve hangi modüle ait ekranın güncellenmesi gerektiğini bilmelidir. Hangi modülün aktif olduğu bilgisi ise ana döngü tarafından `DSM_SetActiveDisplay()` fonksiyonu ile bildirilir.

Normalde HCS modülü her zaman aktiftir, ancak menü açık olduğu durumlarda, yani iletileri işliyorsa aktif ekran MSS'ye geçer.

DSM ekran değişimlerinde sisteme `MSG_REFRESH_DISPLAY` iletisini göndererek aktif olan modülün ekran bilgilerini güncellemesini ister. Bu iletii alan modül de `DSM_SetDisplayState()` fonksiyonunu kullanarak DSM'ye içinde bulunduğu durumla ilgili bilgi verir ve ekranın yenilenmesi gerçekleştirilmiş olur.

DSM ekranı güncellemek için donanım sürücülerine doğrudan erişmek yerine, PYK içinde bulunan 'Ekran Motoru'nu kullanır. Bu şekilde uygulamanın bir parçası olan DSM, üzerinde hiçbir değişiklik yapılmasına gerek duyulmadan platformlar arasında taşınabilir. Modüllerin sistemden gelen iletilerle durum bilgilerini güncellemeleri ve bu bilgilerin ekrana kadar nasıl taşındığı, Şekil 3.19'daki ekranın güncellenme senaryosunda gösterilmektedir.



Şekil 3.19 Ekran güncelleme senaryosu

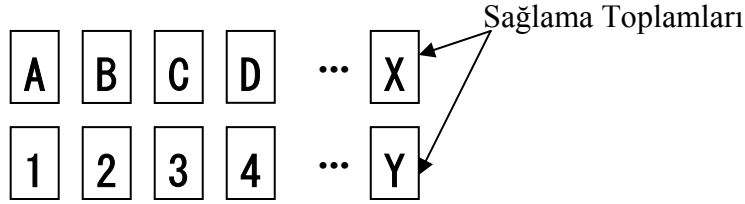
3.5.7 Ekran motoru (Display Engine)

Ekran motoru, PYK içinde olup, uygulama ile ekran donanım sürücülerini arasında yalıtım sağlar. Yalıtım sağlamak için genel olarak LCD ekranlar için kullanılabilecek fonksiyonları gerçekler.

LCDler genel olarak yavaş cihazlardır. Bu nedenle LCD ekranlara birşeyler yazmak uzun (bazı durumlarda bir saniyeye kadar) zaman alır. IDS yazılımı bir işletim sistemi üzerine kurulmadığından, bir işlem bitene kadar diğerlerinin çalışması mümkün değildir ve uzun süren işlerden olabildiği kadar kaçınmak gerekir. Her ne kadar bir ileti kuyruğuna sahip olmamız olayların kaçırılmadan mutlaka işlenmesini garanti etse de bazı noktalarda uzun takılmalar ileti kuyruğunun taşmasına neden olabilir.

Ekran motorunun yalıtım sağlama dışında bir diğer görevi de gereksiz donanım erişimlerini ortadan kaldırarak sistemin daha verimli kullanılmasını sağlamaktır. Bunu yapmak için ekran motoru, ekran verisinin bir kopyasını kendi içindeki tamponlarda tutar. Bu tamponlar, üzerlerinde değişiklik olması durumunda donanıma aktarılır. Bunun dışında donanım erişimi yapılmaz bu sayede uzun süren LCD donanımı güncellemeleri en aza indirgenmiş olur.

Ekran Tamponu



Şekil 3.20 Ekran tamponu sağlama toplamı

Ekran motoru, tamponu iki satır için ayrı olmak üzere iki parça halinde tutar (Şekil 3.20). Her parçanın sonunda bir önceki verinin toplamına ait bir sağlama toplamı (checksum) bulunmaktadır. Uygulamadan gelen ekrana yazma istekleri önce bu tamponlara yazılır. Ekran güncelleme işlemi her 100 milisaniyede bir ana döngü tarafından istenir. Ekran motoru tamponlardaki son veriye ait sağlama toplamını hesaplar ve bir önceki ile karşılaştırır. Eğer yeni sağlama toplamı bir öncekinden farklıysa o sağlama toplamına ait satır donanıma tekrar

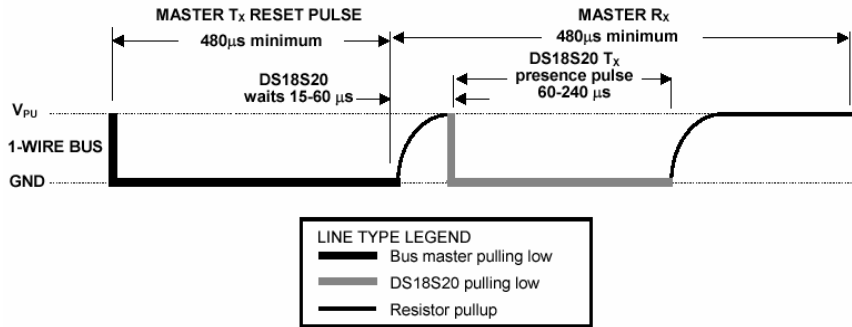
aktarılarak ekranın en güncel bilgi ile yenilenmesi sağlanmış olur. Ekranın değişen yerlerinin satır bazında donanım aktarılması sisteme diğer işlerini yapabilmesi için daha çok zaman bırakmaktadır.

3.4.7 Bir kablolu yol (1Wire Bus) ve sürücüsü

Dallas Semiconductor tarafından tasarlanan bir cihaz haberleşme sistemidir. Düşük-hızlı veri, sinyalleşme ve güç iletimi tek bir kablo üzerinde gerçekleşir. Bunun yanında bir toprak kablosu da gerekir. Her cihazın kendine özel, sabit, içine gömülü bir adresi ve zamanlama kontrol sistemi vardır.

İki kablolu yol ile haberleşmeden farkı daha düşük veri hızı ve daha düşük maliyetli olmasıdır. Genel olarak sayısal termometreler, hava cihazları gibi pahalı olmayan, küçük cihazların haberleşmesinde kullanılır.

Haberleşme sinyalleri darbe genişliğine dayanır. Yol yöneticisi tarafından gönderilen 480 mikrosaniyelik bir darbe, yol üzerinde bulunan cihazların kendilerini sıfırlamasını sağlar. Eğer yol üzerinde çalışan bir cihaz varsa, sıfırlama sinyaline yanıt olarak yaklaşık 200 mikrosaniyelik bir varlık bildirme sinyali gönderir. Böylece yol yöneticisi sıfırlama işleminin başarıyla gerçekleştiğini ve yol üzerinde aktif cihaz olduğunu anlar. Şekil 3.21 bir kablolu yol üzerindeki sıfırlama sinyalini zamanlama bilgileri ile birlikte göstermektedir.



Şekil 3.21 Bir kablolu yol sıfırlama sinyali (Dallas, 1999)

Veri transferi için ise yine sinyal genişlikleri kullanılır. Yol üzerindeki 60-120 mikrosaniye arasındaki bir darbe mantıksal SIFIR anlamına gelirken, 1 mikrosaniyeden biraz uzun bir sinyal mantıksal BİR anlamına gelmektedir.

Bir kablolu yol üzerindeki bütün veri iletişimi yol yöneticisi tarafından başlatılır ve yönetilir. Veri iletimi önemli bit (MSB) önce gönderilecek şekilde yapılır.

Bir kablolu yol üzerinde aynı anda bir çok cihaz bulunabilir. Cihazların birbiri ile karışmamasını sağlamak için her cihazın sadece kendisine ait, 64 bitlik tanıtım numarası vardır. Yol üzerinde bulunan bütün cihazları tanımlayabilen belirli bir arama algoritması vardır. Bu algoritma kullanılarak, tek kablolu yol üzerinde bulunan bütün cihazlar tanımlanarak indekslenir. Tanımlama sırasında cihazın tanıtım numarasının içinden cihazın ne tür bir cihaz olduğu hakkında gerekli bilgi de alınır. Daha sonra yol üzerindeki istenen cihaza erişmek için o cihazın adresi kullanılır.

Eğer tek kablolu yol üzerinde yalnızca bir adet cihaz bulunuyorsa, bu durumda cihazla haberleşmek için adresini bilmeye ve kullanmaya gerek yoktur. Haberleşme başladıktan sonra gönderilen “SKIP ROM” komutunu alan cihaz bundan sonra yapılan haberleşmenin doğrudan kendisine geldiğini bilir.

Tek kablolu yol üzerinde sadece bir adet cihaz kullanmak sürücü karmaşıklığını azaltır, ve bir arama algoritması gerçekleşmesi gerekliliğini ortadan kaldırır. IDS'nin sınırlı kaynakları dolayısı ile sürücü tasarlanırken bu yol izlenmiştir. Sistemde var olan iki adet sıcaklık sensörü, iki farklı yol üzerine konulmuştur. Bu sayede sensörlerle haberleşmek için adreslerini bilmeye gerek yoktur.

Bir kablolu yol protokolünün en küçük gerçeklemesi kapsamında aşağıdaki fonksiyonlar kullanılmıştır.

Error OneWire_BusReset(): Bir kablolu yolu sıfırlar. Dönüş değeri olarak yol üzerindeki cihazdan varlık sinyalini alamazsa hata verir.

BYTE OneWire_ByteRead(): Yol üzerinde bulunan cihazdan bir bayt veri okur.

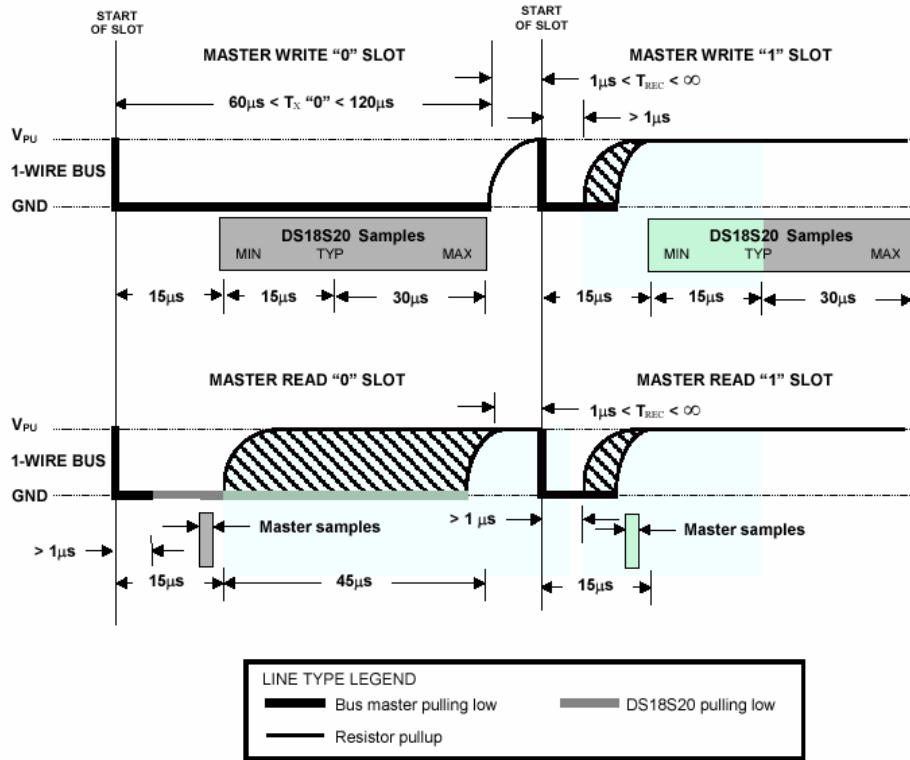
void OneWire_ByteWrite(BYTE byteData): Yol üzerinde bulunan cihaza bir bayt veri yazar.

void OneWire_SkipROM(): Tek cihazlı çalışma modunda cihaza, adres

bilgisi göndermemek için bu fonksiyon çağrılır.

void OneWire_ReadROM(ROMCode DATA *): Cihazın ROM adresini okur.

Şekil 3.22 bir kablolu yol üzerindeki veri bitlerinin transferini zamanlama bilgileri ile birlikte göstermektedir.

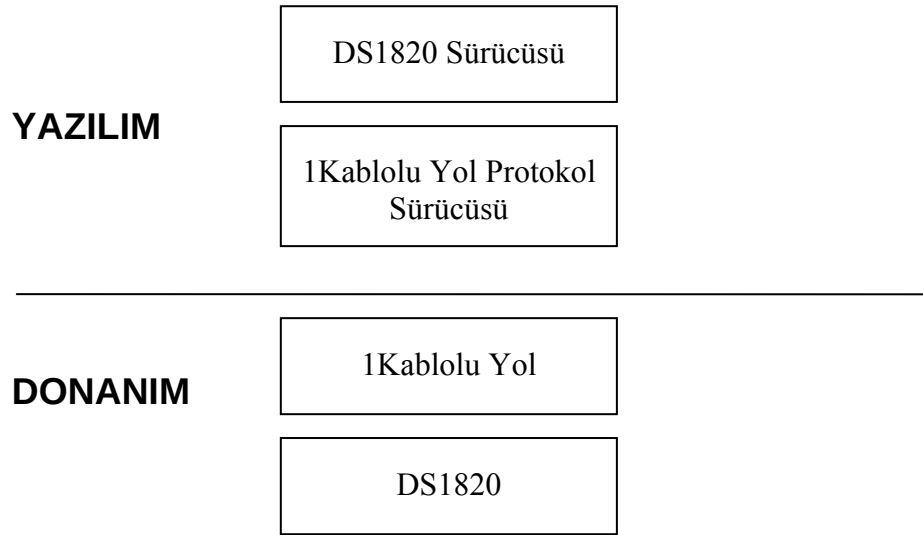


Şekil 3.22 Bir kablolu yol veri sinyalleri (Dallas, 1999)

Bir kablolu yol sürücüsü gerçekleştirirken gerek duyulan zamanlamalar, boş bekleme döngüleri ile sağlanmıştır. Bekleme

sürelerinin mikrosaniyeler düzeyinde olması nedeniyle, bu döngüler genel sistem performansını olumsuz yönde etkilemezler. Ancak yine de sistem verimini artırmak adına sensör okumaları saniyede bir defa yapılır. Bekleme döngüsü sistemin şu anki çalışma frekansına uygun olarak ayarlanmıştır. Bu nedenle sistemin çalışma frekansı değiştirilirse tekrar ince ayar yapılması gereklidir. Aksi takdirde tek kablolu yol sürücüsü yol üzerindeki cihazlara erişemeyecektir.

IDS’de kullanılan DS1820 sıcaklık sensörleri tek kablolu yol üzerinde, Şekil 3.23’te gösterildiği gibi çalışmaktadırlar.



Şekil 3.23 DS1820 sürücü yapısı

3.5.8 İki kablolu yol (2Wire Bus – i2C) ve sürücüsü

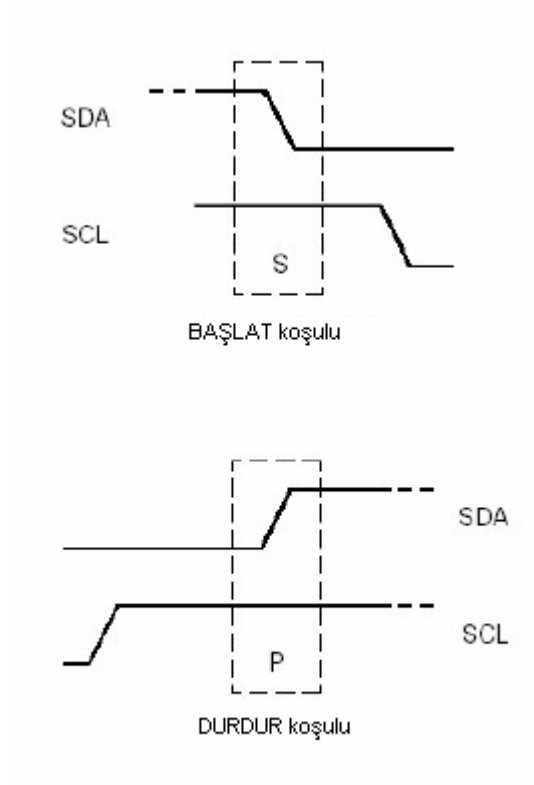
Yongaların birbiri arasında düşük hızda haberleşmelerini sağlamak için Philips tarafından geliştirilmiş bir protokoldür.

i2c haberleşme sistemi 2 aktif kablo ve 1 toprak dan oluşur. Aktif kablolardan biri seri veri yolu, diğeri seri saat yoludur ve her ikisi de çift-yönlüdür.

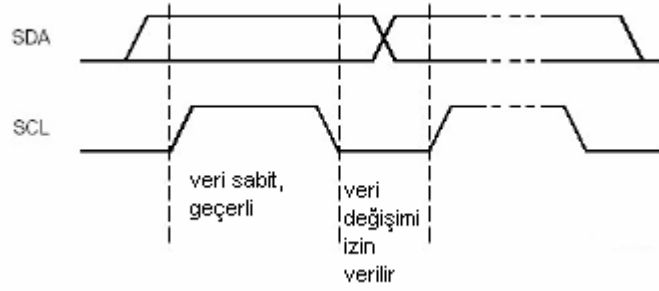
Her cihazın sabit bir adresi vardır ve yol üzerinde toplam 127 adet birbirinden farklı cihaz bulunabilir. Haberleşmeler yol yöneticisi tarafından başlatılır ve denetlenir.

Karşılıklı gönderilen her baytın ardından cihazlar birbirlerine gönderilen baytın alındığına dair bir onay biti gönderirler. Onay bitinin gitmemesi haberleşmenin bitirilmesine neden olur.

Yol boşta iken saat ve veri hatlarının ikisi de mantıksal ‘bir’ seviyesinde dururlar. Haberleşmenin başlama ve bitişini bildiren iki özel sinyal vardır. BAŞLAT sinyali için saat hattı ‘bir’ seviyesindeyken veri hattı sıfıra indirilir. DUR sinyali için ise saat hattı ‘bir’ seviyesindeyken veri hattında sıfırdan bir seviyesine geçiş yapılır. Yol üzerindeki veri bitleri saat sinyali sıfırdayken hazırlanır ve saat sinyali bire çekildiğinde karşıdaki cihaz bu biti alır. Veri iletişimi bayt tabanlıdır ve her baytın bitleri, en büyük bitten başlayarak gönderilir. Meşgul olduğunu belirtmek isteyen cihaz, saat hattını sıfır seviyesinde tutarak, yeni bir haberleşmenin başlamasını engeller. Şekil 3.24’ de iki kablolu yol üzerinde haberleşmenin başlama ve bitiş sinyalleri görülmektedir. Şekil 3.25’ de iki kablolu yol üzerinde veri aktarımı sırasında saat ve veri hatlarının ilişkisi görülmektedir. SCL, seri saat sinyali (serial clock) hattıdır; ve SDA ise seri veri (serial data) anlamına gelmektedir.



Şekil 3.24 İki kablolu yol senkronizasyon sinyalleri (Philips, 2000)



Şekil 3.25 İki kablolu yol veri aktarımı (Philips, 2000)

İki kablolu yol haberleşmede gönderilen paketlerin içindeki baytların sırası Şekil 3.26’ da gösterilmiştir.

BAŞLAT	Cihaz Adresi	Okuma/Yazma	Alt Adres	Veri Baytları...	DUR
--------	--------------	-------------	-----------	------------------	-----

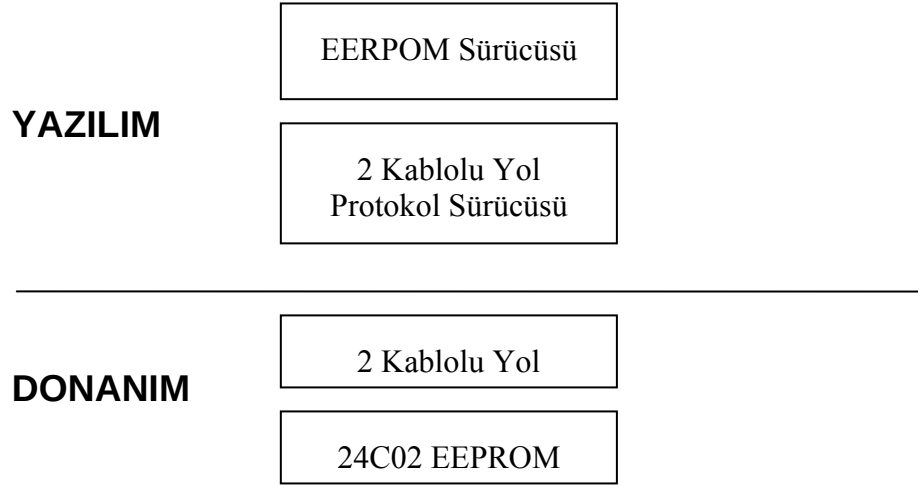
Şekil 3.26 İki kablolu yol paket yapısı

İki kablolu yol üzerinde bulunan EEPROM yongasının bir adresine bir baytlık bir veri yazmak için gereken fonksiyon örnek olarak aşağıdadır. Şekil 3.27’te EEPROM sürücüsünün genel yapısı görülmektedir.

```

Void EEPROM_WriteByte(BYTE address,BYTE dat)
{
    I2C_Start();
    I2C_WriteByte(AT24C02); // Fiziksel cihaz adresi
    I2C_WriteByte(address); // Cihaz üzerindeki alt adres
    I2C_WriteByte(dat); // Veri baytı
    I2C_Stop();}

```



Şekil 3.27 EEPROM sürücü yapısı

3.5.9 Sistemin başlatılması

Bir gömülü sistem üzerinde üst seviye bir programlama dilinde geliştirilen yazılımların çalıştırılabilmesi için, önce donanımın ilklendirilmesi gerekir. Örneğin kullanılan programlama dili C ise main fonksiyonu çağrılmadan önce, genel değişkenlerin değerlerinin atanması gibi, gerekli ilklendirmeler yapılır. Donanım ilklendirmesi, işlemci, bellek, giriş/çıkışlar ve diğer yardımcı ara birimlerin üst seviye programlama dili tarafından kullanılabilir hale getirilmesi demektir.

En alt seviyede sistemin başlatılması dört aşamada gerçekleşir. Bunlar sırasıyla;

- Sıfırlama kodu (Assembly),
- Donanım ilklendirme kodu (Assembly),
- Başlangıç kodu (Assembly),
- main fonksiyonunun çağrılmasıdır.

İklendirmenin ilk aşaması sıfırlama kodudur. Bu kod donanım sıfırlanmasından sonra otomatik olarak çalışmaya başlayan ilk assembly kod parçasıdır. Sadece bir kaç bayt uzunluğundadır ve görevi iklendirme kodunu çalıştırmaktır. Bu kod parçası işlemcinin sıfırlama sonrası kod işleme adresine konulur. 8051 için başlangıç vektörü 0x0000 adresidir. Bu nedenle özel bir sıfırlama kodu yazılmasına gerek yoktur. Ancak bazı işlemcilerde işlemcinin başlangıç vektörü örneğin 0xffffd gibi belleğin sonunda bir yerde olabilir. Bu gibi durumlarda sıfırlama koduna ihtiyaç duyulur.

Donanım iklendirmenin büyük kısmı ikinci bölümde gerçekleşir. Burada bellek, kesmeler, zamanlayıcılar ve giriş/çıkışlar gerekli başlangıç durumlarına ayarlanırlar. Daha az önemli olan diğer donanımlar main başladıktan sonra iklendirilebilirler; seri haberleşme noktası gibi. Ancak, özellikle sayısal çıkışların olabildiğince çabuk belirli değerler almaları çok önemlidir. Bu nedenle çıkışların da bu noktada ayarlanması gereklidir.

Ayrıca işlemcinin kendi çalışmasına özel, örneğin bellek yapılandırması gibi ayarlamalar ve bellek testi de bu noktada yapılmalıdır.

Üçüncü adım, başlangıç kodudur. Bu kod C programının başlaması için gerekli düzenlemeleri yapar. Bunlar, genel değişkenlerin ilk değerlerini alması, eğer kullanılıyorsa dinamik bellek bölgesinin ayrılması, yığının belirlenmesi ve yazmaçların ilk değerlerinin atanması olarak sıralanabilir. Eğer bir işletim sistemi kullanılacaksa, işletim sistemini ve diğer işleri başlatma görevi main tarafından yapılır. Başlangıç kodu bu işleri bitirdikten sonra main fonksiyonunu çağırır.

IDS sisteminin başlangıç kodu olarak Keil C Compiler tarafından sağlanan standart başlangıç kodu kullanılmıştır. Bu kod 8051 assembly dilinde yazılmıştır. Görevi belleği iklendirmek, yığını hazırlamak ve main fonksiyonunu çağırmasıdır. Daha üst seviye cihaz sürücülerinin hazırlanması ve başlatılması main fonksiyonunun başında yapılmıştır.

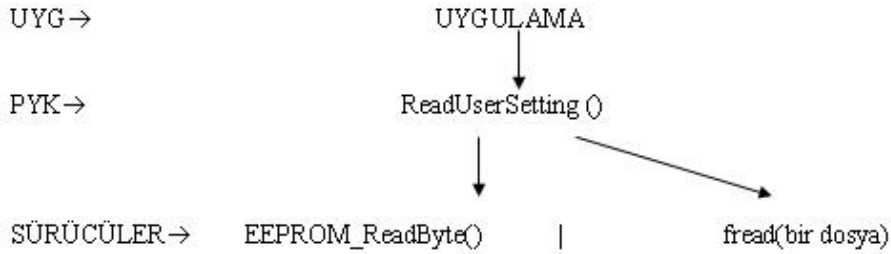
3.5.10 Platform Yalıtım Katmanı - PYK

Bu katman fiziksel cihaz sürücülerini tarafından sağlanan özellikleri mantıksal fonksiyonlara dönüştürür. Uygulama seviyesinde yalıtım

sağlar. Bu sayede yazılımın üstte kalan bölümünde hiç bir değişiklik yapmadan platformlar arasında taşıma yapılabilir. Böyle bir katman tasarımı sistemin PC üzerinde simulasyonunu kolaylaştırır ve yazılım geliştirme sürecini hızlandırır. Yalıtım katmanı kullanmak sistemin modülerliğini artırdığı gibi, üst seviye uygulamanın nesneye yönelik bir programlama diliyle geliştirilmesini kolaylaştırmaktadır (Gamma, 1994).

Örnek verecek olursak, kullanıcı ayarları gerçek sistemde bir EEPROM üzerinde saklanır. Cihaz sürücüler alttaki donanıma ulaşmayı sağlar. EEPROM için iki tane basit fonksiyon tanımlanabilir ve bunlar EEPROM_ReadByte() ve EEPROM_WriteByte() olarak adlandırılabilir. Ancak uygulamadan EEPROM'a erişmek için bu fonksiyonları doğrudan çağırmak yerine PYK içinde yer alan ReadUserSetting(), WriteUserSetting() fonksiyonları kullanılır. Bu şekilde uygulama alttaki donanım platformundan yalıtılmış olur.

PC simlatörü üzerinden örnek vermek gerekirse, kullanıcı ayarlarını tutmak için EEPROM yerine, disk üzerindeki bir dosyayı kullanabiliriz. Ama uygulama cihaz sürücülerine PYK üzerinden eriştiği için, uygulama kodunda bir değişiklik yapılmasına gerek kalmayacaktır. Değişen tek şey PYK olacak ve PYK'nın üstü aynen korunacaktır. Şekil 3.28' de platform yalıtım katmanının kullanıcı ayarlarını alt seviye veri saklama birimlerinden nasıl ayırdığı anlatılmaktadır.



Şekil 3.28 PYK kullanıcı ayarları örneği

3.5.11 Hata denetimi

Gömülü sistemler bulundukları ortamın koşullarına bağlı olarak hata oluşumuna açıktırlar. Örneğin sıcaklık sensörlerine su kaçması sensör okuma hatalarına yol açabilir. Bu durumlarda hata denetiminin sağlıklı bir şekilde yapılarak sistemin tutarlılığına zarar gelmesini engellemek gerekir.

Kullanıldığı yere bağlı olarak gömülü sistemlerde, denetlenmeyen veya gözardı edilen hatalar ölümcül sonuçlara yol açabilir. Örneğin sensörlerden okunan tahmin edilmeyen bir değer, yapılan sonuç hesaplamalarının da beklenmedik bir şekilde sınırların dışına çıkmasına neden olabilir; bir ABS sisteminin yanlış şekilde çalışarak aracın kaza yapmasına neden olabileceği gibi.

Gömülü sistemlerde hata oluşumuna karşı izlenebilecek en güvenli yol, güvenli bir şekilde sistemi kapatmaktır. Bundan önce hatanın geçici olup olmadığını anlamak için birkaç deneme yapılır ve hatanın kalıcı olduğuna karar verilirse sistem olabilecek en güvenli duruma alınarak bütün işlemler durdurulur. Hata durumu kullanıcıya görsel ya da işitsel iletiler yoluyla bildirilir. Bu bir ışığın yanıp sönmeye olabileceği gibi sürekli öten bir sinyal sesi ya da ekranda bir yazı da olabilir.

IDS, oluşabilecek donanım hataları durumunda kendisini koruma moduna alır. Sensör okumaları sırasında hata oluşması durumunda sistem kazanları ve pompaları kapatarak bekleme konumuna geçer. Bu durumda ekranda hata kodu görünür.

Yazılımda hata yapması olası bütün fonksiyonların dönüş değeri “Error” olarak belirtilmiştir. Bir fonksiyonun hata döndürdüğünü anlamak için kullanılmak üzere tasarlanmış çeşitli makro’lar vardır. Bunların amacı oluşan hatayı anlamak ve işlemi hatanın olduğu yerde kesmektir. Hata kodları ve makro’lar error.h dosyasında tanımlanmıştır.

ERR_RAISE(e): Parametre olarak girilen hatanın fonksiyondan döndürülmesini sağlar. Aynı zamanda global hata değişkeninin değerini de ayarlar.

IS_THIS_ERROR(e): Parametre olarak geçirilen işlemin sonucunun hatalı olup olmadığını kontrol eder.

ERR_ASSERT(cond): İşlemin hatalı olması durumunda o anda içinde bulunduğu fonksiyondan da aynı hata değeri ile dönülmesini sağlar. Hata döndüren bütün fonksiyonların bunu kullanması durumunda, oluşan hata zincirleme olarak işlemi başlatan fonksiyona kadar döner.

ERR_REDIRECT(cond, exception): Bazı durumlarda alt tarafta oluşan hataları üst tarafa döndürürken çevirmek gerekebilir. Örneğin 1-kablolu yol üzerinde oluşan bir ‘yol hatası’, uygulamaya ‘su sensörü hatası’ olarak dönebilir. Çünkü üst seviye uygulama için yol hatası bir anlam ifade etmemektedir. Onun için önemli olan su sensörünün okunup okunmadığıdır.

Hata döndürmesi olası bütün fonksiyonların geri dönüş değeri ‘Error’ olmalıdır. Ayrıca hata döndürme olasılığı olan fonksiyonları çağıran diğer fonksiyonlar da, dönen sonucun hata olup olmadığına bakmalıdırlar. Bu şekilde işlem gereken noktada sonlandırılır.

Örnek olarak aşağıdaki kod parçası denetleyici durum makinesi güncelleme fonksiyonu içinde sensör okuma hatası oluşması durumunda fonksiyondan geriye hata döndürülmesini sağlar. Bu fonksiyonu çağıran HCS_ProcessMessage fonksiyonu ise dönen hatayı görerek sistemi hata konumuna sokar.

```
Error CSM_Update()
{
    ERR_ASSERT( CSM_ReadSensors() );
    ....

HCS_ProcessSystemMessage()
{
    ...

    case SS_AUTO:
        switch(msg)
        {
            case MSG_NEW_SECOND:
```

```

if ( IS_THIS_ERROR(CSM_Update()) )
{
    HCS_SetSystemState(SS_ERROR);
}

break;

....

```

3.5.12 Yapılandırma ve Kalibrasyon

Yapılandırma sistemini derleme zamanı yapılandırması ve çalışma zamanı yapılandırması olarak iki gruba ayırabiliriz. Bunlara sırasıyla sistem profili ve kullanıcı tercihleri diyebiliriz.

Sistem profili bilgisi derleme sırasında config.h dosyasından okunur. Bu dosyada sistemin çalışma platformu, hata ayıklama modunda olup olmadığı gibi bilgilerin yanında, aynı zamanda takılıp sökülebilir yazılım seçenekleri bulunmaktadır. Mikrodenetleyici üzerinde hangi bacağın hangi girişe ya da çıkışa bağlanacağı da bu dosyada belirtilir.

Yazılım seçeneklerinin takılıp sökülebilmesi sayesinde, kullanıcının isteklerine özel bir yazılım yapılabildiği gibi, kullanılmayan kod parçalarının da bellekte yer kaplaması önlenmiş olur.

Yazılımın seçimsel yerleri önışlemci tanımlarının içinde tutulur. Böylece kodun içinde ayrıca değişiklik yapılmasına gerek duyulmaz.

Çalışma zamanı yapılandırması iki bölümden oluşur. Bunlar kullanıcı ayarları ve kalibrasyon ayarlarıdır. Bu ayarlar EEPROM üzerinde tutulurlar. EEPROM üzerinde her ayarın kendine özel bir bellek bölgesi vardır ve bu bilgiler NVMMMap.h dosyası içinde tanımlanmıştır.

EEPROMun başlangıçta bütün baytları 0xff olarak gelir. Bu nedenle EEPROMu kullanmadan önce başlangıç değerlerine ayarlamamız gerekir. EEPROM içindeki verinin geçerli olup olmadığını anlamak için sağlama toplamı yöntemi kullanılır. Bu yöntemde veri içindeki bütün baytların toplamı alınır ve veri bloğunun son baytı olarak yazılır. Eğer kayıtlı sağlama toplamı hesaplanan sağlama toplamına eşit değilse cihaz ön değerlerle ilklendirilir. Bu yöntem aynı zamanda elektrik kesilmeleri ya da benzer donanım hataları sonucunda EEPROM bozulmasına karşı da etkilidir. Veri bozulması durumunda EEPROM

içeriği güvenli bir şekilde öndeğerlerle geri yüklenmektedir. Bu şekilde sistemin beklenmedik hatalar yapması engellenmiş olur.

Bunun dışında bir diğer koruma da yazılımın güncellenmesi durumlarında çalışacak şekilde geliştirilmiştir. Bu yapıda kullanıcı ayarları verisinin ilk baytı, veri yapısının boyunu tutar. Bu şekilde yeni gelen yazılımdaki yapının boyu ile EEPROM üzerinde bulunan yapının boyu birbirini tutmuyorsa EEPROM verisi sıfırlanır ve yeniden yazılır. Böylece yazılımda yapılan yenilemeler sonucunda değişen EEPROM veri yapısı cihaz üzerinde var olanla aynı hale gelmiş olur. Şekil 3.29' da EEPROM içindeki verilerin yerleşimi görülmektedir.



Şekil 3.29 Eeprom veri yapısı

Kalibrasyon bilgileri, son kullanıcı tarafından bilinmeyen, sistemin çalışma şeklini değiştiren bilgilerdir. Bu ayarlar genel kullanıcı tarafından bilinmeyen bir tuş kombinasyonu ile girilen bir menü yardımıyla değiştirilir ve sonuçlar EEPROM üzerinde saklanır.

Kalibrasyonun amacı, sistemi bulunduğu ortam koşullarına uygun hale getirmektir. IDS sisteminde kalibrasyon menüsüne girmek için, sistem açılırken 3 saniye boyunca sol tuşa basmak gereklidir.

Her kalibrasyon parametresinin kendine has özellikleri vardır. Bunlar bir C yapısı içinde aşağıdaki gibi tanımlanmıştır.

```
typedef struct
{
    char * name; // Parametrenin adı
    uint8 nvOffset; // EEPROM' da nerede tutulduğu
    int8 deflt; // Varsayılan ilk değeri
    int8 minValue; // En düşük değeri
    int8 maxValue; // En yüksek değeri

} CalibrationTableEntry;
```

Boş ya da bozulmuş EEPROMa kalibrasyon verilerinin ön değerleri yazılırken, bu yapının içinde bulunan ‘varsayılan’ değer kullanılır.

Kalibrasyon menüsünden ayarlanabilen sistem parametreleri aşağıdaki gibi bir tabloda tutulurlar. Tablonun boyu genişletilebilir olup EEPROM boyu ile sınırlıdır. Ayrıca kalibrasyon menüsü kodu aşağıdaki tabloyu doğrudan kullanarak ayarların yapılmasını sağlar. Yani başka bir deyişle, yeni eklenen kalibrasyon özellikleri için menü kodunda herhangi bir değişiklik yapılmasına gerek yoktur. Bu tabloya eklenen değerler otomatik olarak kalibrasyon menüsüne eklenirler.

```
// Kalibrasyon aralığı
CalibrationTableEntry CODE calRangeTable[] =
{
    {"Hysteresis", CSMCAL_HYST, 3, 1, 5},

    {"TableOffset", CSMCAL_TABLE_OFFSET, 0, -10, 10},

    {"W-Sensor", CSMCAL_WSENSOR_OFFSET, 0, -5, 5},
    {"A-Sensor", CSMCAL_ASENSOR_OFFSET, 0, -5, 5},

    {"PreheatTime", CSMCAL_PREHEAT_TIMEOUT, 10, 0, 30},
    {"Stabil.Time", CSMCAL_STABILIZE_TIMEOUT, 3, 0, 30},
    {"StartupTemp", CSMCAL_STARTUP_TEMP, 15, 10, 25},
    {"PumpOffTemp", CSMCAL_PUMP_SHUTDOWN_TEMP, 30, 25, 40},
    {"HighOffset", CSMCAL_HIGH_OFFSET, 5, 2, 15},
    {"LowOffset", CSMCAL_LOW_OFFSET, 10, 2, 15},
    {0, 0, 0, 0} //End of table
};
```

Histerez: Bu deęer sistem histerez penceresinin geniřlięini belirler. Bu da okunan su sıcaklıęının istenen deęere ulařıp ulařmadıęını anlamakta kullanılır.

TableOffset: Bu deęer, dıř sıcaklıęa baęlı olarak su sıcaklıęının okunduęu tablonun deęerlerinin üzerine eklenir. Bylece tablo deęerleri binanın yalıtım zelliklerine uygun hale gelir.

W-Sensor ve A-Sensor: Su sıcaklıęı ve hava sıcaklık sensrlerinden okunan deęerler, sensrn yerleřtirildięi yere baęlı olarak tam doęru olmayabilir. Bu kalibrasyon parametresi ile sensrlerden okunan deęerler dzeltilir.

PreheatTime: n ısıtma sresi (dakika olarak).

Stabil.Time: Pompalar alıřmaya bařladıktan sonra, su sıcaklıęı tutarlı hale gelene kadar geen sre (dakika olarak).

StartupTemp: Isıtmaya bařlamak iin gerekli minimum dıř hava sıcaklıęı.

PumpOffTemp: Isıtma iřlemi bittikten sonra pompaların kapanması iin uygun olan su sıcaklıęı. Bu řekilde sıcak su soęuyana kadar sistemde dolařtırılır.

HighOffset ve LowOffset: Kullanıcı sistemdeki su sıcaklıęını normal, dřk ya da yksek olarak seebilmektedir. Bu parametre ile tablodan okunan deęerin yksek ve dřk seimlerde nasıl deęiřtirileceęi belirlenir.

3.5.13 Hata ayıklama (Debugging)

Kullanılacak 8051 iřlemcisinin zel bir hata ayıklama modu bulunmamaktadır. Bařka bir deyiřle, programı ykleyip adım adım alıřtırmak, durdurmak, deęiřken ve yıęın deęerlerini inceleme řansımız bulunmamaktadır.

Bu sistem zerinde elimizde olan tek hata ayıklama yntemi yazılım izlerini seri kapı zerinden almaktır. Bunun iin sistem aılırken seri kapı 9600 baud hızında programlanır. Bir Null-Modem kablosu yardımıyla IDS bilgisayara baęlanarak, Hyperterminal benzeri bir terminal programı ile programın akıřı izlenebilir.

printf fonksiyonu seri kapaıya yönlendirilmiştir. Bu fonksiyon kullanılarak yazılan herşey seri kapıdan çıkar. Kod içine yazılan *printf* fonksiyonları içinde bulunan yazılar da derleme sırasında koda eklenir ve kodun büyümesine yol açar.

Bunun önüne geçmek için iki adet derleme modu vardır. Bunlardan biri bütün *printf* çıktılarının açık olduđu hata ayıklama konumu, diğeri de hata ayıklama çıktılarının kapatıldığı ve koddan söküldüğü yayınlama konumudur.

Yazılımın hata çıktılarını açıp kapatabilmek isteğimiz için *printf* fonksiyonunu açık olarak kullanılmalıdır. Onun yerine sadece hata ayıklama konumunda geçerli olan DPF() makrosunu kullanılır. DPF makrosunun çalışma şekli config.h dosyasından seçilen hata ayılama moduna göre değişebilmektedir.

DPF makrosu aynı zamanda sistem simülatör üzerinde çalışırken hata ayıklama çıktılarını simülatördeki *printf* karşılığı olan DebugPrintf fonksiyonuna yönlendirir. Aşağıda DPF makrosunun nasıl yazıldığı görölmektedir.

```
#if DEBUG
    #if SIMULATION
        #define DPF(x) DebugPrintf x
    #endif

    #if X51
        #define DPF(x) printf x
    #endif

#else
#define DPF(x)
#endif
```

Bu makro kullanılırken, yapısı gereği çift parantezle kullanılır. Bu sayede kodun içine durağan olarak kodlanmış olan yazılar, yayınlama konumunda koddan tamamen sökülebilir. Örneğin çıktı olarak “Merhaba Dünya” yazmak isteniyorsa şu şekilde yazmalıdır:

```
DPF(("Merhaba Dünya\n"));
```

3.5.14 Uzaktan telefon bağlantısı

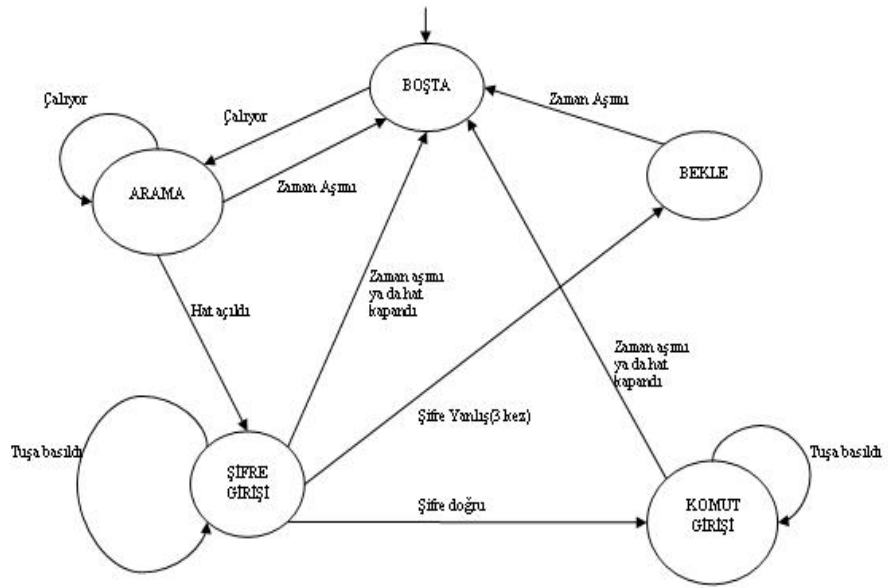
IDS uzaktan telefonla denetlenebilecek şekilde tasarlanmıştır. Bu nedenle içinde bir telefon devresi bulunmaktadır.

Karşı taraftan arandığında cihaz hattı açar, kullanıcı 4 rakamlı şifresini girer ve sistemi otomatik modda açıp kapatabilir. Telefon sisteminin basitliği nedeni ile sadece otomatik durumda açıp kapatma özelliği gerçekleştirilmiştir. Bunun dışında sistemin hiçbir özelliği telefonla uzaktan denetlenemez. Sistem kullanıcıdan gelen komutları aldığını bildirmek için karşıya sinyal sesi gönderir. Kullanıcının seçtiği çalışma durumu, kullanıcı telefonu kapattıktan sonra aktif hale getirilir. Bu şekilde tuşlara yanlılıkla basılması durumunda sistemin sürekli durum değiştirmesi önlenmiş olur.

Kullanıcı şifresini üç defada doğru girmek zorundadır. Aksi durumda sistem telefon hattını kapatır, ve ondan sonraki beş dakika boyunca yapılan aramalara cevap vermez. Bu şekilde, zorla deneme yöntemiyle şifrenin bulunması zorlaştırılıp, sistemin güvenliği artırılmış olur. Telefonla denetlemeye ait sistemin durum makinesi şeması aşağıdadır.

Telefonun kaç defa çaldıktan sonra açılacağı ve zaman aşımaları parametreleri kalibrasyon menüsünden ayarlanabilmektedir. Kullanıcı şifresi ise kullanıcı ayarları menüsünden değiştirilebilir. Şekil 3.30' da uzaktan telefon bağlantısı durum makinesi görülmektedir.

Telefon hattının durumu ve basılan tuş bilgileri ilgili donanımdan toplanır. Durum değişiklikleri telefon durum makinesi tarafından işlenerek gerekli işlemler yapılır. Sistem, telefon durum makinesini düzenli aralıklarla güncelleyerek, zaman aşımalarını hesaplatır ve herhangi bir durumda gereğinden fazla kalmasını engeller.



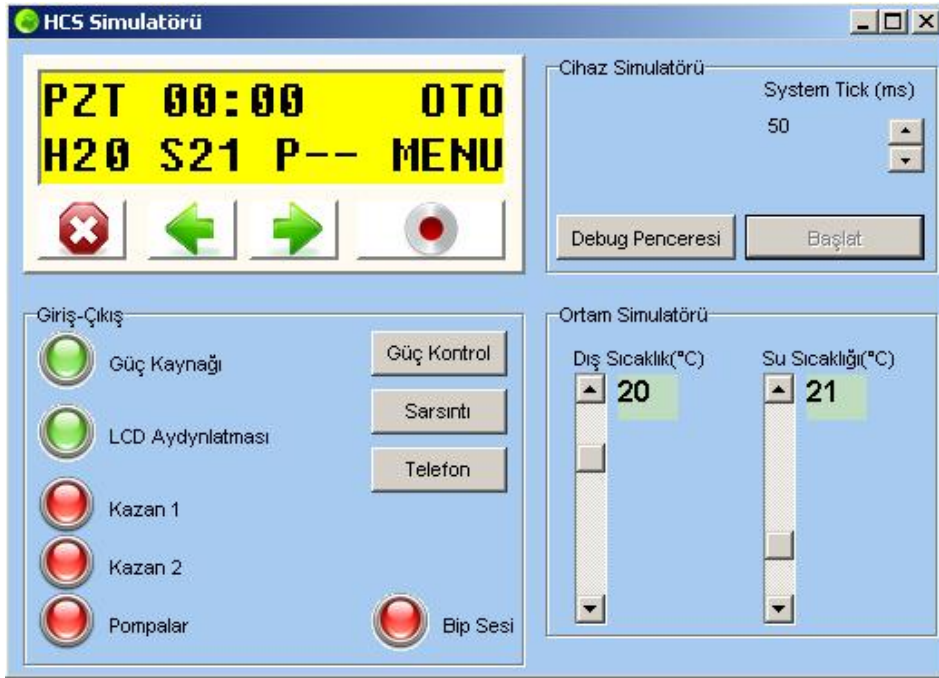
Şekil 3.30 Uzaktan telefon bağlantısı

4. UYGULAMA SİMULATÖRÜ

IDS'nin üst seviye yazılım geliştirme sürecini hızlandırmak ve kolaylaştırmak için bir uygulama simülatorü geliştirilmiştir. Uygulama simülatorü kullanılarak platform yalıtım katmanı üzerinde kalan bütün yazılım PC üzerinde geliştirilebilir.

Bu sistemde uygulama simülatorünü kullanarak yazılım geliştirmek, bizi gömülü sistemler üzerinde yazılım geliştirmenin en büyük sorunlarından biri olan, derlenmiş kodun hedef sisteme aktarılması aşamasından kurtarır. Böylece hedef donanımın özelliklerine bağlı olarak, her yazılım denemesinde yaklaşık en az bir dakika olmak üzere zaman kazanılabilir.

Şekil 4.1' de HCS simülatorü görülmektedir.



Şekil 4.1 HCS Simulaterü

Geliştirilen simülâtör bir CPU simülasyonu değildir. Yani gömülü sistem için derlenmiş kodun yüklenmesi yöntemiyle çalışmaz. Uygulama simülâtörü, üst seviyede uygulama platformunu ve ortamını simüle eder. Simülâtörün kendisi de her yazılım denemesi için ayrıca sistem yazılımıyla birlikte derlenir, ve sistem yazılımı simülâtörün bir parçası olarak çalışır.

Kaynak kodun simülâtörde derlenmeye uygun hale gelmesi için config.h dosyası için bulunan SIMULATION tanımının aktif hale getirilmesi gerekmektedir. Bu tanımın aktif hale getirilmesi dışında kodun geri kalanında herhangi bir değişiklik yapılmasına gerek kalmadan simülâtör kodu derlenip çalıştırılabilir.

Simülâtör geliştirme ortamı olarak Borland C++ Builder seçilmiştir. Bunun nedeni görsel uygulamaların kolay geliştirilebilmesi ve IDS'ye özel C dosyalarının doğrudan derlenebilmesidir.

Simülâtörün görevi PYK altındaki herşeyi uygulamaya şeffaf bir şekilde simüle etmektir. IDS uygulaması açısından bakıldığında bunlar;

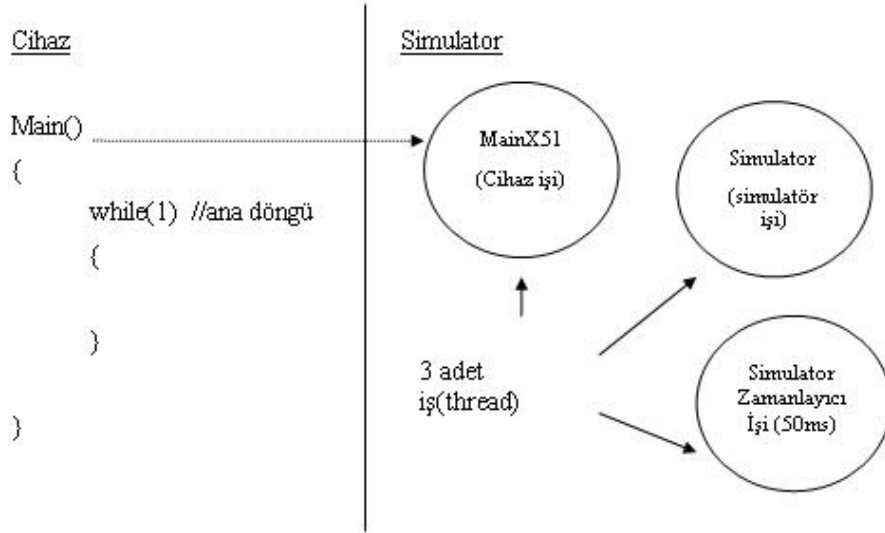
- Ekran
 - Tuşlar
 - Sıcaklık sensörleri
 - Zamanlayıcı
 - Kullanıcı ayarları (EEPROM)
 - İz ekranı
- olarak sıralanabilir.

4.1 Genel Yapı

Uygulama kodu simülâtör üzerinde ayrı bir iş olarak çalışır. Normalde gömülü sistem üzerinde main() fonksiyonu ile başlayan uygulama, simülâtör üzerinde MainX51 fonksiyonu adı altında bir iş olarak çalışır.

Sistem tiklerini üreten zamanlayıcı da ayrı bir iş olarak tanımlanmıştır ve her 50 ms'de bir ileti kuyruğuna bir SYS_MSG_TICK

iletisini gönderir. Ayrıca sistemin yüksek ileti yükü altında davranışını incelemek için, bu iletinin zamanı 1ms'ye kadar indirilebilir. Şekil 4.2' de simulator ve cihaz farkı gösterilmektedir.



Şekil 4.2 Simulatör ve cihaz farkı

Normalde mikrodenetleyici üzerinde tanımlanmış bacakların her biri, simulasyon ortamında birer değişkendirler. Örneğin tuşlara basıldığında o tuşa ait kapı bacağının durumu değişiyormuş gibi, o kapı bacağına ait simulatör değişkeninin değeri değişir.

Sıcaklık sensörlerini simule etmek için iki adet ayar çubuğu konulmuştur. Buralardan ayarlanan değerler, üst seviye sıcaklık okuma fonksiyonları tarafından yazılıma, gerçekten sensörlerden okunmuş gibi iletilirler.

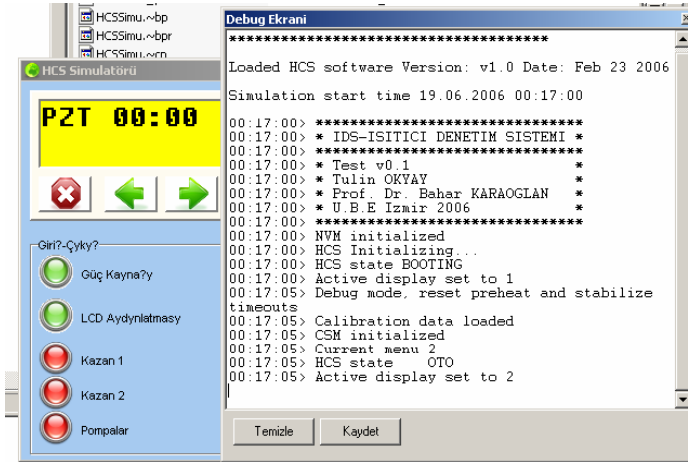
Gerçek donanımda bir EEPROM üzerinde saklanan kullanıcı ayarları ve kalibrasyon bilgileri simulatör üzerinde bir dosya da tutulur. Sistem yazılımının EEPROM'a yaptığı yazma ve okuma işlemleri, PYK

tarafından bu dosyaya yönlendirilir. Bu şekilde kodun içinde bir değişikliğe gerek kalmadan EEPROM varmış gibi kullanılabilir.

4.2 İz Ekranı

İz ekranı (Şekil 4.3) yazılım içinde farklı yerlere konulmuş olan *printf* çıktılarının görüntülediği terminal ekranıdır. Sistemde yazılımın akışının takip edilmesini sağlayarak hata ayıklamaya yardımcı olur, aynı zamanda sistemin uzun süreli testi için de uygundur. Uzun süre çalışmaya bırakılan yazılım içinde oluşan bütün olaylar pencerede gösterilir ve daha sonra incelenmek üzere kaydedilebilir.

İz ekranı simulatörün içinde ayrı bir pencere olarak açılır. Kod için DPF() makrosu kullanılarak seri kapıdan elde edilen bütün hata ayıklama çıktıları, simulatör ortamında çalışılırken bu ekrana yönlendirilir. Bunun için kodun içinde bir değişiklik yapılmasına gerek yoktur. DPF() makrosu gömülü sistem için derlendiği zaman farklı, simulatör için derlendiği zaman farklı davranış gösterir. Bu özellik, debug.h dosyası içinde belirlenmiştir.



Şekil 4.3 İz ekranı

5. SONUÇ

Bu tez kapsamında, gömülü sistemlerle bir denetim sistemi tasarlanmıştır. Sistem yazılımsal ve donanımsal olarak gerçekleştirilmiş, ve başarıyla çalıştırılmıştır. Sistem için geliştirilmiş olan simülâtör yazılım geliştirmede yoğun olarak kullanılmıştır.

Simülâtörün yazılım geliştirme sürecini önemli ölçüde hızlandırdığı görülmüştür. Normalde gerçek sistem üzerinde yapılan her deneme için, kodun yüklenmesi ve çalışır hale getirilmesi yaklaşık iki dakika zaman alırken, simülâtör kullanılarak bu süre 10 saniyenin altına indirilmiştir. Bu esneklik sayesinde uygulamanın kullanıcı arabirimi geliştirilirken programcı istediği kadar deneme yapabilir.

Aynı şekilde kullanıcı arabirimi ile ilgili hata ayıklama da gerçek sisteme ihtiyaç duyulmadan PC üzerinde yapılabilir.

Ayrıca platform yalıtım katmanı aynı kalacak şekilde, herhangi bir uygulamanın kullanıcı arabirimi, henüz ortada donanım yokken yazılabilir. Bu şekilde donanım hazır olduğunda, PYK yeni donanıma taşındıktan sonra, uygulama aynı simülâtörde olduğu gibi çalışmaya devam edecektir.

Sistem yazılımının boyu, tasarım sırasında öngörülen donanım kapasitesi sınırlarının içerisinde kalmıştır. Derlenmiş kod ile ilgili veri boyu (ram) ve kod boyu (rom) bilgileri Çizelge 5.1’de görülmektedir.

Çizelge 5.1 Yazılım boyu

	Hata ayıklama modu	Ürün modu
Veri boyu	~160 bayt	~150 bayt
Kod boyu	~16 kбайt	~12 kбайt

Kaynak kod büyüklüğü, satır sayısı olarak yaklaşık olarak 6000 satır civarındadır.

Projenin yazılım ve donanım olarak, bütün ayrıntıları ile geliştirilmesi için yaklaşık 8 (ADAMxAY) insan gücüne ihtiyaç duyulmuştur.

Prototip maliyeti yaklaşık olarak 80 USD civarındadır. Ancak seri üretim durumunda bu maliyet bunun çok altına düşecektir. Gerçek birim maliyetini üretim sayısı belirleyecektir.

Bu sistemi daha da ileri götürmek isteyenler için gelecekte yapılabilecek işlerden bazıları şöyle sıralanabilir:

- Kod boyu ve bellek kullanımı açısından kod optimize edilebilir.
- Platform yalıtım katmanı daha da genelleştirilerek geliştirilebilir. Bu şekilde PYK'nın altında daha farklı donanımlar desteklenebilir.
- IDS internet üzerinden kontrol edilebilir hale getirilebilir.
- Sistemin sıcaklık kontrol algoritması optimize edilerek verimi artırılabilir.

Pazardaki Çözümler

Şu anda pazarda hazırda bizim yaptığımıza benzer çözümler bulunmaktadır. Her cihazın kendine göre artıları ve eksileri vardır.

Aşağıda pazarda bulunabilecek ürünlerden iki tanesini kısaca tanıtabiliriz.

Dataterm IHC: Mikroişlemci tabanlı akıllı bir ısıtıcı kontrol sistemidir. Minimum yakıt harcıyarak doğru sıcaklık ayarı yapabilmek için tasarlanmıştır. Kendi kendine öğrenebilir ve tamamen otomatiktir. Bulanık mantık tekniğini kullanır.

Optimum başlat/durdur özelliği sayesinde sistemin boşa yakıt harcamasını engellemek için başlatma zamanı dış hava sıcaklığına bağlı olarak seçilir. Dataterm, ısıtma sistemi açık ve kapalı olduğunda bina içindeki sıcaklık değişim hızını belleklar ve görüntüler. Daha sonra bu veri, değişen iklim durumlarına bağlı olarak kararlı bir sıcaklık sağlamak için kendi kendine öğrenme işleminde kullanılır.

Haftanın her günü, sekiz ayrı zaman aralığında istenilen sıcaklık değeri programlanabilir.

Düşen sıcaklık değerini sürekli görüntüler ve kazanı güvenli bir şekilde zamanında çalıştırır.

Celcius ya da Fahrenheit olarak sıcaklık değerlerini, yakıt seviyesini, özel tatil durumu planlarını görüntüleme imkanı sunar.

RS232 ile bilgisayara bağlanabilir. Anahtarlanabilir her türlü ısıtıcı ile kullanılabilir. Şekil 5.1’ de cihazın genel görünüşü verilmektedir.



Şekil 5.1 Dataterm IHC görüntüsü

Honeywell: ST6400 modeli çok özellikli elektronik programlayıcıdır. Isıtma ve sıcak su için her gün en fazla üç ayrı zaman diliminde olmak üzere haftanın 7 günü için ayrı programlama yapılabilir. Böyle esnek bir programlama, merkezi ısıtma sistemi günden güne değişme ihtiyacı olan insanlar için geliştirilmiştir. Şekil 5.2’ de cihazın genel görünüşü verilmektedir.



Şekil 5.2 Honeywell görüntüsü

IDS ve pazardaki diğer çözümlerin karşılaştırma tablosu

Çizelge 5.2 IDS ve pazardaki diğer çözümlerin karşılaştırma tablosu

	IDS	Dataterm	Honeywell
Programlanabilme	x	x	x
Dış sıcaklığa bağlı su sıcaklığı kontrolü	x	x	-
Kalibrasyon menüsü	x	x	-
Ön ısıtma	x	-	-
LCD ekran	x	x	x
Elektrik kesintisinden sonra otomatik devam etme	x	x	-
Kolay entegrasyon	x	x	x
Donma koruması	-	x	x
Deprem sensörü	x	-	-

KAYNAKLAR DİZİNİ

Barr, M., 1999, Programming Embedded Systems, O'reilly, Unites States of America,

Schultz, T.W, 1996, C and the 8051 Hardware, Modular Programming and Multitasking, Prentice Hall PTR, New Jersey,

Burns, A., 2001, Real-time systems and Programming Languages, Addison-Wesley.

Simon, David E., 1999, An Embedded Software Primer, Addison-Wesley

Buschmann, F., 1996, Pattern-oriented software architecture: a system of patterns, Wiley Chicester.

McDonnell, M. D., 2004, An Introduction to Real-time Programming Using the Tics RTOS, Tics Real-time

Gamma, E., 1994, Design Patterns Elements of Reusable Object-Oriented Software, Addison Wesley.

Hansen, P. B., 1973, Concurrent Programming Concepts, ACM Computing Surveys.

Walmsley, M., 2000, Multi-threaded programming in C++, Springer.

Kanellos, M., 2001, Intel's Revolution, CNET News

Dixon, S. L., 1998. Fluid Mechanics, Thermodynamics, Pergamon Press

Philips Semiconductors, 2000, The I2C Bus Specification, Philips

KAYNAKLAR DİZİNİ (devam)

Dallas Semiconductors, 1999, The 1-WireBus Specification, Dallas

Atmel Semiconductors, 2000, 89C55WD Microcontroller Datasheet,
Atmel

ÖZGEÇMİŞ

1979 yılında Eskişehir’ de doğan Tülin Okyay, Dokuz Eylül Üniversitesi Mühendislik Fakültesi Elektrik-Elektronik bölümünü tamamladı (2001). Eğitimine Ege Üniversitesi Uluslararası Bilgisayar Enstitüsüne devam etmekj için yüksek lisansa başvurdu. Vestel Elektronik ARGE bölümünde elektronik mühendisi olarak çalıştı (2003). Şu an Arkas Holding Bimar Bilgi İşlem A.Ş.’ de network destek uzmanı olarak çalışmaktadır (2006).

EKLER

EK 1 Gerçek Zamanlı ve Gömülü Sistem Tanımları

Durma noktası (breakpoint)

Bir program durma noktasına ulaştığında durur. Durma noktası, hata ayıklayıcının program akışını durdurması bildirilen komutun adresidir. Bu noktadan sonra kullanıcı hata ayıklayıcı komutlarını kullanmaya başlayabilir. Bir noktadan sonra kullanıcı, “Devam(continue)” komutuyla programın kaldığı yerden devam etmesini sağlayabilir.

Meşgul bekleme (busy waiting)

Bir programın sürekli olarak bir şartın oluşup oluşmadığına bakarak beklemesi durumudur. Aşağıdaki kod basit bir meşgul bekleme örneğidir.

```
while (TRUE) {  
    if (switchDown) {  
        lampOn();  
        break;  
    }  
}
```

Bu kod olay sürmeli kod yapısının tersine, beklenen şart gerçekleşene kadar döngünün içinde kalır.

Ortaklaşa çokluişlem (cooperative scheduling)

Diğer işler de çalışabilsin diye, bir işin gönüllü olarak işlemciyi bırakması temeline dayanan, bir grup işin ortaklaşa çalışma şeklidir. Başka deyişle, bir iş sonsuza kadar hiç ara vermeden çalışırsa diğer işler ölecektir. Kontrol, işin Bekle(pause) ya da Dur(suspend) işletim sistemi fonksiyonlarını kullanmasıyla ya da bir olay beklemeye başlamasıyla bırakılır.

Çevrimsel çokluışlem (cyclical scheduling)

Çevrimsel çokluışlem (round robin olarak da kullanılır), işletim sistemi çekirdeği olmadan, bir sonsuz döngü içinde işlerin sırasıyla çağrılmasıyla yapılır. Aşağıdaki döngüyü buna örnek olarak verebiliriz.

```
void main(void)
{
    for (;;)
    {
        task0();
        task1();
        task2();
        ...
    }
}
```

Bu senaryoda her iş kendi görevini hızlıca bitirmeli , gerekiyorsa son durumunu saklamalı ve kontrolü sonsuz döngüye geri vermelidir. Bu yaklaşımın iyi tarafı, küçük program ve bellek boyuna gereksinim duymasıdır. İşletim sistemi kullanılmaz ve bütün işler için sadece bir adet yığın kullanılır. Kontrolü ve anlaşılması daha kolaydır. Kötü yanı ise öncelik yoktur, işler kendi durumlarını saklamak zorundadır ve programcıya daha büyük sorumluluk düşmektedir. Aynı zamanda, iş sayısı çok olduğunda büyük döngü gecikmelerine yol açılabilir.

Aktarım (download)

Çalıştırılabilir programın geliştirme bilgisayarından hedef cihaza aktarılması anlamında kullanılır. Genel olarak RS-232 seri haberleşmesi kullanılır. Hedef üzerinde aktarma işlemini gerçekleştirerek, kodu yükleyip çalıştıracak bir yazılım bulunmalıdır. Hedef cihazda böyle bir yazılım yoksa bir devre içi emulatör(ICE- In Circuit Emulatör) gereklidir. ICE'a genelde RS-232 üzerinden bağlanılır ve program

yüklenerek çalıştırılır. ICE özelliği kullanılan işlemcinin içinde gelen donanımsal bir yapıdır.

Olay sürmeli kod (event driven code)

Bir olay gerçekleşene kadar çalışmayan koddur. Aşağıdaki kod buna bir örnektir

```
while (TRUE) {  
    msg = waitMsg(MOTOR_ON);  
    freeMsg(msg);  
    turnMotorOn();}
```

Meşgul bekleme tarzında programlamanın tersine, MOTOR_ON iletisi gelene kadar waitMsg fonksiyonu o an çalışan işi durdurur.

Kesme hizmet fonksiyonu (interrupt service routine, isr)

Bir kesme oluştuğunda işlemci tarafından, donanım seviyesinden çağrılan fonksiyondur.

İleti (message)

Bir işe gönderilen olay bilgilerini içeren bir veri yapısıdır. Bir kuyrukta tutulur. Olayların geliş sırasına göre işlenirler.

round robin

Bütün işlerin sırası geldiğinde çalıştığı bir çokluişlem yöntemidir. Bütün işler çalıştıktan sonra döngü tekrarlanır. Ortaklaşa çokluişlem, zaman dilimli çokluişlem, ve çevrimsel çokluişlem gibi farklı zamanlama teknikleri round robin şeklinde çalışır. Çevrimsel ve round-robin zamanlama terimleri birbiri yerine kullanılabilir.

Hedef ve ev sahibi (target, host)

Bu terimler yazılım geliştirme sürecinde iki bilgisayar var olması durumunda kullanılır. Bir bilgisayar yazılımı geliştirmek için kullanılır ve ev sahibi bilgisayar olarak anılır. Diğer bilgisayar da programı çalıştırmak için kullanılır ve hedef bilgisayar olarak anılır. Hedef yazılımın üzerinde çalıştığı gerçek üründür. Bir çok durumda hedef ve ev sahibi aynıdır. Örneğin PC için, PC üzerinde geliştirilen bir uygulamada

olduđu gibi. Ancak gömölü sistemler için durum böyle değildir. Bir robot kolunu kontrol eden bir kart için yazılım geliştiriliyorsa, bu işlem o kart üzerinde yapılamaz çünkü, klavyesi ya da ekranı yoktur. Bu durumda yazılım PC üzerinde geliştirilir ve test edilmeye uygun olduđu düşünöldüğünde kontrol kartına yüklenir.