YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

TELSİZ ÇOKLU ORTAM ALGILAYICI DÜĞÜM TASARIMI VE İŞLETİM SİSTEMİ UYARLAMASI

Bilgisayar Mühendisi Esra ÇELİK

FBE Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Mühendisliği Programında Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı : Yard. Doç. Dr. A. Gökhan YAVUZ

İSTANBUL, 2010

İÇİNDEKİLER

Sayfa

KISALTMA LİSTESİ

SDIO	Secure Digital Input Output
TDMA	Time Division Multiple Access
MEMS	Mikro-elektromekanik Sistemler
RAM	Random-access Memory
QVS	Quality-aware Voice Streaming
MOS	Mean Opinion Score
OS	Operating System
nesC	Network Embedded Systems C
RAM	Random Access Memory
ROM	Read-Only Memory
EEPROM	Electronically Erasable Programmable Read-Only Memory
ADC	Analog-to-Digital Converter
DAC	Digital-to-Analog Converter
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
RISC	Reduced Instruction Set Computing
GCC	GNU C Compiler
CPU	Central Processing Unit
ICWCS	In-Cave Wireless Communication System
FIFO	First-In-First-Out
DMA	Direct Memory Access
LCD	Liquid Crystal Display
LED	Light Emitting diyote
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
SMCLK	Submain System Clock
ACLK	Auxiliary Clock
MCLK	Master Clock
DCO	Digitally Controlled Oscillator
VCORE	Core Voltage
ESB	Embedded Sensor Board
RTC	Real-Time Clock
SDIO	Secure Digital Input Output
RF	Radio Frequency
KB	Kilobyte
kHz	kilohertz
mA	milliampere
µA	microampere
mm	millimeter
ms	millisecond
db	decibel
V	Volt

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Bu tez günümüz dünyasında telsiz algılayıcı ağlar konusunda gelişmekte olan teknolojiyi daha ileri götürmeyi ve ülkemiz yararına farklı projelerde kullanılmasına öncü olmayı amaçlamıştır. Tezin hazırlanmasında benzer güncel projeler incelenmiş, geliştirilmesinde ise yeni teknolojik gelişmelerden faydalanılmıştır.

Telsiz algılayıcı ağların kullanım alanları arasına yeni birini ekleyeceğini düşündüğümüz bu tez, geliştirilmeye açıktır ve birçok yeni projeye kaynak olacaktır. Telsiz algılayıcı ağların ses haberleşmesinde kullanılmasına imkan verecek bir alt yapı sunduğumuz bu proje, Yıldız Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü Öğretim Üyeleri Yard. Doç. Dr. A. Gökhan Yavuz, Z. Cihan Tayşi ve Amaç Güvensan'ın katılımı ile başlatılmış, genel amaçlı bir telsiz çoklu ortam algılayıcı düğüm tasarımı ve gerçekleştirilmesi projesinin de bir ön çalışması olmuştur.

Çalışmamda bana her türlü desteği veren ve danışmanlığımı yapan değerli hocam Yard. Doç. Dr. A. Gökhan Yavuz'a ve yine her konuda desteğini esirgemeyen Z. Cihan Tayşi'ye teşekkürü bir borç bilirim.

Ağustos, 2010

Esra ÇELİK

ÖZET

Son yıllarda çeşitli alanlarda ortam verisi toplama amacıyla telsiz algılayıcı düğümlerin kullanımı yaygınlaşmıştır. 1990'ların sonunda ise küçük boyutlu yeni nesil telsiz algılayıcı düğümlerin geliştirilmeye başlanmasıyla bu teknolojinin kullanıldığı alanlar da hızla artış göstermiştir. Telsiz algılayıcı ağlar ilk olarak askeri amaçlı savunma uygulamalarında kullanılmıştır. Günümüzde ise kullanım alanları arasında doğal yaşam alanlarının izlenmesi, tarım ürünlerinin gelişimlerinin izlenmesi, endüstriyel kontrol ve izleme sistemleri, ev otomasyon sistemleri, güvenlik uygulamaları ve tedarik zinciri uygulamaları bulunmaktadır. Önceleri telsiz algılayıcı ağ sistemleri, algılayıcılar yardımıyla elde edilen ortam verilerinin belli aralıklarla bir merkeze iletimini sağlamaktaydı. Günümüzde ise ses ve görüntü haberleşmesi sağlayan elektronik birimlerin (kamera, mikrofon gibi) ucuzlaması ve boyutlarının küçülmesi, algılayıcı ağların çoklu ortam verilerinin iletiminde kullanılmasına imkan vermiştir.

Bu tez çalışmasının temel amacı, ses haberleşmesi amacıyla kullanılabilecek yapıda bir telsiz algılayıcı düğüm tasarlamak ve bu düğümde çalışmak üzere seçilecek bir işletim sistemini düğüme uyarlamaktır.

Tez kapsamında ilk olarak geliştirilmiş diğer ses haberleşmesi sağlayabilen telsiz algılayıcı ağlar ve bu ağların yapısında kullanılmış olan düğümler incelenmiştir. Bu inceleme sonucunda tasarlanacak olan düğüm üzerinde kullanılacak elektronik birimler ve işletim sistemi belirlenmiştir.

Telsiz algılayıcı ağlar için geliştirilmiş bir işletim sistemi seçilerek, tasarlanan düğüme uyarlanmış ve ses haberleşmesinin desteklenmesini sağlayacak gerekli sürücülerin yazılımı gerçeklenmiştir. Seçilen elektronik birimlerin ve gerçeklenen yazılımın denenmesi için bir test ortamı edinilmiştir. Bu test ortamında yapılan denemeler sonucunda da düğüm tasarımı geliştirilerek son halini almıştır.

Anahtar kelimeler: Telsiz Algılayıcı Ağlar, Telsiz Çoklu Ortam Algılayıcı Ağlar, Algılayıcı Düğümler, Gömülü İşletim Sistemleri, Gömülü Sistemler.

ABSTRACT

In the recent years wireless sensor nodes' usage on collection of environment data has been growing. In the late 90's the modern deployment of small sensor nodes extended the usage of this technology to several areas. First examples of wireless sensor networks were built for military applications. Nevertheless in the present day wireless sensor networks are being used in habitat monitoring, intelligent agriculture, industrial monitoring and control, home automation, security applications and assert tracking and supply chain management applications. Whilom wireless sensor networks have focused on applications where sensor data were reported periodically to a sink. Recent advances in electronical systems for voice and video communications have enabled the use of wireless sensor networks in multimedia applications.

The aim of this project is to design a wireless sensor node with the ability of voice transmission and to port an open source wireless sensor network operating system to this node.

Initially we have analyzed several wireless sensor networks projects and nodes that are used for voice transmission. As a result, we have constructed the hardware architecture of our node and chosen components and the operating system that will be used.

We ported the Contiki operating system to our node and added some drivers to support voice transmission over the network. In order to test and improve the designed architecture and the software developed we have used the MSP430F5438 experimenter board.

Keywords: Wireless Sensor Networks, Wireless Multimedia Sensor Networks, Sensor Nodes, Embedded Operating Systems, Embedded Systems.

1. GİRİŞ

Telsiz algılayıcı ağların geleneksel kullanımı, düşük çalışma çevrimli (low duty cycle) uygulamalarda, algılayıcı verisinin periyodik aralıklarla rapor edilmesi üzerinedir (Mangharam, 2006). Önceleri telsiz algılayıcı düğüm yapılarının getirdiği enerji kısıtı ve düşük maliyet gereksinimleri çoklu ortam uygulamaları için telsiz algılayıcı ağların kullanımını kısıtlamaktaydı. Ancak günümüzde, mikrodenetleyiciler, telsiz iletişim sistemleri ve algılayıcı teknolojilerindeki gelişmeler sonucu, ses haberleşmesine olanak veren; yüksek işlem gücü, düşük enerji tüketimi ve yüksek veri iletişim hızına sahip, telsiz algılayıcı düğümler geliştirilmeye başlanmıştır.

Bu proje kapsamında, telsiz algılayıcı düğümler üzerinden ses haberleşmesi sağlanmasına imkan sunacak gerekli yazılım bileşenlerinin oluşturulması ve ses haberleşmesi yeteneğine sahip bir telsiz algılayıcı düğüm tasarımı yapılması amaçlanmıştır.

Tasarım aşamasından önce, son yıllarda gerçeklenmiş ses haberleşmesi sağlayan telsiz algılayıcı ağ projeleri incelenmiş, teknolojideki gelişmelerin ne yönde olduğu belirlenmiştir. Sonraki adımda ise proje kapsamında tasarlanmış düğümün donanımsal özelliklerine ve içereceği algılayıcılara karar verilmiş, tasarım için gerekli yazılım bileşenleri gerçeklenmiştir.

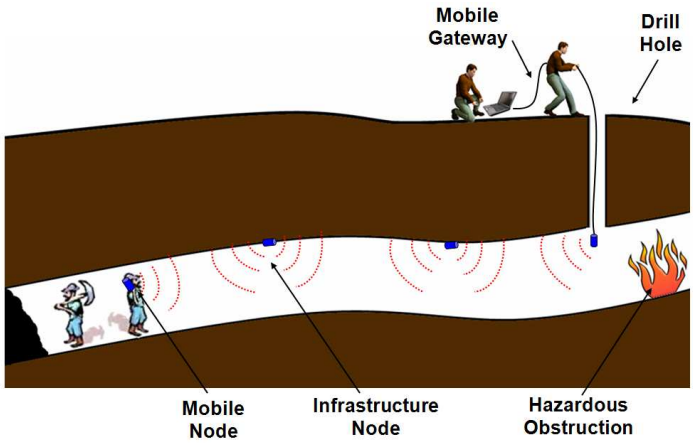
Tezin 2. bölümünde ses haberleşmesi için daha önce oluşturulmuş telsiz algılayıcı ağ projelerinin özellikleri, 3. bölümde bu projelerde kullanılan düğüm yapıları incelenmiştir. 4. bölümde planlanan tasarımın özellikleri, 5. bölümde telsiz algılayıcı ağlarda kullanılmak üzere geliştirilmiş işletim sistemleri incelenmiştir. 6. bölümde geliştirilen telsiz çoklu ortam algılayıcı düğümün tasarımından ve test ortamından, 7. bölümde gerçeklenen yazılımdan bahsedilmiş, 8. bölümde de sonuçlar bildirilmiştir.

2. SES HABERLEŞMESİ İÇİN GELİŞTİRİLMİŞ TELSİZ ALGILAYICI AĞLAR

Mikro-elektromekanik sistemlerdeki gelişmeler sonucu kullanılmaya başlanan düşük maliyetli mikrofon ve hem düşük maliyetli hem de yüksek işlem gücüne sahip mikrodenetleyici birimleri, telsiz çoklu ortam algılayıcı düğümlerin gelişimini hızlandırmıştır [1]. Ortamdan gerçek zamanlı ses verisi algılamak üzere telsiz algılayıcı düğümler geliştirilmeye başlanmıştır. Bu bölümde ses haberleşmesi sağlayan telsiz çoklu ortam algılayıcı ağ tasarımlarına sahip Firefly ve QVS projeleri hakkında bilgi verilmiştir.

2.1 Firefly Projesi [3]

Firefly projesi, kömür madenleri gibi geniş, kapalı, ulaşımı zor ve değişken ortamlarda yaşanan afetlerde kurtarma çalışmalarının haberleşme sıkıntısı sebebiyle zorlaştığına dikkat çekerek, bu alanlarda kullanılmak üzere telsiz algılayıcı ağlar üzerinden ses yayını yapabilen bir sistem sunmaktadır. Bu gibi ortamlarda yaşanan afetlerde ölümlerin genellikle yaralanma sebebiyle değil ulaşılamayan noktalardaki işçilerin oksijensiz kalmaları sebebiyle yaşandığı belirtilmiştir. Şekil 2.1'de amaçlanan haberleşme ve kurtarma yöntemi görselleştirilmiştir. Firefly düğümleri bu gibi acil durumlar için kısa süreli, tek taraflı ses iletiminde kullanılmak üzere geliştirilmiştir.



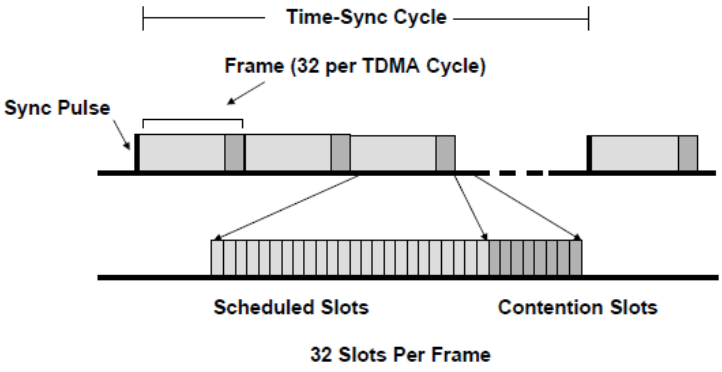
Şekil 2-1 Algılayıcı ağların acil durum haberleşme aracı olarak kullanım örneği [3]

Firefly projesi, geliştirilen düşük maliyetli bir telsiz algılayıcı düğümü, bir algılayıcı ağ işletim sistemini ve gerçek zamanlı bir link katmanı ile ağ zamanlayıcısını kapsamaktadır. TDMA tabanlı bir ağ zamanlayıcısı sunan tasarım, zaman senkronizasyonu ve ses iletişimi için kullanılan dual radyo sistemi içermektedir.

Tasarım; gerçek zamanlı yayında kararlı bir çerçeve sağlamak için, çakışmaları önleme

yeteneğine sahip TDMA bağlantı katmanı ve her düğümde çoklu görev paylaşımı ile düşük miktarlı, düşük karmaşıklığa sahip bir ağ yapılandırması sunmaktadır. (Mangharam, 2006)

Firefly düğümleri; üzerinden ses verisinin aktarıldığı sabit düğümler ve kullanıcıların taşıdıkları hareketli düğümler olmak üzere iki farklı yapıdadır. Tüm düğümler, üzerlerindeki iki radyodan biri ile ses verisi yayını yaparken, diğeri ile düğümler arası zaman senkronizasyonu sağlayarak TDMA tabanlı haberleşmeye olanak sunmaktadır.



Şekil 2-2 Firefly TDMA tabanlı link katmanı [3]

Şekil 2.2'de görüldüğü gibi her zaman devri (time cycle) 32 çerçeveye (frame) ayrılmış, tüm çerçeveler de 32 adet 6ms'lik dilimlere (slot) bölünmüştür. Bu durumda bir zaman devri 6.144 saniyedir. Düğümler yayın hakkı için bir devirde en çok 1024 dilim olmak üzere ihtiyaçları kadar dilim seçerler. Düğümler; bu zaman dilimlerini, uyanık kalacakları zaman aralıklarını belirlemek için kullanarak ideal güç tüketimi sağlayabilmektedirler.

Bu proje telsiz algılayıcı ağlar üzerinden ses haberleşmesi konusunda önemli bir gelişme sağlamıştır. Ancak düğüm tasarımları şimdilik tek taraflı ve kısa süreli ses iletimine izin verdiğinden tam bir haberleşme alt yapısında kullanılmalarına olanak vermemektedir.

2.2 QVS Projesi [2]

QVS (Quality-aware Voice Streaming for Wireless Sensor Networks) Projesi, telsiz algılayıcı ağların ses haberleşmesi için kullanımını amaçlayan güncel projelerde, ses iletiminin kalitelileştirilmesini sağlamak üzere yapılmış bir çalışmadır. (Li, 2009).

Deneysel araştırmalar, telsiz algılayıcı ağlarda paket kayıp oranlarının yüksek olduğunu göstermektedir. Gerçek zamanlı ses haberleşmesinde paket kayıpları iletişimi güçleştirmektedir. QVS, yayındaki ses kalitesini otomatik olarak değerlendirip, verinin

kopyalanması ya da sıkıştırılması için geri besleme sağlayan deneysel bir model sunar. Bu model değişken hat kalitesi ve ağ topolojilerinde güçlü bir ses yayın kalitesi sağlamayı amaçlamaktadır. (Li, 2009)

Ses iyileştirmesinin sağlanması için QVS;

- ses verisini sağlayan düğüm üzerinde, gecikme ve paket kaybı oranı gibi iletim kalite değerlerinin ölçümlerini tutar,
- değişken bağlantı koşullarına adapte olarak dinamik veri sıkıştırma/kopyalama işlemleri yapar,
- her düğüm için ağ kapasitesine göre yayınlanan veri oranlarını, ses kalite gereklerine göre belirler. (Li, 2009)

QVS projesi kapsamında bazı ses kalite ölçüm teknikleri incelenmiştir. İncelenen objektif yöntemde, gönderici tarafında ses verisinin enkode edilmeden önceki hali ile alıcı tarafında dekode edildikten sonraki hali karşılaştırılır. İki veri arasındaki fark ve tüm ses verileri için bu farkların toplamının sıfıra yakınlığı ses kalitesini vermektedir. Bu yöntem ağdaki gecikme oranı, enkode/dekode işleminin verdiği hata payı gibi değişkenleri hesaba katmamaktadır. Bir subjektif yöntem olan MOS (Mean Opinion Score) ise gerçek zamanda ses haberleşmesi sağlamaya çalışan deneklerin aldıkları ses verisini 5 üzerinden puanlamasına dayanmaktadır. Bu yönteme göre, duyulan sesin kalitesini 2.6'nın üzerinde rapor eden dinleyici ses kalitesinin kabul edilebilir düzeyde olduğunu söylemektedir. Bu subjektif yöntem deneysel olduğundan gerçek haberleşme zamanında kullanılmaya uygun değildir.

Proje kapsamında incelenen adaptif çözüm ise R-value yöntemi olmuştur. Bu yöntemde en kaliteli iletim katsayısından gecikme değeri ve paket kaybı çıkarılarak elde edilen anlık kalite değeri gerçek zamanlı veri sıkıştırma/kopyalama işlemleri için kullanılmaktadır. Yöntemde aşağıdaki denklem (Şekil 2-3) kullanılmıştır.

$$R = R_0 - I_d - I_e$$

Şekil 2-3 R-Value Yöntemi

Bu denklemdeki R_0 değeri en yüksek R değerini vermektedir. Bu değer MOS yöntemindeki 4.41 puanına denk gelen 93.2 değeridir. I_d noktadan noktaya gecikme değerini ve I_e de paket kaybını göstermektedir. QVS modelinde R-value yöntemi kullanılarak adaptif hat kalitesi ölçülmüş ve bu kalite oranına göre kopyalama/sıkıştırma işlemleri gerçekleştirilmştir. Bu işlemlere ait bilgi aşağıda verilmiştir. (Li, 2009)

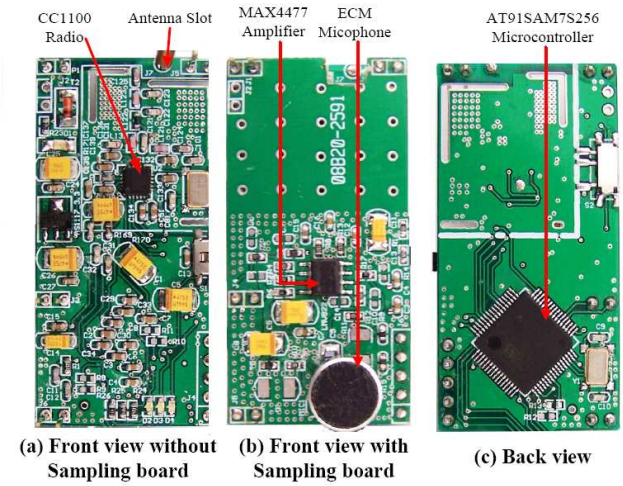
QVS modeli; R-value yöntemi ile hattın kalite parametrelerini toplayarak dinamik ses iyileştirmesi yapmaktadır. Eğer iki düğüm arasında ağ durumu yoğun ise sıkıştırma oranı yükseltilerek veri büyüklükleri azaltılmakta, ağın hata oranı yüksekse paketlerin kopyalanıp gönderilme oranı arttırılmaktadır. Fakat gönderimin arttırılması ağın yoğunluk durumunu etkilediğinden sıkıştırma oranının artmasına sebep olmaktadır. QVS, bu iki oran arasında gerçek zamanda bir denge kurmayı amaçlamıştır. (Li, 2009)

QVS projesi kapsamında geliştirilen modelin test edilmesi için, yüksek bant genişliğinde, kısa süreli ses haberleşmesi sağlayabilen SenEar telsiz algılayıcı düğümü geliştirilmiştir. Bu düğümle ilgili ayrıntılı bilgiye 3. bölümde yer verilecektir.

3. SES HABERLEŞMESİNDE KULLANILAN TELSİZ ALGILAYICI DÜĞÜMLER

Bu bölümde 2. bölümde incelenmiş olan Firefly ve QVS projelerinde gerçeklenen telsiz algılayıcı düğüm yapıları hakkında bilgi verilecektir.

QVS [2] çalışması kapsamında geliştirilen SenEar algılayıcı düğümü, kısa süreli fakat yüksek aktarım hızı ile kaliteli telsiz ses haberleşmesi sağlayan bir algılayıcı düğümdür. Firefly ise temel olarak TDMA tabanlı bir ağ zamanlayıcısı kullanarak iletişim sırasında çakışmaları en aza indirecek bir tasarım sağlamaktadır. Firefly, her yeni TDMA iletişim döngüsünün başlangıç zamanını gösteren donanım tabanlı bir zaman senkronizasyonu sağlar.



Şekil 3-1 SenEar algılayıcı düğümü [2]

Şekil 3-2 Firefly algılayıcı düğümü [25]

Şekil 3.1 ve Şekil 3.2'de devre görünümleri verilen Firefly ve SenEar düğümlerine ait donanım özellikleri Çizelge 3.1'de karşılaştırmalı olarak verilmiştir.

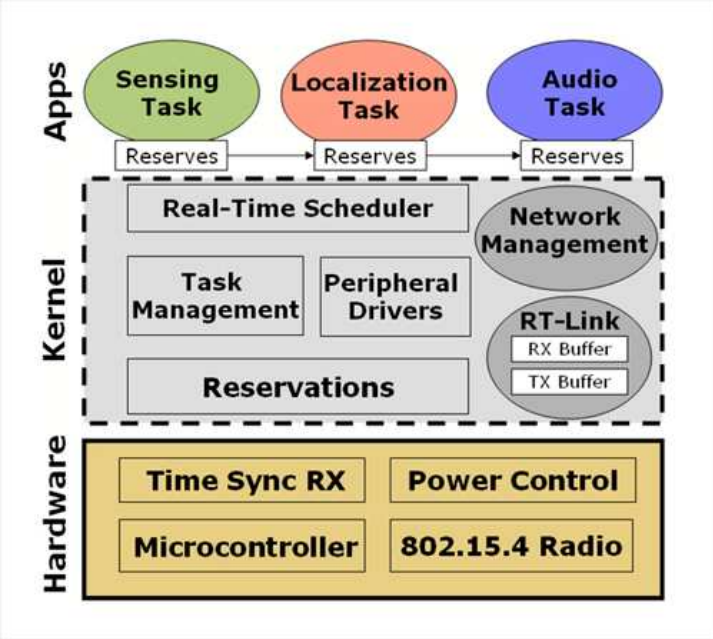
Çizelge 3.1 Firefly ve SenEar düğümlerinin donanımsal özellikleri

	Firefly	SenEar
İşlemci birimi	Atmel Atmega32	Atmel AT91SAM7S256
İşlemci mimarisi	RISC	RM
İşlemci bit uzunluğu	8 bit	32 bit
Bellek miktarı	2 KB	64 KB
Veri aktarım hızı	256 Kbps	500 Kbps
Güç tüketimi	-	60 mA
İşletim sistemi	Nano-RK RTOS	SenEar OS kernel

Çizelge 3.1'de görüldüğü gibi Firefly algılayıcı düğümü 8 bitlik düşük işlemci bit uzunluğuna ve 2 KB ile sınırlı bellek miktarına sahiptir. (Mangharam, 2006)'da ayrıntılı güç tüketimi bilgisine yer verilmemiş olsa da amaçlanan kısa süreli acil durum gereksinimlerine yanıt verebildiğinden bahsedilmiştir. SenEar düğümü yapısında ise 32 bit işlemci ve yüksek veri aktarım hızı dikkat çekmektedir.

3.1 Firefly ve SenEar Düğümleri Üzerinde Çalışan İşletim Sistemleri

Firefly düğümü, Carnegie Mellon Üniversitesi'nde geliştirilen işlem üstünlüğü kullanmayı destekleyen (preemptive), ağ üzerinde rezervasyon tabanlı (reservation-based) gerçek zamanlı bir işletim sistemi olan Nano-RK'yı [25] kullanmaktadır. Nano ismi bu işletim sisteminin hafızada kapladığı alanı temsil etmektedir. İşletim sisteminin çekirdek yapısı 2 KB RAM ve 18 KB ROM alanı kullanımıyla telsiz algılayıcı ağlar için hafızayı elverişli kullanan gömülü bir işletim sistemidir. Nano-RK işletim sistemi zaman paylaşımı prensibiyle çalışmaktadır.



Şekil 3-3 Firefly / Nano-RK çalışma prensibi [25]

Nano-RK, Firefly projesi kapsamında geliştirilmiş ve bu düğüm üzerinde kullanılmıştır. Diğer platformları da destekler hale gelmesi üzerinde çalışmalar devam etmektedir. [25]

SenEar algılayıcı düğümü üzerinde ise QVS projesi kapsamında geliştirilen basit bir çekirdek işletim sistemi çalışmaktadır. Bu çekirdek işletim sistemi, görev yönetimi, işlem kesimi istek kuyruğu (interruption request queue), yazılımsal zamanlayıcı ve çevre birimi sürücülerini içermektedir. Görev yönetim biriminde ilk gelen ilk işlenir (first come first served) yani işlem üstünlüğüne yer verişlmemiştir. QVS projesi kapsamında gelecekte SenEar düğümü üzerinde çalışmak üzere, TinyOS işletim sisteminin kullanılacağı belirtilmiştir. [2]

4. TASARIM ÖZELLİKLERİ

Proje kapsamında öncelikle hedeflenen; ses haberleşmesi için tasarlanmış bir telsiz algılayıcı düğümde çalışmak üzere bir gömülü işletim sisteminin uyarlamasının yapılmasıdır. Telsiz algılayıcı ağlar için kullanılan gömülü işletim sistemleri düğüm yapısına göre farklılıklar göstermektedir. Bu sebeple öncelikle bir telsiz çoklu ortam algılayıcı düğümü tasarlanmış daha sonra gömülü işletim sistemi çalışmaları yapılmıştır.

Ses haberleşmesi için kullanılan telsiz algılayıcı düğümleri oluşturan en temel birimler, ses bilgisinin alımını sağlayan mikrofon birimi, toplanan bilginin işlenmesini ve düğümün kontrol edilmesini sağlayan işlemci birimi, bilginin diğer düğümlere veya merkezlere aktarılmasını sağlayan kısa mesafeli radyo iletişim birimi ve bilginin saklanmasını sağlayan depolama birimidir. Düğümlerin en temel özellikleri sınırlı enerji kaynaklarına ve küçük boyutlara sahip olmalarıdır. Ancak ses haberleşmesinde kullanılacak düğümlerin enerji kaynağı bakımından dayanıklı olmaları beklenmektedir. Çoklu ortam algılayıcı düğümlerinin bazı uygulamalarda elde edilen veri üzerinde bir takım işlemler yapmaları da gerekecektir. Bu işlem yüklerine rağmen beklenen, düğümün olabildiğince uzun süre enerji kaynağını kullanabilmesidir. Bu durum, telsiz algılayıcı düğümlerin yazılımsal ve donanımsal tasarımlarını farklılaştırmaktadır.

Bu bölüm kapsamında ses haberleşmesinde kullanılan telsiz algılayıcı düğümleri oluşturan birimler için gerekli temel özellikler ortaya konmuş, bu özellikler ışığında daha önce tasarlanmış olan telsiz algılayıcı düğümlerde kullanılmış ve kullanılabilmeye aday birimlerin özellikleri karşılaştırmalı olarak incelenmiştir.

4.1 İşlemci Birimi

Ses haberleşmesinde kullanılacak bir telsiz algılayıcı düğümde işlemci birimi seçimi yaparken dikkate alınması gereken en önemli özellik, işlem gücü / enerji tüketimi oranının yeterli süre ses haberleşmesine imkan verecek ideal bir seviyede olmasıdır.

Telsiz çoklu ortam algılayıcı düğüm yapısında kullanılacak işlemci biriminin, yeterli bit genişliğinde bir mimariye ve geniş bir adresleme kapasitesine sahip olmasının yanı sıra çoklu ortam verisini kabul edilebilir süreler içerisinde işleyebilmesi için yüksek çalışma frekansına sahip olması beklenmektedir (McIntire, 2009).

Düğüm yapısında kullanılan işlemci biriminin, enerji tasarrufu sağlayabilmek için, çalışma hızını dinamik olarak değiştirebilmesi ve buna bağlı olarak enerji tüketimini yönetebilme

imkanı sunabilmesi gerekmektedir (Lynch, 2005).

Kullanılacak olan işlemci biriminin bir diğer özelliği ise; düğümün yapısında bulunan ses algılayıcısı, radyo iletişim birimi, ikincil bellek birimi ve algılayıcılar ile iletişimini sağlayacak sayıda ve çeşitte bağlantı arayüzüne imkan vermesi olmalıdır. Ses haberleşmesinde telsiz algılayıcı ağların kullanıldığı uygulamaların birçoğu algılayıcılar üzerinden toplanan bilginin en hassas şekilde değerlendirilmesine ihtiyaç duyar. Bu yüzden işlemci birimi üzerinde yer alan ve algılayıcılardan okunan bilginin değerlendirilmesinde kullanılan analog sayısal çeviricinin de (ADC) bu ihtiyaca cevap verebilmesi gerekmektedir. Bu sebeple işlemci seçimi yapılırken ADC bit hassasiyeti de bir kriter olarak değerlendirilmelidir.

4.2 Telsiz Haberleşme Birimi

Ses haberleşmesinde kullanılacak telsiz çoklu ortam algılayıcı düğümlerin haberleşmeleri sırasında göndermeleri gereken bilgi geleneksel telsiz algılayıcı ağlara oranla daha fazladır (Tayşi, 2006). Telsiz çoklu ortam algılayıcı düğümünde kullanılacak olan telsiz haberleşme birimi; güç tüketimi, veri aktarım mesafesi ve veri aktarım hızına göre değerlendirilmelidir.

Telsiz haberleşme birimi; düğüm çalışma zamanının büyük bölümünde pasif durumda kalacak, yalnızca haberleşme esnasında aktif duruma geçecektir. Bu nedenle haberleşme biriminin pasif durumdaki güç tüketiminin en az seviyede olması önemlidir.

4.3 İkincil Bellek Birimi

Düğümün yapısında, ses verisinin işlenmesi gerektiği durumlarda kullanılmak üzere bir ikincil bellek birimi bulunması uygun olacaktır. Algılayıcıdan alınan ses verisi işlenmesi istendiği durumlarda bu ikincil bellek biriminden faydalanılacaktır.

Düğüm bünyesinde bulunan diğer tüm birimler gibi ikincil bellek birimininde düşük güç tüketimine sahip olması ve kullanılmadığı durumlarda düşük seviyede güç harcaması gerekmektedir. Ayrıca ikincil bellek biriminin okuma ve yazma işlemleri için de düşük güç tüketimine sahip olması beklenmektedir. Pasif çalışma seviyesinden aktif çalışma (okuma veya yazma) seviyesine geçiş süresi de kısa olmalıdır. Bu sürenin uzun olması durumunda mikro denetleyici aktif halde bekler durumda kalacak ve bu durum düğümün güç tüketimini arttıracaktır.

Düğümün yapısında kullanılacak olan ikincil bellek birimi, SPI, I2C veya benzeri standart bir

arabirim üzerinden mikrodenetleyici birime bağlanabilmelidir. Bu şekilde ikincil bellek birimi ile ilgili işlemlerin gerçekleştirilmesi, yazılımsal açıdan kolaylaşacaktır. Düğüm üzerinde standart bir arabirime sahip ikincil bellek biriminin kullanılması, bu birimin ihtiyaç duyulduğunda, yazılımsal bir değişiklik gerektirmeden daha hızlı ve/veya daha yüksek kapasiteli bir ikincil bellek birimi ile değiştirilmesine de imkan verecektir.

4.4 Ses Algılayıcısı

Sesli haberlemede kullanılacak telsiz çoklu ortam algılayıcı düğümün en temel birimlerinden biri de ses algılayıcısı yani mikrofondur. Algılayıcı düğüm için kullanılacak mikrofonun, ortam dinlemesi gibi uygulamalarda da kullanılabilmesi açısından, çok yönlü ses alımına izin vermesi (omni-directional) gerekmektedir.

Diğer birimler gibi ses algılayıcısının da güç tüketiminin düşük seviyede olması beklenir. Proje kapsamında hedeflenen insan sesi aktarımı için kullanılabilecek bir çoklu ortam algılayıcı düğümü tasarlamak olduğundan, ses algılayıcı biriminin tipik insan sesi frekansı olan 4KHz’te çalışmaya uygun olması gerekmektedir. Belirtilen kriterlere uygun ses algılayıcı birimi düğüme eklenen audio jack modülüne takılarak kullanıbilir.

5. TELSİZ ALGILAYICI DÜĞÜMLER İÇİN GELİŞTİRİLMİŞ İŞLETİM SİSTEMLERİ

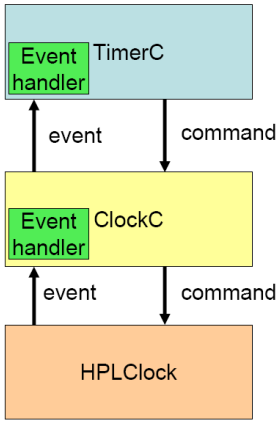
Telsiz algılayıcı ağlarda kullanılan işletim sistemleri, farklı gereksinimleri ve kaynak kısıtlarından dolayı, genel amaçlı işletim sistemlerinden daha az karmaşıktır. Örneğin, hafıza ve hafıza adresleme kısıtları, sanal hafıza (virtual memory) kullanımını gereksiz veya imkansız kılmaktadır. Fakat telsiz algılayıcı düğüm yapıları geleneksel gömülü sistemlerin yapılarıyla benzerlik gösterir ve bununla birlikte gömülü işletim sistemleri telsiz algılayıcı ağ yapılarında kullanılabilir.

Telsiz algılayıcı ağlar üzerinde kullanılmak üzere geliştirilmiş pek çok işletim sistemi mevcuttur. Bu işletim sistemlerinin ortak amacı telsiz algılayıcı düğümlerin enerji tüketimlerini minimum seviyede tutarak pil ömürlerini uzatmaktır. Proje kapsamında kullanılacak işletim sisteminin seçilmesi için, telsiz algılayıcı ağlarda sıklıkla kullanılan TinyOS [26], Mantis [28] ve Contiki OS [27] işletim sistemleri incelenmiştir.

5.1 TinyOS

UC Berkeley Üniversitesi tarafından algılayıcı düğümler üzerinde kullanılmak üzere geliştirilmiştir. NesC isimli, bu işletim sistemi için özel olarak C programlama dilinden geliştirilmiş yeni bir programlama dili kullanılmıştır. TinyOS, bileşen tabanlı (component-based) bir mimariye sahiptir. Basit bir zamanlayıcı ve bir yazılımsal bileşenler bütününden oluşur. TinyOS bünyesinde bulunan bileşen tabanlı kütüphaneler kullanılarak uygulamaya özel çözümler üretilmektedir. Bu kütüphaneler arasında iletişim protokolleri, algılayıcı sürücüleri ve veri toplama araçları (data acquisition) bulunmaktadır. Güç tüketimini en aza indirmek için, üzerinde kullanıldığı algılayıcı düğümün gerekli işlemi kısa zamanda gerçekleştirip uzun süre uyuması için tasarlanmıştır. Sorgulama (polling) tekniğine dayanmayıp kesme (interrupt) tabanlı çalışma yapısına sahiptir.

TinyOS yapısında, modüller ve konfigürasyonlar olmak üzere iki tip bileşen modeli mevcuttur. Modüller uygulama davranışını belirlerken, konfigürasyonlar bileşenleri birbirine bağlar. Şekil 5-1'de bileşen yapısının oluşturduğu hiyerarşi görülmektedir. Komutlar ve olaylar çağıran göreve geri dönüş sağlar. Olaylar komutları çağırabilirken tersi mümkün değildir. [5]



Şekil 5-1 TinyOS Bileşen Hiyerarşisi [5]

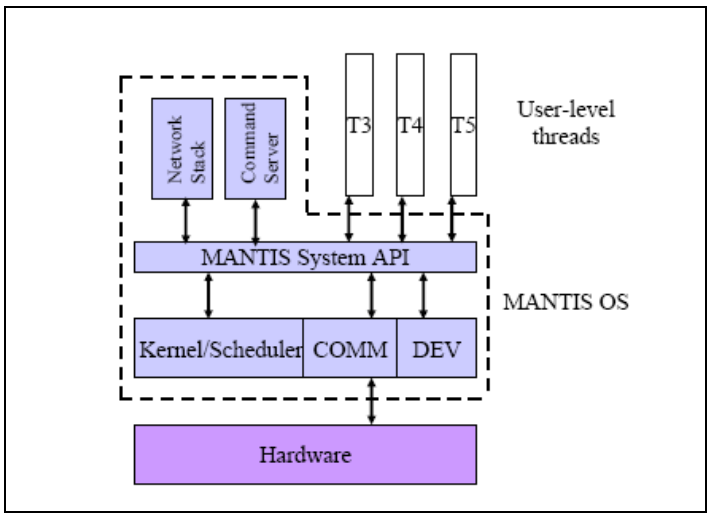
TinyOS veri hafıza modelinde, dinamik hafıza tahsisine izin vermemektedir. TinyOS'un yapısında 2 seviyeli (2 level) bir zamanlama (scheduling) yöntemi kullanılmıştır. Bu yöntemdeki birinci grubu görevler (tasks), ikinci grubu ise olaylar (events) oluşturmaktadır. Görevler (task) öncelikli işi yapmaktadır. Atomik olarak çalışmakta ve sonuca kadar ara vermeden ilerlemektedirler, ancak olaylar (event) tarafından kesintiye uğrayabilirler. Görevler basit "ilk giren ilk çıkar" (first in first out) işlem yönetimi ile yürütülürler. Geliştirilen yapıda olaylar, başka olayları veya görevleri oluşturabilirler. Ancak görevler, başka görevleri veya olayları oluşturamazlar. Olaylar donanımsal kesmeler (interrupts) tarafından oluşturulan düşük seviyeli olaylar ve yazılımlar tarafından oluşturulan yüksek seviyeli olaylar olarak ikiye ayrılmıştır. Olaylar fonksiyon çağırılması (function calls) ile düşük seviyeden yüksek seviyeye doğru yayılabilirler. TinyOS'un yapısında yüksek seviyedeki bileşenler daha düşük seviyedeki bileşenlere komutlar (commands) aracılığıyla istekte bulunur. [35]

TinyOS işletim sistemi birçok değişik donanım üzerinde çalışabilmektedir ve birçok projede kullanılmıştır. Bu donanımlardan bazıları; UC Berkeley Üniversitesi tarafından geliştirilmiş olan WeC, Rene, Rene2, Dot, Mica, Mica2Dor, Mica2 ve Telos'tur. TinyOs'un diğer kullanıldığı projelerden bazıları ise, Calamari[29], CotsBots[30], Great Duck Island[34], Firebug[31], TinyDB[32], TinyGALS[33]'tır.

5.2 MANTIS

MOS (Mantis Operating System), telsiz algılayıcı düğümler için multi-threaded bir işletim sistemi oluşturulması amacıyla Colarado Üniversitesi Bilgisayar Bilimleri Bölümü tarafından geliştirilmiştir. Şekil 5-2'de MOS'un geleneksel katmanlı mutli-threaded mimarisi

görülmektedir. Uygulama threadleri Mantis System API tarafından çekirdek işletim sisteminden ayrılmıştır. Bu sayede Mantis farklı platformları destekler hale getirilmiştir. MOS özet olarak sadeleştirilmiş ve düşük güç tüketimi sağlayan bir zamanlayıcı, kullanıcı düzeyinde bir ağ yığıtı (user-level network stack) ile farklı platformlarda kullanılabilecek aygıt sürücülerinden oluşur. (Bhatti 2005)



Şekil 5-2 Mantis işletim sisteminin blok diagramı (Bhatti 2005)

MOS, C programlama dili kullanılarak yazılması sebebiyle taşınabilir bir yapıya sahiptir. Kullanıcıya sunduğu kütüphaneler sayesinde, çekirdek yapının karmaşıklığına girmeden programlaması basit bir arayüz sunmaktadır. Şekil 5-3'te basit bir algıla/ilet uygulama kod parçacığı görülmektedir. (Bhatti 2005)

```
#include <inttypes.h>
#include "led.h"
#include "dev.h"
#include "com.h"

void start(void)
{
    uint16_t delay;
    uint8_t value;
    comBuf send_pkt;

    value = dev_get(DEV_MICA2_LIGHT);
    mos_led_toggle(0);

    send_pkt.data[0] = value;
    send_pkt.size=1;
    com_send(IFACE_RADIO, &send_pkt);
}
```

Şekil 5-3 Mantis - Algıla/İlet fonksiyonu içeren örnek kod

MOS yapısındaki çekirdek, zamanlayıcı, ağ yığıtı toplamda 500 byte RAM ve yaklaşık 14 KB ikincil bellek alanı kaplamaktadır. (Bhatti 2005)

5.3 Contiki OS

Contiki OS, İsveç Bilgisayar Bilimleri Enstitüsü'nde araştırmacı olarak görev yapan Adam Dunkels tarafından geliştirilmiştir. C dilinde yazılmış bir işletim sistemi olan Contiki OS, açık kaynaklı, taşınabilir ve hafıza kısıtı olan gömülü sistemler için geliştirilmiştir. Ayrıca ağ desteği ve çoklu-görev (multi-tasking) uyumluluğu sağlayan gerçek zamanlı bir yapıya sahiptir. Çoğu zaman kapladığı alan 4kB'tan fazla olmamaktadır. Bu sebeple kaynak kısıtlı sistemlerde kullanılmaya elverişlidir. (Dunkels, 2005)

Telsiz algılayıcı ağlarda pek çok düğümü aynı iş için programlanma ihtiyacı vardır. Contiki OS ağ üzerinden tekrar programlama ve güncelleme yeteneğine sahip olduğundan telsiz algılayıcı düğümlerde programlama kolaylığı sağlamaktadır. Çizelge 5-1'de Contiki OS'un kapladığı hafıza alanları ortalama olarak verilmiştir.

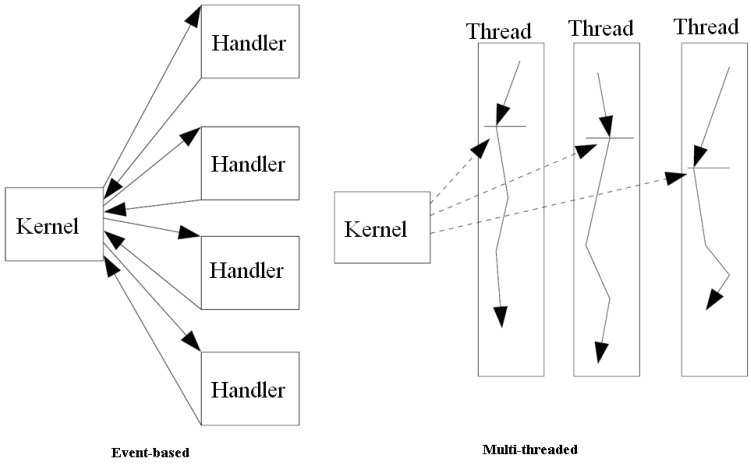
Çizelge 5.1 Contiki OS büyüklüğü (byte)

Modül	MSP430 Kodu	AVR Kodu	RAM
Çekirdek	810	1044	10 + e + p
Program loader	658	-	8
Multi-threading kütüphanesi	582	678	8 + s
Zamanlayıcı kütüphanesi	60	90	0
Hafıza yöneticisi	170	226	0
Event log replicator	1656	1934	200
µIP TCP/IP stack	4146	5218	18 + b

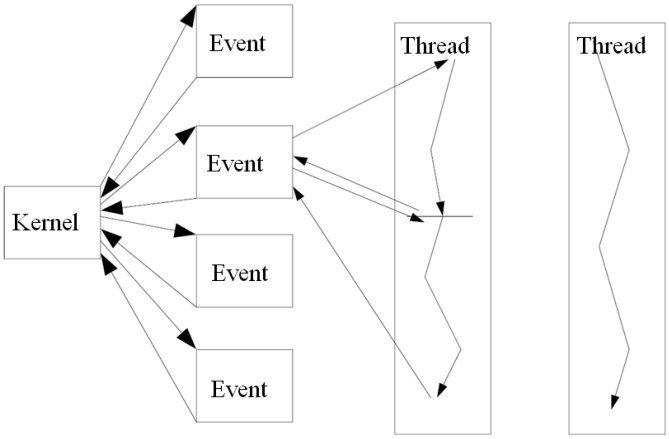
Geleneksel işletim sistemi yapılarında tüm yazılım sistemi bir bütün binaryden oluşur. Contiki OS'ta ise düğümler, çekirdek yapıyı sabit tutarak aralarında yeni uygulamalar paylaşabilmektedir. Düğümler ağ ya da EEPROM üzerinden tekrar programlanabilir durumdadır. Bu tasarımın ağ üzerinde hızlı yeniden programlama yapabilme avantajı vardır.

Contiki OS; olay tabanlı (event-based) yazılımların sağladığı küçük boyut, düşük context-

switch yükü avantajlarıyla, threaded çalışmanın getirdiği uzun işlem yapabilme özelliklerini bir arada sağlayan bir işletim sistemidir. Şekil 5-4'te olay tabanlı ve multi threaded işletimin karşılaştırması görülmektedir. Şekil 5-5'te görülen Contiki OS çekirdek yapısı olay tabanlı çalışmaktadır. Böylece ana süreçlerin daha az yer kaplaması ve süreçler arası geçisin hızlandırılması sağlanmıştır. Multi-threaded yapısı ise bir kütüphane olarak gerçeklenmiştir ve bu yapı uzun süreli hesaplama gerektiren süreçlerin diğer süreçleri tıkamasını engellemek amacıyla kullanılabilmektedir.



Şekil 5-4 Olay tabanlı ve multi-threaded tasarımlar



Şekil 5-5 Olay tabanlı ve threaded yapıyı birleştiren tasarım

Olay tabanlı işletim sistemlerinin çekirdek yapıları (kernel) durum makineleri (state machines) gibi tasarlanmıştır ve geleneksel kod yazma tekniğine daha uzaktır. En önemli farkları bekle() (wait()) ifadesi ya da işlem üstünlüğü kullanma yetenekleri olmamasıdır. Bu eksiklik, uzun süre çalışan hesaplamalar sırasında başka olayların merkezi işlemci birimini

kullanmalarını engellemesidir. Çizelge 5.2'de olay tabanlı ve multi-threaded yazılım yapıları karşılaştırılmıştır.

Çizelge 5.2 Olay tabanalı ve multi threaded işletimin karşılaştırması [3]

	Olay tabanlı		Multi-threaded
-	wait() tanımı yapılamaz	+	wait() tanımı yapılır
-	işlem üstünlü yoktur (no preemption)	+	işlem üstünlüğü vardır (preemption possible)
-	State machines	+	Sıralı kod akışı
+	Küçük boyutlu kod	-	Büyük boyutlu kod
+	Süreçlerin tıkanması önlenir	-	Süreç tıkanma problemi
+	Verimli hafıza kullanımı	-	Daha fazla hafıza gereksinimi

5.4 İşletim Sistemi Seçimi

Proje kapsamında ilk olarak kullanılmaya başlanan işletim sistemi Mantis olmuştur. Bölüm 5.2'de verilen bilere ek olarak bu işletim sistemi, MSP430 mikrodenetleyicisi için Unix GCC (GNU Compiler Collection) tabanlı MSPGCC [4] derleyicisini kullanmaktadır.

İşletim sisteminin çekirdeği içerisinde Unix tabanlı işletim sistemlerinde kullanılan zamalayıcılara (scheduler) benzer bir zamanlayıcı kullanılmıştır. Zamanlama yöntemi olarak öncelik seviyelerine sahip round robin yöntemi kullanılmaktadır. Threadlerin eş zamanlılığın sağlanması için ikili ve sayan semaphore'lar kullanılmaktadır. Threadler arasındaki geçişleri donanımsal olarak oluşturulan bir zamanlayıcı kullanılır. Bu zaman aralığının normal değeri 10 ms'dir. Ancak bu değer yazılımsal olarak değiştirilebilir. Bu geçişler semaphore işlemleri veya sistem çağrıları ile de gerçekleştirilebilir. Oluşan kesmeler arasında sadece zamanlayıcı kesmesi çekirdek tarafından değerlendirilir. Diğer tüm kesmeler ilgili aygıt sürücülerine yönlendirilir. (Bhatti 2005)

Mantis OS; Imote2, MicroBlaze, MicaZ, TelosB gibi popüler bazı telsiz algılayıcı düğüm platformlarına uyarlanmıştır. Bu işletim sisteminde tanımlı platformlara ait ayrı bir klasör olmayıp, ortak uygulama ya da çekirdek yazılımlarında ayrıştırılmış kod parçaları halinde tanımlamalar yapılmıştır. Bu durum yeni platform oluşturmada kullanım kolaylığını azaltsa da çekirdeğin yapısını dağınık olmaktan kurtarmıştır.

Proje kapsamında incelenen ve kullanımına karar verilen bu işletim sistemine Mart 2010 tarihinde destek verilmemeye başlanmış ve tüm internet kaynakları silinmiştir. Bunun üzerine TinyOS ile Contiki OS arasında bir değerlendirme yapılmış (Çizelge 5.3) ve Mantis gibi C programlama diliyle yazılmış ve MSP430 mikrodenetleyicileri için MSPGCC derleyicisini kullanan Contiki OS kullanımına geçilmiştir.

Çizelge 5.3 Tiny OS & Contiki OS karşılaştırması

TinyOS	Contiki
Bileşen tabanlı, işlem önceliği kullanmayan (non-preemptive) çoklu görev yürütümü	Olay tabanlı, isteğe bağlı işlem önceliği ile (optional preemptive) çoklu görev yürütümü
Statik link	Dinamik link
nesC programlama dili	C programlama dili

Çizelge 5.3'te TinyOS ve Contiki OS karşılaştırması yapılmıştır. TinyOS'un aksine Contiki OS dinamik linklemeye izin vermektedir. Bu sayede ağ üzerinde düğümlerin programlamasından sonra ek yazılımlar yüklenmesi ve modifikasyonlar yapılması Contiki OS'ta çekirdek yapısının tekrar yüklenmesi gerekmeden çok daha kısa sürede gerçekleştirilebilmektedir. Contiki OS yapısının bir diğer avantajı da, TinyOS yalnızca FIFO (ilk giren ilk çıkar) olay kuyruklamasını desteklerken, Contiki'nin hem FIFO, hem de öncelik sırasıyla çalışan bir havuz ile olay kuyruklarını yönetmesidir.

6. TELSİZ ÇOKLU ORTAM ALGILAYICI DÜĞÜM TASARIMI

Tezin bu kısmında tasarlanan düğümü oluşturan birimlerde kullanılan parçaların seçilme işlemleri, seçilmiş olan parçaların test edilmesi için oluşturulmuş olan ortam ve düğümün tasarımı detaylı olarak anlatılmaktadır.

Tez kapsamında tasarlanan düğümün mikrodenetleyici birimi, telsiz haberleşme birimi, ikincil bellek birimi, kullanıcı arayüzleri, ses giriş zinciri, ses çıkış zinciri, güç yönetim birimi ve diğer algılayıcılar olarak 8 birimden oluşması tasarlanmıştır. Bu birimleri oluşturan parçaların seçimi ise tezin 4. bölümünde belirlenmiş olan tasarım özellikleri göz önünde bulundurularak yapılmıştır.

6.1 Mikrodenetleyici Birimi

Geliştirilen telsiz algılayıcı düğümün yapısında kullanılmak üzere, 2. bölümde yer alan telsiz algılayıcı ağların yapısında kullanılmış olan düğümler ve 3. bölümde incelenmiş olan genel amaçlı telsiz algılayıcı düğümlerin yapılarında kullanılmış olan mikrodenetleyici birimler incelenmiştir. Bu birimler, Atmel firması tarafından geliştirilmiş olan 32-bit Atmel AT91SAM7S256 (SenEar üzerinde) ve 8-bit Atmel ATmega32 (Firefly üzerinde) mikrodenetleyici birimleridir. Ayrıca Texas firmasının geliştiridiği, MSP ailesinden MSP430F149 ve MSP430F5438 mikrodenetleyicileri incelenmiştir.

Geleneksel telsiz algılayıcı düğümlerde kullanılan mikrodenetleyici birimleri, ses haberleşmesinde yetersiz kalabilmektedir. Bunun başlıca nedeni, çoğu mikrodenetleyicinin sesin kodlanması için gerekli işlem gücüne sahip olmamalarıdır. Ayrıca ses haberleşmesinde mikrodenetleyici biriminin geleneksel durumdan daha uzun süre aktif durumda çalışacağı öngörülmektedir. Bu da mikrodenetleyici biriminin işlem gücünün yüksek, güç tüketiminin ise düşük seviyede olmasını daha da önemli kılmaktadır.

Çizelge 6.1'de incelenen mikrodenetleyicilerin genel mimari özellikleri görülmektedir. Firefly düğümü üzerinde kullanılan 8-bit mimariye sahip Atmega1281 mikrodenetleyicisi, ses haberleşmesinde kısıtlı işlem gücü sebebiyle performans kaybına neden olmaktadır (Mangharam, 2006). 16-bir mimariye sahip mikrodenetleyiciler arasında MSP430F5438 yüksek çalışma hızı, geniş hafıza alanı, yeterli tipte ve sayıda bağlantı kanalı ile dikkat çekmektedir. Ayrıca bu mikrodenetleyici Texas Instruments firmasının geliştirdiği yeni nesil MSP430X geliştirilmiş işlemci mimarisine sahiptir.

Çizelge 6.1 İncelenen mikrodenetleyicilerin genel özellikleri [12,13,14,15]

	Mimari	Komut Sayısı	Flash (KB)	ADC (bit)	Çalışma Frekansı	Bağlantı Kanalları
AT91SAM7S256	32-bit RISC	-	256	10	55 MHz	I²C, SPI, SSC, UART/USART, USB
ATmega1281	8-bit RISC	135	128	10	16MHz	I²C, SPI, UART/USART
MSP430F149	16-bit RISC	51	60	12	8MHz	SPI, UART/USART
MSP430F5438	16-bit RISC	51	256	12	18MHz	I²C, IrDA, SPI, UART/USART

Çizelge 6.2'de incelenen mikrodenetleyicilerin enerji tüketimleri görülmektedir. Görüldüğü gibi işlem gücü / enerji tüketimi oranının en ideal olduğu mikrodenetleyici MSP430F5438'dir. ADC hassasiyeti Atmel mikrodenetleyicilerine göre daha fazla olan MSP430F5438 mikrodenetleyicisinin bağlantı kanallarının çeşitliliği de önemli bir avantajdır.

Çizelge 6.2 İncelenen mikrodenetleyicilerin enerji tüketimleri [12,13,14,15]

	Max Frek. (3V)	Uykuda	Atıl (1 MHz)	Aktif (1 MHz)	Aktif (8 MHz)	Aktif (16 MHz)	Aktif (55 MHz)
AT91SAM7S256	55 MHz	20 µA	-	-	8.4 mA	13.5 mA	50 mA
ATmega1281	8 MHz	25 µA	0.4 mA	2 mA	8 mA	-	-
MSP430F149	8 MHz	0.5 µA	55 µA	420 µA	1.9 mA	-	-
MSP430F5438	18 MHz	1.69 µA	86 µA	0.57 mA	3.62 mA	7.2 mA	-

Çizelge 6.3'te ise incelenen mikrodenetleyicilerin uyanma süreleri görülmektedir. Yapılan incelemeler sonucu, uyanma süresi, flash kapasitesi, çalışma frekansı ve harcadığı güç miktarı göz önüne alınarak gerçeklenecek düğüm tasarımında Texas Instruments firmasının geliştirdiği MSP430F5438'nin kullanılmasına karar verilmiştir.

Çizelge 6.3 İncelenen mikrodenetleyicilerin uyanma süreleri [12,13,14,15]

	Uyanma zamanı
AtMega1281	2 ms + 4 clk
AT91SAM7S256	45 µs
MSP430F149	< 6 µs
MSP430F5438	< 5 µs

MSP430F5438 mikrodenetleyicisi, Texas Instruments firmasının yeni nesil MSP430X mimarisine sahiptir. Bu yeni mimarinin MSP430 mimarisine göre en önemli noktası 64kB üzerinde adres alanına sahip oluşudur. Bellek miktarının da MSP430X ile 64kB sınırından kurtulup 128kB+ değerlere ulaştığı görülmektedir. Ayrıca yeni mimarinin MSP430 mimarisi ile kod uyumluluğu da mevcuttur. Böylece yeni mimaride önceden kodlanmış programlar yeniden derlenmeye gerek duymadan çalışabilmektedir. MSP430X mimarisi kesim (interrupt) cevaplama zamanı ve komut devir (instruction cycle) sayıları konusunda da MSP430'a üstünlük sağlamaktadır. MSP430X mimarisinde çevrebirimlere ve belleğe ulaşım bir devir daha azdır. Bu kod yoğunluğunu ve çalışma süresini düşüren önemli bir etkendir.

6.2 Telsiz İletişim Birimi

Geliştirilen düğümün yapısında kullanılacak telsiz iletişim biriminin belirlenmesi için 3. bölümde incelenmiş olan düğümler üzerinde kullanılan telsiz iletişim birimler araştırılmıştır. Bu telsiz iletişim birimleri, Chipcon firması tarafından üretilen CC1100 ve CC2420'dir. Bu telsiz iletişim birimleri ile ilgili detaylı bilgi Çizelge 6.4'te karşılaştırılmalı olarak verilmiştir.

Geliştirilen düğümün yapısında kullanılan CC2420 telsiz iletişim birimi çıkış gücünün düşürülerek veri aktarımı sırasında güç tüketiminin azaltılmasına imkan vermektedir. Çıkış gücü 0 dBm ile -25 dBm arasındaki değiştirilebilmekte böylece güç tüketimi 8.5 mA'e kadar düşürülmektedir.

Çizelge 6.4 İncelenen telsiz iletişim birimlerinin genel özellikleri [16,17]

	Çalışma Frekansı (MHz)	Veri aktarım hızı (kbps)	Çıkış Gücü (dBm)	Alıcı Hassasiyeti (dBm)	Güç Tüketimi		
					Pasif (A)	İletim (mA)	Alım (mA)
CC1100	315/433/868/915	1.2 - 500	10	-111	95 µA	31.1 (10dBm / 868 MHz)	14.4 (1.2kbps / 868 MHz)
CC2420	2483.5	250	0	-95	1	17.4	18.8

Proje kapsamında amaç ses haberleşmesi sağlamak olduğundan telsiz iletişim biriminin veri aktarım hızı yüksek olmalıdır. Bu gibi uygulamalarda sıklıkla kullanılan CC2420 telsiz iletişim biriminin düğüm tasarımında kullanılmasına karar verilmiştir.

6.3 İkincil Bellek Birimi

Tez kapsamında incelenen ses haberleşmesinde kullanılmak üzere tasarlanmış telsiz çoklu ortam algılayıcı düğümlerinde ikincil bellek bulunmamaktadır. Firefly projesinde, düğüme ikincil bellek eklenebilmesi için, bir SDIO yuvası bırakılmıştır. QVS projesinde kullanılan SenEar düğümünde ise, kullanılan mikrodenetleyicinin sağladığı 256 KB ikincil bellekten yararlanılmıştır. Çizelge 6.5'te bu proje kapsamında tasarıma eklenmesi için incelenen bellek tipleri görülmektedir. Tasarlanan düğümün yapısında bellek birimi olarak diğer alternatiflere göre daha yüksek saklama kapasitesi ve veri aktarım hızına sahip olan M25P80'nin kullanılmasına karar verilmiştir.

Çizelge 6.5 M25P80 ikincil bellek biriminin genel özellikleri [18]

	Bağlantı Tipi	Bellek Miktarı (Kbit)	Yazma Süresi (ms)	Veri Aktarım Hızı (KHz)	Güç Tüketimi		
					Pasif	Aktif Okuma	Aktif Yazma
M25P80	SPI	8192	1.5	25000	1 µA	4000 µA	15 mA
M24M01S	I2C	1024	10	400	2	2000	-
AT45DB041B	SPI	4096	7	5000	20 µA	4000 µA	15 mA

6.4 Ses Algılayıcısı

Tez kapsamında geliştirilen telsiz çoklu ortam algılayıcı düğüm tasarımında ses algılayıcısı olarak, CUI firmasının geliştirdiği, düşük güç tüketimi sağlayan, çok yönlü CMC-5044PF-A mikrofonu kullanılmıştır. Bu mikrofonun temel özellikleri Çizelge 6.6'da görülmektedir. Ayrıca tasarlanan düğümde audio jack yardımıyla kullanıcı istediği mikrofonu kullanabilecektir.

Çizelge 6.6 CMC-5044RF-A mikrofonunun temel özellikleri [19]

Özellik	Değer
Yön Seçiciliği	Çok yönlü (omnidirectional)
Duyarlılık	-44 ±3 db (f=)
Çalışma Voltajı	2V (standart), 10V (maksimum)
Çıkış Empedansı	2.2k Ω
Çalışma Frekansı	100 – 20.000 Hz
Güç Tüketimi	0.5 mA (maksimum)
Ölçü	ø6 x 5 mm

Geliştirilen devre tasarımında mikrofon ve MSP430F5438 ADC kanalı arasına bir op-amp fayda devresi konmuştur. Bu devrede Texas Instruments firmasının geliştirdiği düşük enerji tüketimi sağlayan TLV2760 hassasiyet yükseltici biriminden [20] yararlanılmıştır. TLV2760 için gerekli olan enerji MSP430F5438'nin bir bacağından sağlanmıştır. Böylece, TLV2760'ın gerekli olmadığı durumlarda enerjisinin yazılımsal olarak kesilebilmesi sağlanmıştır. Kullanılan bu op-amp tipik konuşma frekansı olan 4 kHz limitine ayarlanarak istenmeyen gürültülerin bir kısmı engellenmiştir. EK 3'te bu ses giriş zincirine ve EK 4'te ses çıkış zincirine ait devre şematiği görülmektedir.

6.5 Diğer Algılayıcılar

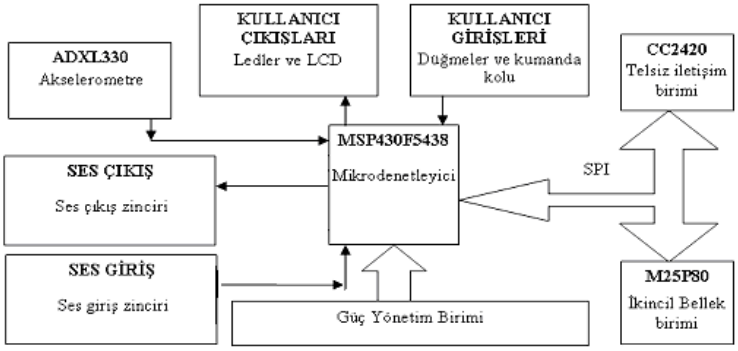
Geliştirilen tasarımda bir de 3-eksenli akselerometre kullanılmıştır. Bunun için Analog Devices firmasına ait ADXL330 [21] akselerometresi seçilmiştir. Her bir X, Y, Z ekseni için üç analog sinyal MSP430F5438 mikrodenetleyicisinin ADC modülünün üç giriş kanalına bağlanmıştır.

6.6 Kullanıcı Arayüzleri

Düğümde kullanıcı girişleri için biri reset düğmesi (buton) olmak üzere üç düğme ve bir adet beş yönlü kumanda kolu (joystick) kullanılmıştır. Ayrıca tasarımda kullanıcı arayüz çıkışları için iki led, bir de LCD ekran kullanılmıştır. EK 5'da kullanıcı arayüzlerinin işlemci birimi bacak bağlantıları görülmektedir.

6.7 Geliştirilen Düğümün Yapısı

Geliştirilmiş olan telsiz çoklu ortam algılayıcı düğüm, mikro denetleyici birimi, telsiz iletişim birimi, ikincil bellek birimi, güç yönetim birimi, ses giriş zinciri, ses çıkış zinciri ve akselerometre biriminden oluşmaktadır. Şekil 6-1'de düğümü oluşturan birimler ve bu birimler arasındaki bağlantı yapısı gösterilmektedir.



Şekil 6-1 Geliştirilen düğümün blok şeması

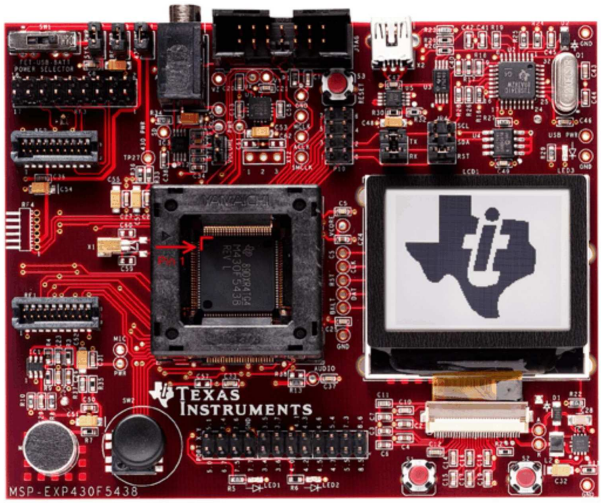
Ek 1'de; tasarlanan düğüme ait mikrodenetleyici, güç yönetimi ve bazı kullanıcı arayüzlerine ait baskılı devre çizimleri, Ek 2'de; telsiz haberleşme birimine ait baskılı devre çizimleri, Ek 3'te; düğüm tasarımında kullanılan ses girişi şematiği, Ek 4'te; düğüm tasarımında kullanılan ses girişi şematiği bulunmaktadır.

7. YAZILIM GELİŞTİRME

Bu aşamada öncelikle MSP430F5438 mikrodenetleyicisinin test edilebileceği bir test ortamı edinilmiş ve yazılım geliştirme işlemleri bu ortamda yapılmıştır. Contiki OS uyarlama işlemleri bu test ortamı üzerinde gerçekleştirilmiştir. Uyarlama süreci için öncelikle uygun derleyici hazırlanmış, daha sonra da Contiki OS üzerinde merkezi işlemci birimi tanımlamaları ve düğüm çevrebirimleri sürücüleri gerçeklenmiştir.

7.1 Test Ortamı

Proje kapsamında geliştirilen yazılımların ve tasarlanan düğümde kullanılması kararlaştırılan MSP430F5438 mikrodenetleyisinin test edilmesi için, Şekil 7.2'de görülen, Texas firmasının geliştirdiği MSP430F5438 Experimenter Kit (MSP-EXP430F5438) [22] devresi kullanılmıştır.



Şekil 7-1 MSP430F5438 Experimenter Board [22]

Bu devre içerdiği ses giriş çıkış zincirleriyle proje kapsamında tasarlanan düğüme örnek oluşturmuş ve gerçeklenen yazılımların test edilmesi için kullanışlı bir ortam sağlamıştır. Ayrıca içerdiği dot-matrix LCD, USB kapısı (port) ve RF eklentileri ile geniş bir test alanı sunmuştur.

7.2 Contiki OS Uyarlama Süreci

Contiki OS yüksek taşınabilirliğe sahip, çoklu görev yürütümünü destekleyen açık kaynak kodlu bir işletim sistemidir. İşletim sisteminin de geliştiricisi olan İsveç Bilgisayar Bilimleri Enstitüsü'nden Adam Dunkels'ın geliştirdiği yığıtsız (stackless) "protothread" tasarımını

kullanmaktadır.

Contiki OS MSP430 tabanlı mikrodenetleyiciler için MSPGCC derleyicisini desteklemektedir. Proje kapsamında ilk olarak düğüm tasarımında kullanılan MSP430X ailesinden MSP430F5438 mikrodenetleyicisinin MSPGCC tarafından desteği araştırılmıştır. Ancak son kararlı sürüm olan mspgcc20081230 sürümünün kullanılan mikrodenetleyiciyi desteklemediği görülmüştür. Bu sürüm Çizelge 7.2'deki mikrodenetleyicileri desteklemektedir.

Çizelge 7.1 MSPGCC kararlı sürümünün desteklediği mikrodenetleyici listesi [4]

msp430x110	msp430x112			
msp430x1101				
msp430x1111	msp430x1121			
msp430x122	msp430x123			
msp430x1222	msp430x1232			
msp430x133	msp430x135			
msp430x1331	msp430x1351			
msp430x147	msp430x148	msp430x149		
msp430x1471	msp430x1481	msp430x1491		
msp430x155	msp430x156	msp430x157		
msp430x167	msp430x168	msp430x169	msp430x1610	msp430x1611
msp430x311	msp430x312	msp430x313	msp430x314	msp430x315
msp430x323	msp430x325	msp430x336	msp430x337	
msp430x412	msp430x413			
msp430xE423	msp430xE425	msp430xE427		
msp430xW423	msp430xW425	msp430xW427		
msp430x435	msp430x436	msp430x437		
msp430x447	msp430x448	msp430x449		

GCC tabanlı, MSP430 mikrodenetleyicileri için geliştirilmeye başlanmış yeni bir derleyici de MSPGCC4'tür [8]. GCC 4.4.3 sürümü tabanlı bu derleyicinin henüz kararlı bir sürümü bulunmamakta ve geliştirilmesine devam edilmektedir. Geliştirilen bu sürümün MSP430X ailesini de desteklemesi öngörülmektedir.

MSPGCC derleyicisinin MSP430X mikrodenetleyicileri için derleme yapmasını sağlayacak kütüphane ve program yaması Alman Innoventis firması tarafından geliştirilmiştir [9]. Kullanıcıları tarafından "NEW_VERSION" olarak tanımlanan bu yamanın MSP430x54xx kütüphanesinde bazı hatalı veya eksik tanımlamalar mevcuttur. Proje kapsamında msp430x54xx.h dosyası, kullanıcı klavuzu [10] göz önünde bulundurularak Ek 9'daki gibi modifiye edilmştir. Kütüphane dosyasında yapılan önemli değişiklikler; status flag register,

bit ve low power mode tanımlamalarının eklenmesi, RTC, ADC ve DMA adreslerinin tanımlanıp ilgili kütüphane dosyalarının msp430x54xx.h dosyasının içeriğine dahil edilmesidir.

Innoventis firmasının geliştiriği yamanın kullanımı ve MSP430F5438 için gerekli tanımlamalar sayesinde MSP-EXP430F5438 üzerinde çalışan obje kodları üretilmiştir.

7.2.1 Contiki OS Uyarlanması

Telsiz algılayıcı ağlarda kullanılan düğümler çok çeşitli birimlerden ve mikrodenetleyicilerden oluşmaktadır. Farklı donanım yapıları, geliştirilen işletim sistemlerinin taşınabilirliğini zorunlu kılmaktadır. Platformların varolan çok az benzer karakteristiğinden biri, segmentasyon ve hafıza koruması içermeyen mikrodenetleyici yapılarıdır (Dunkels, 2004). Çoğunlukla program kodu yeniden programlanabilir ROM üzerine, veri de RAM üzerine yerleştirilir. Contiki taşınabilirlik problemini, mikrodenetleyici erişimini soyutlayıp, yüklenebilir programlara ve servislere destek vererek çözmeyi amaçlamıştır. Contiki OS'i yeni bir platforma uyarlamak teoride birkaç modifikasyon gerektirse de düşük seviyeli donanım hatalarını ayıklamak bir problem oluşturmaktadır (Stan, 2007).

Dunkels, tasarladığı işletim sisteminin çekirdeğinde bir multi-threading kütüphanesi oluşturarak olay tabanlı mimarilerin eksikliklerini kapatmayı amaçlamıştır. Bu opsiyonel kütüphane ile kullanıcılar uzun süreli hesaplamaları sırasında multithreading çalışmanın getireceği faydalardan da yararlanabilmektedir. Dunkels geliştirdiği bu tasarıma "protothread" adını vermiştir. Protothreadler yığıtsız (stackless) yapıdadır ve threadlerin kendi durumlarını kaydetmeleri için gerekli, 2 byte RAM alanı kullanan tek bir C fonksiyonundan oluşur. Kütüphanenin gerçekte yaptığı, olay tabanlı çekirdek üzerinde işlem üstünlüğü ve engelleme (blocking) sağlayan bir yapı ortaya koymasıdır. Protothreadler sayesinde kullanıcı durum makinaları kullanmadan olay tabanlı çekirdek yapısını kullanmayı sağlar.

Contiki OS'de süreçler arası iletişim her zaman çekirdek (kernel) üzerinden yapılır. Donanım ile uygulamalar arasında bir tabaka yoktur. Aygıt sürücüleri ve uygulamalar donanıma direk olarak erişebilmektedir. Süreçlerin durumları, sürecin kendi hafıza alanı içinde tutulur ve çekirdek yalnızca süreç durumunu gösterir bir işaretçi (pointer) tutar. Tüm süreçler aynı adresleme alanını paylaşır ve farklı bir alanda çalışmazlar. Temel olarak Contiki OS; çekirdek (core) ve uygulamalardan oluşur. Çekirdek; kernel, program yükleyici, gerçek zamanda sıkça kullanılacak işletim programları, iletişim sürücülerini de içeren bir iletişim yığıtından oluşur.

Çekirdek bir defa ikili imaja (binary image) derlenir ve daha sonra modifikasyon gerektirmez. Çünkü Contiki modelinde kullanıcı programları çekirdeği bilirken, çekirdek yapı kullanıcı programlarından habersizdir.

Contiki OS'in getirdiği protothread yapıları programlama açısından kullanıcıya, durum makinalarını kullanmadan daha basit programlama yapma olanağı sağlamaktadır. Protothreadler yalnızca koşullu engelleme bekleme (blocking wait) durumları yaratmak için kullanılır. Tümü tek yığıtta çalışır ve ortam değişimi (context switching), yığıt geri sarma ile yapılır. Contiki süreçleri yalnızca protothreadlerdir. Şekil 7-2'de protothreadlerin basit uygulamaları görülmektedir.

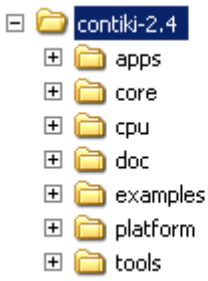
```
struct pt { unsigned short lc; };
# define PT_INIT ( pt ) pt->lc = 0
# define PT_BEGIN( pt ) switch ( pt->lc ) { case 0 :
# define PT_EXIT ( pt ) pt->lc = 0 ; return 2
# define PT_WAIT_UNTIL ( pt, c ) pt->lc = __LINE__ ; case __LINE__ : \
                                 if(!( c )) return 0
# define PT_END( pt ) } pt->lc = 0 ; return 1
```

Şekil 7-2 Protothread makroları

Protothreadlerde tüm süreçler PT_INIT (pt) makrosu ile tanımlanmış, PT_BEGIN (pt) ifadesi ile başlatılmış ve PT_END (pt) ifadesi ile sona erdirilmiş olması gerekmektedir. Bu makrolara erişim Contiki OS'da bu makrolara erişim; PROCESS (processPointer, "process name"), PROCESS_BEGIN () ve PROCESS_END () tanımlamaları ile kısıtlanmıştır. Ek 6'de Contiki OS'da kullanılan basit bir süreç örneği bulunmaktadır.

Tez kapsamında tasarlanan ses haberleşmesi sağlayan telsiz algılayıcı düğümün gelecek planlamasında mağara-içi haberleşmesinde kullanılması amaçlanmaktadır. Bu doğrultuda yeni platformun ismi ICWCS (In-Cave Wireless Communication System) olarak belirlenmiştir. Amaçlanan ICWCS platformu için Contiki OS uyarlaması yapmaktır.

Şekil 7-3'te Contiki İşletim Sisteminin temel dizin yapısı görülmektedir. Yeni bir platform uyarlamak için kullanılacak temel dizinler */platform* ve */cpu* dur.



Şekil 7-3 Contiki İşletim Sistemi dizin yapısı

Oluşturulacak yeni platform için */platform* dizini altında yeni bir klasör oluşturularak platform özel tanımlar bu klasör içinde yapılmalıdır. Proje kapsamında */platform/icwcs* dizini açılmıştır. Öncelikli olarak Contiki'nin temel yapısını oluşturan tanımlamaları içeren */platform/native* dizini yeni oluşturulan dizine aktarılmış ve gerekli düzenlemeler yapılmıştır. */platform/native* dizini sıklıkla kullanılan aygıt sürücülerini ve Contiki özel tanımlamaları içerir.

Uyarlama kapsamında gerçeklenmesi en kritik dosya */platform/platform_ismi/contiki-conf.h* dosyasıdır ki bu dosya tüm platform özel tanımlamaları, derleyici konfigürasyonlarını, saat konfigürayonlarını, ağ yönetimi konfigürasyonlarını, düşük seviye hafıza adreslerini, pin konfigürasyonlarını ve gerekli diğer tanımlamaları içermektedir (Stan, 2007). Derleme zamanında platforma özel işlemlerin, derlenecek dosyaların, ve mikrodenetleyici tipinin tanımlandığı yer ise */platform/platform_ismi/Makefile.platform_ismi* dosyasıdır. Donanım ilklendirmelerinin yapıldığı, diğer süreçlerin başlatıldığı ve ana süreç kodunun bulunduğu dosya da */platform/platform_ismi/contiki-platform_ismi-main.c* dosyasıdır.

Gerçeklenen platforma özel algılayıcıların tanımı ise */platform/platform_ismi/dev* altında yapılmaktadır. Tez kapsamında */platform/icwcs/dev* altında kullanılan platform için audio, lcd ve button sürücüleri gerçeklenmiştir.

Mikrodenetleyiciye özel kodlar ise */cpu/cpu_ismi* dizininde bulunmaktadır. Bu dizin sıklıkla kullanılan mikrodenetleyicilerin uyarlamalarını içermektedir (Stan,2007). Contiki 2.4 sürümünde */cpu/msp430* dizini altında bulunan sürücüler Contiki OS uyarlaması yapılan TmoteSky, ESB gibi platformlara ait özel tanımlamaları içermekte bir ortak bir kullanım alanı oluşturmamaktadır. Proje kapsamında geliştirilen ICWCS platformu için farklı tanımlamalar yapmak gerektiğinde */cpu/msp430x* dizini oluşturulmuş ve gerekli düzenlemeler bu dizin altında yapılmıştır.

7.2.1.1 Mikrodenetleyici

Contiki OS yapısında mikrodenetleyici çalışma frekansının ayarlanması için /cpu/msp430x/msp430x.c dosyası içinde harici kristal, MCLK (Master Clock), DCO (Digitally Controlled Oscillator) ve VCORE (Core Voltage) tanımlama ve ilklendirilmeleri yapılmıştır. Şekil 7-4'te ilklendirmelerde kullanılan değerler görülmektedir. Bu ilklendirmeye göre işlemci çalışma hızı 16MHz olarak ayarlanmış; ACLK, MCLK ve SMCLK kaynakları verilmiştir. Ayrıca çalışma hızı uygulamaya bağlı olarak değiştirilebilmektedir.

```
/* Set lowest possible DCOx, MODx */
  UCSCTL0 = 0x00;
/* Select suitable range */
  UCSCTL1 = DCORSEL_16MHZ;
/* Set DCO Multiplier FLL Loop Divider /2 */
  UCSCTL2 = DCO_MULT_16MHZ + FLLD_1;
/* Set ACLK Source Select XT1CLK,
 * SMCLK Source Select DCOCLKDIV,
 * MCLK Source Select DCOCLKDIV */
  UCSCTL4 = SELA__XT1CLK |
            SELS__DCOCLKDIV |
            SELM__DCOCLKDIV;
```

Şekil 7-4 CPU tanımlamaları

MSP430F5438 mikrodenetleyicisi, uzun süreli eylemsizliklerin fark edilip işlemcinin yeniden başlatılmasını sağlayan Watchdog zamanlayıcı arayüzüne sahiptir. Watchdog konfigürasyonu da /cpu/msp430x/watchdog.c dosyası içinde yapılmıştır. İlklendirme Şekil 7-5'teki gibi yapılarak 250 ms'lik bir zamanlayıcı kurulmuştur.

```
WDTCTL = WDTPW + WDTSSEL_1 + WDTTMSEL
         + WDTCNTCL + WDTIS_5;
```

Şekil 7-5 Watchdog ilklendirmesi

7.2.1.2 Saat ve zamanlayıcılar

Contiki'de kullanıcı isteklerine cevap vermek üzere geliştirilmiş dört çeşit zamanlayıcı mevcuttur (Stan, 2007):

- timer: pasif, yalnızca kendi bitiş zamanını tutar.
- etimer: aktif, süresi dolduğunda bir olay oluşturur.
- ctimer: aktif, süresi dolduğunda bir fonksiyon çağırır.
- rtimer: gerçek zaman tutar, süresi dolduğunda bir fonksiyon çağırır.

Her mikrodenetleyici için saat tanımlamaları /cpu dizini altında yapılmalıdır. Proje kapsamında /cpu/msp430x/clock.c dosyası oluşturulmuş ve tüm işlemciler için tanımlanması gereken *clock_init(), clock_delay()* ve *clock_time()* fonksiyonları tanımlanmıştır. Yukarıda belirtilen zamanlayıcılardan rtimer hariç diğer zamanlayıcılar bu fonksiyonlardan beslenmektedir. Saat tanımlamaları Şekil 7-6'daki gibi ilklendirilmiştir. Bu konfigürasyona göre Timer0_A0 zamanlayıcısı için 32.768Hz'lik ACLK seçilmiş, interrupt (kesim) etkinleştirilmiş ve 128ms'lik INTERVAL değeri zamanlayıcının geri sayım değeri olarak atanmıştır. Zamanlayıcının koşum esnasında sürekli çalışması için Timer0_A0 sürekli moda çalışmaya ayarlanmıştır.

```
/* Select ACLK 32768Hz clock, divide by 2 */
TA0CTL = TASSEL_1 | TACLR | ID_1;
/* CCR1 interrupt enabled
 * interrupt occurs when timer equals CCR1. */
TA0CCTL0 = CCIE;
/* Interrupt after 128 ms. */
TA0CCR0 = INTERVAL;
/* Start Timer0_A0 in continuous mode. */
TA0CTL |= MC_1;
```

Şekil 7-6 Clock ilklendirmesi

Rtimer için diğer platformlarda olduğu gibi /cpu/msp430x/rtimer_arch.c dosyası oluşturularak ilgili fonksiyonlar tanımlanmıştır. Rtimer kullanımı için tanımlanması gereken fonksiyon *rtimer_arch_schedular (rtimer_clock_t t)* fonksiyonudur. Bu fonksiyon belirlenen *t* zamanında sıradaki fonksiyonu çalıştırır. Proje kapsamında rtimer işlevinin gerçekleştirilmesi için MSP430F5438 Timer1_A0 zamanlayıcısı kullanılmıştır. Şekil 7-6'da bu zamanlayıcıya bağlı kesim metodu (interrupt method) görülmektedir.

```
interrupt(TIMER1_A0_VECTOR) timera0 (void)
{
  ENERGEST_ON(ENERGEST_TYPE_IRQ);

  rtimer_run_next();   // calls rtimer_arch_schedule(rtimer_clock_t t)
  if(process_nevents() > 0) {
    LPM4_EXIT;
  }

  ENERGEST_OFF(ENERGEST_TYPE_IRQ);
}
```

Şekil 7-7 rtimer interrupt

7.2.1.3 Seri arabirim (Serial Interface)

MSP-EXP430F5438, UART iletişimi ve hata ayıklama için kullanılabilecek bir USB bağlantısı içermektedir. UART iletişiminin konfigürasyonu için /cpu/msp430x/uart.c dosyasında UCA1CTL0 register alanı kullanılarak Şekil 7-8'deki tanımlamalar yapılmıştır. 8bit iletişim uzunluğu ayarlanmış ve SMCLK (Submain System Clock) kullanılmıştır. Bu konfigürasyon için MSP430F5438 User's Guide'da [22] belirtilen en düşük iletim hatasına yol açacak 57.600 baud rate iletim hızı seçilmiştir.

```
UCA1CTL1  = UCSWRST;     // Software reset
UCA1CTL0 = UCMODE_0;     // Set UART
UCA1CTL0 &= UC7BIT;      // Set 8bit communication
UCA1CTL1  = UCSSEL_2;    // Select SMCLK
/* Set Baud Rate Control Register
16Mhz & 57600 Tx error rate = 0.3
UCBRx = 277, UCBRSx = 7, UCBRFx = 0 */
UCA1BR0 = 16;
UCA1BR1 = 1;
UCA1MCTL = 0xE;          // oversampling disabled.
UCA1CTL1 &= UCSWRST;     // clear
UCA1IE  = UCRXIE;        // enable interrupts
```

Şekil 7-8 UART ilklendirmesi

7.2.1.4 LEDler

Contiki'de tabanında; kırmızı, yeşil ve sarı olarak tanımlanmış 3 farklı LED (Light-emitting diode) desteği verilmiştir. Proje kapsamında geliştirilen tasarımda ise bir kırmızı, bir de sarı olmak üzere iki adet LED bulunmaktadır. Bu sebeple yeşil LED tanımlamaları sarı LED'e atanmıştır.

LED desteği için /cpu/msp430x/leds-arch.c dosyasında aşağıdaki fonksiyonlar tanımlanmıştır.

- void leds_arch_init(void): LED'lerin ilklendirilmesi yapılır (hepsi kapalı).
- void leds_arch_set(unsigned char leds): Verilen bit maskesine göre LED durumlarını değiştirir.
- unsigned char leds_arch_get(void): LED durumlarını bir bit maskesi halinde döndürür.

Ek 7'de leds-arch.c dosyasının ve yukarıda belirtilen fonksiyonların içeriği bulunmaktadır.

7.2.1.5 LCD

Contiki'de diğer platformlarda olduğu gibi platforma özgü LCD sürücüsü /platform/icwcs/dev dizini altına koyulmuştur. Bu sürücü MSP-EXP430F5438 demo yazılımı ile birlikte gelen sürücülerden alınmıştır [22]. Demo yazılımında kullanılan sürücü IAR yazılımı [36] ile derlenmek üzere yazılmış fakat, MSPGCC ile de uyumludur. LCD aygıtının

ilklendirilmesinden, piksel piksel yazma, blok ya da karakter yazma, kaydırma çubuğu (scroll bar) çizme gibi pek çok fonksiyon içermektedir.

7.2.1.6 Audio

Düğüm için geliştirilen audio sürücü, /platform/icwcs/dev dizini altında bulunur ve kayıt, tekrar çalma fonksiyonlarını içerir. Kayıt fonksiyonu temel olarak dahili flash belleği açar ve Timer_B zamanlayıcısını kullanarak ADC örneklemesini başlatır. İşlem tamamlandığında flash belleği kapatılır, DMA (Direct Memory Access) devre dışı bırakılır ve zamanlayıcı durdurulur. Tekrar çalma fonksiyonu da benzer şekilde flash bellekte kaydedilmiş ses verisinin, DAC (Digital-to-Analog Converter) modülü ve DMA kontrolcüsü sayesinde ses çıkışına verilmesini sağlar. Audio sürücüsü ile ilgili temel fonksiyonlar Ek 8'da verilmiştir.

8. SONUÇ

İlk olarak askeri savunma uygulamalarında kullanılan telsiz algılayıcı ağlar, daha sonra doğal yaşam alanlarının izlenmesi, tarım ürünlerinin gelişimlerinin izlenmesi, endüstriyel kontrol ve izleme sistemleri, ev otomasyon sistemleri, güvenlik uygulamaları ve tedarik zinciri uygulamalarında kullanılmaya başlanmıştır. Günümüzde ise ses algılayıcıları kullanılarak ortam ses verisinin telsiz algılayıcı ağlarda yayınlanabildiği görülmektedir. Bu gelişme, teknolojideki ilerlemelerle birlikte telsiz algılayıcı düğüm yapılarında kullanılan işlemci birimlerinin, işlem güçlerinin artması ve güç tüketimlerinin azalması, ayrıca ses ve görüntü haberleşmesi sağlayan elektronik birimlerin (kamera, mikrofon gibi) ucuzlaması ve boyutlarının küçülmesi ile mümkün olmuştur.

Tez çalışması kapsamında, telsiz algılayıcı ağlarda ses haberleşmesi yapılabilmesi için bir telsiz çoklu ortam algılayıcı düğüm tasarımı yapılmış ve bir gömülü işletim sistemi tasarlanan düğüm yapısına uyarlanmıştır. Geliştirilen yazılım ve düğüm tasarımında kullanılan işlemci birimi hazır bir deneme platformu üzerinde denenmiştir.

Bugüne kadar geliştirilen diğer ses iletimi sağlayan telsiz algılayıcı düğümler tek yönlü ses iletimine izin vermekte ve karşılıklı haberleşme için kullanılamamaktadır. Bu projede tasarlanan düğüm üzerine yerleştirilmiş audio jack ve gerçeklenen yazılımda ses çalabilme olanağı veren sürücü sayesinde tek bir düğüm hem ses algılayabilme hem de ses dinletebilme özelliğine sahiptir.

Düğüm yapısında kullanılan MSP430F5438 işlemci birimi Texas Instruments firması tarafından geliştirilmiş yeni nesil MSP430X mikrodenetleyicilerinden biridir. Bu mikrodenetleyici diğer 16bit alternatiflere göre düşük güç tüketimi ve yüksek işlem gücü sağlamaktadır.

Telsiz algılayıcı düğümlerde kulanılmak üzere geliştirilmiş işletim sistemlerinden en yaygın olanlarından biri Contiki İşletim Sistemidir. Proje kapsamında geliştirilen düğüme Contiki İşletim Sistemi ve MSPGCC derleyicisi uyarlanmıştır.

Geliştirilen tez çalışmasına ek olarak, tasarlanan düğüm üzerindeki CC2420 RF sağlayıcısını destekleyecek sürücü geliştirilip, işletim sistemi üzerinde ses haberleşmesine uygun yapıda bir Link ve MAC katmanı oluşturularak verimli bir ağ yapısı oluşturulabilir.

KAYNAKLAR

D. McIntire, “Energy Benefits of 32-bit Microprocessor Wireless Sensing Systems”, Sensoria Corporation White Paper, Ekim 2009.

Z. Cihan TAYŞİ, “Genel Amaçlı Bir telsiz Algılayıcı Düğüm Tasarımı ve Gerçeklenmesi” YTÜ Fen Bilimleri Ens., Master Tezi, Temmuz 2006.

Lynch, C., O’Reilly, F., (2005) “Processor Choice For Wireless Sensor Networks”, Workshop on Real-World Wireless Sensor Networks, Haziran 2005.

Bhatti, S., Carlson, J., Dai, H., Deng, J., Rose, J., Sheth, A., Shucker, B., Gruenwald, C. , Torgerson, A., Han, R., (2005) "MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms", ACM/Kluwer Mobile Networks & Applications (MONET), Special Issue on Wireless Sensor Networks, vol. 10, no. 4, Ağustos 2005.

Stan, A., “Porting the Core of the Contiki operating system to the TelosB and MicaZ platforms”, Guided Research Final Report, Mayıs 2007.

Dunkels, A., Gronvall, B., Voigt, T., “Contiki – a lightweight and flexible operating system for tiny networked sensors”, Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, Kasım 2004.

Dunkels, A., “Contiki/ESB Reference Manual”, Swedish Institute of Computer Science, Temmuz 2005.

Mangharam, R., Rowe, A., Rajkumar, R., Suzuki, R., “Voice Over Sensor Networks”, Real-time Systems Symposium, Aralık 2006.

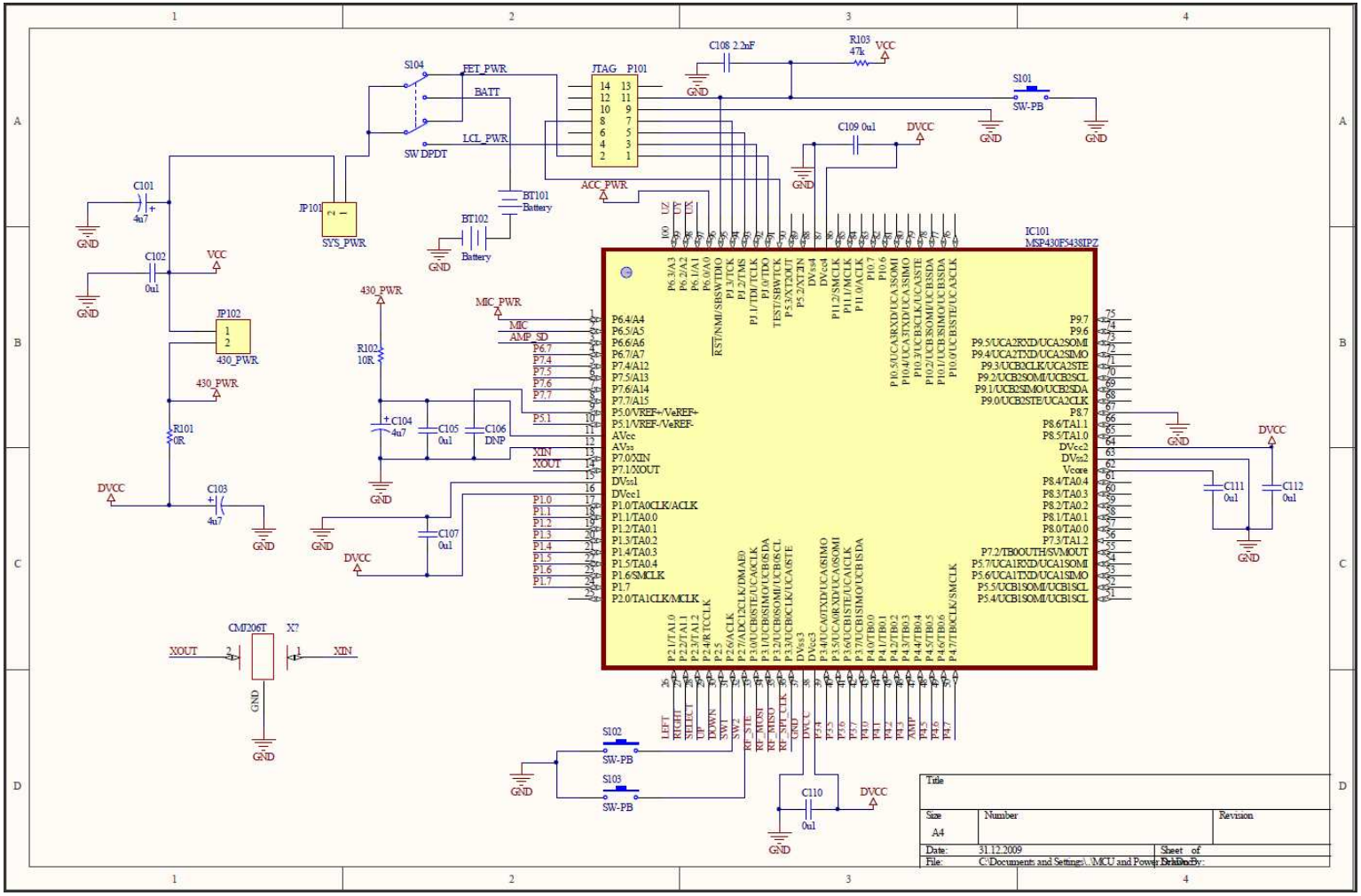
Li, L., Xing, G., Sun, L., Liu, Y., “QVS: Quality-aware Voice Streaming for Wireless Sensor Networks” , Proceedings of the 29th Annual IEEE International Conference on Distributed Computing Systems, Haziran 2009.

INTERNET KAYNAKLARI

[1] http://www.eng.buffalo.edu/~tmelodia/papers/multimedia_survey.pdf
[2] http://www.cse.msu.edu/~glxing/docs/voice_icdcs09.pdf
[3] http://www.ece.cmu.edu/firefly/
[4] http://mspgcc.sourceforge.net/
[5] http://www.princeton.edu/~wolf/EECS579/imotes/tos_tutorial.pdf
[6] http://www.slideshare.net/sudharsan2020/introduction-to-tiny-os-and-contiki-os
[7] http://focus.ti.com/lit/ml/slap109/slap109.pdf
[8] http://mspgcc4.sourceforge.net/
[9] http://www.innoventis.de/downloads/mspgcc/
[10] http://www.ti.com/litv/pdf/slau208f
[11] http://www.eecs.iu-bremen.de/archive/bsc-2007/stan.pdf
[12] http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf
[13] ATmega1281, "http://www.atmel.com/dyn/resources/prod_documents/doc6175.pdf"
[14] MSP430F149, "http://focus.ti.com/lit/ds/symlink/msp430f133.pdf"
[15] MSP430F5438, "http://focus.ti.com/lit/ds/symlink/msp430f5418.pdf"
[16] CC2420, "http://focus.ti.com/lit/ds/symlink/cc2420.pdf"
[17] CC110, "http://focus.ti.com/lit/ds/symlink/cc1100.pdf"
[18] M25P80, "www.st.com/stonline/books/pdf/docs/8495.pdf"
[19] CMC-5044PF-A, "http://www.cui.com/getPDF.aspx?filename=CMC-5044PF-A.pdf"
[20] TLV2760, "http://focus.ti.com/docs/prod/folders/print/tlv2760.html"
[21] ADXL330, "http://www.analog.com/static/imported-files/data_sheets/ADXL330.pdf"
[22] http://focus.ti.com/docs/toolsw/folders/print/msp-exp430f5438.html
[23] FireFly, "http://www.nanork.org/attachment/wiki/Documentation/FireFly_datasheet.pdf"
[24] http://www.sics.se/~adam/slides/contiki-emnets.ppt#276,21,Implementing preemptive threads 1
[25] http://www.nanork.org/
[26] www.tinyos.net
[27] www.sics.se/contiki
[28] http://mantis.cs.colorado.edu
[29] www.cs.berkeley.edu/~kamin/calamari/
[30] firebug.sourceforge.net/
[31] www-bsac.eecs.berkeley.edu/projects/cotsbots/
[32] telegraph.cs.berkeley.edu/tinydb/
[33] ptolemy.eecs.berkeley.edu/papers/03/TinyGALS
[34] www.greatduckisland.net
[35] Hill, J., "A Software Architecture Supporting Networked Sensors", "www.cs.berkeley.edu/~kwright/nest_papers/TinyOS_Masters.pdf"
[36] www.iar.com

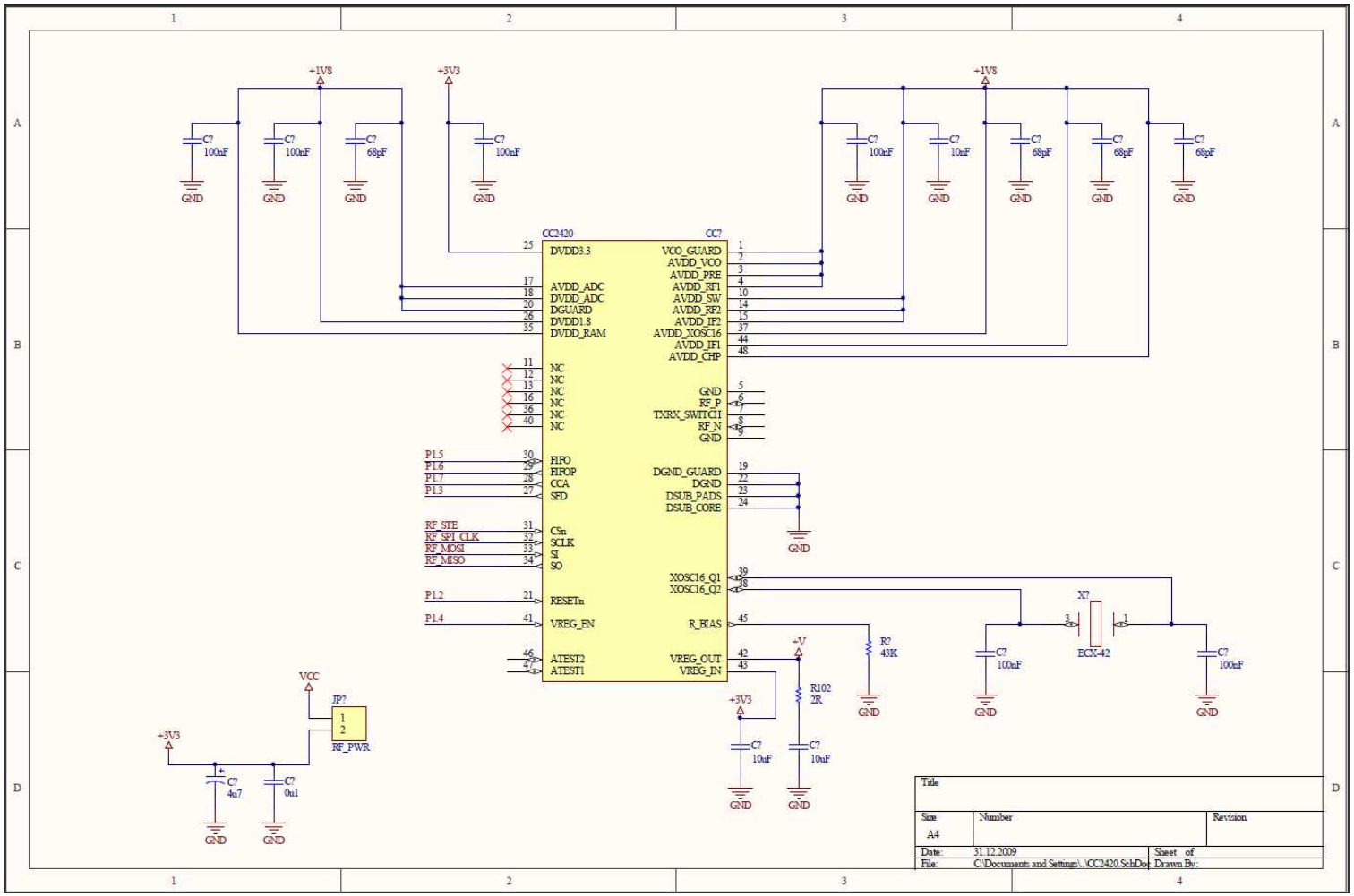
EKLER

Ek 1 Mikrodenetleyici, güç yönetimi ve bazı kullanıcı arayüzlerine ait devre çizimleri



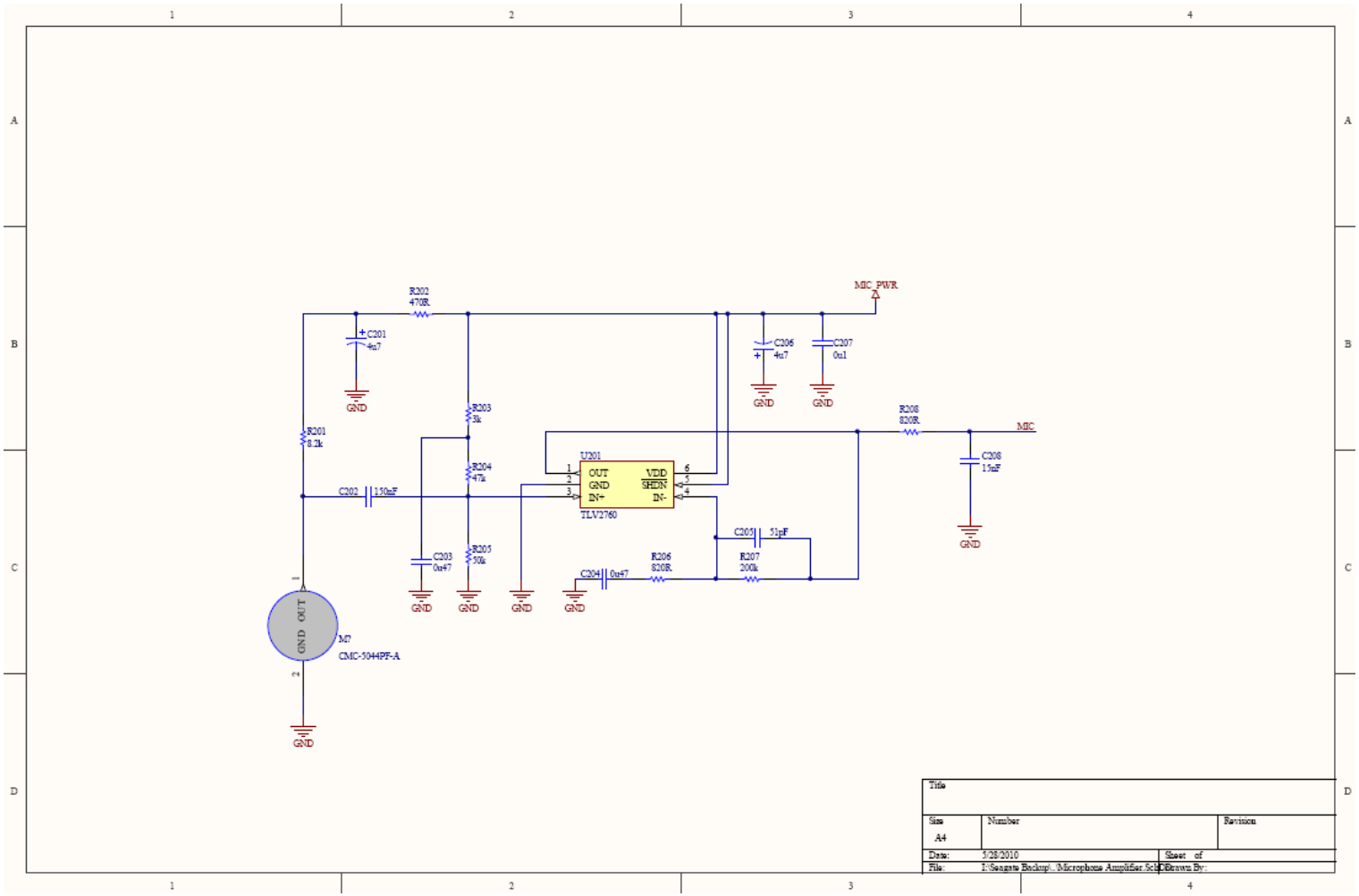
Şekil 8-1 Mikrodenetleyici, güç yönetimi ve bazı kullanıcı arayüzlerine ait şematik

Ek 2 Telsiz haberleşme birimine ait devre çizimleri



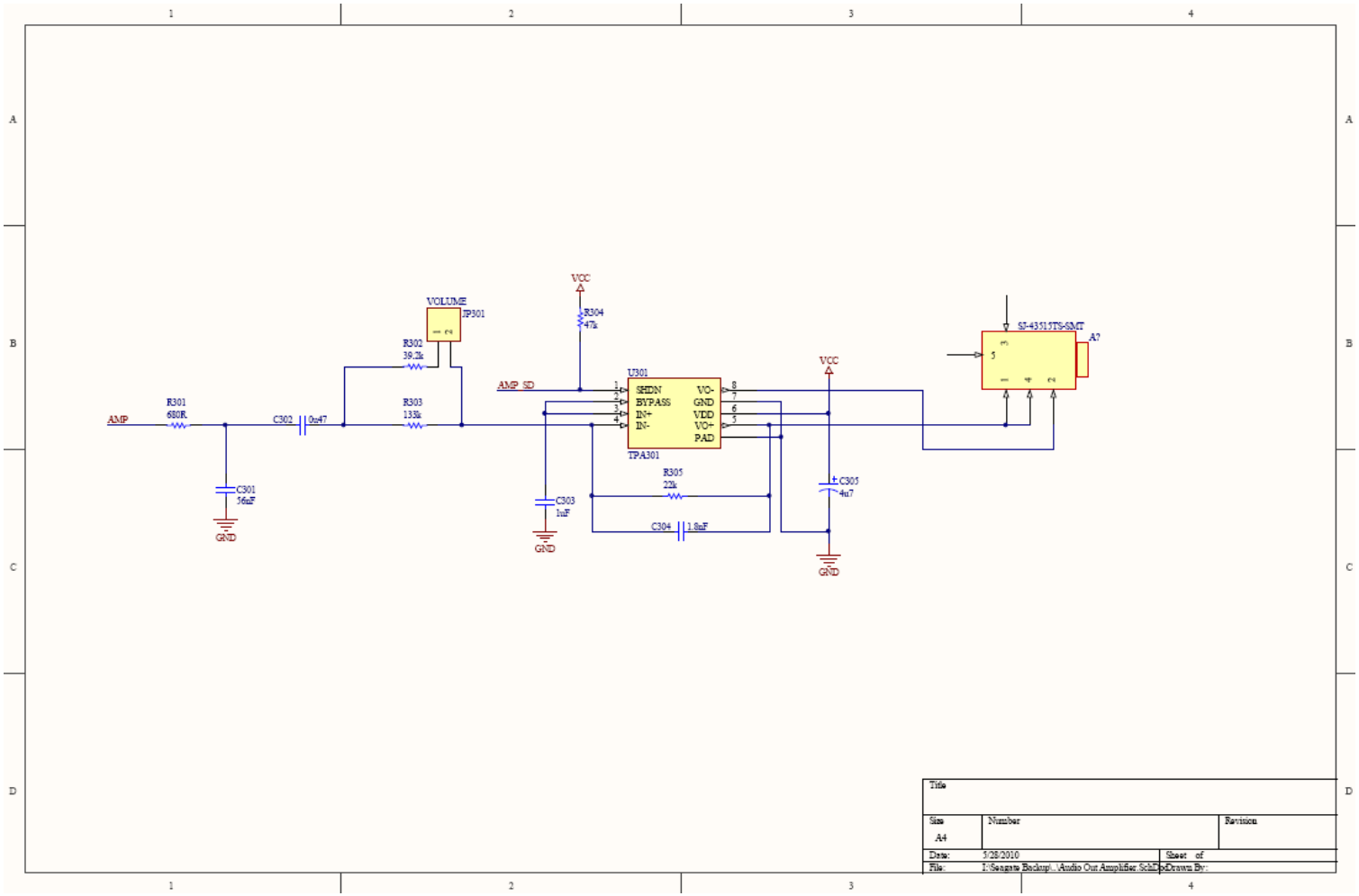
Şekil 8-2 Telsiz Haberleşme Birimine ait şematik

Ek 3 Düğüm tasarımında kullanılan ses girişi şematiği



Şekil 8-3 Düğüm tasarımında kullanılan ses giriş şematiği

Ek 4 Düğüm tasarımında kullanılan ses çıkışı şematiği



Şekil 8-4 Düğüm tasarımında kullanılan ses çıkışı şematiği

Ek 5 Kullanıcı Arayüzü Bağlantı Bacakları

Çizelge 8.1 Düğümün kullanıcı girişlerinin bacak bağlantıları

Giriş	Bacak
5 yönlü kumanda kolu (Sol)	P2.1
5 yönlü kumanda kolu (Sağ)	P2.2
5 yönlü kumanda kolu (Merkez)	P2.3
5 yönlü kumanda kolu (Yukarı)	P2.4
5 yönlü kumanda kolu (Aşağı)	P2.5
Düğme 1	P2.6
Düğme 2	P2.7
Reset Düğmesi	RST / NMI
Led 1	P1.0
Led 2	P1.1

Ek 6 Contiki OS basit kod örneği

```
/*---------------------------------------------------------------------------*/
/* Süreç deklare ediliyor. */
PROCESS(test_leds_process, "Test leds");

/* Modül yüklendiğinde sürecin başlaması sağlanıyor. */
AUTOSTART_PROCESSES(&test_leds_process);
/*---------------------------------------------------------------------------*/

/* Süreç kodu uygulanıyor. */
PROCESS_THREAD(test_leds_process, ev, data)
{
  PROCESS_BEGIN(); /* Süreçler bu ifade ile başlamak zorundadır. */

  /* değişkenlerin statik tanımlanmasına dkkat edilmelidir. */
  Static struct etimer etimer;
  static uint8_t active = 0;

  /* Ledler başlatılıyor. */
  leds_init();
  leds_off(LEDS_ALL);

  while(1) { /* sonsuz döngü */

    /* aktif zamanlayıcı başlatılıyor. */
    etimer_set(&etimer, CLOCK_SECOND);

    /* set edilen zamanın tamamlanması bekleniyor. */
    PROCESS_WAIT_UNTIL(etimer_expired (&etimer));

    if(!active) {
      /* Kırmızı ledi yak */
      leds_on(LEDS_RED);
      leds_off(LEDS_YELLOW);
    } else {
      /* sarı ledi yak */
      leds_off(LEDS_RED);
      leds_on(LEDS_YELLOW);
    }
    active ^= 1;
  }
  PROCESS_END();  /* Süreçler bu ifade ile bitmek zorundadır. */

}
/*---------------------------------------------------------------------------*/
```

Ek 7 leds-arch.c dosyası

```
#include "contiki-conf.h"
#include "dev/leds.h"

#include <io.h>

/*---------------------------------------------------------------------------*/
void
leds_arch_init(void)
{
  LEDS_PxDIR |= (LEDS_CONF_RED | LEDS_CONF_GREEN | LEDS_CONF_YELLOW);
  LEDS_PxOUT |= (LEDS_CONF_RED | LEDS_CONF_GREEN | LEDS_CONF_YELLOW);
}
/*---------------------------------------------------------------------------*/
unsigned char
leds_arch_get(void)
{
  return ((LEDS_PxOUT & LEDS_CONF_RED) ? LEDS_RED : 0)
    | ((LEDS_PxOUT & LEDS_CONF_GREEN) ? LEDS_GREEN : 0)
    | ((LEDS_PxOUT & LEDS_CONF_YELLOW) ? LEDS_YELLOW : 0);
}
/*---------------------------------------------------------------------------*/
void
leds_arch_set(unsigned char leds)
{
  LEDS_PxOUT = (LEDS_PxOUT & ~(LEDS_CONF_RED|LEDS_CONF_GREEN|LEDS_CONF_YELLOW))
    | ((leds & LEDS_RED) ? LEDS_CONF_RED : 0)
    | ((leds & LEDS_GREEN) ? LEDS_CONF_GREEN : 0)
    | ((leds & LEDS_YELLOW) ? LEDS_CONF_YELLOW : 0);
}
/*---------------------------------------------------------------------------*/
```

Ek 8 audio.c dosyası

```
static void setupRecord(void)
{
  LED_PORT_OUT |= LED_1;                    // Turn on LED
  AUDIO_PORT_OUT |= MIC_POWER_PIN;
  AUDIO_PORT_OUT &= ~MIC_INPUT_PIN;
  AUDIO_PORT_SEL |= MIC_INPUT_PIN;

  TBCTL = TBSSEL_2;                         // Use SMCLK as Timer_B source
  TBR = 0;
  TBCCR0 = 2047;                            // Initialize TBCCR0
  TBCCR1= 2047- 100;
  TBCCTL1 = OUTMOD_7;

  UCSCTL8 |= MODOSCREQEN;
  ADC12CTL2 = ADC12RES_0;                   // Select 8-bit resolution
  ADC12CTL0 = ADC12ON + ADC12SHT02 ;
  ADC12CTL1 = ADC12SHP + ADC12CONSEQ_2 + ADC12SSEL_2 + ADC12SHS_3;
  //Sequence of channels, once
  ADC12MCTL0 = MIC_INPUT_CHAN | ADC12EOS  ; //VeREF+ and VeREF-
  ADC12CTL0 |= ADC12ENC;                    //Enable
  ADC12CTL0 |= ADC12SC;                     //Start conversion

  DMACTL0 = DMA0TSEL_24;                    // ADC12IFGx triggers DMA0
  DMA0SA = (void (*)())ADC12MEM0_;          // Src address = ADC12 module

}


static void shutdownRecord(void)
{
  TBCTL &= ~MC0;
  ADC12CTL0 &= ~( ADC12ENC + ADC12ON );

  FCTL1 = FWKEY;                            // Disable Flash write
  FCTL3 = FWKEY + LOCK;                     // Lock Flash memory

  // Power-down MSP430 modules
  ADC12CTL1 &= ~ADC12CONSEQ_2;              // Stop conversion immediately
  ADC12CTL0 &= ~ADC12ENC;                   // Disable ADC12 conversion
  ADC12CTL0 = 0;                            // Switch off ADC12 & ref voltage

  TBCTL = 0;                                // Disable Timer_B
  LED_PORT_OUT &= ~LED_1;                   // Turn off LED

  AUDIO_PORT_OUT &= ~MIC_POWER_PIN;         // Turn of MIC
  AUDIO_PORT_SEL &= ~MIC_INPUT_PIN;

  //saveSettings();                           // Store lastAudioByte to Flash


}


static void record(void)
{
  FCTL3 = FWKEY;                            // Unlock the flash for write
  FCTL1 = FWKEY + BLKWRT;

  DMA0CTL = DMADSTINCR_3 + DMAEN + DMADSTBYTE +  DMASRCBYTE + DMAIE;
  // Enable Long-Word write, all 32 bits will be written once
  // 4 bytes are loaded

  TBCCTL1 &= ~CCIFG;
  TBCTL |= MC0;

  __bis_SR_register(LPM0_bits + GIE);       // Enable interrupts, enter LPM0

  TBCTL &= ~MC0;
  DMA0CTL &= ~( DMAEN + DMAIE);

  FCTL3 = FWKEY + LOCK;                     // Lock the flash from write
}
```

```
void audioRecord(unsigned char mode)
{
  unsigned char i;

  setupRecord();

  flashErase(MemstartTest, Memend);
  DMA0DA = (void (*)())MemstartTest;
  DMA0SZ = (long) (Memend - MemstartTest);

  record();

  if (DMA0SZ != ( long) (Memend - MemstartTest))
    lastAudioByte = Memend - DMA0SZ;
  else
    lastAudioByte = Memend;

  shutdownRecord();
}


void audioPlayBack(unsigned char mode)
{
  // Power-up external hardware
  AUDIO_PORT_DIR |= AUDIO_OUT_PWR_PIN;
  AUDIO_PORT_OUT &= ~AUDIO_OUT_PWR_PIN;
  AUDIO_OUT_SEL |= AUDIO_OUT_PIN;           // P4.4 = TB4
  LED_PORT_OUT |= LED_1;                    // Turn on LED

  PlaybackPtr = (unsigned long)(Memstart);

  PlaybackPtr ++;

  /* Setup Timer0_A for playback */
  // Use SMCLK as Timer0_A source, enable overflow interrupt
  TBCTL = TBSSEL_2 + TBIE;
  // Set output resolution (8 bit. Add 10 counts of headroom for loading TBCCR1
  TBCCR0 = 255 ;
  TBCCR4 = 255 >> 1;                        // Default output ~Vcc/2
  // Reset OUT1 on EQU1, set on EQU0. Load TBCCR1 when TBR counts to 0.
  TBCCTL4 = OUTMOD_7 + CLLD_1;
  // Start Timer_B in UP mode (counts up to TBCCR0)
  TBCTL |= MC0;

  // Activate LPM during DMA playback, wake-up when finished
  __bis_SR_register(LPM0_bits + GIE);       // Enable interrupts, enter LPM0

  // Power-down MSP430 modules
  TBCTL = 0;                                // Disable Timer_B PWM generation
  AUDIO_OUT_SEL &= ~AUDIO_OUT_PIN;          // P4.4 = TB4
  LED_PORT_OUT &= ~LED_1;                   // Turn off LED
}
```

Ek 9 msp430x54xx.h dosyası

```
#if !defined(__msp430x54xx)

#define __msp430x54xx
/* msp430x54xx.h
 *
 * mspgcc project: MSP430 device headers
 * MSP430x54xx family header
 *
 * (c) 2006 by Steve Underwood <steveu@coppice.org>
 * Originally based in part on work by Texas Instruments Inc.
 *
 * 2008-10-08 - sb-sf (sb-sf@users.sf.net)
 * - created, based on msp430xG461x.h
 *
 * 2009-10-08 - modifications by J.M.Gross <mspgcc@grossibaer.de>
 * - addeditional includes (TimerB, CRC)
 * - added usage of base addresses for UCS, timers etc.
 * - added SR bit constants from common.h (temporary solution)
 * 2009-11-20 - modifications by J.M.Gross <mspgcc@grossibaer.de>
 * - split definitions for 8/11port and 2/4 USCI CPUs
 * - added PMM module
 * 2010-01-29 - modifications by J.M.Gross <mspgcc@grossibaer.de>
 * - added missing stuff from old common.h
 * - cleaned the SFR definitions
 *
 * 2010-04-10 - modifications by Esra Celik
 * - removed faulty include header pmm.h
 * - added new version Power_Management.h
 * - added new version ADC.h
 * - added new version RTC.h
 * - added new version DMA.h
 *
 * $Id: msp430x54xx.h,v 1.5 2009/06/04 21:55:18 cliechti Exp $
 */

#include <iomacros.h>

#define __MSP430_PMM5_BASE__    0x120
#define __MSP430_CRC16_BASE__   0x150
#define __MSP430_WDT_A_BASE__   0x150
#define __MSP430_UCS_BASE__     0x160
#define __MSP430_SYS_BASE__     0x180
#define __MSP430_PORT1_BASE__   0x200
#define __MSP430_PORT2_BASE__   0x200
#define __MSP430_PORT3_BASE__   0x220
#define __MSP430_PORT4_BASE__   0x220
#define __MSP430_PORT5_BASE__   0x240
#define __MSP430_PORT6_BASE__   0x240
#define __MSP430_PORT7_BASE__   0x260
#define __MSP430_PORT8_BASE__   0x260
#define __MSP430_PORTJ_BASE__   0x320
#define __MSP430_HAS_T0A5__
#define __MSP430_T0A_BASE__     0x340
#define __MSP430_HAS_T1A3__
#define __MSP430_T1A_BASE__     0x380
#define __MSP430_HAS_TB7_5__
#define __MSP430_TB7_BASE__     0x3c0
#define __MSP430_MPY32_BASE__   0x4C0
#define __MSP430_USCI5_BASE_0__ 0x5C0
#define __MSP430_USCI5_BASE_1__ 0x600
#define __MSP430_ADC12_A_BASE__ 0x700

#if defined(__MSP430_5438__) || defined(__MSP430_5436__) ||  defined(__MSP430_5419__)
#define __MSP430_PORT9_BASE__   0x280
#define __MSP430_PORT10_BASE__  0x280
#define __MSP430_PORT11_BASE__  0x2A0
#define __MSP430_USCI5_BASE_2__ 0x640
#define __MSP430_USCI5_BASE_3__ 0x680
#endif
```

```
#include <msp430/wdt_a.h>
#include <msp430/sys.h>
//#include <msp430/pmm.h>
#include <msp430/gpio_5xxx.h>
#include <msp430/mpy32.h>
#include <msp430/timera.h>
#include <msp430/timerb.h>
#include <msp430/crc16.h>
#include <msp430/unified_clock_system.h>
#include <msp430/usci.h>
#include <msp430new/ADC.h>
BUILDADC(0x0700);
#include <msp430new/Power_Management.h>
BUILDPMM(0x0120);
// RTC.h
#include <msp430new/RTC.h>
BUILDRTC(0x04A0);
//DMA.h
#include <msp430new/DMA.h>
BUILDDMA(0x0500);
BUILDDMACHANNEL(0, 0x0510);
BUILDDMACHANNEL(1, 0x0520);
BUILDDMACHANNEL(2, 0x0530);

/* Status register flags */
#define C                   0x0001
#define Z                   0x0002
#define N                   0x0004
#define V                   0x0100
#define GIE                 0x0008
#define CPUOFF              0x0010
#define OSCOFF              0x0020
#define SCG0                0x0040
#define SCG1                0x0080

/* low power mode defines */
#ifdef __ASSEMBLER__ /* Begin #defines for assembler */
#define LPM0                CPUOFF
#define LPM1                SCG0+CPUOFF
#define LPM2                SCG1+CPUOFF
#define LPM3                SCG1+SCG0+CPUOFF
#define LPM4                SCG1+SCG0+OSCOFF+CPUOFF
#else /* Begin #defines for C */
#define LPM0_bits           CPUOFF
#define LPM1_bits           SCG0+CPUOFF
#define LPM2_bits           SCG1+CPUOFF
#define LPM3_bits           SCG1+SCG0+CPUOFF
#define LPM4_bits           SCG1+SCG0+OSCOFF+CPUOFF

#define LPM0      _BIS_SR(LPM0_bits)    /* Enter Low Power Mode 0 */
#define LPM0_EXIT _BIC_SR_IRQ(LPM0_bits) /* Exit Low Power Mode 0 */
#define LPM1      _BIS_SR(LPM1_bits)    /* Enter Low Power Mode 1 */
#define LPM1_EXIT _BIC_SR_IRQ(LPM1_bits) /* Exit Low Power Mode 1 */
#define LPM2      _BIS_SR(LPM2_bits)    /* Enter Low Power Mode 2 */
#define LPM2_EXIT _BIC_SR_IRQ(LPM2_bits) /* Exit Low Power Mode 2 */
#define LPM3      _BIS_SR(LPM3_bits)    /* Enter Low Power Mode 3 */
#define LPM3_EXIT _BIC_SR_IRQ(LPM3_bits) /* Exit Low Power Mode 3 */
#define LPM4      _BIS_SR(LPM4_bits)    /* Enter Low Power Mode 4 */
#define LPM4_EXIT _BIC_SR_IRQ(LPM4_bits) /* Exit Low Power Mode 4 */
/* LPM5 is only available on 54xxA devices and requires PMMREGOFF=1 and LPM4 */
#endif /* End #defines for C */


/* bit definitions (for those who cannot HEX), taken from common.h */
#define BIT0                0x0001
#define BIT1                0x0002
#define BIT2                0x0004
#define BIT3                0x0008
#define BIT4                0x0010
#define BIT5                0x0020
#define BIT6                0x0040
#define BIT7                0x0080
#define BIT8                0x0100
#define BIT9                0x0200
#define BITA                0x0400
#define BITB                0x0800
#define BITC                0x1000
```

```
#define BITD                0x2000
#define BITE                0x4000
#define BITF                0x8000


/* special function register SFRIE1 (interrupt enable) */
#define SFRIE1_             0x0100
sfrw(SFRIE1, SFRIE1_);
sfrb(IE1, SFRIE1_);
#define WDTIE               (1<<0)  /* Watchdog interrupt enable             */
#define OFIE                (1<<1)  /* Oscillator fault interrupt enable     */
/* RESERVED                 (1<<2)*/
#define VMAIE               (1<<3)  /* Vacant memory address interrupt enable */
#define NMIIE               (1<<4)  /* NMI interrupt enable                  */
#define ACCVIE              (1<<5)  /* FLASH controller access violation     */
#define JMBINIE             (1<<6)  /* JTAG mailbox input interrupt enable   */
#define JMBOUTIE            (1<<7)  /* JTAG mailbox output interrupt enable  */

/* special function register SFRIFG1 (interrupt flags) */
#define SFRIFG1_            0x0102
sfrw(SFRIFG1, SFRIFG1_);
sfrb(IFG1, SFRIFG1_);
#define WDTIFG              (1<<0)  /* Watchdog interrupt flag               */
#define OFIFG               (1<<1)  /* Oscillator fault interrupt flag       */
/* RESERVED                 (1<<2)*/
#define VMAIFG              (1<<3)  /* Vacant memory address interrupt flag  */
#define NMIIFG              (1<<4)  /* NMI interrupt flag                    */
/* RESERVED                 (1<<5)*/
#define JMBINIFG            (1<<6)  /* JTAG mailbox input interrupt flag     */
#define JMBOUTIFG           (1<<7)  /* JTAG mailbox output interrupt flag    */


/* special function register SFRRPCR (RESET pin control) */
#define SFRRPCR_            0x0104
sfrw(SFRRPCR, SFRRPCR_);
#define SYSNMI          (1<<0)  /* RST/NMI pin (0:Reset, 1: NMI) */
#define SYSNMIIES       (1<<1)  /* NMI edge select (0:rising edge). Can trigger NMI */
#define SYSRSTUP        (1<<2)  /* Reset resistor pin pullup (0: pulldown, 1: pullup) */
#define SYSRSTRE        (1<<3)  /* Reset pin resistor enable (0: disabled, 1: enabled) */



/* device specific FLASH controller (maybe to be moved into flash.h later, but currently
incompatible) */

#define __MSP430_FLASH_BASE__ 0x140
#define FCTL1_              __MSP430_FLASH_BASE__             /* Flash control 1 */
sfrw(FCTL1, FCTL1_);
// MSP430F54xx has no FCTL2!
#define FCTL3_              __MSP430_FLASH_BASE__ + 0x04      /* Flash control 3 */
sfrw(FCTL3, FCTL3_);
#define FCTL4_              __MSP430_FLASH_BASE__ + 0x05      /* Flash control 4 */
sfrw(FCTL4, FCTL4_);

#define FRKEY               0x9600        /* Flash key returned by read */
#define FWKEY               0xA500        /* Flash key for write */
#define FXKEY               0x3300        /* for use with XOR instruction */


/* FCTL1 bits */
#define ERASE               0x0002        /* Enable bit for flash segment erase */
#define MERAS               0x0004        /* Enable bit for flash mass erase */
#define SWRT                0x0020        /* Enable bit for flash smart write */
#define WRT                 0x0040        /* Enable bit for flash write */
#define BLKWRT              0x0080        /* Enable bit for flash segment write */

/* aliases by MSPGCC */
#define NOERASE             (0)           /* reset erase operations */
#define SEGERASE            (ERASE)       /* erase single segment */
#define BANKERASE           (MERAS)       /* erase one bank of flash memory */
#define MASSERASE           (ERASE|MERAS) /* erase all banks of main memory */
#define BWWRITE             (WRT)         /* Byte/Word write mode */
#define LWWRITE             (BLKWRT)      /* long word write mode */
#define LWBWRITE            (WRT|BLKWRT)  /* long word block write mode */

/* FCTL3 bits */
#define BUSY              0x01  /* flash controller is busy erasing/writing) */
```

```
#define KEYV              0x02  /* Flash security key violation */
#define ACCVIFG           0x04  /* Access violation interrupt flag */
#define WAIT              0x08  /* flash memory ready for next byte/word write */
#define LOCK              0x10  /* flash memory operations locked */
#define LOCKA               0x40  /* segment A write protected, B..D excluded from mass erase
(read) toggle lock state (write) */

/* FCTL4 bits */
#define VPE               0x01   /* voltage change while programming */
#define MRG0              0x10   /* marginal read mode 0 active */
#define MRG1              0x20   /* marginal read mode 1 active, preference over mode 0*/
#define LOCKINFO          0x80   /* info memory protected against write or erase */

/* interupt vectors */
#define RTC_A_VECTOR        0x52    /* 0xFFD2 Basic Timer / RTC */
#define PORT2_VECTOR        0x54    /* 0xFFD4 Port 2 */
#define USCIB3_RXTX_VECTOR  0x56    /* 0xFFD6 USCI B3 RX/TX */
#define USCIA3_RXTX_VECTOR  0x58    /* 0xFFD8 USCI A3 RX/TX */
#define USCIB1_RXTX_VECTOR  0x5A    /* 0xFFDA USCI B1 RX/TX */
#define USCIA1_RXTX_VECTOR  0x5C    /* 0xFFDC USCI A1 RX/TX */
#define PORT1_VECTOR        0x5E    /* 0xFFDE Port 1 */
#define TIMER1_A1_VECTOR    0x60    /* 0xFFE0 Timer1_A3 CC1-2, TA1 */
#define TIMER1_A0_VECTOR    0x62    /* 0xFFE2 Timer1_A3 CC0 */
#define DMA_VECTOR          0x64    /* 0xFFE4 DMA */
#define USCIB2_RXTX_VECTOR  0x66    /* 0xFFE6 USCI B2 RX/TX */
#define USCIA2_RXTX_VECTOR  0x68    /* 0xFFE8 USCI A2 RX/TX */
#define TIMER0_A1_VECTOR    0x6A    /* 0xFFEA Timer0_A5 CC1-4, TA0 */
#define TIMER0_A0_VECTOR    0x6C    /* 0xFFEC Timer0_A5 CC0 */
#define AD12_A_VECTOR       0x6E    /* 0xFFEE ADC */
#define USCIB0_RXTX_VECTOR  0x70    /* 0xFFF0 USCI B0 RX/TX */
#define USCIA0_RXTX_VECTOR  0x72    /* 0xFFF2 USCI A0 RX/TX */
#define WDT_VECTOR          0x74    /* 0xFFF4 Watchdog Timer */
#define TIMER0_B1_VECTOR    0x76    /* 0xFFF6 Timer_B7 CC1-6, TB */
#define TIMER0_B0_VECTOR    0x78    /* 0xFFF8 Timer_B7 CC0 */
#define USER_NMI_VECTOR     0x7A    /* 0xFFFA Non-maskable */
#define NMI_VECTOR          0x7C    /* 0xFFFC Non-maskable */
#define TIMERB1_VECTOR       TIMER0_B1_VECTOR /* Timer0_B7 CC1-6, TB */


#endif /* #ifndef __msp430x54xx */
```

ÖZGEÇMİŞ

Doğum tarihi	25.04.1985	
Doğum yeri	Ankara	
Lise	2000-2003	Hacı Ömer Tarman Anadolu Lisesi
Lisans	2003-2007	Yıldız Üniversitesi Elektrik Elektronik Fak. Bilgisayar Mühendisliği Bölümü
Yüksek Lisans	2007-2010	Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Müh. Anabilim Dalı

Çalıştığı kurum(lar)

	2008-....	TÜBİTAK Bilişim Teknolojileri Enstitüsü, Araştırmacı