İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

AÇIK KAYNAK KODLU İŞLEMCİ VE İŞLETİM SİSTEMLERİ KULLANILARAK NESNELERİN İNTERNETİ İÇİN ENERJİ ETKİN DÜĞÜM TASARIMI VE FPGA ÜZERİNDE GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

Mehmet Onur DEMİRTÜRK

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Sıddıka Berna ÖRS YALÇIN

TEMMUZ 2019

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

AÇIK KAYNAK KODLU İŞLEMCİ VE İŞLETİM SİSTEMLERİ KULLANILARAK NESNELERİN İNTERNETİ İÇİN ENERJİ ETKİN DÜĞÜM TASARIMI VE FPGA ÜZERİNDE GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

Mehmet Onur DEMİRTÜRK
(504151233)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Sıddıka Berna ÖRS YALÇIN

TEMMUZ 2019

İTÜ, Fen Bilimleri Enstitüsü'nün 504151233 numaralı Yüksek Lisans Öğrencisi Mehmet Onur DEMİRTÜRK, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "AÇIK KAYNAK KODLU İŞLEMCİ VE İŞLETİM SİSTEMLERİ KULLANILARAK NESNELERİN İNTERNETİ İÇİN ENERJİ ETKİN DÜĞÜM TASARIMI VE FPGA ÜZERİNDE GERÇEKLENMESİ" başlıklı tezini aşağıdaki imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Doç. Dr. Sıddıka Berna ÖRS YALÇIN** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Prof. Dr. Güneş Zeynep KARABULUT KURT** İstanbul Teknik Üniversitesi	.....
	Dr. Öğr. Üyesi Tuba AYHAN MEF Üniversitesi	.....
		.....

Teslim Tarihi : **22 Temmuz 2019**
Savunma Tarihi : **25 Temmuz 2019**

Sevgili Aileme...

ÖNSÖZ

Nesnelerin interneti kapsamında gelişen teknolojiyle, insan hayatını kolaylaştıran bir çok uygulama ortaya çıkmıştır. Bu kapsamda kullanılan cihazların çeşitli ihtiyaçları vardır. Düşük enerji tüketimi de bu ihtiyaçlardan biridir. Yapılan tez çalışmasında nesnelerin interneti için düşük enerji tüketimi temel kıstas olarak belirlenmiş bir duyarga düğümü geliştirilmiştir. Bu amaçla düğümün kırmık üstü sistem tasarımı ve FPGA üzerinde gerçeklemeleri yapılmıştır.

Bu çalışma; İTÜ Bilimsel Araştırma Projeleri Birimince desteklenmiştir. Proje numarası: 41185.

Çalışmalarımda danışmanlığımı üstlenen, tecrübelerini paylaşan ve desteğini esirgemeyen sayın Doç. Dr. Sıddıka Berna Örs Yalçın'a,

Tez çalışmam sırasında her türlü konuda bana yardımcı olan, sorularımı bıkmadan usanmadan cevaplayan ve çalışmalarımı takip eden değerli arkadaşlarım Y.Müh. Latif Akçay, Y.Müh Emre Göncü ve Y.Müh Özen Özkaya'ya,

Beni yetiştirip, koruyup, gözetip bu günlere gelebilmem için her türlü fedakârlığı yapan, sevgisini ve desteğini her zaman ve her yerde hissettiğim Anneme ve Babama,

Güzel ablalarım Aybüke ve Aybegüm'e ve kurdukları güzel ailelerine; ağabeyim, ilk arkadaşım Fazlı'ya,

Mesafeler farketmeksizin yanımda oldukları için tüm dostlarıma,

Sevgi, saygı ve minnetlerimi sunarım.

Temmuz 2019

Mehmet Onur DEMİRTÜRK
Elektrik-Elektronik Mühendisi

İÇİNDEKİLER

Sayfa

KISALTMALAR

AHB : Advanced High Speed Bus
AMBA : Advanced Microcontroller Bus Architecture
APB : Advanced Peripheral Bus
API : Application Programming Interface
ASIC : Application Spesific Integrated Circuit
BCC : Bare-C Cross Compiler
BLE : Bluetooth Low Energy
CPU : Central Processing Unit
DSP : Digital Signal Processing Unit
EDA : Electronic Design Automation
FPGA : Field Programmable Gate Array
GPIO : General Purpose Input Output
GRLIB : Gaisler Library
GRMON : Gaisler Monitor
IoT : Internet of Things
IP : Intellectual Property
RISC : Reduced Instruction Set Computing
RTOS : Real Time operating System
RTS : Real Time System
SPI : Serial Peripheral Interface
VCD : Value Change Dump
WCET : Worst Case Execution Time
WSN : Wide Sensor Network

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

AÇIK KAYNAK KODLU İŞLEMCİ VE İŞLETİM SİSTEMLERİ KULLANILARAK NESNELERİN İNTERNETİ İÇİN ENERJİ ETKİN DÜĞÜM TASARIMI VE FPGA ÜZERİNDE GERÇEKLENMESİ

ÖZET

Nesnelerin interneti (Internet of things - IoT) benzersiz olarak adreslenebilen milyonlarca nesnenin internete bağlanarak oluşturduğu dünya çapında bir ağdır. Günümüzde IoT uygulamaları endüstriyel otomasyon, sağlık, akıllı bina, şehir ve trafik sistemleri gibi birçok alanda karşımıza çıkmaktadır. Gün geçtikçe bu konsept daha yaygın bir hale gelmekte ve her gün daha fazla gelişmektedir.

IoT cihazları üst uç (high-end) ve alt uç (low-end) cihazlardan oluşmaktadır. Alt uç cihazlar genellikle enerji, işlemci ve hafıza kapasitesi gibi kaynaklar açısından oldukça kısıtlıdır. Duyarga, eyleyici ve giyilebilir cihazlar bu alt uç cihazlara örnektir. Kısıtlı kaynaklara sahip nesnelerin uzun yıllar batarya değişimi yapılmadan uygulamalara hizmet vermesi beklenmektedir. Bu da IoT'de enerji etkin gerçekleme ihtiyacını doğurur.

Tez kapsamında düşük enerji tüketimli bir duyarga düğümünün tasarımı amaçlanmıştır. Bu tasarımın kolaylıkla yapılabilmesi ve mühendislik maliyetlerinin düşürülmesi amacıyla yazılım-donanım ortak tasarım yöntemi kullanılmıştır. Bu doğrultuda donanımın gerçeklemesi aşamasında FPGA (Field Programmable Gate Array - Alanda Programlanabilir Kapı Dizileri) kullanılmıştır. Geliştirilen sistemde kullanılan kodların açık kaynak olmasına dikkat edilmiştir.

IoT kapsamında tasarlanan duyarga düğümleri; algılayıcı birimler, işlemciler ve haberleşme birimi olmak üzere üç kısımdan oluşmaktadır. Çalışmada işlemci ihtiyacının karşılanması için açık kaynak kodlu yazılımsal işlemcilerden OpenRISC ve LEON3 işlemcileri araştırılmış ve FPGA üzerinde gerçeklemesi yapılmıştır. Yazılımsal işlemciler, mimarisi ve davranışı bir donanım tanımlama diliyle tanımlanmış, FPGA ya da ASIC üzerinde gerçeklenebilen ve ihtiyaca göre yapıları değiştirilebilen işlemcilerdir. Bu işlemcilerin karşılaştırması literatürden görüldüğü kadarıyla genellikle komut kümesi mimarisi, derleyici dili, doğrulama durumu, dokümantasyon paylaşımı, sözcük uzunluğu, veriyolu standardı, saat frekansı özellikleri kullanılarak yapılmıştır. Bu projede ise yazılımsal işlemcilerin enerji tüketimine göre kıyaslaması yapılmıştır. Bu amaçla, yaygın olarak kullanılan kıyaslama programları işlemciler üzerinde çalıştırılarak işlemciler test edilmiştir. Bu programlar kendi kendine yeten, C diliyle yazılmış programlardır. Ek kütüphane dosyası, işletim sistemi ya da dosya sistemine ihtiyaç duymaz. Yapılan testler doğrultusunda LEON3 işlemcisinin daha az güç tükettiği ve uygulamaları daha kısa sürede tamamladığı yani daha az enerji harcadığı görülmüştür. Tasarımda LEON3 işlemcisi seçilerek çalışmalara devam edilmiştir.

IoT kapsamında yapılan birçok uygulamada nesnelerin işletim sistemine ihtiyacı olduğu görülmüştür. Bu nedenle gerçeklemesi yapılan işlemci üzerine bir işletim

sistemi kurulmasına karar verilmiştir. Bu amaçla Linux kernel, Contiki, RIOT ve FreeRTOS işletim sistemleri incelenmiştir. Geleneksel işletim sistemlerinden Linux kernel çok fazla belleğe ihtiyaç duymaktadır ve incelenen diğer işletim sistemlerine kıyasla daha fazla enerji tüketmektedir. Bu nedenle IoT uygulamalarında kullanmaya elverişli olmadığı görülmüştür. Contiki işletim sistemi ise IoT uygulamalarında oldukça yaygın olarak kullanılan, düşük güç tüketimli ve az bellek kullanan bir işletim sistemidir. Ancak bu işletim sisteminin gerçek zaman kabiliyeti kısıtlıdır. RIOT ve FreeRTOS işletim sistemleri ise gerçek zamanlı, düşük güç tüketimli ve az bellek kullanan işletim sistemleridir. FreeRTOS işletim sistemi hâlihazırda LEON3 işlemcisi için uyarlandığı için bu işletim sisteminin kullanılmasına karar verilmiştir.

FreeRTOS işletim sisteminin kullanımı oldukça kolaydır. C programlama diliyle yazılmış üç dosya işletim sisteminin çekirdeğini oluşturmaktadır. Bununla birlikte işletim sistemi üzerinde çalışacak uygulamalar C programlama diliyle yazılabilmektedir. Yazılan uygulama işletim sisteminin uygulama arayüzü fonksiyonları kullanılarak görevler şeklinde tanımlanır. Yine bu fonksiyonlar kullanılarak görevlere öncelik atanabilir. Yazılan uygulama, çekirdeği üretecek dosyalar ve FreeRTOS kütüphanesi kullanılarak derlenir böylece uygulamaya işletim sisteminin özellikleri kazandırılmış olur.

Çalışmada geleneksel işletim sistemleriyle gerçek zamanlı işletim sistemlerinin farkının anlaşılabilmesi için Linux kernel ve FreeRTOS işletim sistemleri derlenerek işlemci üzerinde çalıştırılmıştır. Üretilen imaj dosyaları karşılaştırıldığında Linux kernelin 6MB boyutunda, FreeRTOS'un ise 65kB gibi küçük bir boyutta olduğu görülmüştür. FreeRTOS'un çalıştırdığı görev sayısına göre boyutunun arttığı gözlenmiştir. Ayrıca Linux kerneli derlemenin çok fazla vakit aldığı gözlenmiştir.

Açık kaynak kodlu işlemci ve işletim sistemleri kullanılarak hazırlanan kırmık üstü sisteme algı cihazı olarak ultrasonik duyarga eklenmesine karar verilmiştir. Bu amaçla LEON3 işlemcisi için duyargayı çalıştıracak bir modül yazılarak işlemci genişletilmiştir. Bu modül işlemciyle APB (Advanced Peripheral Bus) üzerinden haberleşmektedir. Son olarak düşük güç tüketimli bir haberleşme protokolü olan Bluetooth düşük enerji (Bluetooth low energy - BLE) cihazı sisteme eklenmiştir. Bu cihaz işlemciyle UART arayüzü üzerinden haberleşmektedir. Bu nedenle cihaz işlemciye bağlanırken ek bir sürücüye ihtiyaç olmamıştır. Algı cihazı, işlemci, işletim sistemi ve haberleşme biriminden oluşan duyarga bu şekilde tamamlanarak IoT uygulamalarını gerçeklemeye hazır hale getirilmiştir.

Hazırlanan bu sistem kullanılarak kişilerin bina içindeki varlığını tespit etmek için FreeRTOS işletim sistemi üzerinde 5 görevden oluşan bir uygulama yazılmıştır. Tanımlanan görevlerden ilki çağrıldığında, bir katsayı hesabı yapılmaktadır. Hesaplanan katsayı belirli bir yazmaç adresine kaydedilmektedir. Bu katsayı ortam sıcaklığına bağlı olarak değişmektedir. Uygulamada sıcaklık bilgisi el ile girilmektedir. Tanımlanan görevlerden üçü, üç ayrı duyarganın kontrolünde kullanılmaktadır. Bu görevler ile ultrasonik duyargalar çalıştırılarak elde edilen sayaç verileri belirli bellek yazmaçlarına kaydedilmektedir. Daha sonra bu sayaç verileri, ilk görevden elde edilen katsayıya bölünerek santimetre cinsinden mesafe hesabı yapılmaktadır. Son olarak bu mesafeler oda boyutuyla karşılaştırılır. Ölçülen mesafe oda boyutundan küçük ise "11111111" değil ise "00000000" verisi üretilir. Böylece her bir duyarganın ayrı ayrı bir nesneyi görüp görmediği kaydedilmiş olur. Tanımlanan beşinci görevde ise duyargalardan üretilen veriler karşılaştırılarak odada birinin var olup olmadığı

mesajı üretilir. Üretilen bu mesaj BLE modülü ile yayınlanır. Görevlerin ana fonksiyon içerisinde çağrılmasıyla hazırlanan yazılım derlenerek LEON3 işlemcisine aktarılmıştır. Üç duyarganın bağlı olduğu sistem üzerinde çalıştırılan bu uygulama test edilmiştir ve doğru çalıştığı belirlenmiştir.

Yapılan bu çalışmayla tasarımcılara enerji etkin düğüm tasarımında izleyebilecekleri bir kaynak sağlanmıştır.

ENERGY EFFICIENT NODE DESIGN for INTERNET of THINGS and IMPLEMENTATION on FPGA by USING OPEN SOURCE PROCESSORS and OPERATING SYSTEMS

SUMMARY

The Internet of Things is a world-wide network of millions of objects that are uniquely addressable by connecting to the Internet. Nowadays, IoT applications are encountered in many fields such as industrial automation, health, smart building, city and traffic systems. This concept is becoming more common and developing every day.

Firstly the general structure, features and needs of the Internet of things are studied within the context of the thesis. The application areas of the IoT are also examined. This network of millions of objects has been found to be in a heterogeneous structure.

In this heterogeneous network, IoT devices are divided into high-end and low-end devices. Low-end devices are has very limited resources such as energy, processor and memory capacity. Sensor nodes, actuator nodes and wearable devices are examples of these low-end devices. Things with limited resources are expected to serve applications without changing the battery for many years. This leads to the need for energy efficient implementation in the IoT.

The aim of this thesis is to design a sensor node with low energy consumption. In order to design this sensor node easily and to reduce the engineering costs, hardware-software co-design methodology was used. According to this, FPGA (Field Programmable Array) was used to implement this hardware design. It has been ensured that the codes used in the developed system are open source codes.

Design model for a IoT application includes four layers. These layers are processor and peripherals, hardware abstraction, operating system and service layers. The lowest layer is the layer which includes processor, sensor and communication devices. Hardware abstraction layer is the layer between software and hardware. Drivers and hardware porting are included in this layer. The compatibility between the operating system layer and the processor is also ensured in this layer. The top layer is the service layer, where there are several sub-services to be used in the IoT network application.

Many applications can be done within the scope of IoT, but every application and every service brings along different needs. Therefore, according to the application to be selected within the scope of the IoT, the needs of the system should be determined from the top-down and the design should be carried out from the bottom to the top.

An application to determine the presence of people in the emergency evacuation system has been determined as a motivation point in design. In addition to the energy efficiency of the node designed for use in the application, it should be a real-time system. Real-time systems, as in this application, are expected to output at certain time constraints. This need can be achieved by the real time capability of the operating system and software to be used.

Sensor nodes designed within the scope of IoT consist of three parts: sensors, a processors and a communication unit. OpenRISC and LEON3 processors were searched from open source softcore processors to use in this design. These processors are implemented on FPGA.

Architecure and behaviour of the open source softcore processors are defined by using a hardware description language. These are processors that can be implemented on FPGA or ASIC and can be modified according to needs. As seen from the literature, the comparison of these processors is generally done by using command set architecture, compiler language, validation status, documentation sharing, word length, bus standard, clock frequency properties. In this project, open source processors were analyzed according to power consumption and execution time of processes. For this purpose, commonly used benchmarking programs were executed on processors and processors were tested. These programs are self-contained software programs written in C language. No additional library file, operating system or file system is required for this programs. According to the tests performed, the LEON3 processor consumes less power and programs executes in a shorter time on LEON3, which means it consumes less energy. The studies were continued by selecting the LEON3 processor in the design.

In many applications made within the scope of IoT, it is seen that things need operating system. For this reason, it was decided to install an operating system on the processor. For this purpose, Linux kernel, Contiki, RIOT and FreeRTOS operating systems were analyzed. Linux kernel from traditional operating systems needs a lot of memory and consumes more energy than the other operating systems. Therefore, it is seen that it is not suitable for use in IoT applications. The Contiki operating system is a widely used operating system with low power consumption and low memory usage in IoT applications. However, the real time capability of this operating system is limited. RIOT and FreeRTOS operating systems are real-time, low-powered and low-memory using operating systems. Since the FreeRTOS operating system is already ported to the LEON3 processor, it has been decided to use this operating system.

The core of the FreeRTOS operating system can be compiled by using only three files written in C programming language. This operating system is very easy to use. Applications to be run on the operating system can be written in C programming language. Written applications are defined as tasks by using the application programming interface functions of this operating system. Priorities can be assigned to tasks by using these functions. The files that will generate the kernel and the written application files are compiled together using the FreeRTOS library. Thus, the operating system features are provided to the applications.

In order to understand the difference between traditional operating systems and real-time operating systems, the Linux kernel and FreeRTOS operating systems were compiled and run on the processor. Compared to the image files produced, Linux kernel is 6MB in size and FreeRTOS is in a small size of 65kB. It was observed that the size of the FreeRTOS application increased according to the number of tasks. It is also observed that compiling the Linux kernel takes a lot of time.

It was decided to add an ultrasonic sensor as a sensing device to this system. For this purpose, the LEON3 processor is expanded by adding a module to run the sensor. This module communicates with the processor via the APB (Advanced Peripheral Bus). With the operation of the added module, the signal applied from the trigger pin of the

sensor emits an ultrasonic sound wave at a frequency of 40 kHz. When this sound wave hits any object and returns to the sensor, the echo pin becomes active. By measuring the time between these two signals, the distance of the object from the sensor is calculated. The calculated time is sent to the registers specified on the processor. These data can then be read from these registers for use in the written application.

Finally, the Bluetooth low energy (BLE) device, which is a low-power communication protocol, has been added to the system. This device communicates with the processor via UART. Therefore, there was no need for an additional driver when the device was connected to the processor. The node consisting of a sensor, a processor, an operating system and a communication unit has been completed in this way and is ready to implement IoT applications.

Using this system, an application consisting of 5 tasks has been written on the FreeRTOS operating system in order to detect the presence of people in the building. When the first of the defined tasks is called, a coefficient calculation is made. The calculated coefficient is saved to a specific register address. This coefficient varies depending on the ambient temperature. In this application, the temperature information is entered manually. Three of the defined tasks are used to control three separate sensors. With these tasks, the ultrasonic sensors are operated and the counter data obtained from the sensors are stored in certain memory registers. Then, the counter data is divided by the coefficient obtained from the first task and the distance is calculated in centimeters. Finally, these distances are compared with the size of the room. If the measured distance is smaller than the room size, "11111111" data is generated. If not, "00000000" data is generated. Thus, it is recorded whether or not each sensor sees an object. In the fifth task defined, the data generated from the sensors are compared and a message is generated to see if someone exists in the room. This generated message is transmitted with the BLE module. The software prepared by calling the tasks in the main function was compiled and transferred to the LEON3 processor. This application, which runs on the system, has been tested and found to be working correctly.

With this study, a resource that designers can follow in energy efficient node design is provided.

1. GİRİŞ

Gelişen teknolojiyle birlikte nesnelerin interneti (Internet of Things - IoT) kapsamında bir çok akıllı nesne ve cihaz üretilmiş ve bu teknolojilere dayanan bir çok uygulama hayata geçirilmiştir.

IoT cihazları hem üst uç hem de alt uç cihazlardan oluşmaktadır. Alt uç cihazlar genellikle enerji, işlemci ve hafıza kapasitesi dahil olmak üzere kaynaklar açısından oldukça kısıtlı nesnelerdir [1]. IoT'nin en alt katmanını oluşturan bu nesnelerin kaynaklar bakımından kısıtlı olması bu kapsamdaki gelişmelerin hızını büyük bir oranda kesmektedir. Düşük güç tüketimi, bu kısıtların doğurduğu en büyük ihtiyaçlardan biridir. Birçok nesneye ev sahipliği yapan sistemlerin batarya değişimi yapılmadan uzun yıllar kullanılması beklenmektedir. Bu ihtiyacın karşılanabilmesi için hem donanım hem de yazılım tarafında çözümler aranmaktadır.

Sistemlerin tasarımında tasarım boyutu, kullanım alanı, güç tüketimi, performans gibi kısıtların yanında araştırma ve pazara çıkma süresi gibi kısıtlarla karşılaşılmaktadır [2]. Donanım olarak yapılan tasarımlarda düşük kullanım alanı, düşük güç tüketimi ve yüksek performans özelliklerine ulaşılabilmesine karşın doğrulama, hata ayıklama, araştırma ve pazara çıkma süresi oldukça uzun olmaktadır. Yazılımla yapılan tasarımlarda ise bu süreler oldukça kısa olmasına karşın yüksek performansları elde etmek oldukça zordur. Bu noktada yazılım-donanım ortak tasarım yöntemi ortaya çıkmaktadır. Böylece her iki tasarım yönteminin avantajlarından faydalanılabilmektedir [3].

Bu çalışmada düşük enerji tüketimi özelliği temel kıstas olarak belirlenmiş bir duyarga düğümünün geliştirilmesi amaçlanmıştır. Bu amaçla yazılım-donanım ortak tasarımı yöntemine dayanarak kırmık üstü sistem tasarımı ve FPGA (Field Programmable Gate Array - Alanda Programlanabilir Kapı Dizileri) üzerinde gerçeklemeleri yapılmıştır.

Tezin ikinci bölümünde nesnelerin internetinin mimarisi, uygulama alanları, ihtiyaçları ve kısıtlarıyla ilgili bilgiler verilmiştir. Üçüncü bölümde gerçekleştirilecek sistemin

planı yapılmıştır. Dördüncü bölümde sistemin donanım gerçeklemesi, beşinci bölümde ise yazılım gerçeklemesi anlatılmıştır. Altıncı ve son bölümde ise sonuçlara ve gelecek çalışmalarla ilgili bilgilere yer verilmiştir.

2. NESNELERİN İNTERNETİ

Nesnelerin interneti 1990 yıllarında basit uygulamalar olarak ortaya çıkmaya başlamış ve ilk kez 1999 yılında Kevin Ashton tarafından bir şirkete yaptığı sunumda kavram olarak kullanılmıştır. Günümüzde ise nesnelerin interneti ağı ve uygulamaları büyük gelişmeler göstermiştir. Hem günlük yaşantıda hem de akademik çalışmalarda sıklıkla karşımıza çıkan bir konsepte dönüşmüştür.

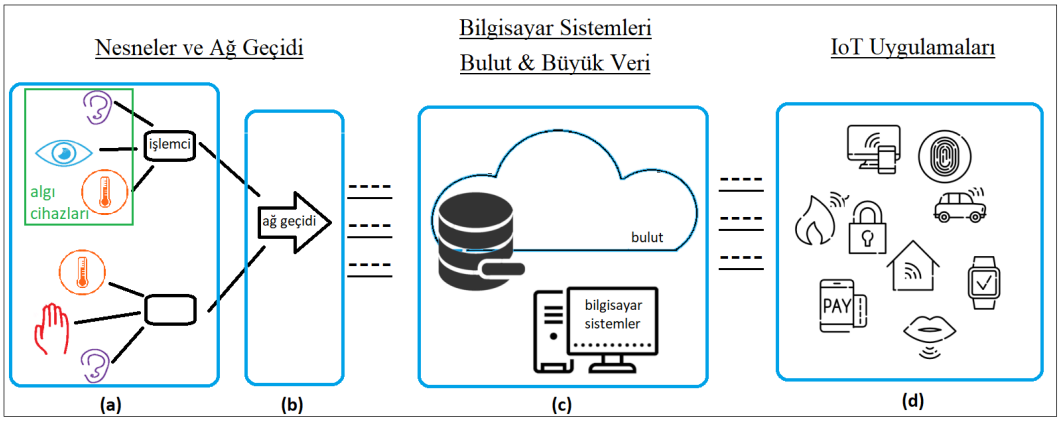
Nesnelerin interneti için birçok farklı tanım yapılmıştır. ITU'nun 2012 yılında nesnelerin interneti için yaptığı tanıma göre "Nesnelerin interneti, birlikte çalışabilir bilgi ve iletişim teknolojilerine dayanarak fiziksel ve sanal nesneleri birbirine bağlayan ve böylece bilgi toplumu için gelişmiş hizmetleri mümkün kılan küresel bir altyapıdır" [4].

Al-Fuqaha vd. ise "Nesnelerin İnterneti, fiziksel nesnelerin bir işi yapabilmek amacıyla görüp, duyabilmesini, düşünebilmesini, bilgi paylaşmak ve karar vermek için birbiriyle konuşabilmesini mümkün kılar" şeklinde bir tanım yapmıştır [5]. Ayrıca nesnelerin internetinin hesaplama, gömülü sistem, haberleşme teknolojileri ve duyarga ağlarından faydalanarak nesneleri "akıllı nesnelere" dönüştürdüğünü belirtmişlerdir.

Açıkça ifade etmek gerekirse nesnelerin interneti, duyarga ve eyleyici düğümleri, mobil cihazlar, gömülü ve/veya giyilebilen cihazlar, kısacası işlemci ve haberleşme birimine sahip nesnelerin oluşturduğu ve internete bağlanabildikleri bir ağdır. Nesneler iletişim kanalları üzerinden belirli protokollerle birbiriyle veya bir üst ortamla haberleşebilir ve internete erişebilir. Nesneler üzerinden gelen veriler bulut ortamında toplanabilir ve bilgisayar sistemleriyle çeşitli uygulamalar için kullanılabilir.

2.1 Nesnelerin İnterneti Mimarisi

Nesneler, ağ geçidi, bilgisayar sistemleri, bulut, büyük veri ve nesnelerin interneti uygulamalarından oluşan IoT genel mimarisi Şekil 2.1'de gösterilmiştir. Nesnelerin interneti ağı, düğümler şeklinde ilerleyen bir yapıdadır. İlk aşamada işlemciler sonraki



Şekil 2.1 : Nesnelerin interneti mimarisi genel gösterimi : (a)nesneler, (b) ağ geçidi, (c) bilgisayar sistemleri, bulut ve büyük veri, (d) nesnelerin interneti uygulamaları.

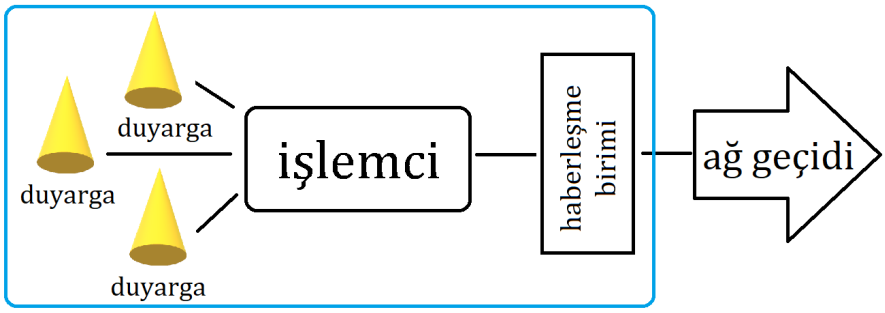
aşamalarda ise ağ geçitleri ve bulut, düğüm noktaları olarak sayılabilir. Bu bölümde IoT mimarisinin elemanları ve IoT kapsamında gündeme gelen uygulamalardan bahsedilecektir.

2.1.1 Nesneler

IoT ağı, duyarga ve eyleyici düğümleri, mobil cihazlar, gömülü ve/veya giyilebilen cihazlar gibi nesnelerden oluşur. Duyarga, tasarımı gereği belirlenen sıcaklık, konum, hareket, titreşim, mesafe, hız, nem, koku, basınç gibi bilgilerin toplayarak ağa iletilmesini sağlayan cihazlara verilen addır. Eyleyici ise basitçe anlatmak gerekirse duyarganın tersi yönünde çalışan gelen komutla fiziksel bir iş yapan cihazdır. Bu cihazları kullanarak sistemden verilerin toplanabilmesi ya da sisteme komut verilebilmesi için her bir nesnede bir işlemci ve haberleşme birimi bulunması gerekir.

Duyargadan işlemciye gelen sinyaller, burada ihtiyaca göre çeşitli işlemlerden geçirilir. Anlamlı hale getirilen veri, belirlenecek haberleşme protokolüne göre bir haberleşme cihazı kullanılarak internet ortamına iletilir. Eyleyicide ise haberleşme cihazından işlemciye gelen komutla bu cihaz harekete geçirilir.

Duyarga ve eyleyici gibi cihazlar GPIO, UART, SPI, I2C gibi seri haberleşme protokolleriyle işlemciye bağlanır. İşlemciler ise kablolu ya da kablosuz haberleşme yöntemleriyle internete erişir. Kablosuz haberleşmede BLE, Zigbee, NFC, Wi-Fi, 6LowPan gibi çeşitli kablosuz haberleşme protokolleri kullanılır [6–11]. Bu şekilde bir üst düğüm noktası olan ağ geçidine ya da doğrudan internete erişim sağlanabilir. Duyarga düğümü şeklindeki nesne yapısı Şekil 2.2'de gösterilmiştir.



Şekil 2.2 : Duyarga düğümünün genel yapısı.

2.1.2 Ağ geçidi

Ağ geçidi, nesnelerin TCP/IP protokolleri kullanılarak internete erişiminin sağlandığı bir katmandır. Nesneler ve ağ geçidinden oluşan ve internete açılan yapıda ağ cihazları da düğüm noktaları konumundadır.

Ağdaki nesneler farklı işlemci, bellek ve batarya özelliklerine sahip olabilirler. Bu nedenle düğümler şeklinde oluşan ve ilerleyen yapı, büyük bir önem arz etmektedir. Örneğin; erişim mesafesi arttıkça harcanan enerjinin de artması küçük bataryaya sahip nesneler için bir sorun haline gelecektir. Ya da her bir nesne üzerinde ağ protokollerinin gerçeklenmesi, düşük belleğe sahip nesneler için bir soruna dönüşecektir. Bunun dışında her bir nesnenin bulut ya da büyük veriye tek tek erişmesi işlem yoğunluğuna neden olacağı için yaşanacak gecikme, zamanın kritik olduğu IoT uygulamalarında probleme neden olacaktır [12]. Ağ geçidi ve düğümler mekanizmasının kullanımıyla bu sorunların önüne geçilebilmektedir.

Düğüm mekanizmasında ihtiyaca göre ön işlemler ağ geçidi ya da duyarga düğümü üzerinde gerçekleştirilebilir. Uzun vadede çalışan uygulamalar için bu ön işlemler ağ geçidinde olabilirken, kısa vadede gerçek zamanlı olarak çalışması gereken uygulamalarda duyarga düğümü üzerinde bu işlemlerin gerçeklenmesi beklenir.

2.1.3 Bilgisayar sistemleri bulut ve büyük veri

Büyük sayıdaki nesnenin ve cihazın IoT kapsamında bir araya gelmesiyle büyük miktarda verinin toplanması, iletilmesi ve işlenmesi ihtiyacı doğmaktadır. Bu nedenle "büyük veri" sunucuları kullanılarak bu veri muhafaza edilebilir. Bilindiği gibi büyük veri karmaşık hesaplamalara ve bilgi çıkarma işlerine ihtiyaç duyar.

Ancak alışılagelmiş cihaz ve yazılımlarla bu büyük miktardaki verinin işlenmesi ve kullanılması oldukça zordur. Bu aşamada bulut hizmetleri ortaya çıkmaktadır [5].

NIST tarafından ortaya atılan ve geliştirilen bulut bilişimi teknolojisi kullanıcıların uygulama, hizmet, depolama sunucuları gibi işlem ve hesaplama kaynaklarına istediği zaman erişilebildiği bir ağ modelidir. IoT için büyük verinin işlenmesi ve saklanmasında en iyi yolun bulut bilişimi olduğu söylenebilir [13, 14].

Nesnelerden elde edilen veriler bulut bilimle işlenebileceği gibi bazı bilgisayar sistemlerinde de doğrudan kullanılabilir. Ancak kullanılacak veri miktarının ve işlem yükünün nispeten daha az olması beklenir.

2.1.4 Nesnelerin interneti uygulamaları

Bulut bilişimi ve büyük veri hizmetlerinin de gelişmesiyle nesnelerden elde edilen verilerin kullanımı kolaylaşmış, hayatımızı kolaylaştıracak ve birçok gelişimin önünü açacak uygulamalar ortaya çıkmıştır. Aşağıda IoT uygulama alanları ve bu kapsamdaki ortaya çıkan örnek uygulamalar verilmiştir.

Doğal afet tahmini: Duyarga ve duyargaların otonom olarak kullanılmasıyla oluşturulacak simülasyonlar sayesinde doğal afet tahmini yapılmasını ve karşı önlemler alınmasını içeren uygulama alanıdır [15].

Endüstriyel uygulamalar: IoT uygulamaları endüstride de karşılık bulmaktadır. Endüstri 4.0 kapsamında akıllı fabrika uygulamalarıyla fiziksel işlemlerin izlenmesi, sanallaştırılarak simülasyonlarının yapılabilmesi ve merkezi olmayan kararların alınabilmesi hedeflenmektedir. Ayrıca siber fiziksel sistemlerin birbirleriyle ve insanlarla gerçek zamanlı olarak işbirliği içinde çalışabilmesi planlanmaktadır [16]. Böylelikle hızlı ve kaliteli üretimin sağlanması beklenmektedir.

Su ve kuraklık durumu izleme: Farklı yerlerdeki su ve kuraklık durumunun izlenebilmesi için yapılabilecek bir uygulamadır. Uygulama kapsamında sadece uzun vadeli izleme değil, ters akıntı ya da taşkın durumlarında kazaların engellenmesi için gerçek zamanlı olarak alarm verilmesi de sağlanabilir [15].

Akıllı ev uygulamaları: Aydınlatama, ısıtma, soğutma, enerji tüketim yönetimi, ev güvenliği, acil durum tespiti gibi konularda uygulamalarda IoT kullanılabilir [5, 17].

Sağlık uygulamaları: Kişilerin sağlık ve aktivite durumlarının gözlenmesi, bağımsız yaşayabilmesi için destek sağlanması, ortam destekli yaşam, ilaç kullanımının izlenebilmesi gibi uygulamalar bu başlık altında toplanmaktadır [5, 18].

Tarım uygulamaları: Farklı duyargaların yardımıyla verilerin alınıp işlenmesi ve çiftçiyi bilgi sağlaması amacıyla yapılan uygulamalardır. Tohumlama, gübreleme, sulama durumlarının kontrolü, özel bakım isteyen toprakla ilgili bilgilendirme gibi işler bu alandaki uygulamalara örnektir. Bu sayede tarımsal üretim ve kalite artacaktır [5].

Akıllı ulaşım sistemleri: IoT kapsamında trafik ve taşımacılıkla ilgili kontrol ve yönetim uygulamaları gerçekleştirilebilir. Trafik kurallarına uyulmasını sağlamak ve kurallara aykırı durumların tespiti, kaza ve güvenlik durumlarında acil işlemlerin yapılabilmesi ve trafik sıkışıklığının önlenmesi amacıyla çeşitli uygulamalar gerçekleştirilebilir [15].

Akıllı şehirler: Akıllı şehirlerin tasarımında IoT'den faydalanılabilir. Hava kalitesinin izlenmesi, acil durumların tespiti, etkili ışıklandırma, bahçe sulaması ile ilgili çalışmalar yapılabilir. Yine, çöp ve atıkla ilgili yönetim, akıllı park sistemleri, ortak alanların temizliği ve şehir enerji yönetimi bu başlık altında yapılabilir [15, 19].

Bunların dışında yeni alışveriş uygulamaları, akıllı ölçüm ve izleme sistemleri, akıllı güvenlik sistemleri, alarm sistemleri, gıdaların muhafaza, dağıtım ve tüketimiyle ilgili uygulamalar IoT kapsamında gerçekleştirilmektedir.

2.2 Nesnelerin İnternetinin İhtiyaçları ve Karşılaştığı Kısıtlar

IoT uygulamaları Bölüm 2.1.4'te de anlatıldığı üzere hayatın her köşesinde karşımıza çıkmaktadır. IoT kapsamında uygulamalar ve uygulama alanları çeşitlilik gösterdikçe ağa bağlı nesneler de çeşitlilik göstermeye başlamıştır. Yapılan çalışmalar internete bağlanan nesnelerin sayısının 2020 yılında 50 milyara ulaşacağını ifade etmektedir [20]. Ayrıca bu nesnelerin birbiriyle ilişkisi de giderek artacaktır. Farklı miktarlarda kaynaklara sahip nesneler IoT ağının parçası olabilseler de genel kanı IoT'nin büyük bir parçasının kaynağı kısıtlı düğümlerden oluşacağı yönündedir. Sürekli gelişmekte olan bu ağda nesnelerin kaynaklar açısından oldukça kısıtlı olması gelişimin hızını

önemli ölçüde kesen bir problemdir. Bu bölümde IoT'de ortaya çıkan kısıtlar ve IoT'nin ihtiyaçları anlatılacaktır.

2.2.1 Bellek kısıtı

Nesnelerin interneti kapsamında alt uç cihazların özellikle bellek konusunda kısıtlı olduğu bilinmektedir. Bilgisayar ve mobil cihazlarda kullanılabilecek bellek kapasitesi gigabayt hatta terabayt seviyelerindeyken alt uç cihazlarda bu bellek kapasitesi kilobaytlar seviyesindedir. Bu sınırlama hem kısa süreli (RAM) hem de uzun süreli bellek için geçerlidir (ROM) [1,21]. Bu kısıtlamaya karşın IoT'nin ihtiyaç ve işlerliğini karşılayacak uygulamaların yazılabilmesi beklenmektedir.

2.2.2 Enerji tüketimi

Nesnelerin enerji ihtiyacı birçok durumda bataryalarla karşılanmaktadır ve bu nesnelerin batarya değişimi yapılmadan uzun yıllar hizmet vermesi beklenmektedir. Ayrıca çok fazla nesnenin ortamda olması nedeniyle enerji kaynakları kısıtlı kalabilmektedir. Bu yüzden düşük güç tüketimli ve enerji etkin tasarımların yapılması ihtiyacı doğmaktadır [1, 22]. Bu kısıta karşılık hem donanım hem yazılım seviyesinde çözümler sunulmaktadır.

2.2.3 İşlemci ihtiyacı

Her bir nesnenin bir işlemciye ihtiyacı vardır böylece algı cihazlarından gelen ham veri anlamlandırılarak üst katmana iletilebilir. Haberleşme protokolleri ve veri ön işlemeleri bu işlemci üzerinde gerçekleştirilir. Nesnenin ve uygulamanın ihtiyacına göre donanımsal (hardcore) ya da yazılımsal (softcore) işlemciler seçilebilir.

2.2.4 Haberleşme arayüzü

IoT cihazlarının ortak noktalarından biri de kendi aralarında ya da internet ortamıyla haberleşmeleridir. Bu yüzden nesnelerin haberleşme arayüzüne sahip olması beklenir. Haberleşme teknikleri sadece düşük güç tüketimli radyo haberleşme teknolojilerini (ZigBee, BLE, LoRaWAN, Wi-Fi) değil aynı zamanda kablolu haberleşme (Ethernet, bus sistemleri vs.) teknolojilerini de kullanabilir [23, 24].

Kablosuz duyarga ağlarından (wireless sensor network - WSN) farklı olarak IoT'de nesnelerinin doğrudan internete erişmesi de beklenebilir [25]. Nesnelere erişim için nesnelerin birer kimliğe ve adrese sahip olmaları beklenir. Bu amaçla nesneler doğrudan kendi IP adresiyle internete erişebilir ya da ağ geçitleri sayesinde ortak bir adres üzerinden erişimi sağlayabilir.

2.2.5 Ölçeklenebilirlik

Ölçeklenebilirlik, var olan servislerin ve uygulamaların kalite ve işleyişini bozmadan sisteme yeni cihazların ve servislerin eklenebilmesini ifade eder. Birçok farklı donanım platformu ve haberleşme protokollerinin mevcut olması bu ekleme ve geliştirmeleri zorlaştırmaktadır. Bu nedenle IoT uygulamaları en başından itibaren bu özellik göz önünde bulundurularak tasarlanmalıdır.

2.2.6 Farklı yapıda olma

Nesnelerin interneti kapsamında ortaya çıkacak, farklı yapıdaki cihazları içeren (heterojen) ağ birçok zorluğu beraberinde getirmektedir. Bunların başında ortak standartların ve mimarilerin belirlenememesi gelmektedir. Yine bu ağa bağlı birçok farklı donanımın yazılım tarafında da desteklenmesi ihtiyacı doğmaktadır. Bu ihtiyacı karşılamak için IoT odaklı işletim sistemleri, yazılım ve derleyiciler ortaya çıkmaktadır. Heterojenliğin ortadan kaldırılması için sistemlerin platformdan bağımsız olması gerekmektedir. Bu yüzden ölçeklenebilirlik ve heterojenliğin birlikte incelenmesi gerekmektedir [26].

2.2.7 Gerçek zaman kabiliyeti

Gerçek zamanlı sistemler (Real Time Systems - RTS), belirli zaman kısıtlarında çıkış vermesi beklenen sistemlerdir [27]. RTS'ler zorunlu ve hafif kısıtlı gerçek zamanlı sistemler olmak üzere ikiye ayrılır. Zorunlu RTS, için süre kısıtı kesindir ve doğru zamanda gerçeklenemeyen bir davranış sistemde hatalara neden olur ve buna bağlı olarak ciddi zararlara neden olabilir. Hafif kısıtlı sistemlerde bu durum hatalara neden olmaz bunun yerine performans düşümü gözlenir.

IoT kapsamında belli zaman kısıtları altında tepki gerektiren uygulamalar yapılmaktadır. Örneğin izleme ve kontrol sağlayan akıllı sağlık uygulamalarında gerçek zamanlı

çalışma oldukça kritiktir [1, 28]. Bunun dışında günlük bakımların yapıldığı tarım uygulamalarında, akıllı trafik sistemlerinde ve yangın alarmı gibi acil durumlarla ilgili uygulamalarda da gerçek zamanlı çalışabilirlik beklenmektedir. Bu yüzden kullanılacak yazılımların ve işletim sistemlerinin gerçek zaman kabiliyetine sahip olması gerekmektedir.

2.2.8 Güvenlik ve gizlilik

IoT cihazlarının internete bağlı oluşu güvenlik ve gizlilikle (mahremiyetin korunması) ilgili büyük ihtiyaçlar doğurmaktadır. IoT'de ortak bir standart ve mimari olmadığı için güvenlik ve gizlilikle ilgili sorunların çözümü zorlaşmaktadır. Ağın heterojenliği, bellek ve enerji kaynaklarından kısıtlı oluşu nedeniyle çözüm üretmek de kolay değildir [5].

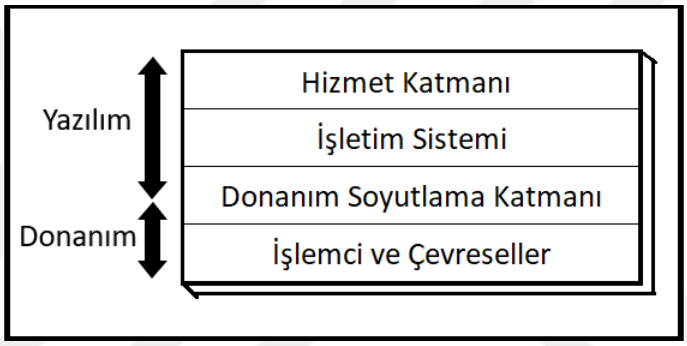
2.2.9 Güvenilirlik

Güvenilirlik, sistemin belirlenen özelliklere göre doğru çalışması [29], verinin sağlıklı ve doğru şekilde iletildiğinden emin olunmasıdır. Güvenilirliğiin sağlanması için bilgi ve servislere zamanında erişim de oldukça önemlidir. Güvenilirlik her uygulama için gerekli olsa da acil durum uygulamalarında daha da kritik bir hale geldiği söylenebilir [30].

Güvenilir bir sistem, verilerin algılanması, işlenmesi ve iletilmesinde ortaya çıkacak hataların azaltılmasını, hataların ortaya çıktığı durumlarda ise bu hataların giderilmesini sağlar. Verimli bir IoT ağında veri kayıpları engellenmeli böylece karar sürecinin uzaması ve hatalı sonuçların üretilmesi önlenmelidir. Bu amaçla güvenilirlik hem donanım hem de yazılım tarafında sağlanmalıdır.

3. NESNELERİN İNTERNETİNDE UYGULAMAYA YÖNELİK TASARIM

IoT kapsamında birçok uygulama yapılabilir ancak her uygulama ve her hizmet farklı ihtiyaçları da beraberinde getirmektedir. Bu nedenle IoT kapsamında bir tasarımın yapılabilmesi için öncelikle tasarım modelinin belirlenmesi uygun olacaktır. Tez çalışmasında izlenecek tasarım modeli Şekil 3.1'de verilmiştir [1]. Bu modele göre sistemin ihtiyaçları yukarıdan aşağı belirlenmeli ve tasarım en alt uçtan başlayarak yukarı doğru gerçeklenmelidir.



Şekil 3.1 : Nesne tasarım modeli.

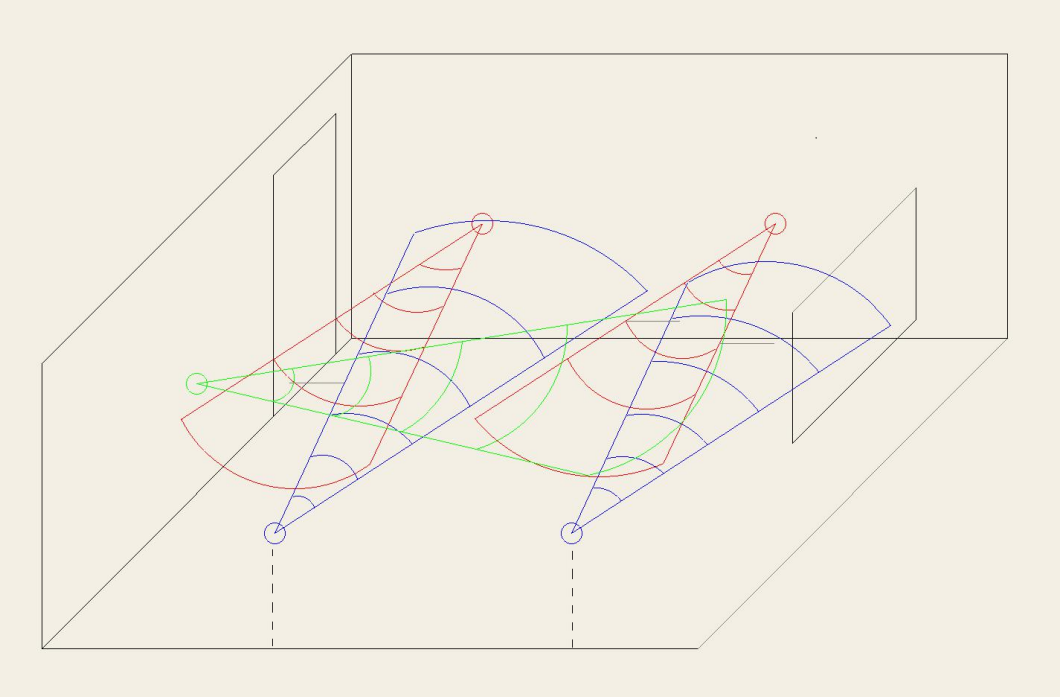
Bu bölümde uygulamaya yönelik tasarımda kullanılacak model detaylandırılarak modelin katmanları ve bu katmanlarda kullanılacak elemanlarla ilgili bilgiler verilmiştir.

3.1 Hizmet Katmanı: Acil Durum Tahliye Sisteminde Kişilerin Bina İçindeki Varlığının Tespiti

Verilen modele göre en üst katman gerçekleştirilecek hizmetin seçilmesini içermektedir. Tez kapsamında, akıllı ev uygulamalarında kullanılmak üzere acil durum tahliye sistemi çıkış noktası olarak belirlenmiştir. Acil durum tahliye sistemi, akıllı ev uygulamalarında kullanılan bir hizmettir. Bu sistem alarm durumunun izlenmesi, alarmın verilmesi, bina kat planı, konum belirlenmesi, kişi varlığının belirlenmesi gibi alt hizmetleri içermektedir [31]. Tez kapsamında yangın alarmı durumunda odalarda kişilerin var olup olmadığını bildiren alt hizmetin gerçekleştirilmesine karar verilmiştir. Buradan sağlanacak bilgi hem tahliye rotasının belirlenmesinde hem de

acil durum müdahale ekiplerinin bilgilendirilmesinde kullanılabilecektir [32]. Kişi varlığının tespiti için odalara ultrasonik duyargaların yerleştirilmesi planlanmıştır. Ultrasonik duyargalarla mesafe ölçümü yapılacak böylece odada birilerinin olup olmadığı tespit edilebilecektir.

Şekil 3.2'de ultrasonik duyargaların oda duvarlarına yerleştirildiği bir model verilmiştir. Kullanılan ultrasonik duyargalar 30 derecelik etkin tarama alanına sahiptir ve 2-400 cm mesafede ölçüm yapabilmektedir [33]. Şekil 3.2'de gösterilen odanın en, boy ve yüksekliği sırasıyla 600, 566 ve 260 cm'dir. Görüldüğü gibi bu odada tüm alanın taranabilmesi için birden fazla duyargaya ihtiyaç vardır. Ayrıca doğruluğun arttırılabilmesi amacıyla duyarga sayısının arttırılması gerekmektedir. Farklı yükseklik ve açılarda duyargaların yerleştirilmesi mümkündür. Odanın boyutu arttıkça yerleştirilmesi gereken duyarga sayısı da artacaktır. Bu açıdan bakıldığında bütün bir binanın sisteme dahil edilebilmesi için çok fazla sayıda duyarga kullanılması gerektiği aşikardır.



Şekil 3.2 : Duvarlara duyargaların yerleştirildiği bir oda modeli.

Enerji etkin tasarımın yapılması bu sistemin temel ihtiyaçlarından biridir [1, 22]. Çok fazla sayıda duyarganın bulunduğu bu sistemde her bir duyarga için kablo alt yapısıyla güç sağlamak oldukça zor olacaktır. Bu yüzden duyargaların batarya ile kullanılması daha uygundur. Nesnelerin batarya değiştirmeden uzun yıllar çalışması

beklenmektedir. Bu amaçla kullanılacak cihazların güç tüketimi karşılaştırması yapılmalı, düşük güç modlarının olup olmadığına dikkat edilmeli ve buna göre seçimleri yapılmalıdır. Ayrıca kullanılacak duyarga sayısının ihtiyaca göre eklenip çıkarılabildiği esnek bir yapının olması faydalı olacaktır. Bunun dışında haberleşme cihaz ve protokolleri de düşük güç tüketimi göz önünde bulundurularak seçilmelidir. Yazılım tarafında da enerji etkinliği göz önünde bulundurulmalıdır. Derleyicilerde optimizasyon seçenekleri bu noktada etkilidir [34, 35]. Yine nesnelerde işletim sistemlerinin gerçek zamanlı işletim sistemlerinden seçilmesi enerji etkinliğinde önem arz etmektedir [36].

Nesnelerin interneti sürekli gelişen bir yapıdadır bu yüzden yapılacak tasarımların esnek olabilmesi, sistemin kolay bir şekilde yapılandırılabilir ve programlanabilir olması önemlidir. Bu noktada açık kaynak kodlu işletim sistemlerinin ve açık kaynak kodlu yazılımsal işlemcilerin kullanılması bir çözüm olarak sunulmaktadır [37–39]. Yazılımın tüm kaynak kodlarının detaylı bir şekilde görülebilmesi ve ihtiyaca göre değiştirilebilir oluşu tasarımcılara bu kolaylıkları sağlamaktadır. Açık kaynak kodlar kullanım amacına göre lisanslandırılabilir. Örneğin, bu kodlar eğitim amaçlı kullanımlarda ücretsiz olup ticari amaçlı kullanımlarda ücretli olacak şekilde lisanslandırılabilir.

Yine haberleşme ve enerji kablosu problemi göz önünde bulundurularak düşük güç tüketimli kablosuz haberleşme arayüzünün kullanılması sonucuna ulaşılmaktadır.

Yangın tahliye sistemlerinde tahliye süresi, alarmın geldiği anla güvenli bölgeye çıkışın tamamlandığı an arasında geçen süredir [40, 41]. Bu sürenin kısa tutulması güvenlik açısından oldukça önemlidir. Bunun doğrultusunda gerçek zamanlı yangın tespiti ve tahliye sistemleri ile ilgili birçok çalışma yapılmıştır [42–45]. Yangın tahliye sisteminde alarm gelmesi durumunda sisteme bir kesme (interrupt) verilerek sistemde belirlenen hizmetler, kısıtlı zaman içerisinde çalıştırılmalıdır. Bu yüzden kullanılacak yazılımların ve işletim sistemlerinin gerçek zaman kabiliyetine sahip olması gerekmektedir.

Servisin bu ihtiyaçları doğrultusunda açık kaynak kodlu işlemci ve işletim sistemleri kullanılarak nesnelerin interneti için enerji etkin duyarga tasarımının yapılmasına ve FPGA üzerinde gerçeklenmesine karar verilmiştir.

3.2 İşletim Sistemi Katmanı

Şekil 3.1'de verilen modele göre bir sonraki katman işletim sistemi katmanıdır. IoT'nin güvenlik, kurulum maliyeti, esnek yapı ihtiyacı gibi ihtiyaçlarının olması, ayrıca tasarlanan nesne ile birçok duyarganın yönetilecek olması işletim sistemi ihtiyacını doğurmaktadır [39]. Bir duyarga verisinin işlendiği yere iletilmesinde geçen süreyi göz önüne alırsak, işletim sisteminin olmadığı durumda bu iletilme süresi duyarga sayısıyla orantılı olarak artacaktır. Ancak işletim sistemi kullanılarsa hem veri duyarga üzerinde ön işlemlerden geçirilebilecek hem de verinin iletilmesi de belli bir organizasyona göre yapılabilecektir. İşletim sistemi sayesinde hizmet için gerekli bu yazılım kolay bir şekilde gerçeklenebilecektir.

İşletim sistemlerinin incelenmesinde sistemin genel mimarisi, zamanlama modeli, bellek paylaştırma, programlama modeli gibi teknik özelliklere bakılmaktadır.

Genel mimari, sistemin monolitik (monolithic) ya da mikroçekirdek (mikrokernel) yapıda olmasıyla ilgilidir. Mikroçekirdek yaklaşım, çekirdeğin daha fonksiyonel olmasını bunu yaparken de sistemin geri kalanında çok az bellek kullanmayı, daha fazla alan ve esneklik sağlamayı ve olabildiğince kararlı bir yapı sunmayı hedefler. Burada kararlı yapıdan kasıt, sistemde oluşabilecek bir hatanın tüm sistemi etkilemediği durumdur. Monolitik yaklaşımda ise sistemin bütün elemanları bir bütün olarak geliştirilir. Bu elemanlar birleştirilerek tek bir sistem olacak şekilde derlenir [1,46]. Çalışma sırasında bütün işlemler çekirdekle aynı bellek ve öncelikleri paylaşır. Bu mimari basit ve verimli bir tasarım sağlar ancak eleman sayısı arttıkça karmaşıklık artmaktadır.

İş zamanlama modeli (scheduling model), işletim sisteminin enerji etkinliği, gerçek zamanlı oluşu ve programlama modeli gibi önemli özelliklerini de etkileyen bir özelliktir. Geçişli (preemptive) ve geçişsiz yani işbirlikçi (non-preemptive, cooperative) zamanlayıcılar olmak üzere iki tip model mevcuttur. Geçişli zamanlayıcılar çekirdek olmayan herhangi bir görevi herhangi bir zamanda kesebilir ve başka bir görevin belirli bir sürede çalışıp tamamlanmasını sağlayabilir. Yani bu model her bir görev için işlemci süresini belirler. İşbirlikçi yapıda ise hiç bir görev hatta çekirdek bile diğer görevleri kesemediği için her bir iş parçacığının kendini tamamlaması gerekir. [1].

Bellek paylaştırma, kısıtlı belleğin kullanımı için gerekli bir özelliktir. Bellek statik ya da dinamik paylaştırması yapılabilir. Statik paylaşımda program çalışmaya başlamadan önce gereken bellek ayrılır ve program tamamlanana kadar bellek kullanımı devam eder. Statik bellek paylaşımı bazı durumlarda fazladan bellek tahsisi (over-provisioning) gerektirir. Çalışma sırasında ihtiyaçların değiştirilmesi gerektiği durumlarda fazla esneklik sunmaz. Dinamik bellek paylaşımında ise programın çalıştığı sırada ihtiyaca göre bellek kullanımı arttırılıp azaltılabilir. Bu bellek paylaşımıyla sistem tasarımı statik bellek paylaşımındakinden daha karmaşıktır. Ayrıca "malloc()" gibi dinamik bellek fonksiyonlarının gerçek zaman kabiliyetini olumsuz etkileyebilmesi nedeniyle gerçek zamanlı işlemler için TLSF gibi özel uygulamalara ihtiyaç vardır [1, 47].

Programlama modeli, olaya dayalı sistemler (event-driven) ve çok iş parçacıklı (multithreading) sistemler için ikiye ayrılabilir. Olaya dayalı sistem daha çok WSN işletim sistemlerinde kullanılmaktadır. Bu modelde her bir görev kesme gibi dış olaylarla (event) tetiklenir. Çok iş parçacıklı modelde her görev kendi iş parçacığı grubuyla çalışır ve farklı görevler arasındaki iletişim, uygulama programlama arayüzü (Application Programming Interface - API) ile sağlanır.

Yukarıda açıklanan özellikler temel alınarak açık kaynak kodlu işletim sistemlerinden Linux kernel, Contiki, FreeRTOS ve RIOT tez kapsamında incelenmiştir [48–51].

3.2.1 Linux kernel işletim sistemi

1991 yılında Linus Torvalds tarafından geliştirilen Linux kernel UNIX benzeri bir işletim sistemi çekirdeğidir [48]. Linux ailesi işletim sistemleri bu çekirdek üzerine inşa edilir ve çeşitli işlemcilerle kullanılmak için derlenebilir. Monolitik bir mimariye sahiptir. Linux kernel güçlü bir işletim sistemidir ancak MB seviyelerinde yüksek RAM ve ROM belleğe ihtiyaç duymaktadır. Ayrıca üzerinde birçok görev (task) tanımlı olması enerji tüketimini önemli oranda artırmaktadır çünkü görevlerin tamamlanma süresi uzamaktadır [52, 53].

3.2.2 Contiki işletim sistemi

Adam Dunkels tarafından 2003'te kısıtlı kaynaklara sahip gömülü sistemler için geliştirilen Contiki popüler açık kaynak işletim sistemlerinden biridir oldukça fazla

cihazla kullanılabilmektedir [49]. Başta WSN için bellek olarak çok kısıtlı 8-bit işlemciler için çıkmıştır daha sonra 16-bit ve 32-bit işlemciler için de kullanılmaya başlanmıştır. Monolitik mimariye sahiptir. Contiki'nin kullandığı zamanlama modeli işbirlikçi zamanlama (cooperative thread scheduling) modelidir. Gerçek zaman desteği kısmen mevcuttur. Contiki düşük güç tüketimli internet haberleşmesi sağlar. IPv6, 6LoWPAN, RPL, CoAP internet standartlarını destekler [54–57]. İnternet üzerinde kendi geliştirici topluluğuyla geliştiriciler birbirine destek sağlamaktadır. Minimum bellek ihtiyacı 2 kB RAM ve 30 kB ROM bellek kadardır. Contiki temel olarak C programlama dilini bazı kısıtlamalar dahilinde desteklemektedir. Contiki işletim sisteminde olaya dayalı programlama yapılmaktadır. Olaya dayalı tasarım tipik WSN uygulamalarında kullanışlıdır ancak verimli ve fonksiyonel ağ gerçeklemelerini yapmak konusunda dezavantajlıdır [1, 36, 58].

Kısıtlı kaynaklara sahip nesnelerde Linux gibi geleneksel işletim sistemlerinin kullanılması mümkün değildir diğer yandan da ağın çeşitli ihtiyaçlarını Contiki gibi WSN'e özgü hafif işletim sistemleri (lightweight OS) karşılayamamaktadır. Bu yüzden de gereksiz geliştirme ve bakım maliyetlerinden de kaçınacak yeni bir işletim sistemi gereklidir. Bu noktada gerçek zamanlı IoT işletim sistemleri karşımıza çıkmaktadır [36].

Gerçek zamanlı işletim sistemleri (Real Time Operating System - RTOS), gömülü sistem cihazlar üzerinde gerçek zamanlı olarak çalışan, görev, zaman, bellek yönetimi, sistemin giriş çıkış denetimi, görevler arası iletişim ve senkronizasyonu gibi hizmetleri sağlayan işletim sistemleridir [39]. RTOS'lar güvenilir, iyi performanslı, düşük alan kaplayan ve ölçeklenebilir yapıdadır. Çoklu görevleri (multitasking) gerçekleştirebilir oluşu RTOS için temel özelliklerdendir. Çoklu görev özelliği bir zamanlama algoritmasıdır bu algoritmayla işlerin hangi sırayla yapılacağı belirlenir. RTOS'ta her görev için bir öncelik tanımlanabilir. Zamanlama planı yapılırken en yüksek önceliğe sahip olan iş en önce gerçekleşir. FreeRTOS ve RIOT OS bu işletim sistemlerine örnektir.

3.2.3 RIOT işletim sistemi

RIOT OS 2013 yılında duyurulmuştur. Bu işletim sistemi, gerçek zaman ihtiyaçlı WSN senaryoları için tasarlanmış FireKernel'in mikroçekirdek mimarisi baz alınarak

geliştirilmiştir [59]. Minimum bellek ihtiyacı 1.5 kB RAM ve 5 kB ROM bellek kadardır. C ve C++ dillerini destekler. Moduler mikroçekirdek yapıda oluşu tek bir elemandaki arızalara karşı sistemi dayanıklı kılar. RIOT çok iş parçacıklı programlama modelini kullanmaktadır. Geliştiricilerin birden fazla iş parçacığı oluşturmasına olanak verir ve dağınık (distributed) sistemler çekirdek mesaj arayüzü kullanılarak kolayca gerçeklenebilir. Oluşturulabilecek iş parçacığı sayısı yalnızca mevcut bellek ve iş parçacıklarının yığıt boyutuyla sınırlıdır.

Gerçek zaman ihtiyaçlarının yerine getirilebilmesi için RIOT, çekirdeği görevleri belirli sürelerde yapması için zorlar ve kısıtlar. Ayrıca gerçek zamanı garanti edebilmek için çekirdeğe özel statik bellek paylaşımı kullanılır. Uygulama tarafında ise dinamik bellek paylaşımı sağlar.

Uyku modu süresinin uzatılması IoT cihazlarının enerji etkin olması için önemlidir. Bu yüzden RIOT periyodik olaylardan bağımsız çalışan tiksiz (tickless) bir zamanlayıcı sunmaktadır. Bekleyen bir görev olmadığı sürece RIOT sistemin boşta olduğu "idle" iş parçacığına geçiş yapar. Böylece (çevresellere de bağlı olarak) mümkün olan en derin uyku moduna girer. Sistemi bu noktada sadece dışarıdan gelen ya da çekirdekte üretilen kesmeler uyandırabilir.

RIOT ağ bağlantısını desteklemektedir. 6LoWPAN, IPv6, RPL, UDP ve CoAP gibi standart IP protokollerini desteklemektedir.

RIOT kaynak kodu LGPL altında lisanslanmıştır ve GitHub üzerinden bu kodlara erişilebilir. Geliştirici topluluğu birbirine destek sağlamaktadır.

3.2.4 FreeRTOS işletim sistemi

FreeRTOS Richard Barry tarafından 2002 yılında geliştirilmiştir. Bedava ve açık kaynaklı bu işletim sisteminin MIT lisansı altında dağıtımı yapılmaktadır [51]. FreeRTOS küçük, basit, portatif ve kullanımı kolay olacak şekilde tasarlanmıştır. Bu sebeple bu işletim sistemi büyük bir topluluk tarafından desteklenir. 35'ten fazla mikrodenetleyici için destek sunmaktadır.

FreeRTOS birçok farklı mimari ve geliştirme aracı için basit ve bağımsız çözümler sunmaktadır. Minimum bellek ihtiyacı 6-12 kB aralığındadır. Oldukça basit bir mikroçekirdek mimarisine sahiptir ve yalnızca 3 adet C dosyasıyla bu çekirdek

gerçeklenir. Bu basit çekirdeğin sağladığı fonksiyonlar yalnızca,iş parçacığı yönetimi (thread handling), mutex, semaphore ve yazılımsal zamanlayıcılardır (software timer). Mutex ve semaphore iş parçacıkları arasında senkronizasyon ve sıralamayı sağlayan mekanizmalardır [60].

Standart ön ayarlarında FreeRTOS, geçişli, öncelik tabanlı Raund Robin zamanlama algoritmasını kullanır. Bu zamanlama modeli, periyodik saat kesmeleriyle tetiklenir. RIOT'ta da olduğu gibi FreeRTOS bu zamanlayıcının durdurulduğu tiksiz bir moda sahiptir, bu mod sistemin boşta olduğu durum süresince kesmelerin gelmesini önler. İsteğe bağlı olarak kullanılan bu modun kullanımıyla sistem derin güç tasarrufu durumuna geçer.

FreeRTOS kendi halinde ağ erişimi sağlayamaz ancak FreeRTOS ekosistemindeki çeşitli araç ve kütüphaneler eklenerek ağ erişimi sağlanabilir. Bunlardan en bilineni FreeRTOS+TCP'dir, bu eklenti Ethernet tabanlı IPv4 yığıtını ve TCP, UDP protokollerini desteklemektedir [1, 51].

FreeRTOS temel olarak C programlama dilindedir ancak kullanıcıların C++ dilinde uygulamalar gerçekleştirmesine de olanak sağlar. FreeRTOS için portatif sürücü modelleri ya da çevresel soyutlama arayüzleri yoktur. Bunun yerine sağlayıcı firmaların kart destek paketleriyle çalışır [1].

İncelenen işletim sistemleri için yapılan karşılaştırma Çizelge 3.1'de verilmiştir.

Çizelge 3.1 : Açık kaynak işletim sistemleri karşılaştırması.

	Linux kernel	Contiki	RIOT	FreeRTOS
Mimari	monolitik	monolitik	mikroçekirdek, RTOS	mikroçekirdek, RTOS
Zamanlama modeli	geçişli	işbirlikçi	geçişli, tiksiz	geçişli, isteğe bağlı tiksiz
Programlama modeli	çok iş parçacıklı	olaya dayalı	çok iş parçacıklı	çok iş parçacıklı
Desteklediği diller	C	C (kısmi)	C,C++	C
Gerçek zaman	kısmi destek	kısmi destek	var	var
Minimum RAM	1 MB	< 2 kB	1,5 kB	300 bayt
Minimum ROM	1 MB	< 30 kB	5 kB	5-10 kB

Bu incelemelerden sonra söylenebilir ki tasarlanan nesnede düşük alan kullanımı, düşük güç tüketimi ve gerçek zaman özelliği nedeniyle RIOT veya FreeRTOS işletim sistemlerinin kullanılması diğer işletim sistemlerine göre daha uygun olacaktır.

3.3 Donanım Soyutlama Katmanı

Donanım soyutlama katmanı, donanım ve yazılım arasındaki iletişimin sağlanması için yapılan uyarlama işlemlerini içeren katmandır. İyi bir donanım soyutlama katmanı işlemci, bellek ve kesme işleyicisi (interrupt handler) için iyi tanımlanmış bir arayüz sağlar.

3.4 İşlemci ve Çevreseller Katmanı

Tasarım modelinin en alt katmanında işlemciler ve çevreseller yer almaktadır. Tez kapsamında açık kaynak kodlu işlemciler incelenmiştir bu konunun daha iyi anlaşılabilmesi için bazı kavramların açıklanması gerekmektedir.

3.4.1 Donanım tasarımında temel kavramlar

FPGA (Field Programmable Gate Array - Alanda Programlanabilir Kapı Dizileri), programlanabilir mantık blokları, blok dizisini çevreleyen giriş-çıkış blokları ve bu bloklar arasındaki ara bağlantılardan oluşan sayısal tümleşik devrelerdir [61]. Paralel algoritmaların gerçeklemesinde kullanılabilen FPGA oldukça geniş bir uygulama alanına sahiptir. Bu devreler donanım tanımlama dilleri (Hardware description language - HDL) kullanılarak programlanabilir [62].

HDL ile yazılan kodlar sentezleyici adı verilen bir programla işlenir, bu program metin tabanlı kod dizisinden FPGA üzerindeki programlanabilir yapı için tanımlar içeren bir başka dosya üretir. Dosyanın FPGA'ye yüklenmesiyle gerçekleme tamamlanmış olur [61, 62].

ASIC (Application Spesific Integrated Circuit- Uygulamaya özel Tümleşik devre), belirli bir işlemi, görevi yerine getirmek üzere özel olarak tasarlanmış tümleşik devrelerdir. FPGA'e göre yapıları daha küçüktür ve daha az enerji tüketirler. Ancak üretim maliyeti ve üretim öncesi araştırma maliyetleri oldukça yüksektir. Bu yüzden ASIC üretilmeden önce prototiplerin gerçeklemesi amacıyla FPGA'ler kullanılır [63].

Sistemlerdeki karmaşıklık zaman geçtikçe artmaktadır. Bir sistemdeki her bir donanım elemanı için baştan tasarım yapmak hem kullanışsız hem de pahalı olacaktır. Bu sebeple önceden tanımlanmış ve optimize edilmiş uygulamaya özel donanım bloklarını (Intellectual Property – IP) tasarımda kullanmak faydalı olacaktır [37, 64]. Yazılımsal işlemciler de bu IP bloklarındandır.

3.4.2 Açık kaynak kodlu yazılımsal işlemciler

Yazılımsal işlemciler (Soft-core processors), mimarisi ve davranışı bir donanım tanımlama diliyle tanımlanmış, FPGA ya da ASIC üzerinde gerçeklenebilen ve ihtiyaca göre yapıları değiştirilebilen işlemcilerdir [37]. Yazılımsal işlemcilerin birçok avantajı vardır. Bunların ilki işlemcilerin esnek bir yapısının olması ve çok kolay bir şekilde uygulamaya uygun olarak özelleştirilmesidir. Bu özelleştirme, çevresellerin eklenip çıkarılabilmesi ya da doğrudan işleminin yeteneklerinin değiştirilmesi şeklinde olabilir. Yazılımsal işlemcilerin bir diğer avantajı da teknolojiden bağımsız oluşudur. Bu işlemciler herhangi bir FPGA ya da ASIC için sentezlenebileceği için kullanımı azalan teknolojilerle birlikte yok oluşu gibi bir durum söz konusu olamaz. Son olarak bu işlemcilerin yüksek soyutlama seviyelerinde (abstraction level) HDL olarak tanımlanması tasarımın anlaşılmasını kolaylaştıran bir avantaj sağlar.

IoT sistemlerinde düşük güç tüketimi ve performansın sağlanabilmesi için FPGA kullanımı uygun olmamakla beraber, prototip amaçlı kullanımı hem geliştirme sürecini hem de test sürecini hızlandıracaktır. Böylece üretim öncesi mühendislik maliyetin düşürülmesi sağlanacaktır. Esnek yapıdan faydalanarak tasarlanan sistem daha sonra ASIC haline dönüştürülebilir.

Tez kapsamında OpenRISC ve Leon3 açık kaynak kodlu yazılımsal işlemcileri incelenmiştir.

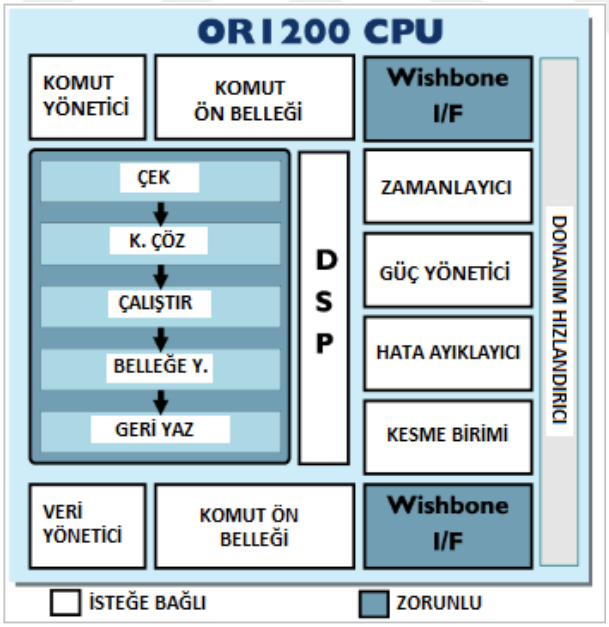
3.4.2.1 OpenRISC1200 (OR1200)

OpenRISC 1000, açık kaynak kodlu komut mimarileri geliştirmek üzere, indirgenmiş komut kümesi mimarisi RISC (Reduced Instruction Set Computing) prensipleriyle tasarlanan ve OpenCores.org'taki gönüllü geliştiricilerin katılımıyla yürütülen bir projedir [65]. Bu kapsamda yazılan kodlar tamamen açık ve ücretsizdir. Kullanıcılar bu tasarımı istediği gibi kullanabilir ve üzerinde değişiklik yapabilir.

OpenRISC 1200 sentezlenebilir bir 32-bit işlemci çekirdeğidir ve 2000 yılında Damjen Lampert tarafından OpenRISC 1000 mimarisiyle yapılan ilk gerçeklemedir [66]. Verilog RTL olarak gerçeklenmektedir. OR1200 işlemcisinin özellikleri kullanıcı tarafından değiştirilebilir. Yüksek performanslı işlem yapabilme yeteneğine sahip bu işlemci Wishbone arayüzüyle uyumludur [67]. Yüksek hızlarda çalışan bellek ve bellek yönetimine sahiptir. 32-bitlik komut kümesi 32-bit ve 64-bitlik verilerle çalışabilir. İşlemci Harvard mimarisine göre tasarlanmıştır.

Bu işlemci ayrıca CPU-DSP (sayısal sinyal işleme birimi-digital signal processing unit), komut ve veri önbelleği, güç-enerji yönetim birimi ve arayüzü, zamanlayıcı, kesme kontrol birimi ve arayüzü fonksiyonel özelliklerine sahiptir.

OR1200 işlemcisinin yapısı 3.3'te gösterilmektedir. Şekilde koyu renkte verilen bloklar işlemcinin en minimal halde çalışması için gereken temel kısımlardır. Beyaz ile verilen bloklar ise isteğe bağlı olarak işlemciye eklenebilen ya da çıkarılabilen kısımlardır. Kullanılacak uygulamadaki ihtiyaca göre bu kısımların çıkarılabilmesi, düşük güç tüketimi için fayda sağlamaktadır.



Şekil 3.3 : OR1200 işlemci yapısı [66].

3.4.2.2 LEON3

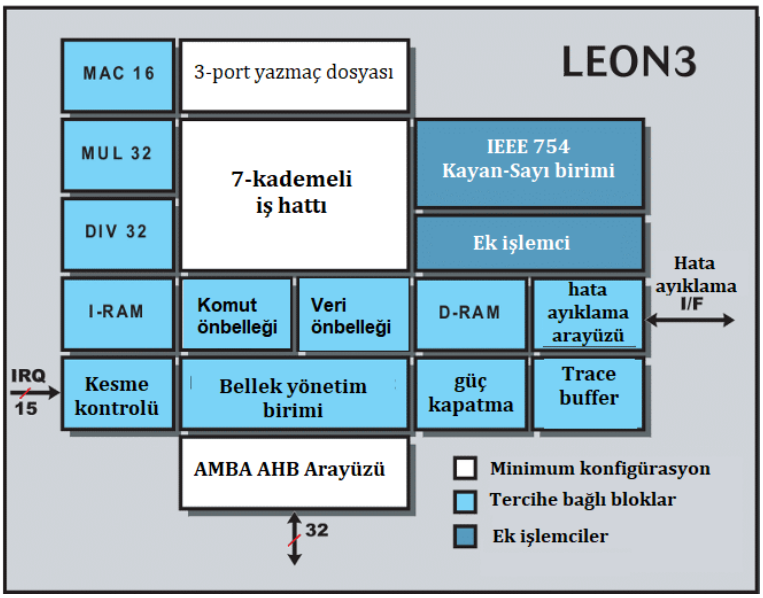
LEON3, Sparc V8 RISC mimarisi tabanlı, açık kaynak kodlu, 32-bitlik mikroişlemcidir ve Gaisler AB firması tarafından geliştirilmiştir. Çoklu işlemi destekleyen bu işlemci sentezlenebilir VHDL modelleriyle oluşturulmuştur. Büyük oranda

özelleştirilebilen bu işlemci modeli, özellikle kırmık üstü sistemlerde (System on Chip - SoC) kullanılmaya uygundur. LEON3 işlemcisinin eğitim ve araştırma amaçlı kullanımı GNU GPL lisansı altında dağıtılmakta olup ücretsizken ticari amaçlı kullanımı ücretlidir [68].

EDA (Electronic Design Automation- elektronik tasarım otomasyonu) araç ve tasarım akışına uygunluğu sayesinde karmaşık çok işlemcili sistemler kolaylıkla gerçekleştirilebilir ve bu sayede pazara çıkış süresi ve geliştirme maliyetleri büyük oranda azaltılabilmektedir.

LEON3 işlemcisi 7 kademeli iş hattından (pipeline) oluşmaktadır. İşlemci çekirdeği, dağınık komut belleği, önbellekler, yazmaçlar ve bunları kontrol eden birimlerden oluşur. Ayrıca kayan noktalı sayı birimi de isteğe bağlı olarak çekirdeğe eklenebilir. Yazmaç sayısı SPARC standartlarına limitleri dahilinde değiştirilebilir. İşlemci Harvard mimarisine göre tasarlanmıştır.

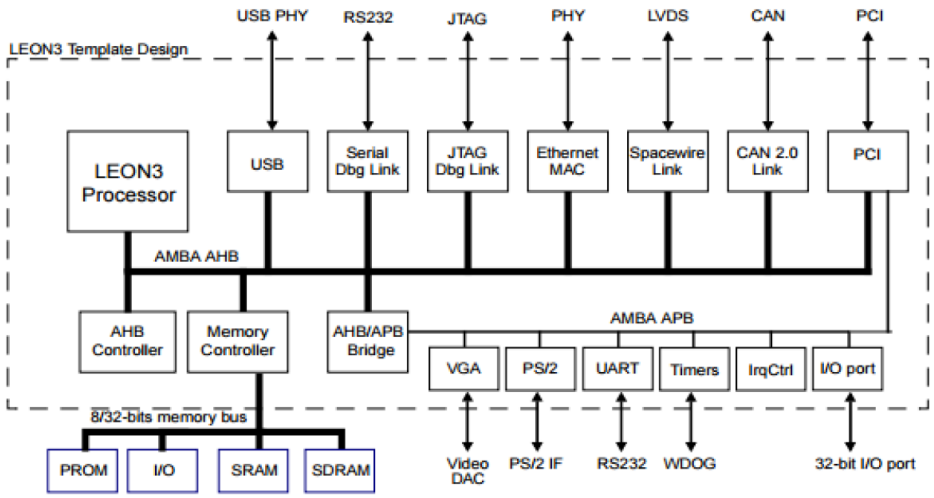
LEON3 oldukça esnektir ve yüksek oranda özelleştirilebilir bir yapıdadır. Tak oynat özelliği sayesinde geliştirme süresi oldukça azalmaktadır. Bunun yanında 0,13 mikron tasarımda 400MHz saat frekansına ulaşabilen yüksek performanslı bir işlemcidir. Saat işareti kapılama ve güç kapatma modu sayesinde düşük güç tüketimi sağlayabilmektedir. Şekil 3.4'de Leon3 işlemcisinin genel yapısı gösterilmektedir. Verillen bu şekil üzerinde işlemcinin çalışması için gerekli minimal bloklar ve çıkarılabilir bloklar da gösterilmiştir.



Şekil 3.4 : LEON3 işlemci yapısı [68].

LEON, VHDL ile oluşturulan GRLIB kütüphanesini desteklemektedir. Kütüphane içindeki her IP'nin kendine has kütüphane ismi vardır. Herhangi bir kütüphane ya da paket eklenmesi ya da çıkarılması sırasında evrensel olarak tanımlanmış olan dosyaların değiştirilmesine gerek yoktur. Yeni IP'lerin eklenmesi ya da eski IP'lerin çıkarılması diğer IP'leri etkilememektedir [69].

Gaisler kütüphanesinde IP'lerin veriyolunun etrafında olduğu bir tasarım yapılmıştır [69]. AMBA (Advanced Microcontroller Bus Architecture) 2.0 AHB (Advanced High Speed Bus)/APB (Advanced Peripheral Bus) pazarda çok yaygın oluşu ve herhangi bir lisans kısıtı olmaması sebebiyle bu yapıda kullanılmış veriyollarıdır. Gaisler kütüphanesiyle tasarlanmış LEON3 yapısı Şekil3.5'te görülmektedir. Bu tasarımda AHB'ye yüksek bant genişliği gerektiren yapılar , APB'ye ise daha yavaş iletişimin problem olmayacağı IP'ler bağlanmıştır. Bu iki veriyolu bir köprü üzerinden birleştirilmiştir. APB'nin yapısı AHB'ye nispeten daha basittir. Tasarlanan yeni IP'ler bu arayüze daha kolay bağlanabilmektedir.



Şekil 3.5 : LEON3 ve GRLIB yapısı [69].

İncelenen işlemcilerin özellikleri Çizelge 3.2'de verilmiştir.

Çizelge 3.2 : Açık kaynak kodlu yazılımsal işlemci karşılaştırması.

İşlemci adı	OpenRISC	LEON3
Geliştirici organizasyon	Open Cores	Cobham Gaisler
Sözcük uzunluğu	32-bit	32-bit
Veriyolu arayüzü	Wishbone	AMBA 2.0
Saat frekansı	300MHz	400MHz
İş hattı kademesi	5 kademeli	7 kademeli

3.4.3 Bluetooth düşük enerji

Bluetooth düşük enerji (Bluetooth low energy - BLE) IEEE 802.15.1 standardındaki kablosuz haberleşme protokolüdür [6]. Düşük miktarda güç harcayarak kısa menzilli radyo sinyalleriyle yayın yapar. 100 metre kadar bir menzili vardır. Gecikme süresi klasik bluetooth'a göre 15 kez daha kısadır [70]. BLE 0.01-10mW güç aralığında yayın yapabilir bu sebeple IoT uygulamalarında kullanılması uygundur [71]. Hem düşük güç tüketimli olması hem de SPI, I2C, UART gibi seri bağlantı arayüzleriyle çalışabilmesi nedeniyle BLE en çok tercih edilen kablosuz haberleşme teknolojilerinden biri haline gelmiştir.

BLE cihazları düşük gücün sağlanabilmesi için uyku modunda çalışır ve yalnızca bir bağlantı başladığında uyku modundan çıkar. Bağlantı süresinin milisaniyelerle ölçülecek kadar kısa olması nedeniyle bu yöntemin kullanılmasıyla güç tüketimi düşük seviyelerde tutulabilir.

3.4.4 Duyargalar

Duyarga, günlük hayatta artan insan ihtiyaçlarının elektronik cihazlarla karşılanabilmesi için geliştirilmiş temel cihazlardan biridir. Bu cihazlar sayesinde bilgisayarlar, insanlar gibi fiziksel dünyanın çeşitli büyüklüklerini ölçebilir, anlayabilir ve çevresine tepki verebilir.

Duyargaların ölçüm yapabilmesi için farklı türleri geliştirilmiştir. Duyargalar, ölçülen büyüklüğe göre, çıkış büyüklüğüne göre, besleme ihtiyacına göre veya birçok farklı özelliğe göre sınıfladırılabilirler [72].

Ölçülen giriş büyüklüğüne göre duyargalar şu şekildedir;

- Mekanik duyargalar: Hız, pozisyon, ivme, ses (dalga boyu ve yoğunluğu), uzunluk, alan, miktar, kuvvet, momet vb. gibi büyüklükler mekanik duyargalar ile ölçülebilir.
- Termal duyargalar: Isı ve sıcaklık gibi büyüklükler termal duyargalar ile ölçülebilir.
- Elektriksel duyargalar: Gerilim, direnç, akım, kapasite, endüktans, elektrik alanı, frekans ve dielektrik katsayısı gibi büyüklükler elektriksel duyargalar ile ölçülür.
- Manyetik duyargalar: Geçirgenlik, manyetik moment, akı yoğunluğu ve alan yoğunluğu gibi büyüklükler manyetik duyargalar ile ölçülebilir.

- Işıma duyargaları: Yoğunluk, dalga bou, polarizasyon, faz, gönderme ,yansıtma gibi büyüklükler ışıma duyargaları ile ölçülebilir.
- Kimyasal duyargalar: Reaksiyon hızı, oksidasyon/redaksiyon hızı, içerik, yoğunlaşma ve pH miktarı gibi büyüklükler ise kimyasal duyargalar ile ölçülebilir.

Çıkış büyüklüğüne göre duyargalar analog ve sayısal duyargalar olmak üzere iki sınıfa ayrılır. Sayısal duyargalar algıladıkları fiziksel büyüklükleri ayrık sinyaller şeklinde üreterek mikrodenetleyicilere iletirken, analog duyargalar ise 0-5 V arasında ya da 4-20 mA arasında değerleri algılayacak biçimde bağlanırlar. Elektriksel analog sinyal algılandıktan sonra analog-sayısal çevirici ile birlikte sayısal işaretlere dönüştürülerek mikrodenetleyicilere gönderilir.

Besleme ihtiyacına göre duyargalar pasif ve aktif duyargalar olmak üzere iki sınıfa ayrılır. Pasif duyargalar, hiçbir ek enerji kaynağına ihtiyaç duymayan ve dışarıdan gelen bir uyarana (stimulus) doğrudan elektriksel bir karşılık üreten duyargalardır. Giriş olarak gelen uyaran duyarga ile bir çıkış sinyaline dönüştürülür. Isılçift (thermocouple) ve piezoelektrik duyargalar bunlara örnektir. Aktif duyargalar ise çalışmak için harici bir enerji beslemesine ihtiyaç duyan duyargalardır. Bu enerji kullanılarak giriş sinyali analog ya da sayısal formata uygun bir çıkış sinyaline dönüştürülür.

Bunların haricinde duyargalar; uyaran türüne (akustik, optik, mekanik, termal, biyolojik vs.), yapıldığı materyale (iletken, yarı-ileken, biyolojik madde), kullanıldığı uygulama alanına göre de sınıflandırılabilir.

Bu bilgiler ışığında değerledirecek olursak uygulamada kullanılacak ultrasonik duyargalar; analog, aktif ve mekanik duyarga sınıflarında yer alır. Bu duyarga ile mesafe ölçümü yapılırken tetikleme sinyali ile duyarganın ses dalgaları yayması sağlanır. Ses dalgalarının bir cisme çarpıp geri dönmesi arasında geçen süre kullanılarak mesafe hesabı yapılır. Bu duyarganın çalışma detayları Bölüm 4.3.2'de anlatılmıştır.

4. DONANIMIN GERÇEKLENMESİ

Bu bölümde donanımın gerçeklenmesinde kullanılan ortam ve gerçekleme için izlenen adımlar anlatılmıştır. Bu adımlar sırasıyla; işlemcilerin FPGA üzerinde gerçeklenmesi ve karşılaştırılması, duyarga ve haberleşme çevresellerinin işlemciye eklenmesi şeklindedir.

4.1 Gerçekleme Ortamı

Tasarımın kolay bir şekilde gerçeklenebilmesi için Linux Mint işletim sistemi ile çalışan bir bilgisayar üzerinde çalışılmıştır. İşlemci ve işletim sistemleriyle ilgili derleyici ve diğer yazılımlar UNIX benzeri sistemlerde çalışan uygulamalar olduğu için bu bilgisayar tercih edilmiştir.

İşlemcilerin karşılaştırılabilmesi için bu işlemciler Atlys FPGA kartı üzerinde gerçeklenmiştir [73]. Çünkü hem OpenRISC hem de LEON3 bu kart için uyarlanmış işlemcilerdir. Atlys geliştirme kartının üzerinde:

- Spartan-6 XC6SLX45 FPGA
- 128 mb ddr ram
- SPI FLASH (16 MB)
- LED (8 adet)
- Düğme (push button) (5 adet), Anahtar (switch) (8 adet)
- Pmod
- Ethernet
- HDMI
- USB2 portu
- USB-UART ve USB-HID portu
- AC-97 kodekli line-in, line-out, mic ve kulaklık girişi
- Bütün enerji hatlarındaki gücü gösteren gerçek-zaman güç monitörü

- Ayarlar ve veri depolaması için 16 MB x4 SPI Flash
- 100 MHz CMOS osilatör

bulunmaktadır.

Bu kart üzerindeki FPGA'in programlanabilmesi ve donanımların sentezlenebilmesi için Xilinx ISE Design Suite 14.7 programı kullanılmıştır [74]. Programın ücretsiz olan WebPack sürümü kullanılmıştır. Bu program ile kart arasındaki bağlantının sağlanabilmesi için gerekli sürücüler yüklenmiştir.

Güç analizinin yapılabilmesi için Windows 10 işletim sistemi yüklü bir bilgisayarda Digilent firmasının ADEPT uygulaması kullanılmıştır [75].

4.2 İşlemcilerin Gerçeklemesi ve Karşılaştırılması

Tasarımda kullanılacak işlemcinin seçilebilmesi için Bölüm 3.4.2'de anlatılan OpenRISC ve LEON3 işlemcileri sentezlenerek Atlys kartı üzerinde gerçeklenmiştir. İşlemciler doğru çalıştığını sınamak için temel uygulamalar yazılmış ve derlenerek kart üzerinde çalıştırılmıştır. Doğrulama işlemlerinden sonra çeşitli uygulamalar enerji kıyaslaması yapmak için belirlenmiş ve her iki işlemci için kart üzerinde çalıştırılmıştır. Enerji kıyaslaması sonucu elde edilen verilere göre, gelecek adımlarda kullanılacak işlemci belirlenmiştir.

4.2.1 OpenRISC gerçeklemesi ve ilk uygulamaları

ORPSoC-v2 verilog dilinde yazılmış OR1200 CPU üzerine inşa edilmiş ve çeşitli FPGA geliştirme kartları için desteklenen kırmık ütü sistem tasarımıdır. Bu sistem CPU'nun yanında çevesel birimlerin kod dosyalarını ve çeşitli yardımcı yazılımları içermektedir [76, 77]. Bu yardımcı yazılımlar sayesinde yazılımsal işlemci, Atlys kartı için kolaylıkla sentezlenebilir. Bu sentez işleminde yazılımlar Xilinx ISE programıyla ortak çalışmaktadır. Yine yardımcı yazılımlarla FPGA üzerine gömülecek dosya (orpsoc.bit dosyası) kolay bir şekilde üretilebilir. Bu yardımcı yazılımlar kullanılmayacaksa verilog dosyalarından oluşan kırmık üstü sistem doğrudan Xilinx ISE üzerinde görüntülenebilir ve benzetimleri yapılabilir ve bit dosyası üretilebilir.

Sistemde tanımlı modüllerin sisteme eklenmesi ya da sistemden çıkarılması ORPSoC-v2 içerisinde yer alan ve bir verilog dosyası olan "orpsoc-defines.v" üzerindeki değişikliklerle sağlanmaktadır. Bu iş için bir arayüz mevcut değildir.

İşlemciler üzerinde çalıştırmak üzere C,C++ gibi çeşitli dillerde yazılan uygulama kodlarının makina kodlarına dönüştürülebilmesi için derlenmesi gerekmektedir. OpenRISC işlemcisinde bu iş için "or1k" derleyicisi kullanılmaktadır.

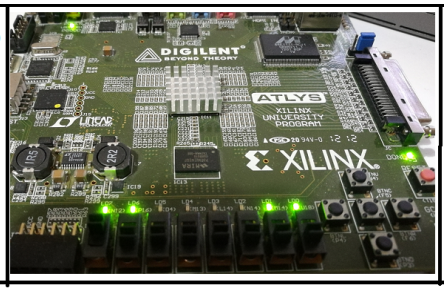
Uygulamaların derlenmesi işlemci üzerinde çalıştırılabilmesi için aşağıdaki komutlar kullanılmıştır:

- or1k-elf-gcc -mboard=atlys uygulama_ismi.c -o uygulama_ismi.elf
- or1k-elf-objcopy -O binary uygulama_ismi.elf uygulama_ismi.bin
- bin2binsizeword uygulama_ismi.bin uygulama_ismi_size.bin
- promgen -spi -p mcs -w -o uygulama_ismi.mcs -s 16384 -u 0 orpsoc.bit -data_file up 1c0000 uygulama_ismi_size.bin

Verilen komutlarla C dosyası olarak yazılan uygulama ve "orpsoc.bit" dosyası kullanılarak .mcs uzantılı bir dosya üretilmiştir. Bu dosya Atlys kartının flashına yüklenebilmektedir. Bu sayede kart her açıldığında flashtan ön yükleme yapılarak, üzerinde yazılan uygulamanın çalıştığı işlemci gerçeklenmektedir.

Derleyici ile LedTest uygulaması OpenRISC işlemcisi için derlenmiştir. Bu uygulamayla sistemin GPIO (General Purpose Input Output - genel amaçlı giriş çıkış) adreslerine erişilebildiğini göstermek amacıyla kart üzerindeki ledlerin kontrolü sağlanmıştır. Ledleri belirli bir düzende yakan bu uygulama için C kodu ve uygulama sonrası ledlerin durumu Şekil 4.1'de verilmiştir [77, 78].

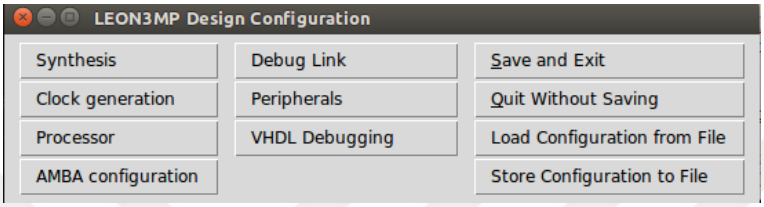
```c
#include "board.h"
#define writeGPIO(addr,val) (*(unsigned char*) (addr) = (val))
#define readGPIO(addr)      (*(unsigned char*) (addr))
int main (int argc, char *argv[])
{
   writeGPIO((unsigned char *)(GPIO_0_BASE + 0x3), 0xff);
   writeGPIO((unsigned char *)(GPIO_0_BASE + 0x4), 0xff);
   writeGPIO((unsigned char *)(GPIO_0_BASE + 0x5), 0xff);

   writeGPIO((unsigned char *)GPIO_0_BASE, 0xc3);
   while(1);
 return 0;
}
```



Şekil 4.1 : OpenRISC için yazılan LedTest uygulaması ve ledlerin durumu.

4.2.2 LEON3 gerçeklemesi ve ilk uygulamaları

LEON3 işlemcisinin gerçeklenmesi için Gaisler IP kütüphanesi (GRLIB) kullanılmaktadır. Bu kütüphane SoC için tekrar kullanılabilir IP çekirdeklerini içermektedir [69]. GRLIB kütüphanesi Gaisler sitesinden indirilebilmektedir. İndirilen kütüphane klasörü içinde tasarımlar "leon3-<üretici adı>-<kart adı>" şeklinde isimlendirilmiştir. Tez kapsamında kulla nılan Atlys kartı için ilgili dosyaya girilerek Linux komut terminalinde "make xconfig" komutu çalıştırılır ve sistemin arayüzü ekrana gelir. LEON3 tasarım yapılandırma ekranı, Şekil 4.2'de verilmiştir.

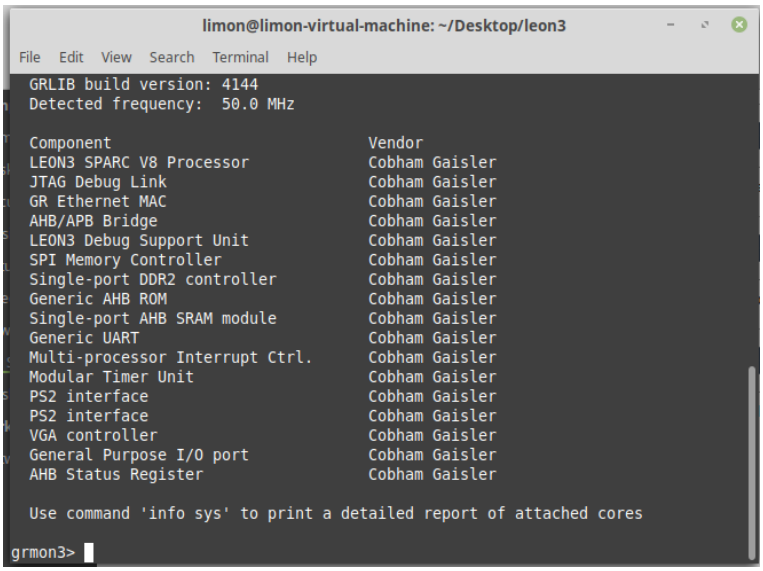


Şekil 4.2 : LEON3 tasarım yapılandırma ana ekranı.

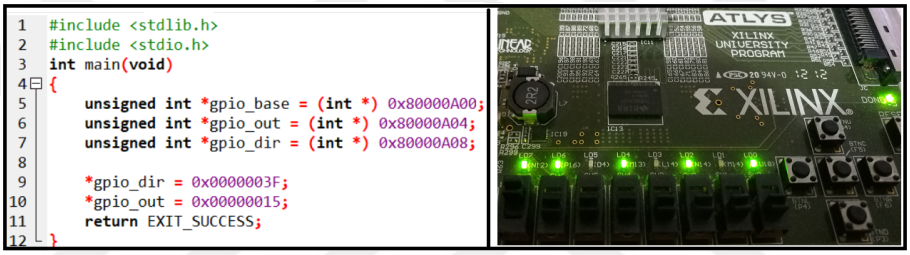
Tasarım yapılandırma ekranında işlemcinin özellikleri ve eklenecek çevreseller belirlenir. Ekrandan çıkılmasıyla "config.vhd" dosyası üretilmiş olur. Daha sonra yine terminal üzerinden "make ise" komutu çalıştırılarak "leon3mp.vhd" üst modülüyle tanımlanmış işlemcinin ISE üzerinden sentezlenmesi sağlanır. Sentezleme sonunda "leon3mp.bit" dosyası da üretilmiş olur. Bu dosyanın kart üzerine aktarılmasıyla LEON3 işlemcisi kullanılmaya hazır hale gelir.

C ve C++ uygulamalarının LEON3 işlemcisinde çalışabilmesi için Bare-C Çapraz derleyicisi (Bare-C Cross Compiler-BCC) ile uygulamaların derlenmesi gerekmektedir [79]. Bu derleyici yine Gaisler sitesinden edinilebilir. LEON3 işlemcisi için BCC kullanılarak "Hello world!" ve "LedTest" uygulamaları derlenmiştir. Derlenen kodların kart üzerinde yüklenebilmesi ve doğruluğunun görüntülenebilmesi için Gaisler ekranı (Gaisler Monitor-GRMON) adı verilen hata ayıklama aracı kullanılmıştır [80]. Bu programın kullanılması için Atlys kartı programlama girişi üzerinden USB kablo ile bilgisayara bağlanmıştır. Program, komut terminalinde "grmon -u -digilent" komutu ile Atlys kartına bağlanmak üzere başlatılır. GRMON başlama ekranının görüntüsü Şekil 4.3'te verilmiştir.

LEON3 için derlenen LedTest uygulamasının C kodu ve uygulama sonrası ledlerin durumu Şekil 4.4'te verilmiştir.

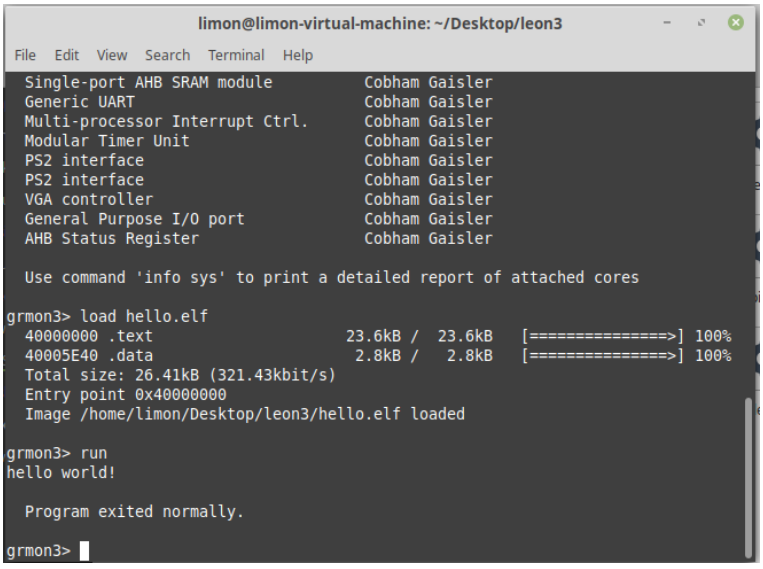


Şekil 4.3 : GRMON başlama ekranı.



Şekil 4.4 : LEON3 için yazılan LedTest uygulaması ve ledlerin durumu.

Programı başlatmak için girilen ek seçeneklerden "-u" ile UART çıkışına verilen mesajlar GRMON ekranı üzerinde görüntülenebilmektedir. Örnek olarak "Hello world!" uygulamasının çıktısı Şekil 4.5'te verilmiştir.



Şekil 4.5 : Hello world uygulamasının terminaldeki çıktısı.

4.2.3 İşlemcilerin güç analizi

Gömülü sistemlerde programlanabilir birimlerin güç tüketiminin bu birimler üzerinde çalışan programa ve işlenen veriye göre değiştiği bilinmektedir [81–83]. Bu nedenle sistemlerin geliştirilmesinde bu durum göz önünde bulundurularak güç tüketimini azaltacak tasarımlar yapılmaktadır. Güç tüketimi azaltılması için yapılan bu çalışmalarda güç tahmini yapılması gerekmektedir [84]. Tasarımlardaki güç tüketimin tahmin edilebilmesi için simülasyon tabanlı ve ölçüm tabanlı olmak üzere iki yönteme başvurulmaktadır.

Xpower yazılımı simülasyon tabanlı güç tüketimi tahmini yapmak amacıyla yaygın olarak kullanılan bir uygulamadır [85]. Bu yazılım, tüketim tahmini için "Post-Place & Route" benzetimi sonrası üretilen Değer Değişim Dökümü (Value Change Dump - VCD) dosyasını kullanmaktadır. Bu dosya benzetimdeki her işaret için 0 ve 1 değişimlerini içermektedir. Bu değişim bilgileri ve tasarımın fiziksel bilgileri kullanılarak Xpower uygulamasıyla güç tahmini yapılmaktadır. Ancak kullanılan OpenRISC işlemcisi için "Post-Place & Route" benzetimlerinin doğru bir şekilde yapılamadığı görülmüştür. İşlemcinin tasarımı sırasında kullanılan yazım biçimi ve olağan test dosyaları ile bu benzetimi yapmak mümkün değildir. Benzetimin yapılabilmesi için bu sorunlar giderilmeye çalışılmıştır ama kullanılabilecek çözüm yöntemleri hem güç analizinin doğruluğunu etkilemekte hem de oldukça fazla vakit almaktadır. Bununla birlikte bu büyük tasarımların simülasyon üzerinden analizinin yapılması için büyük miktarda veri üretilmektedir ve verinin elde edilip işlenmesi için uzun sürelere ihtiyaç vardır. Bu nedenle güç analizi için ölçüm tabanlı yöntemin kullanılmasına karar verilmiştir. Bu ölçüm yöntemi için daha önce yalnızca OpenRISC üzerinde güç analizi yapılan bir çalışmadan [86] faydalanılmıştır. Aynı yöntem ile LEON3 işlemcisi de analiz edilip işlemciler karşılaştırılmıştır.

4.2.3.1 İşlemcilerin yapılandırılması

OpenRISC ve LEON3 işlemcilerini karşılaştırabilmek için öncelikle bu iki işlemci olabildiğince denk bir şekilde yapılandırılmalıdır. Bu amaçla işlemci üzerinde zorunlu olarak belirtilen birimlerin haricinde isteğe bağlı birimler de tasarımlara eklenerek denk hale getirilmeye çalışılmıştır. Ayrıca işlemcilerin güç tüketiminin komutlara bağlı

olarak değiştiğini gözlemleyebilmek için işlemci özellikleri ve işlemciye bağlı çevresel birimler seçilirken, tasarım olabildiğince minimal bir yapıda tutulmaya çalışılmıştır.

Buradan yola çıkarak OpenRISC ve LEON3 işlemcileri Çizelge 4.1'deki elemanları içerecek şekilde yapılandırılmıştır.

Çizelge 4.1 : İşlemciler denk olacak şekilde eklenen birimler.

OpenRISC	LEON3
Wishbone arayüzü (zorunlu)	AMBA 2.0 arayüzü (zorunlu)
5-kademeli iş hattı (zorunlu)	7-kademeli iş hattı (zorunlu)
Hata ayıklama birimi(eklendi)	Hata ayıklama birimi (gerekli)
Komut önbelleği	Komut önbelleği
Veri önbelleği	Veri önbelleği
Bellek yönetim birimi	Bellek yönetim birimi
Güç yönetici	Güç yönetici
Zamanlayıcı	Zamanlayıcı
Kesme Kontrolü	Kesme Kontrolü
DDR bellek	DDR bellek
GPIO	GPIO

Zorunlu birimler haricinde eklenen birimlerin eklenme nedeni şu şekilde açıklanabilir:

- LEON3 işlemcisine programların yüklenebilmesi için hata ayıklama biriminin aktif olması gerektiğinden bu birim OpenRISC işlemcisi için de donanıma eklenmiştir.
- Güç tüketimini azaltacak şekilde her iki işlemciye de güç yönetici birimleri eklenmiştir.
- İşletim sistemleri ile çalışabilmeleri için her iki işlemcide de bellek yönetim birimi, zamanlayıcı ve kesme kontrol birimleri donanıma eklenmiştir.
- İşlemcilerin kullanılacağı uygulama alanlarının kısıtlanmaması adına her iki işlemcide de DDR bellek açık tutulmuştur. Örneğin işlemciyle bir kamera kullanmak ve görüntü işlemek istenirse bellek ihtiyacı doğacaktır
- Komut ve veri önbellekleri işlem süresinin azaltılmasında oldukça önemli oldukları için bu birimler de açık tutulmuştur. Bu önbelleklerin boyutları ve özellikleri ise birbirine denk olacak şekilde yapılandırılmıştır. Önbellek boyutunun işlem süresine yani enerjiye etkisi Bölüm 4.2.3.4'te incelenmiştir.

İşlemciler için isteğe bağlı eklenebilir birimlerden çarpma, bölme, kayan nokta birimleri ile Ethernet, klavye ve fare çevreselleri tasarımdan çıkarılarak minimallik sağlanmıştır.

Denk hale getirilen işlemciler için FPGA birimlerinin kullanım oranları (utilization) ve sayıları sentez sonrasında Çizelge 4.2 ve Çizelge 4.3'teki halleriyle gözlenmiştir.

Çizelge 4.2 : Sentezlenen LEON3 için FPGA kullanım oranları.

Önbellek boyutu	Önbellek yok		8kB Önbellek		32kB Önbellek	
	Birim sayısı	Kullanım oranı	Birim sayısı	Kullanım oranı	Birim sayısı	Kullanım oranı
Yazmaç dilimi	4757	8%	4910	8%	4920	9%
LUT dilimi	6898	25%	7543	27%	7539	27%
Kapladığı dilim	2706	39%	2845	41%	2854	41%
MUXCY	244	1%	280	2%	280	2%
LUT Flip Flop çifti	8316	-	8855	-	8837	-
Bağlı IOB'ler	107	49%	107	49%	107	49%
RAMB16BWER	15	12%	25	21%	55	47%
RAMB8BWER	0	0%	0	0%	0	0
Toplam BRAM hacmi	30 kB		50 kB		110 kB	

Çizelge 4.3 : Sentezlenen OpenRISC için FPGA kullanım oranları.

Önbellek boyutu	Önbellek yok		8Kb Önbellek		32Kb Önbellek	
	Birim sayısı	Kullanım oranı	Birim sayısı	Kullanım oranı	Birim sayısı	Kullanım oranı
Yazmaç dilimi	3831	7%	3900	7%	3903	7%
LUT dilimi	6310	23%	6760	24%	6750	24%
Kapladığı dilim	2363	34%	2427	35%	2377	34%
MUXCY	732	5%	748	5%	748	5%
LUT Flip Flop çifti	6829	-	7174	-	7148	-
Bağlı IOB'ler	83	30%	83	38%	83	38%
RAMB16BWER	6	5%	16	13%	40	34%
RAMB8BWER	2	1%	2	1 %	3	1%
Toplam BRAM hacmi	14 kB		34 kB		83 kB	

4.2.3.2 Ölçüm ortamı

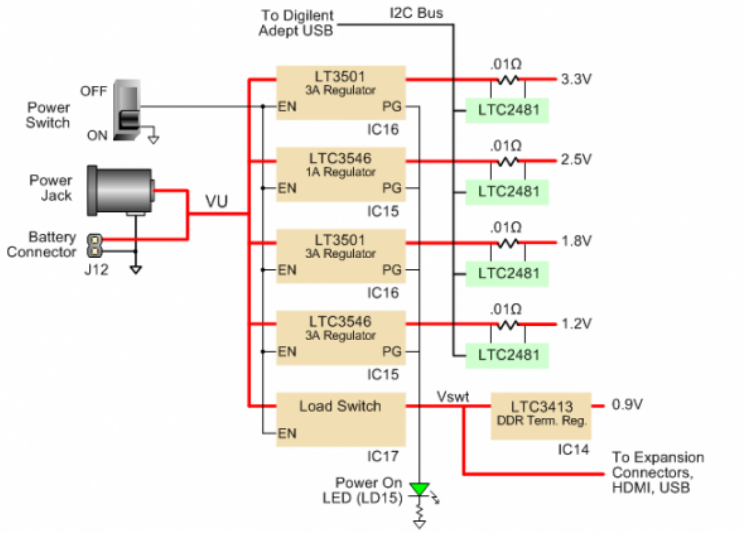
Tasarımın gerçeklemesinde kullanılan Atlys kartı üzerinde güç kaynağı izleme ekipmanları vardır. Bu ekipmanlar LTC2481 sigma-delta analog-sayısal dönüştürücülerdir. Digilent ADEPT uygulaması kullanılarak bu ekipmanlar sayesinde güç izlenebilmektedir [75]. Kart üzerinde güç kaynağına bağlı beş adet gerilim düzenleyici bulunur. Bu gerilim düzenleyicilerle sağlanan 3,3V, 2,5V 1,8V 1,2V ve 0,9V gerilimleriyle geliştirme kartı üzerindeki farklı birimler beslenmektedir. Bunlar Çizelge 4.4'te verilmiştir. Bu güç kaynaklarından ilk dördü ADEPT programında görüntülenebilmektedir.

Çizelge 4.4 : Atlys güç kaynakları.

Kaynak	Devreler	Cihaz	Amper (max/typ)
3,3 V	FPGA I/O, video,saat, USB portları, ROM, ses	IC16:LT3501	3A/900mA
2,5 V	FPGA aux,VHDC,GPIO, Ethernet PHY I/O	IC15:LTC3546	1A/400mA
1,2 V	FPGA core, Ethernet PHY core	IC15: LTC3546	3A/0,8 – 1,8A
1,8 V	DDR, FPGA DDR I/O	IC16:LT3501	3A/0,5 – 1,2A
0,9 V	DDR terminasyon gerilimi	IC14:LTC3413	3A / 900mA

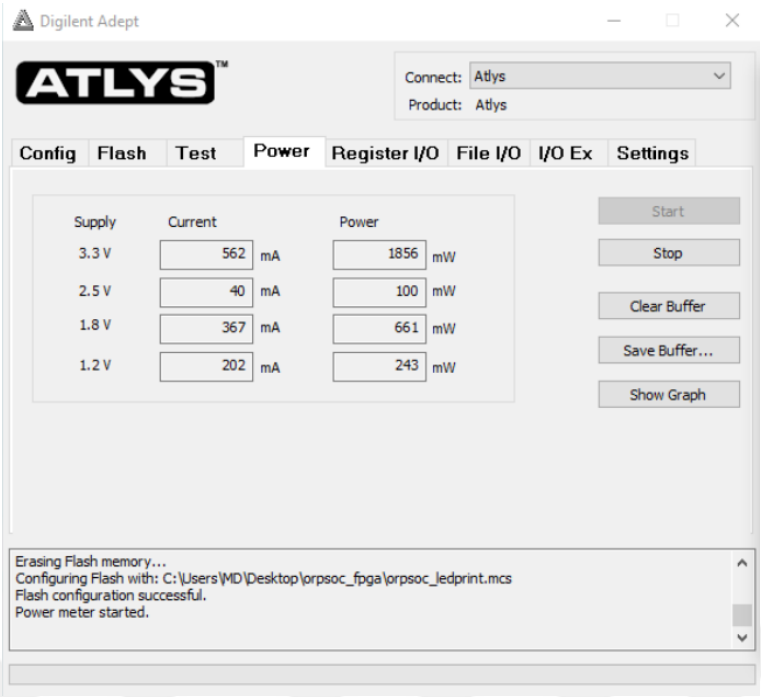
Atlys kartının güç kaynağı ve güç izleme cihazlarını içeren şematik Şekil 4.6'da gösterilmektedir. Görülebileceği üzere gerilim düzenleyiciler delta-sigma dönüştürücülerin solunda yer almaktadır. Bu sayede izlenecek gücün sadece Çizelge 4.4'ün "Devreler" sütununda belirtilen birimlerin tükettiği güç olacağı anlaşılmaktadır.

LTC2481 delta-sigma analog-sayısal dönüştürücüleriyle, kaynaktan çekilen akımın 0,2 saniye süresi içindeki ortalaması alınarak 16-bitlik sayısal işaret elde edilmektedir [87]. Bu durumda yüksek frekanslarda çalışan bir donanım için harcanan gücün anlık bilgisi elde edilemez ancak ortalama güç tüketimi bilgisine ulaşılabilir. Anlık değişimin gözlenebilmesi için örnekleme frekansı, çalışma frekansının en az iki katı olmalıdır.



Şekil 4.6 : Atlys güç kaynakları şematiği [75].

Atlys kartı üzerinde donanımların gerçeklemesinden sonra ADEPT programı üzerinden ölçülen akım ve güç bilgileri, veri dosyası şeklinde elde edilebilmektedir. ADEPT programının arayüzü ve sinyal izleme ekranı Şekil 4.7 ve Şekil 4.8'deki gibidir.



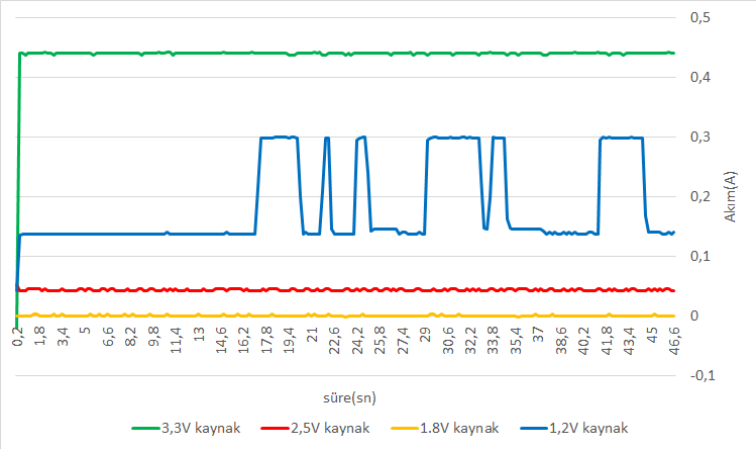
Şekil 4.7 : ADEPT program arayüzü [75].

Şekil 4.8 : ADEPT sinyal izleme ekranı [75].

4.2.3.3 Ölçüm için örnek uygulamaların belirlenmesi

Paralel çarpma uygulaması

Güç analizinde kullanılacak ADEPT uygulamasının doğru çalışıp çalışmadığını anlayabilmek için öncelikle işlemcilerden bağımsız olarak bir çarpma devresi tasarlanmıştır. Bu devre bir üst modül ve iki alt modülden oluşmaktadır. Üst modülde tasarımın saat girişi bir giriş anahtarıyla 'VE' işlemine sokulmuştur. Anahtar "0" konumundayken devrede saat girişi durdurulmakta "1" konumundayken saat girişi aktif olmaktadır. Alt modüller ise 1 adet BRAM hafıza elemanı ve 1 adet çarpıcı



Şekil 4.9 : Çarpma uygulamasında harcanan akımlar.

devreden oluşmaktadır. Saat girişi aktifken çarpıcı devre BRAM'den 32-bitlik verileri okuyarak 10 paralel çarpma işlemi yapmaktadır. Böylece anahtar kontrolüyle yüksek ve düşük güç tüketiminin karşılaştırılabileceği bir sistem oluşturulmuştur. Deneyin sonunda anahtarın rastgele açılıp kapatıldığı hallerde, kartın çektiği akım sinyalleri ADEPT uygulaması üzerinde Şekil 4.9'daki gibi gözlenmiştir. Şekilde verilen grafikte zaman ekseni saniye cinsinden ifade edilmiştir. Örnek uygulama grafikte 46 saniye boyunca gözlenmiştir. 0,2sn'de bir veri girişinin olduğu bu durumda her bir verinin ayrı ayrı gösterilmesi mümkün olmadığı için çizgisel grafikten faydalanılmıştır.

Programdan elde edilen veriler yüksek ve düşük tüketim için Çizelge 4.5'teki gibidir. Devrede DDR bellek kullanılmadığı için 1,8V'luk kaynakta güç tüketimi gözlenmemiştir. Çevresellerin bağlı olduğu 2,5V kaynak ise aktiftir ancak güç değişiminin olmadığı görülür. Çarpıcı devresinde anahtarın değişimiyle BRAM'den veri çekilmeye başlanmış ve çekirdek üzerinde çarpım işlemleri gerçekleşmiştir. Bu yüzden 3,3V ve 1,2V'luk kaynaklar üzerinde güç değişimi gözlenmiştir. Beklenildiği gibi anahtarın "1"olduğu durumda güç tüketimi yüksek, "0" olduğu durumda tüketim düşüktür.Böylece ADEPT uygulamasının doğru şekilde çalıştığı anlaşılmıştır ve kullanılmasına karar verilmiştir.

Çizelge 4.5 : Çarpma devresi için güç tüketimi verileri.

Güç kaynakları	Anahtar açıkken (mW)	Anahtar kapalıyken(mW)
3,3 V	1245	1212
2,5 V	106	106
1,8 V	0	0
1,2 V	357	162

WCET kıyaslama projeleri

İşlemciler üzerinde test edilecek programlar en kötü durum koşum zamanı kıyaslama projeleri arasından (Worst Case Execution Time Benchmark - WCET) seçilmiştir. Programlar, Mälardalen Üniversitesi'nin resmi sayfasından alınmıştır [88]. Bu kıyaslama programlarında amaç, programın en uzun çalışma süresinin ne olduğunu bulmaktır. Deney için seçilen kıyaslama programları kendi kendine yeten C kodlarıdır. Ek kütüphane dosyası, işletim sistemi ya da dosya sistemine ihtiyaç duymamaktadır. Çalışmada kullanılan programlar, Çizelge 4.6'da özellikleriyle birlikte verilmiştir.

Çizelge 4.6 : WCET kıyaslama programları.

Programın adı	Tanım	Yorum	Bayt sayısı	Kod satırı
Compress	Veri sıkıştırma programı	Rastgele veriyi bir buffer üzerinde sıkıştırır.	13411	508
Nsichneu	Genişletilmiş Petri Net simülasyonu	250'den fazla koşul ifadesi içeren otomatik olarak oluşturulan kod .	118351	4253

Atlys üzerinde gerçeklenen işlemcilerin saat hızı ise 50MHz'dir yani saat periyodu 20 ns'dir. ADEPT uygulamasıise 0,2 sn'de bir harcanan ortalama gücün bilgisimi üretmektedir. 0,2sn'de 10^7 periyot geçer. Bu sebeple program 1 kez çalıştırılırsa güç değişimi gözlenemeyecektir. Seçilen programlar için güç değişiminin ADEPT üzerinde görüntülenebilmesi için bu programlar peş peşe çalışacak şekilde döngülere sokulmuştur. Programlara ayrıca 'NOP' komutundan oluşan ve bekleme yapılmasını sağlayan döngüler eklenmiştir. Deneyler, kıyaslama programı ve NOP komutlarından oluşan işleri 10'ar kez tekrarlayacak şekilde ayarlanmıştır. Böylece farklı komutlar için işlemcilerin farklı seviyede güç harcadığı gözlenecektir.

Deneyler için belirlenen döngü ve komut sayıları Çizelge 4.7'de verilmiştir.

Çizelge 4.7 : Testler için döngü sayıları.

Programın adı	Kıyaslama programı tekrarlama sayısı	NOP komutu sayısı
Compress	20.000	50.000.000
Nsichneu	20.000	50.000.000

4.2.3.4 Ölçüm deneyleri ve sonuçları

Kıyaslamalar için öncelikle Atlys kartı üzerinde LEON3 işlemcisi Bölüm 4.2.3.1'de anlatıldığı üzere önbellek olmayan, 8 kB ve 32 kB önbellek olan yapılandırmalarda gerçeklenmiştir. Her bir gerçekleme üzerinde, hazırlanan Compress-NOP ve Nsichneu-NOP test programları çalıştırılmıştır. ADEPT programı kullanılarak elde edilen akım-zaman grafikleri Şekil 4.10'de verilmiştir. Grafikler 60 saniyelik süreyi göstermektedir. Verilen grafiklerde test sırasında "NOP" komutlarının koştuğu, kıyaslama programının koştuğu ve test tamamlandıktan sonra sistemin boşta çalıştığı durumların üçünün de görülebilmesi sağlanmıştır. Bunun için işlemcinin boşta çalışmaya başladığı kısım, son 4 saniyede görülecek şekilde grafikler kaydırılmıştır.

İşlemciler yüksek hızlarda çalışmaktadır ve sürekli olarak hafızadan veri okur. Sistem belleğinden gelen veriler ise çoğunlukla işlemcinin hızına yetişemez ve işlemci verinin ulaşmasını beklemek zorunda kalır. Bu problemi çözmek için işlemci ön bellek kullanılır. Ön bellek çalışmakta olan programa ait komutların, verilerin geçici olarak saklandığı bellek birimleridir. Bu birimler FPGA çekirdeği üzerinde gerçeklenmektedir ve sistem belleği de DDR üzerindedir. Bu bilgiler göz önüne alınarak grafikler incelendiğinde şu sonuçlara ulaşılmıştır:

- NOP komutlarının koştuğu kısımların kıyaslama döngülerinin koştuğu kısımlara göre daha az güç harcadığı görülmektedir.
- Programlar tamamlanıp sistem boşta çalışmaya başladığı zaman 1,2V ve 1,8V'luk kaynaklarda dinamik güç tüketiminin önemli ölçüde azaldığı görülmektedir.
- Tüm yapılandırmalar için, Compress-NOP programı Nsichneu-NOP programına göre daha kısa sürede tamamlanmaktadır. Çizelge 4.6'da verildiği üzere bayt ve kod satırı sayılarının bunda etkili olduğu söylenebilir.
- Önbellek boyutu arttıkça işlemlerin gerçekleme süreleri kısalmaktadır ancak FPGA üzerinde kullanılan alan arttığı için tüketilen güç de artmaktadır. FPGA kullanım alanlarıyla ilgili verilere Çizelge 4.2'den incelenebilir.
- 8 kB ve 32 kB önbellek kullanımı Nsichneu-NOP testi için incelendiğinde önbellek boyutunun artmasıyla sürenin oldukça kısaldığı gözlenirken Compress-NOP

testinde sürede belirgin bir değişiklik olmadığı görülmüştür. Bu da 8 kB önbellek boyutunun Compress-NOP testi için yeterliyken Nsichneu-NOP testi için yeterli olmadığını göstermektedir.

- Önbelleğin yeterli olduğu durumda, önbellek dolana kadar DDR belleğe erişildiği için 1,8V'luk kaynakta güç tüketiminin arttığı gözlenmiştir.
- 8 kB ve 32 kB önbellek kullanımı Compress-NOP testi için karşılaştırıldığında sadece, FPGA çekirdeğini besleyen 1,2V'luk kaynağın güç tüketimin arttığı gözlemlenmiştir. Bu da FPGA kullanım alanının artmasından kaynaklanmaktadır.

LEON3 işlemcisiyle yapılan testler sonucunda elde edilen güç, süre ve enerji verileri Çizelge 4.8 ve Çizelge 4.9'da paylaşılmıştır.

LEON3 işlemcisi kullanılarak yapılan testler daha sonra OpenRISC işlemcisi için de tekrar edilmiştir. Bunun sonucunda ADEPT programıyla elde edilen akım-zaman grafikleri Şekil 4.11'de verilmiştir.

Verilen grafikler incelendiğinde, LEON3'tekiyle aynı şekilde 'NOP' komutunun koştuğu kısımların kıyaslama programının koştuğu yerlerden daha az güç tükettiği gözlenmiştir. Ayrıca yine LEON3'teki gibi sistemin boşta çalıştığı durumda 1,8V'luk kaynakta dinamik güç tüketiminin önemli ölçüde azaldığı gözlenmiştir. Önbellek boyutunun gerçeklenme süreleriyle ilişkisi LEON3'tekiyle benzerdir. Yine önbellek boyutu artması nedeniyle FPGA kullanım alanının arttığı bu nedenle 1,2V'luk kaynağın güç tüketimin arttığı sonucuna ulaşılmıştır. OpenRISC işlemcisin FPGA kullanım alanlarıyla ilgili verileri Çizelge 4.3'ten incelenebilir.

LEON3'ten farklı olarak önbelleğin yeterli olduğu ve olmadığı durumlarda 1,8V'luk kaynağın tükettiği güç belirgin olarak değişmemiştir. DDR belleğe bağlı olan bu kaynakta güç tüketiminin önbellek boyutundan bağımsız olarak aynı güç seviyelerinde seyrettiği gözlenmiştir. İşlemcilerin; önbellek yerleştirme ve değiştirme planları, kullandığı dallanma öngörüsü, bellek yönetimi vb. gibi sistem belleği ile önbellek arasındaki ilişkiyi kuran işlemleri daha farklı yapmasıyla açıklanabilir.

OpenRISC işlemcisiyle yapılan testler sonucunda elde edilen güç, süre ve enerji verileri Çizelge 4.10 ve Çizelge 4.11'de paylaşılmıştır.

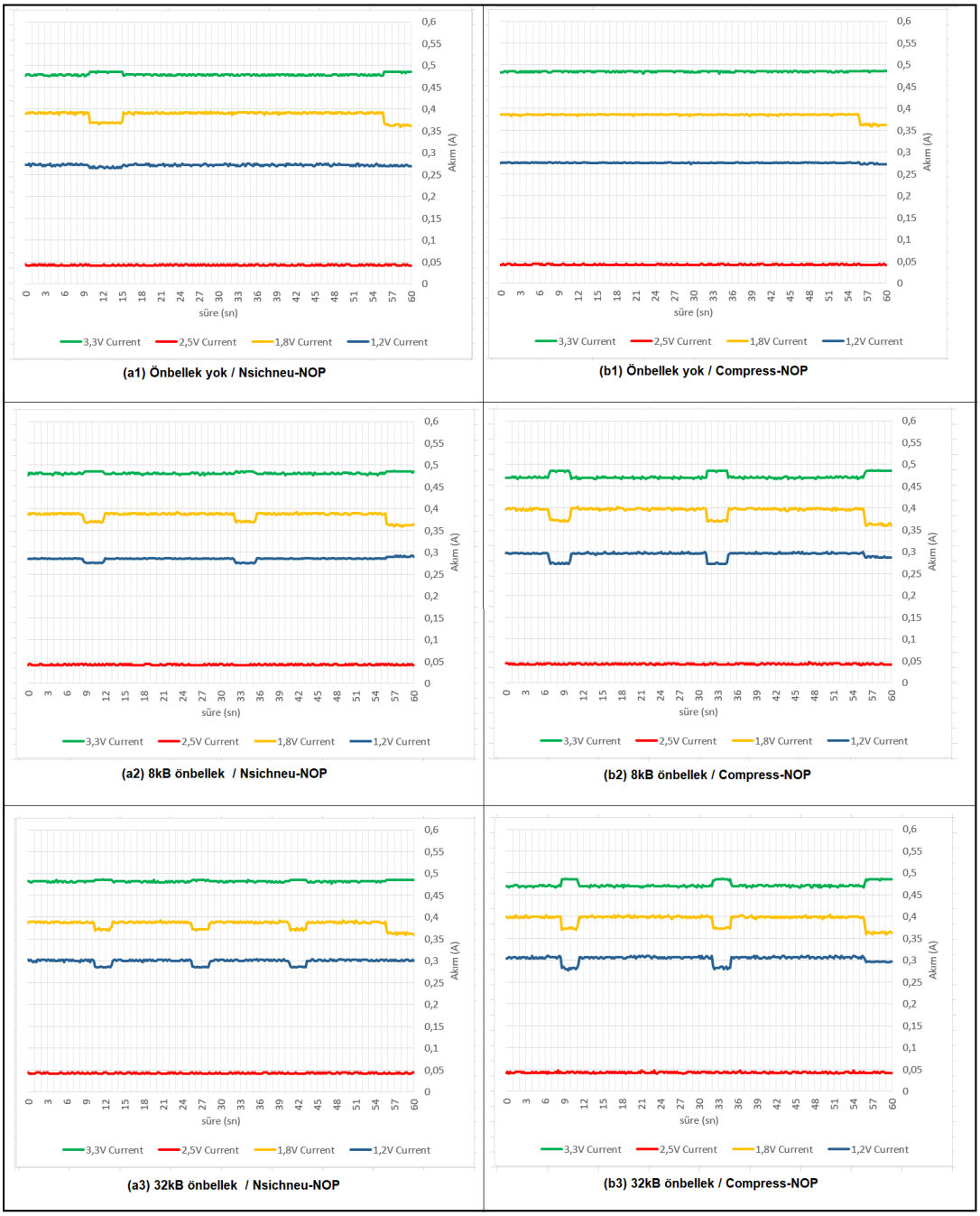
Şekil 4.10 : LEON3 ile yapılan testlerde akım-zaman grafikleri: (a1) Önbellek yok & Nsichneu-NOP, (a2) 8kB Önbellek & Nsichneu-NOP, (a3) 32kB Önbellek & Nsichneu-NOP, (b1) Önbellek yok & Compress-NOP, (b2) 8kB Önbellek & Compress-NOP, (b3) 32kB Önbellek & Compress-NOP.

Çizelge 4.8 : LEON3 üzerinde koşan Compress-NOP testi için güç ve enerji tüketimi sonuçları.

Önbellek boyutu	Önbellek yok			8Kb Önbellek			32Kb Önbellek		
Test aşaması	NOP	Kıyas. P.	Boşta	NOP	Kıyas. P.	Boşta	NOP	Kıyas. P.	Boşta
3,3V kaynak harcanan güç (mW)	1,67944	1,670431	1,740054	1,685499	1,644117	1,740054	1,68423	1,651331	1,740054
2,5V kaynak harcanan güç (mW)	0,100429	0,100292	0,09996	0,09996	0,096298	0,09996	0,100441	0,099745	0,09996
1,8V kaynak harcanan güç (mW)	0,225585	0,265016	0,140944	0,209685	0,359546	0,140515	0,210723	0,354427	0,140058
1,2V kaynak harcanan güç (mW)	0,152714	0,162705	0,136231	0,16009	0,208468	0,140658	0,166087	0,226875	0,147623
Toplam güç (mW)	2,158168	2,198443	2,155234	2,117189	2,308429	2,121187	2,161481	2,332377	2,127694
Gerçekleme süresi (sn)	7,8	45,2	-	2,6	6	-	2,6	5,8	-
Harcanan enerji (J)	16,83371	99,36964	-	5,603608	13,85057	-	5,619851	13,52779	-
Toplam enerji (J)	116,2033		-	19,45418		-	19,14764		-

Çizelge 4.9 : LEON3 üzerinde koşan Nsichneu-NOP testi için güç ve enerji tüketimi sonuçları.

Önbellek boyutu	Önbellek yok			8Kb Önbellek			32Kb Önbellek		
Test aşaması	NOP	Kıyas. P.	Boşta	NOP	Kıyas. P.	Boşta	NOP	Kıyas. P.	Boşta
3,3V kaynak harcanan güç (mW)	1,677675	1,670396	1,740054	1,68423	1,675523	1,740878	1,681692	1,646591	1,740054
2,5V kaynak harcanan güç (mW)	0,100441	0,100267	0,09996	0,099479	0,09996	0,09996	0,099479	0,099663	0,09996
1,8V kaynak harcanan güç (mW)	0,225487	0,25601	0,141158	0,209685	0,25055	0,140119	0,208993	0,285957	0,139873
1,2V kaynak harcanan güç (mW)	0,150324	0,158199	0,135089	0,158244	0,177904	0,140344	0,164242	0,204632	0,146941
Toplam güç (mW)	2,153927	2,184873		2,151639	2,203937	2,121301	2,154407	2,236843	2,126828
Gerçekleme süresi (sn)	7,8	48,8		2,6	16		2,6	8,4	
Harcanan enerji (J)	16,80063	106,6218		5,594261	35,26299		5,601459	18,78948	
Toplam enerji	123,4224		-	40,85725		-	24,39094		-



Şekil 4.11 : OpenRISC ile yapılan testlerde akım-zaman grafikleri: (a1) Önbellek yok & Nsichneu-NOP, (a2) 8kB Önbellek & Nsichneu-NOP, (a3) 32kB Önbellek & Nsichneu-NOP, (b1) Önbellek yok & Compress-NOP, (b2) 8kB Önbellek & Compress-NOP, (b3) 32kB Önbellek & Compress-NOP..

Çizelge 4.10 : OpenRISC üzerinde koşan Compress-NOP testi için güç ve enerji tüketimi sonuçları.

Önbellek boyutu	Önbellek yok			8Kb Önbellek			32Kb Önbellek		
Test aşaması	NOP	Kıyas. P.	Boşta	NOP	Kıyas. P.	Boşta	NOP	Kıyas. P.	Boşta
3,3V kaynak harcanan güç (mW)	1,59986	1,597603	1,600984	1,599344	1,558626	1,59911	1,598682	1,551252	1,59836
2,5V kaynak harcanan güç (mW)	0,106928	0,107806	0,107059	0,10816	0,106208	0,107911	0,109331	0,10855	0,107911
1,8V kaynak harcanan güç (mW)	0,661754	0,694972	0,654079	0,667702	0,715214	0,654284	0,67216	0,719106	0,654897
1,2V kaynak harcanan güç (mW)	0,322371	0,330872	0,32796	0,327431	0,356857	0,346089	0,338436	0,368391	0,356176
Toplam güç (mW)	2,690914	2,731253	2,681597	2,702637	2,736905	2,707394	2,718609	2,7473	2,717344
Gerçekleme süresi (sn)	5,2	101,6		3,2	21,2		2,8	21,2	
Harcanan enerji (J)	13,99275	277,4953		8,648439	58,02238		7,612104	58,24275	
Toplam enerji (J)	291,4881			66,67082			65,85486		

Çizelge 4.11 : OpenRISC üzerinde koşan Nsichneu-NOP testi için güç ve enerji tüketimi sonuçları.

Önbellek boyutu	Önbellek yok			8Kb Önbellek			32Kb Önbellek		
Test aşaması	NOP	Kıyas. P.	Boşta	NOP	Kıyas. P.	Boşta	NOP	Kıyas. P.	Boşta
3,3V kaynak harcanan güç (mW)	1,599543	1,57967	1,59911	1,596949	1,585571	1,59986	1,598485	1,589219	1,59986
2,5V kaynak harcanan güç (mW)	0,106928	0,108608	0,109047	0,106943	0,108682	0,108195	0,107249	0,109331	0,108195
1,8V kaynak harcanan güç (mW)	0,663831	0,704024	0,654488	0,668644	0,699804	0,652648	0,669857	0,699325	0,65367
1,2V kaynak harcanan güç (mW)	0,319718	0,326707	0,324825	0,331632	0,343199	0,348679	0,342863	0,361694	0,361083
Toplam güç (mW)	2,69002	2,719008	2,68747	2,704168	2,737256	2,709382	2,718454	2,759569	2,722808
Gerçekleme süresi (sn)	5,2	40,6		3,2	20,2		2,8	12,4	
Harcanan enerji (J)	13,9881	110,3917		8,653337	55,29257		7,611671	34,21866	
Toplam enerji (J)	124,3798			63,94591			41,83033		

Testlerin sonucunda elde edilen veriler incelendiğinde LEON3 işlemcisinin OpenRISC işlemcisine göre hem daha az güç tükettiği hem de testleri daha kısa sürede tamamladığı görülmüştür. Sonuç olarak LEON3 işlemcisinin daha az enerji tükettiği anlaşılmıştır. Bu nedenle tasarımda LEON3 işlemcisinin kullanılmasına karar verilmiştir.

4.3 Donanıma Duyarga ve Haberleşme Cihazlarının Eklenmesi

LEON3 işlemcisine yeni IP'lerin eklenerek donanımın genişletilebildiği Bölüm 3'te anlatılmıştı. Acil durum tahliye sisteminde kişilerin bina içindeki varlığının tespiti için ultrasonik duyargaların ve BLE haberleşme cihazının kullanılacağı da aynı bölümde açıklanmıştı. Bu bölümde LEON3 işlemcisine bu cihazların nasıl eklendiği anlatılacaktır.

4.3.1 Bluetooth cihazının eklenmesi

HC06 BLE modülü piyasada kolaylıkla bulunabilen haberleşme cihazlarından biridir [89]. Bu cihaz Şekil 4.12'de görülmektedir.

Şekil 4.12 : HC06 bluetooth modülü [89].

Cihazın üzerinde 4 adet pin bulunmaktadır. Bu pinlerden ikisi besleme için kullanılmaktadır diğer iki pin ise RX, TX pinleridir. Bu pinler kullanılarak UART arayüzü üzerinden cihaz ile bağlantı kurulabilmektedir. Bu sebeple BLE cihazı için bir sürücü yazılması gerekmemiştir.

BLE modulünün varsayılan ayarları,

- Baud oranı:9600N81,
- Cihaz ismi: linvor,

- Şifre: 1234

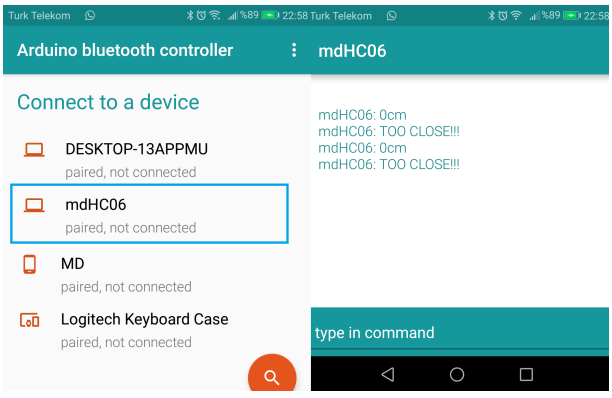
şeklindedir. AT komutları kullanılarak bu ayarlar değiştirlebilir.

BLE modülü LEON3 işlemcisine bağlanmadan önce ön ayarlamalar yapılmalıdır. Çünkü LEON3 işlemcisinin UART arayüzü 38400 baud oranına sahiptir. Cihazın varsayılan baud oranıyla eşleşme olmazsa haberleşme doğru bir şekilde gerçekleşemez. Bu amaçla cihaza UART kablosuyla bağlanılmıştır ve bir terminal programı kullanılarak aşağıdaki komutlar cihaza gönderilmiştir:

- AT, bu komut gönderilerek haberleşmenin sağlandığı test edilmiştir.
- AT+BAUD6, bu komut ile baud oranı 38400'e çekilmiştir.
- AT+NAMEmdHC06, cihazın adı "mdHC06" olacak şekilde ayarlanmıştır.
- AT+PIN0000, cihazın şifresi "0000" olarak ayarlanmıştır.

Cihazın ayarları yapıldıktan sonra sıra Atlys kartı üzerindeki değişikliğe gelir. LEON3 işlemcisinin varsayılan Atlys gerçeklemesinde UART haberleşmesi mini USB üzerinden sağlanmaktadır. BLE modülünün bağlantısını yapmak amacıyla kart üzerinde UART'ın RX, TX pinleri çevresel modül (Peripheral Module - PMOD) arayüzünün ilk iki pini olacak şekilde "leon3mp.ucf" dosyasında değiştirilmiştir. Böylece "printf" komutu gibi çıkış fonksiyonları veriyi BLE modülüne iletir.

BLE modülünün bağlantısı yapıldıktan sonra düzgün çalıştığı test edilmiştir. Bu amaçla hazırlanan donanım ile bir cep telefonu bluetooth üzerinden eşleştrilmiş ve LEON3 üzerinden bir mesaj yayınlanmıştır. Yayınlanan mesaj bir android uygulama kullanarak Şekil 4.13'teki haliyle gözlenmiştir.



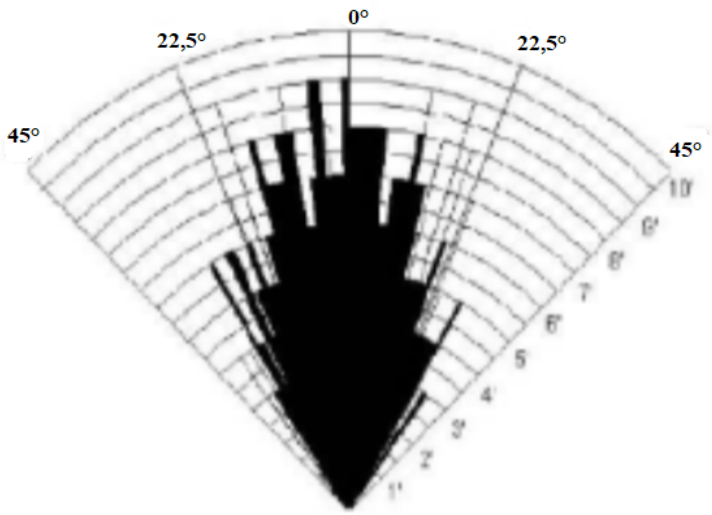
Şekil 4.13 : BLE üzerinden yollanan mesajın android uygulama ile okunması.

4.3.2 Ultrasonik duyarganın eklenmesi

Donanımın gerçeklemesinde HCSR-04 modelindeki piyasada kolaylıkla bulunabilen ultrasonik duyarga kullanılmıştır [33] ve bu duyarga Şekil 4.14'te gösterilmektedir.

Şekil 4.14 : HCSR-04 ultrasonik duyargası [33].

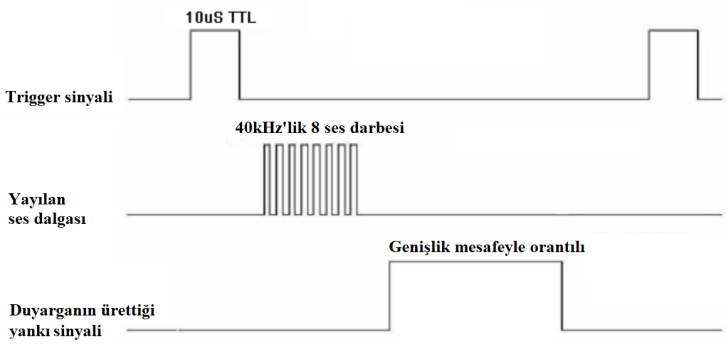
Bu cihazın kullanımı oldukça kolaydır, 2-400 cm arası uzaklıkları düzgün bir şekilde ölçebilmektedir. Etkili ölçüm aralığı yatayda 30 derecedir. Şekil 4.15'te ölçüm hassasiyetini gösteren diyagram verilmiştir.



Şekil 4.15 : HCSR-04 ölçüm hassasiyetini gösteren diyagram [33].

Bu duyarga cihazında 4 adet pin bulunmaktadır. Bunların ikisi besleme ve toprak pinleridir diğer ikisi ise yankı (echo) ve tetikleme (trigger) pinleridir. Tetikleme pininden uygulanan 10 mikrosaniye uzunluğundaki sinyal ile tetiklenen duyarga, 40kHz'lik 8 darbe halinde ultrasonik bir ses dalgası yayar. Ses dalgası iletilince yankı pini aktif hale gelerek "lojik 1" değerini alır ve bu ses dalgası herhangi bir cisme çarpıp

duyargaya geri dönünceye kadar aktif halde kalır. Ultrasonik duyarganın çalışma prensibini gösteren zaman grafiği Şekil 4.16'da verilmiştir.



Şekil 4.16 : Duyarganın zaman grafiği [33].

Santigrat cinsinden ortam sıcaklığının T olduğu durumda cismin duyargaya olan uzaklığı (x), echo pininin aktif olduğu süre bilgisi (t) kullanılarak şu şekilde hesaplanabilmektedir:

$$V = 331\sqrt{1 + (T/273)} \tag{4.1}$$

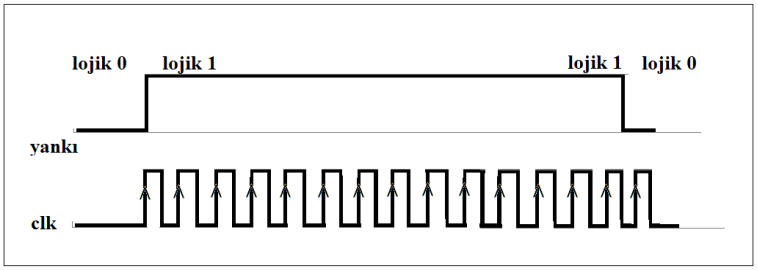
$$2x = V \times t \tag{4.2}$$

Denklem 4.1'de verilen V, sesin ortamdaki hızıdır ve bu hız ortam sıcaklığından etkilenmektedir. Yangın alarmı hallerinde kullanılacağı varsayılan duyargalar için bu bilginin değişken olarak girilmesi doğru olacaktır. Bu yüzden hesaplama için hazırlanan yazılımda bu detay göz önünde bulundurularak çalışmalara devam edilmiştir.

İşlemci üzerine bir işletim sistemi kurularak birden fazla duyarga doğrudan kontrol edilmeye çalışıldığında bu iş için birden fazla görevin yazılımla tanımlanması gerekmektedir. Bu da echo pinlerinin aktif olduğu sürelerin hesaplanmasında gecikmelere, sapmalara ve buna bağlı olarak hatalara neden olabilmektedir. Bu durumu önlemek için duyarga cihazını tetikleyen ve yankı pininin aktif olduğu süreyi hesaplayan bir modülün VHDL dili ile tasarlanarak işlemciye IP olarak eklenmesine karar verilmiştir.

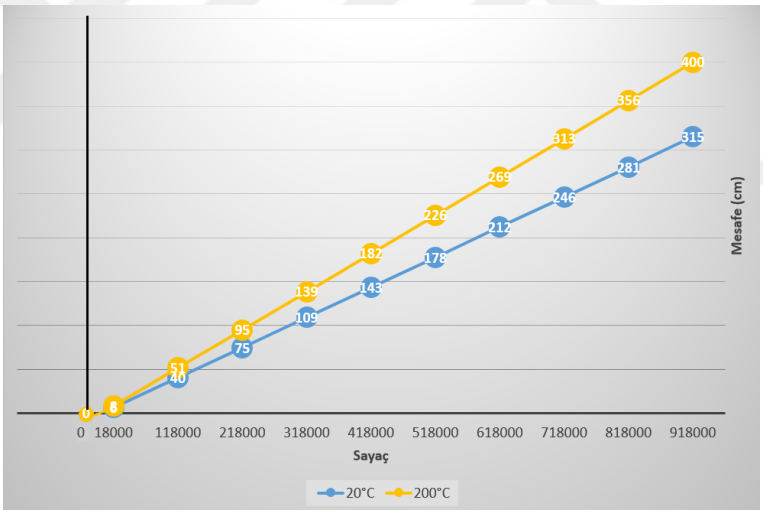
Tasarlanan bu modülde yankı sinyali, 50Mhz frekansındaki işlemci saat sinyalinin yükselen kenarında örneklenmektedir (Şekil 4.17). Yankı sinyali lojik 0'dan lojik

1’e geçtiğinde bir sayaç çalışmaya başlar, lojik 1’den lojik 0’a geçtiğinde ise sayaç değeri bir yazmaca aktarılır. Daha sonra, işletim sisteminde tanımlanacak yazılımlarda yazmaçtaki bu değer ve yukarıda verilen formüller kullanılarak mesafe bilgisi hesaplanacak ve diğer işlemlerde kullanılabilecektir.



Şekil 4.17 : Yankı sinyali ve işlemci saatinin ilişkisi.

Sayaç verisinden mesafenin hesaplanmasında sıcaklığın etkisini gösteren grafik Şekil 4.18’de verilmiştir.Örnek olarak 20 ve 200°C sıcaklıklar için mesafe hesabı yapılarak karşılaştırılmıştır.



Şekil 4.18 : Mesafenin hesaplanmasında sıcaklığın etkisi.

Sesin yayılma hızı sıcaklık arttıkça artmaktadır bu nedenle mesafe sabitken sayaç verisi azalacaktır. Yani mesafe olduğundan kısa görünecektir. Şekil 4.18’den de görüleceği üzere sıcaklık bilgisi değişken olarak tanımlanmazsa mesafe arttıkça ölçüm hatası da artacaktır. Farklı sıcaklıklar için sayaç verisi üzerinden elde edilen mesafeler Çizelge 4.12’de verilmiştir.

Bu modülün LEON3 işlemcisine IP olarak eklenebilmesi için APB veri yolunun kullanılmasına karar verilmiştir. Çünkü kullanılan duyarga basit yapılıdır ve APB

Çizelge 4.12 : Farklı sıcaklıklar için hesaplanan mesafeler.

Sayaçtaki değer	20°C için hesaplanan mesafe(cm)	50°C için hesaplanan mesafe(cm)	100°C için hesaplanan mesafe(cm)	200°C için hesaplanan mesafe(cm)
1000000	343	360	387	436
900000	309	324	348	392
800000	274	288	310	349
700000	240	252	271	305
600000	206	216	232	261
500000	171	180	193	218
400000	137	144	155	174
300000	103	108	116	131
200000	69	72	77	87
100000	34	36	39	44
0	0	0	0	0

veriyolunun özellikleri bu modülün kontrolü için yeterlidir. Tasarlanan modülün APB veri yoluna eklenmesi için şu işlemler yapılmalıdır:

- grlib/lib/opencores/ dosya yoluna eklenmek istenen IP'ye ait klasör oluşturulur.
- grlib/lib/opencores/dir.txt dosyasına IP'nin adı yazılır.
- "duyarga.vhd" ve "duyarga_amba_interface.vhd" oluşturulur. Bunlar sırasıyla duyarga modülü ve duyarganın AMBA ile haberleşmesi için gerekli arayüz modülleridir.
- grlib/lib/opencores/ dosya yolunda "vhdlsyn.txt" adında bir dosya oluşturulur ve tüm VHDL modüllerinin adı yazılır.
- grlib/lib/grlib/amba içerisindeki "devices.vhd" dosyasına oluşturduğumuz IP eklenmelidir ve numara atanmalıdır.
- Eklenmek istenen IP (duyarga modülü) en üst modül olan "leon3.vhd" içerisinde tanımlanmalıdır.

Bu işlemler tamamlandıktan sonra "leon3mp.vhd" dosyası sentezlenir ve Atlys'e yüklenir. Böylece donanım, belirlenen IoT uygulamasını gerçeklemeye hazır hale getirilmiş olur.

5. YAZILIMIN GERÇEKLENMESİ

Bu bölümde yazılımın gerçeklenmesinde izlenen adımlar anlatılmıştır. Bu adımlar sırasıyla; işletim sistemlerinin kurulumu, incelenmesi ve belirlenen hizmet için yazılımının yapılması şeklindedir.

5.1 İşletim Sistemlerinin Kurulumu ve İncelenmesi

Bölüm 3.2'de Linux kernel işletim sisteminin IoT uygulamalarına uygun olmadığı belirtilmişti. Contiki işletim sisteminin ise gerçek zaman kabiliyetinin kısıtlı olması yapılacak uygulama için uygun bulunmamıştır. RIOT ya da FreeRTOS işletim sistemlerinden birinin tasarlanan nesnede kullanılmasının daha uygun olacağı sonucu doğmuştur. FreeRTOS'un hâlihazırda LEON3 işlemcisiyle çalışacak şekilde uyarlanmış olması nedeniyle bu işletim sisteminin kullanılmasına karar verilmiştir.

Geleneksel işletim sistemlerinden Linux kernel ve gerçek zamanlı işletim sistemlerinden FreeRTOS arasındaki farkın görülebilmesi için her iki işletim sistemi de LEON3 için derlenerek Atlys kartı üzerinde çalıştırılmıştır.

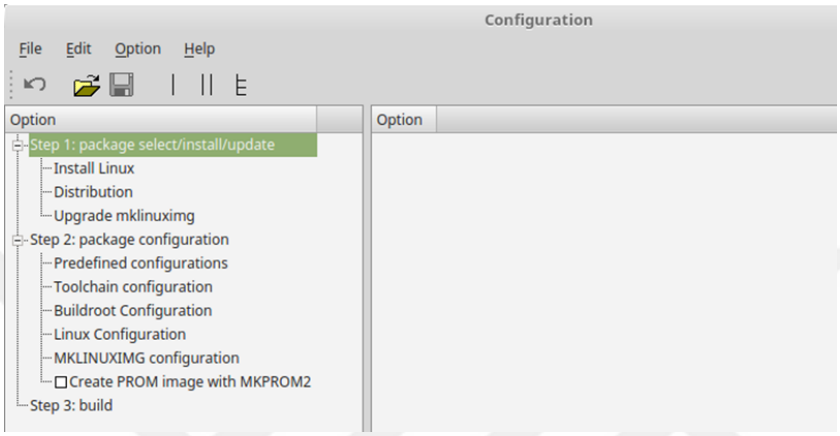
5.1.1 LEON3 üzerinde Linux kernelin çalıştırılması

LEON3 için C ve C++ kodlarının derlenmesinde BCC kullanıldığından bahsetmiştik. Linux kernel işletim sisteminin LEON3 için derlenmesinde ve Linux kernelin üzerinde çalışacak C ve C++ kodlarının derlenmesinde ise GCC derleyicisi kullanılmaktadır. Ancak bu işlemlerin yapılabilmesi için GCC'nin LEON3 işlemcisine kod üretecek şekilde çapraz derlenmiş olması gerekir [90].

Derleme işlemleri için Cobham Gaisler web sitesinden "sparc-gaisler-linux-4.9" çapraz derleyicisi indirilerek kurulmalıdır [91]. Daha sonra Linux kernel kaynak kodları ve Cobham Gaisler tarafından sunulan yardımcı yazılımları içeren "linuxbuild" klasörü aynı siteden indirilir. Kurulum işlemleri bu klasör içinde yapılacak ve en son imaj dosyaları burada oluşturulacaktır. Ardından çeşitli Linux kernel imaj dosyaları

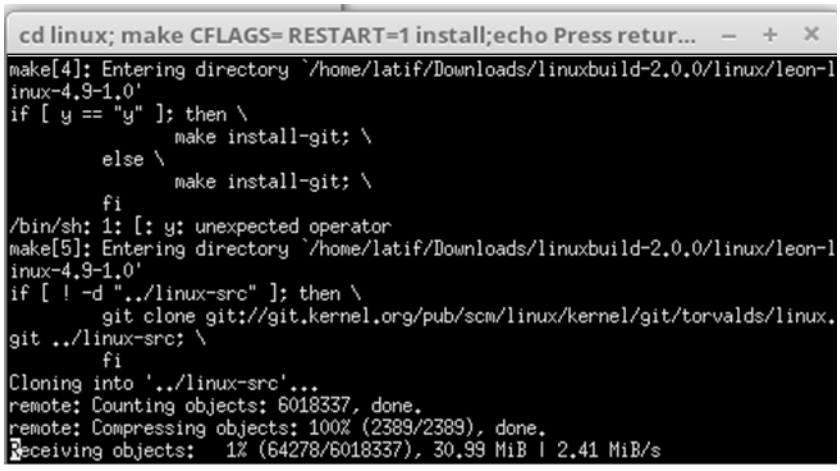
oluşturmaya yarayan ve yine Cobham Gaisler tarafından sunulan "mklinuximg" isimli araç için indirme ve kurulum işlemleri yapılır.

İndirme ve kurulum işlemlerinin tamamlanmasından sonra derleme aşamasına geçilir. Bunun için daha önce indirilip açılmış olan linuxbuild-2.0.0 klasörüne girilir ve terminal açılarak işleme başlanır. Terminal ekranında "make xconfig" komutu girilerek linuxbuild aracı çalıştırılır ve Şekil 5.1'deki pencere açılır.



Şekil 5.1 : Linuxbuild ekranı.

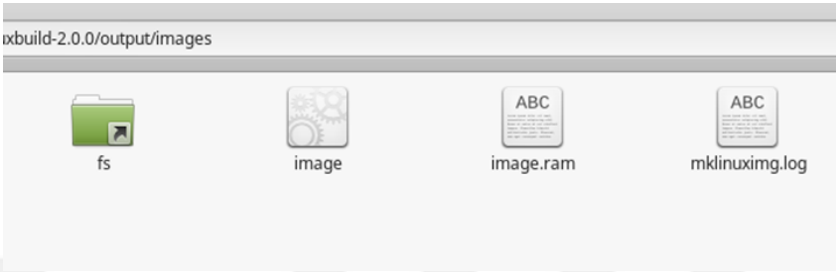
Bu pencere üzerinden ilk olarak Install Linux sekmesine tıklanıp gelen seçeneklerden "Execute Linux Installation of latest stable leon-linux" seçeneğine çift tıklanmalıdır. Bu sayede Şekil 5.2'deki terminal ekranı açılır ve araç kullanılarak internetten en son stabil Linux kernel versiyonu çekilerek güncelleştirme işlemleri otomatik olarak yapılır.



Şekil 5.2 : Linux indirme terminal ekranı.

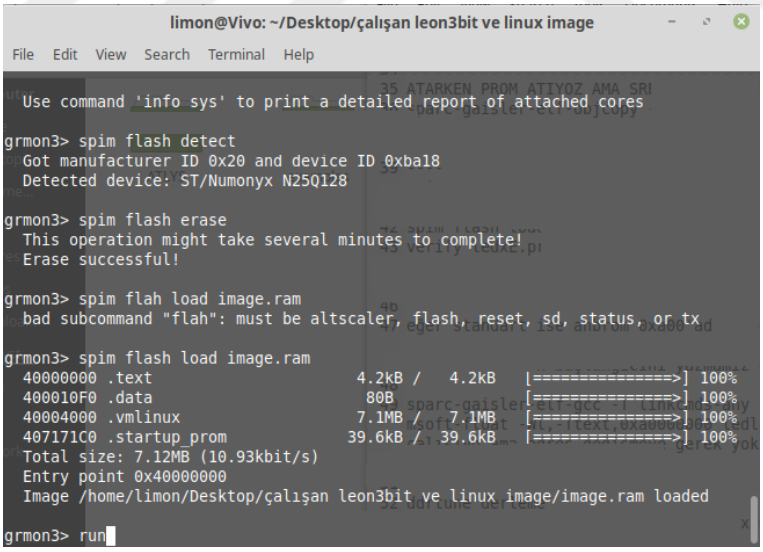
İşlem tamamlandıktan sonra terminal ekranı kapatılır ve yine Linuxbuild aracına dönülür. Araç ekranında "Step2" kısmında kernel özelleştirmeleri yapılabilmektedir.

Üçüncü aşamada ise derleme işlemlerinin yapılması sağlanarak imaj dosyası üretilir. Bunun için "Step3: build" üzerine tıklanır ve gelen seçeneklerden "Execute make build" üzerine çift tıklanır. Gelen uyarı penceresi "yes" seçilerek kapatılır ve açılan terminal üzerinden build işleminin hatasız sona ermesi beklenir. Böylece Linuxbuild aracıyla yapılacak işler tamamlanmış olur. Üretilen imaj dosyaları Şekil 5.3'te gösterilmiştir.



Şekil 5.3 : Linuxbuild aracıyla üretilen imaj dosyaları.

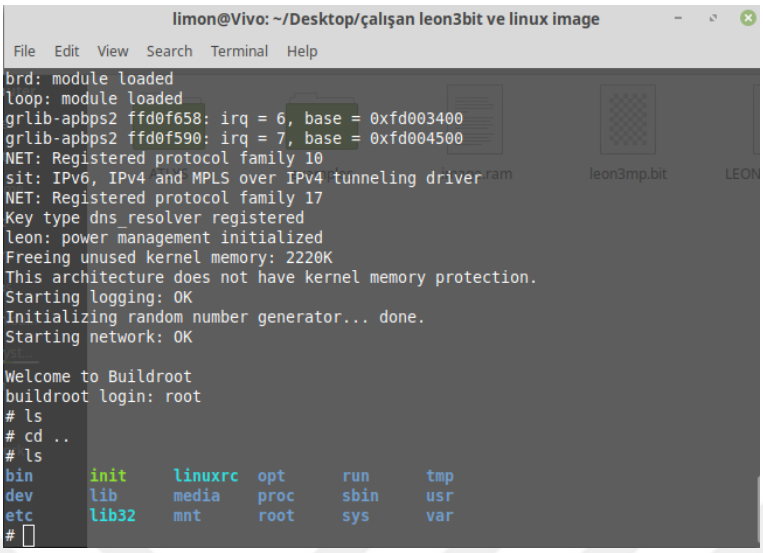
Üretilen imaj dosyası GRMON aracı kullanılarak LEON3 üzerinde çalışmak üzere karta aktarılır. Kart her başlatıldığında otomatik olarak işletim sisteminin çalışabilmesi için bu imaj dosyası kartın flash belleğine yazılmaktadır. Yüklenen dosyaya ait bilgileri içeren yükleme ekranı Şekil 5.4'te verilmiştir.



Şekil 5.4 : Linux imaj dosyasının yüklenme ekranı.

Şekil 5.4'te de görüldüğü gibi yüklenen imaj dosyası 7.1 MB büyüklüğündedir. Bu büyüklüteki bir dosyanın GRMON ile flash üzerine yazılması 20-30 dakika almaktadır. Bellek kısıtının yanında bu yükleme süresi de Linux kernel için bir dezavantaj olarak ele alınabilir. Yükleme yapıldıktan sonra GRMON üzerinde "run" komutuyla işletim

sistemi çalışmaya başlar. Aynı terminal üzerinden artık bash komutlarıyla linux kernele erişilip, sistemin çalıştığı Şekil 5.5'te görülebilmektedir.



Şekil 5.5 : Linux imaj dosyasının yüklenme ekranı.

5.1.2 LEON3 üzerinde FreeRTOS'un çalıştırılması

LEON3 için derlenmiş FreeRTOS V6.0.4 kütüphanesinin dağıtımı Bare-C çapraz derleyicisi içinde yapılmaktadır. [79]. BCC, Cobham Gaisler web sitelerinden indirilebilmektedir [92]. İndirilen BCC dosyası içerisinde "libfreertos.a" adında önceden derlenmiş kütüphane dosyası yer almaktadır. Bu kütüphane "FreeRTOSConfig.h" adındaki konfigürasyon dosyası ve çekirdeğin oluşturulması için gerekli 3 temel C dosyası kullanılarak derlenir. Bunlar sırasıyla "tasks.c", "queue.c" ve "list.c" dosyalarıdır ve işletim sisteminin temel zamanlama modeli özelliklerini sağlar. Ayrıca isteğe bağlı olarak "timers.c" ve "croutine.c" dosyaları da çekirdeğe eklenebilir. Farklı özellikteki kütüphaneler konfigürasyon dosyasının üzerinde değişiklik yapıldıktan sonra "make recompile" komutu terminal üzerinde çalıştırılarak derlenebilmektedir.

FreeRTOS API fonksiyonlarıyla yazılan C dilindeki uygulamalar, FreeRTOS'un temel dosyaları ve libfreertos.a kütüphanesi kullanılarak BCC ile derlenir böylece uygulamalar işletim sisteminin özelliklerine kavuşmuş olur. Derleme sonunda üretilen dosya GRMON aracı kullanılarak karta yüklenir. Bu dosyanın boyutu yazılan C dilindeki uygulamanın boyutuyla ve oluşturulan görev sayısıyla büyümektedir.

5.1.2.1 FreeRTOS ile ilk uygulama

FreeRTOS işletim sisteminin kullanılabilmesi için öncelikle temel API fonksiyonları incelenmelidir. Bu amaçla xTaskCreate() ve "vTaskStartScheduler()" fonksiyonları anlatılacaktır.

"xTaskCreate()" API fonksiyonu, görev oluşturma amacıyla kullanılan fonksiyondur. C diliyle tanımlanan fonksiyonların zamanlayıcıda hangi öncelikte ve hangi parametrelerle çalıştırılacağı bu fonksiyonla belirlenir. Aşağıda bu fonksiyonun temel yapısı verilmiştir:

```
deger_tipi xTaskCreate( deger_tipi pvTaskCode,
                        deger_tipi pcName,
                        deger_tipi usStackDepth,
                        deger_tipi *pvParameters,
                        deger_tipi uxPriority,
                        deger_tipi *pxCreatedTask
                      );
```

Verilen yapıda "pvTaskCode" fonksiyonun ismini, "pcName" görev için verilen tanımlayıcı ismi, "usStackDepth" görev yığıtı için ayrılacak sözcük sayısını ifade etmektedir. "pvParameters" oluşturulacak görevde kullanılacak parametreyi, "uxPriority" hangi görevin önce çağırılacağını belirler.

"vTaskStartScheduler()" API fonksiyonu, RTOS zamanlayıcıyı başlatır. Bu fonksiyon çağırıldıktan sonra RTOS çekirdeği hangi görevin ne zaman çalışacağını kontrol etmeye başlar. Zamanlayıcı başlatılınca otomatik olarak "idle" görev oluşturulur.

FreeRTOS kullanımının anlaşılabilmesi için aşağıda yaptığımız ilk örnek uygulama verilmiştir.

```
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"

#define mainCREATOR_TASK_PRIORITY ( tskIDLE_PRIORITY + 3 )

void myTask1(int *pvParameters) {
    while(1){
    printf("birinci␣%d␣!␣\n",pvParameters);
    pvParameters++;
    vTaskDelay( 100 );
    }
}
void myTask2(int *pvParameters) {
    while(1){
    printf("merhaba␣%d␣!\n",pvParameters);
    pvParameters+50;
    vTaskDelay( 200 );
    }
}
```

```
int main() {
    const char* payload="birinci";
    const char* payload2="ikinci";

        xTaskCreate( myTask1,
            (signed portCHAR *) "deneme",
            configMINIMAL_STACK_SIZE,
            payload,
            tskIDLE_PRIORITY+2, NULL );

        xTaskCreate( myTask2,
            (signed portCHAR *) "deneme2",
            configMINIMAL_STACK_SIZE,
            payload2,
            tskIDLE_PRIORITY+2, NULL );

        vCreateSuicidalTasks(mainCREATOR_TASK_PRIORITY);
        vTaskStartScheduler();
        while(1) {}
}
```

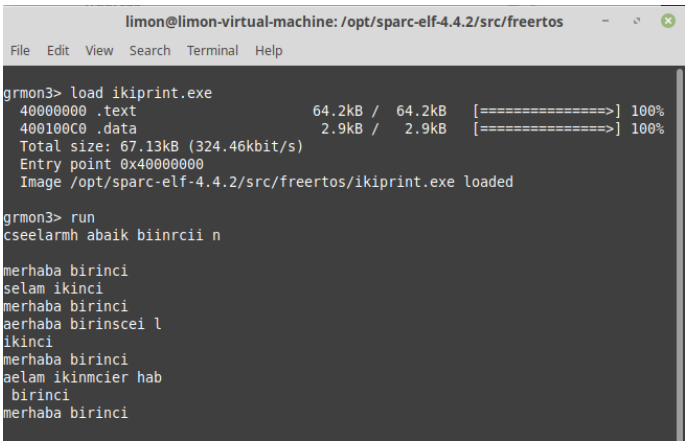
Verilen uygulamada "myTask1" ve "myTask2" isimli iki görev tanımlanmıştır. Görevlerden ilkinde "merhaba birinci!" diğerinde "selam ikinci!" mesajlarının oluşturulması beklenmektedir. Uygulamada ana fonksiyon için yüksek öncelik tanımlanmıştır, oluşturulan diğer görevlerin önceliği ise birbirine eştir.

Bu uygulama BCC ile derlenerek GRMON üzerinden karta yüklenmiştir. Derlenen uygulama 64.2kB boyutundadır, Linux kernelin kullandığı belleğin neredeyse yüzde biri kadardır. Bellek kıyaslaması yönünden Linux kernele göre avantajlı olduğu böylece ispatlanmış olmaktadır.

Yazılan uygulamaya ait ekran çıktısı Şekil 5.6'da verilmiştir. İki mesajın yazılması uygulaması Bare-C olarak derlenecek olsaydı birinci mesaj tamamlanmadan ikinci mesaj yazılmayacaktı. İşletim sisteminin olduğu durumda ise Şekil 5.6'da da görüldüğü gibi ekrana birbirine karışık mesajlar yazılmıştır. Bunun nedeni işlemler arasında herhangi bir önceliğin atanmamış olmasıdır. Buradan da anlaşılacağı üzere işletim sistemi işleri aynı anda yürütmeye çalışmaktadır. Bu uygulamanın sonunda FreeRTOS işletim sisteminin doğru olarak kurulduğunu ve çalıştığı anlaşılmaktadır.

5.1.2.2 Çoklu görevin enerji etkinliğine etkisi

FreeRTOS işletim sisteminin kullandığı zamanlama modeli işlerin paralel olarak yürütülmesini bu şekilde zaman ve enerjiyi etkin olarak kullanmayı hedefler. Bu sayede işlemci üzerinde, FreeRTOS ile hazırlanan programların işletim sistemi olmadan yazılan programlardan daha hızlı çalışması beklenir.

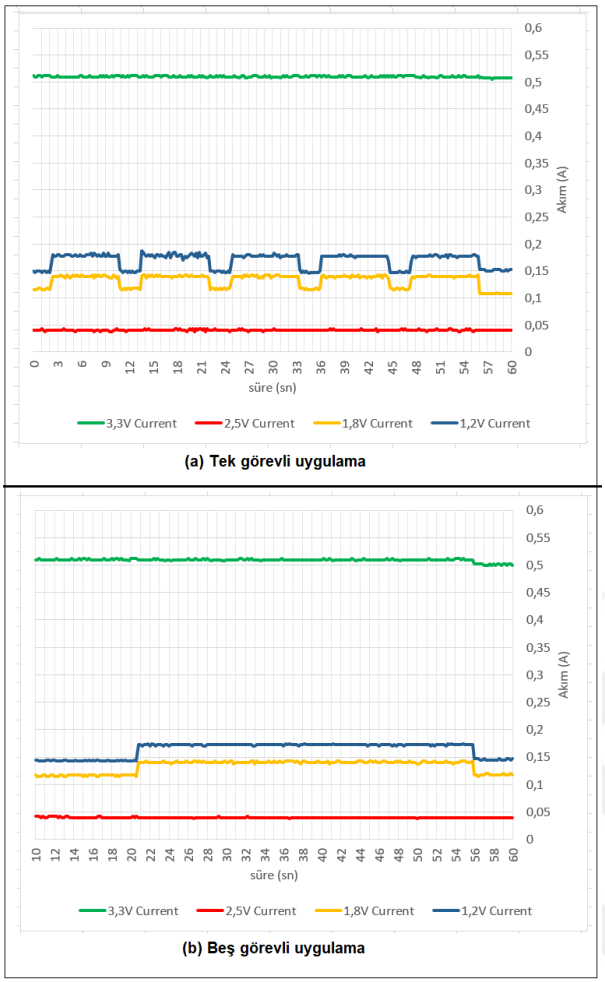


Şekil 5.6 : FreeRTOS ile yapılan ilk uygulamanın ekran çıktısı.

Ayrıca FreeRTOS çekirdeğinin üzerinde ihtiyaca göre görevlerin eklenip çıkarılabilmesi ve bu şekilde görevlerin asgari sayıda tutulmasıyla enerji etkinliği arttırılabilir. Çünkü görev sayısı arttıkça görevlerin tamamlanma süresi bununla beraber enerji tüketimi artmaktadır. Yapılan bir çalışmada FreeRTOS'un görev sayısıyla değişen iş tamamlanma süresini gösteren grafik Şekil 5.7'de verilmiştir [53].

Şekil 5.7 : Görev sayısıyla gerçekleme süresinin ilişkisi [53].

Bu karşılaştırmanın doğruluğunu ispatlamak için öncelikle FreeRTOS üzerinde Nsichneu-NOP testinin görev olarak tanımlandığı bir uygulama çalıştırılmıştır. Daha sonra yine beşi de Nsichneu-NOP testinden oluşan beş görevli bir uygulama hazırlanmıştır. Tek görevden ve beşli görevden oluşan uygulamalara ait tamamlanma süreleri-ortalama akım tüketimi grafikleri ADEPT kullanılarak Şekil 5.8'deki gibi gözlenmiştir. Bu testlere ve aynı testin işletim sistemi yokken gerçeklendiği durumuna ait güç, zaman ve enerji verileri Çizelge 5.1'de paylaşılmıştır. Elde edilen veriler doğrultusunda işletim sistemi varken çoklu görevin tanımlandığı testte daha az enerji harcandığı doğrulanmıştır.



Şekil 5.8 : Akım - gerçeklenme süresi grafikleri: (a) Tek görevli uygulama, (b) Beş görevli uygulama.

Çizelge 5.1 : Çoklu görev sayısının enerji tüketimine etkisini gösteren test sonuçları.

Çalıştırma biçimi	İşletim sistemi yok	Tekli görev	Beşli görev
Test aşaması	Kıyaslama programı	Kıyaslama programı	Kıyaslama programı
3,3V kaynak harcanan güç (mW)	1,646591	1,683861	1,682983
2,5V kaynak harcanan güç (mW)	0,099663	0,100251	0,099925
1,8V kaynak harcanan güç (mW)	0,285957	0,251899	0,25294
1,2V kaynak harcanan güç (mW)	0,204632	0,212915	0,20673
Toplam güç (mW)	2,236843	2,248926	2,242584
Gerçekleme süresi (sn)	8,4x5= 42	8,6x5= 43	35,2
Harcanan toplam enerji (J)	93,9474	96,7038	78,93896

5.2 Kişilerin Bina İçindeki Varlığının Tespiti İçin Uygulamanın Yazılması

Bölüm 4'te hazırlanan donanım kullanılarak kişilerin bina içindeki varlığını tespit etmek için FreeRTOS işletim sistemi üzerinde 5 görevden oluşan bir uygulama yazılmıştır.

Tanımlanan görevlerden ilki olan "temp_task" çağırıldığında, Denklem 4.1 ve Denklem 4.2'den faydalanılarak bir katsayı hesabı yapılmaktadır. Hesaplanan katsayı belirli bir yazmaç adresine kaydedilmektedir. Bu katsayı ortam sıcaklığına bağlı olarak değişmektedir. Uygulamada bu sıcaklık bilgisi el ile girilmektedir. Tanımlanan "temp_task" görevinin C kodu aşağıda verilmiştir:

```
void temp_task( void *pvParameters ) {
    int k,result,n,T,freq2;
    int freq=50000000;
    int sicak=20;

    T=sicak*10000;
    n=109561*(10000+(T/273));
        freq2=freq*2;
        int temp,sqt;
        sqt=n/2;
        temp=0;
    while(sqt!=temp)
        {
         temp=sqt;
         sqt=(n/temp+temp)/2;
        }
    result= sqt;
    k= freq2/result;
    *temperature = k;
    printf("katsayi k=%d \n",*temperature);
    vTaskDelay( 400 );
}
```

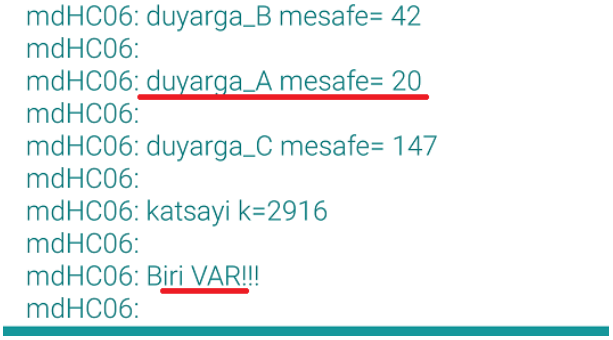
Her bir duyarga için bir görev olmak üzere; tanımlanan "sense_task1", "sense_task2" ve "sense_task3" görevlerinin çağırılmasıyla ultrasonik duyargalar çalıştırılarak elde edilen sayaç verileri belirli bellek yazmaçlarına kaydedilmektedir. Daha sonra bu sayaç verileri, ilk görevden elde edilen katsayıya bölünerek santimetre cinsinden mesafe hesabı yapılmaktadır. Son olarak bu mesafeler oda boyutuyla karşılaştırılır. Ölçülen mesafe oda boyutundan küçük ise "11111111" değil ise "00000000" verisi üretilir. Böylece her bir duyarganın ayrı ayrı bir nesneyi görüp görmediği kaydedilmiş olur. Aşağıda "sense_task1" görevinin C kodu verilmiştir:

```
void sense_task1( void *pvParameters) {
    while (1) {
    int x,r1,k;
    k= *temperature;
    x=*readLEDS;
    r1= x/k;
    printf("duyarga_A mesafe= %d \n",r1);
       if(r1<40){
           *senseA=0x11111111;
           printf("senseA= %p \n",*senseA); }
        else {
           *senseA=0x00000000;
           printf("senseA= %p \n",*senseA);}
    vTaskDelay( 400 );
 } }
```

Son olarak tanımlanan "task_denetle" görevinde ise duyargalardan üretilen veriler karşılaştırılarak odada birinin var olup olmadığı mesajı üretilir. Üretilen bu mesaj "printf" komutuyla yazdırılır bu komutla yazdırılan mesaj donanımın tasarımı gereği BLE modülü ile yayınlanacaktır. "task_denetle" görevinin C kodu aşağıda verilmiştir:

```
void task_denetle(  void *pvParameters ) {
        int r1,r2,r3;
    if (*senseA| *senseB | *senseC)
        { printf("Biri VAR!!! \n");}
    else
        {printf("kimse yok \n");}
    vTaskDelay( 400 ); }
```

Görevlerin ana fonksiyon içerisinde çağırılmasıyla hazırlanan yazılım derlenerek LEON3 işlemcisine aktarılmıştır. Üç duyarganın bağlı olduğu sistem üzerinde çalıştırılan bu uygulamayla yapılan testte 40cm'den daha yakında bir cismin var olup olmadığı gözlenmiştir. Her bir duyargadan ölçülen mesafe bilgileri ve buna göre 40cm'den daha yakında bir cisim varsa "Biri VAR!!!" mesajı BLE modülüyle yayınlanmıştır. Şekil 5.9'da yayınlanan mesajın ekran görüntüleri verilmiştir. Böylece sistemin doğru bir şekilde çalıştığı görülmüştür.



Şekil 5.9 : Kişilerin oda içerisindeki varlığının tespiti uygulamasında görüntülenen mesaj.

6. SONUÇLAR VE GELECEK ÇALIŞMALAR

Tez kapsamında nesnelerin interneti uygulamalarında kullanılmak üzere düşük güç tüketimli duyarga nesnesi tasarımı ve FPGA üzerinde gerçeklemesi yapılmıştır.

Nesnelerin internetinin heterojen bir yapıda olması ve standart bir tasarım modeline sahip olmaması nedeniyle öncelikle tasarlanacak nesne için örnek bir kullanım alanı belirlenmiştir. Bu örnek uygulamadan yola çıkarak nesnenin düşük güç tüketimli ve gerçek zaman kabiliyetine sahip olması gerektiği sonucuna ulaşılmıştır.

Bu amaçla açık kaynak kodlu yazılımsal işlemcilerden LEON3 ve OpenRISC işlemcileri Atlys geliştirme kartı üzerinde gerçeklenerek güç kıyaslaması yapılmıştır. Kıyaslama için yaygın olarak kullanılan algoritmalar seçilmiş ve alınan sonuçlar paylaşılmıştır. Kıyaslama sonucunda LEON3 işlemcisinin hem daha az güç tükettiği hem de işlemleri daha hızlı gerçekleştirdiği görülmüştür. Böylece tasarımda LEON3 işlemcisinin kullanılmasına karar verilmiştir.

Sonraki adımda LEON3 işlemcisine bir ultrasonik duyarga modülü ve düşük güç tüketimli bir haberleşme cihazı olan BLE modülü eklenmiştir. Böylece prototip olarak tasarlanan duyarga nesnesi, belirlenen IoT uygulamasını gerçeklemek için hazır hale getirilmiştir.

Tasarımda işlemci üzerinde çalıştırılmak üzere FreeRTOS işletim sisteminin kullanılmasına karar verilmiştir. Bu işletim sistemi öncelikle geleneksel işletim sistemlerinden Linux-kernel ile karşılaştırılmıştır. Karşılatırma sonucunda FreeRTOS'un hem daha az güç tükettiği ve daha az yer kapladığı hem de daha hızlı gerçeklenebildiği görülmüştür. Böylece tasarımda FreeRTOS seçiminin uygunluğu ispatlanmıştır. LEON3 işlemcisi üzerine bu işletim sistemi kurularak çalışmalara devam edilmiştir.

Son olarak belirlenen IoT uygulaması FreeRTOS ile hazırlanarak işlemci üzerine yüklenmiştir. Yapılan testler ile sistemin doğru bir şekilde çalıştığı görülmüştür.

Yapılan bu çalışmayla tasarımcılara IoT için düğüm tasarımında izleyebilecekleri bir kaynak sağlanmıştır. Ayrıca tasarımcılara açık kaynak işlemcilerin güç ve enerji kıyaslamasında kullanabilecekleri teknik bir analiz sunulmuştur.

Gelecek çalışmalarda tasarlanan duyarga düğümünün optimizasyonu yapılarak enerji tüketimi düşürülmeye çalışılacaktır. Tasarlanan prototipin yeniden programlanabilir oluşu sayesinde tasarım farklı uygulamalarda da kullanılmak üzere özelleştirilecektir ve ihtiyaç olması halinde üzerine güvenlik modülleri de eklenecektir.

KAYNAKLAR

[1] **Hahm, O.**, **Baccelli, E.**, **Petersen, H. ve Tsiftes, N.** (2016). Operating systems for low-end devices in the internet of things: a survey, *IEEE Internet of Things Journal*, *3*(5), 720–734.

[2] **Coelho, C.**, **Silva, D. ve Fernandes, A.** (1998). Hardware-software codesign of embedded systems, *Proceedings. XI Brazilian Symposium on Integrated Circuit Design (Cat. No. 98EX216)*, IEEE, s.2–8.

[3] **Ernst, R.** (1998). Codesign of embedded systems: Status and trends, *IEEE Design & Test of Computers*, *15*(2), 45–54.

[4] **SECTOR, S. ve ITU, O.** (2012). SERIES Y: GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS AND NEXT-GENERATION NETWORKS Next Generation Networks–Frameworks and functional architecture models, *International Telecommunication Union: Geneva, Switzerland*.

[5] **Al-Fuqaha, A.**, **Guizani, M.**, **Mohammadi, M.**, **Aledhari, M. ve Ayyash, M.** (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications, *IEEE Communications Surveys Tutorials*, *17*(4), 2347–2376.

[6] **Gomez, C.**, **Oller, J. ve Paradells, J.** (2012). Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology, *Sensors*, *12*(9), 11734–11753.

[7] **Farahani, S.** (2011). *ZigBee wireless networks and transceivers*, Newnes.

[8] **Want, R.** (2011). Near field communication, *IEEE Pervasive Computing*, (3), 4–7.

[9] **Ferro, E. ve Potorti, F.** (2005). Bluetooth and Wi-Fi wireless protocols: a survey and a comparison, *IEEE Wireless Communications*, *12*(1), 12–26.

[10] **Mulligan, G.** (2007). The 6LoWPAN architecture, *Proceedings of the 4th workshop on Embedded networked sensors*, ACM, s.78–82.

[11] **Al-Sarawi, S.**, **Anbar, M.**, **Alieyan, K. ve Alzubaidi, M.** (2017). Internet of Things (IoT) communication protocols: Review, *2017 8th International Conference on Information Technology (ICIT)*, s.685–690.

[12] **Url-1**, `https://www.iotforall.com/what-is-the-cloud/`, alındığı tarih: 07.04.2019.

[13] **Rao, B.P.**, **Saluia, P.**, **Sharma, N.**, **Mittal, A. ve Sharma, S.V.** (2012). Cloud computing for Internet of Things & sensing based applications, *2012 Sixth International Conference on Sensing Technology (ICST)*, IEEE, s.374–380.

[14] **Bryant, R.**, **Katz, R.H. ve Lazowska, E.D.**, (2008), Big-data computing: creating revolutionary breakthroughs in commerce, science and society.

[15] **Khan, R.**, **Khan, S.U.**, **Zaheer, R. ve Khan, S.** (2012). Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges, *2012 10th International Conference on Frontiers of Information Technology*, s.257–260.

[16] **Url-2**, `https://www.endustri40.com/`, alındığı tarih: 07.04.2019.

[17] **Cook, D.J.**, **Crandall, A.S.**, **Thomas, B.L. ve Krishnan, N.C.** (2013). CASAS: A smart home in a box, *Computer*, *46*(7), 62–69.

[18] **Dohr, A.**, **Modre-Opsrian, R.**, **Drobics, M.**, **Hayn, D. ve Schreier, G.** (2010). The internet of things for ambient assisted living, *2010 seventh international conference on information technology: new generations*, Ieee, s.804–809.

[19] **Zanella, A.**, **Bui, N.**, **Castellani, A.**, **Vangelista, L. ve Zorzi, M.** (2014). Internet of Things for Smart Cities, *IEEE Internet of Things Journal*, *1*(1), 22–32.

[20] **Evans, D.** (2011). The internet of things: How the next evolution of the internet is changing everything, *CISCO white paper*, *1*(2011), 1–11.

[21] **Bormann, C.**, **Ersue, M. ve Keranen, A.** (2014). Terminology for constrained-node networks, **Teknik Rapor**.

[22] **Min, R.**, **Bhardwaj, M.**, **Cho, S.H.**, **Ickes, N.**, **Shih, E.**, **Sinha, A.**, **Wang, A. ve Chandrakasan, A.** (2002). Energy-centric enabling technologies for wireless sensor networks, *IEEE wireless communications*, *9*(4), 28–39.

[23] **Dong, W.**, **Chen, C.**, **Liu, X. ve Bu, J.** (2010). Providing OS support for wireless sensor networks: Challenges and approaches, *IEEE Communications Surveys & Tutorials*, *12*(4), 519–530.

[24] **Saraswat, L. ve Yadav, P.S.** (2010). A comparative analysis of wireless sensor network operating systems, *International Journal of Engineering and Technoscience*, *1*(1), 41–47.

[25] **Jedermann, R.**, **Pötsch, T. ve Lloyd, C.** (2014). Communication techniques and challenges for wireless food quality monitoring, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *372*(2017), 20130304.

[26] **Çavdar, T. ve Öztürk, E.** (2018). Nesnelerin interneti için yeni bir mimari tasarımı, *Sakarya University Journal of Science*, *22*(1), 39–48.

[27] **Laplante, P.A.** (1996). *Real-time systems design and analysis: an engineer's handbook*, Wiley-IEEE Press.

[28] **Milenković, A.**, **Otto, C. ve Jovanov, E.** (2006). Wireless sensor networks for personal health monitoring: Issues and an implementation, *Computer communications*, *29*(13-14), 2521–2533.

[29] **Macedo, D.**, **Guedes, L.A. ve Silva, I.** (2014). A dependability evaluation for Internet of Things incorporating redundancy aspects, *Proceedings of the 11th IEEE International Conference on Networking, Sensing and Control*, IEEE, s.417–422.

[30] **Maalel, N.**, **Natalizio, E.**, **Bouabdallah, A.**, **Roux, P. ve Kellil, M.** (2013). Reliability for emergency applications in internet of things, *2013 IEEE International Conference on Distributed Computing in Sensor Systems*, IEEE, s.361–366.

[31] **Gokceli, S.**, **Zhmurov, N.**, **Kurt, G.K. ve Ors, B.** (2017). IoT in action: design and implementation of a building evacuation service, *Journal of Computer Networks and Communications*, *2017*.

[32] **Mendelson, E.**, (2011), System and method for providing alarming notification and real-time, critical emergency information to occupants in a building or emergency designed area and evacuation guidance system to and in the emergency exit route, uS Patent 7,924,149.

[33] **Technologies, C.**, `http://web.eece.maine.edu/~zhu/book/lab/`, alındığı tarih: 16.04.2019.

[34] **Kandemir, M.**, **Vijaykrishnan, N.**, **Irwin, M.J. ve Ye, W.** (2000). Influence of compiler optimizations on system power, *Proceedings of the 37th Annual Design Automation Conference*, ACM, s.304–307.

[35] **Bara, L.**, **Boncalo, O. ve Marcu, M.** (2015). Hardware support for performance measurements and energy estimation of OpenRISC processor, *2015 IEEE 10th Jubilee International Symposium on Applied Computational Intelligence and Informatics*, IEEE, s.399–404.

[36] **Baccelli, E.**, **Hahm, O.**, **Gunes, M.**, **Wahlisch, M. ve Schmidt, T.C.** (2013). RIOT OS: Towards an OS for the Internet of Things, *2013 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, IEEE, s.79–80.

[37] **Tong, J.G.**, **Anderson, I.D. ve Khalid, M.A.** (2006). Soft-core processors for embedded systems, *2006 International Conference on Microelectronics*, IEEE, s.170–173.

[38] **Dorta, T.**, **Jiménez, J.**, **Martín, J.L.**, **Bidarte, U. ve Astarloa, A.** (2010). Reconfigurable multiprocessor systems: a review, *International Journal of Reconfigurable Computing*, *2010*, 7.

[39] **Abdelsamea, M.H.A.**, **Zorkany, M. ve Abdelkader, N.** (2016). Real time operating systems for the Internet of things, vision, architecture and research directions, *2016 World Symposium on Computer Applications & Research (WSCAR)*, IEEE, s.72–77.

[40] **Pauls, J.** (1987). Calculating evacuation times for tall buildings, *Fire Safety Journal*, *12*(3), 213–236.

[41] **Proulx, G.** (1995). Evacuation time and movement in apartment buildings, *Fire safety journal*, *24*(3), 229–246.

[42] **Liu, S.j. ve Zhu, G.q.** (2014). The application of GIS and IOT technology on building fire evacuation, *Procedia engineering*, *71*, 577–582.

[43] **Chu, L. ve Wu, S.J.** (2012). A real-time fire evacuation system with cloud computing, *Journal of Convergence Information Technology*, *7*(7).

[44] **Töreyin, B.U.**, **Dedeoğlu, Y.**, **Güdükbay, U. ve Cetin, A.E.** (2006). Computer vision based method for real-time fire and flame detection, *Pattern recognition letters*, *27*(1), 49–58.

[45] **Healey, G.**, **Slater, D.**, **Lin, T.**, **Drda, B. ve Goedeke, A.D.** (1993). A system for real-time fire detection, *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, s.605–606.

[46] **Liedtke, J.** *ve diğerleri* (1995). On-kernel construction, *Proceedings of the 15th ACM Symposium on OS Principles*, s.237–250.

[47] **Masmano, M.**, **Ripoll, I.**, **Crespo, A. ve Real, J.** (2004). TLSF: A new dynamic memory allocator for real-time systems, *Proceedings. 16th Euromicro Conference on Real-Time Systems, 2004. ECRTS 2004.*, IEEE, s.79–88.

[48] **Kernel.org**, `https://www.kernel.org/`, alındığı tarih: 08.04.2019.

[49] **Contiki**, `http://www.contiki-os.org/`, alındığı tarih: 07.04.2019.

[50] **RIOT**, `https://riot-os.org/`, alındığı tarih: 07.04.2019.

[51] **FreeRTOS**, `https://www.freertos.org/`, alındığı tarih: 07.04.2019.

[52] **Stankovic, J.A.**, **Spuri, M.**, **Di Natale, M. ve Buttazzo, G.C.** (1995). Implications of classical scheduling results for real-time systems, *Computer*, *28*(6), 16–25.

[53] **Thomas, S.S.**, **Thomas, S.A.**, **Paul, J. ve Shibu, K.K.** (2018). An Intelligent Adaptive Scheduler for Operating Systems Experimented Using FreeRTOS, *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, IEEE, s.1592–1597.

[54] **Deering, S. ve Hinden, R.** (2017). Internet protocol, version 6 (IPv6) specification, **Teknik Rapor**.

[55] **Winter, T.**, **Thubert, P.**, **Brandt, A.**, **Hui, J.**, **Kelsey, R.**, **Levis, P.**, **Pister, K.**, **Struik, R.**, **Vasseur, J.P. ve Alexander, R.** (2012). RPL: IPv6 routing protocol for low-power and lossy networks, **Teknik Rapor**.

[56] **Shelby, Z.**, **Hartke, K. ve Bormann, C.** (2014). The constrained application protocol (CoAP), **Teknik Rapor**.

[57] **Association, I.S.** *ve diğerleri* (2012). IEEE Standard for Local and metropolitan area networks—Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer. IEEE Std 802.15. 4e-2012 (Amendment to IEEE Std 802.15. 4-2011), *IEEE Computer Society: New York, NY, USA*.

[58] **Dunkels, A.**, **Gronvall, B. ve Voigt, T.** (2004). Contiki-a lightweight and flexible operating system for tiny networked sensors, *29th annual IEEE international conference on local computer networks*, IEEE, s.455–462.

[59] **Will, H.**, **Schleiser, K. ve Schiller, J.** (2009). A real-time kernel for wireless sensor networks employed in rescue scenarios, *2009 IEEE 34th Conference on Local Computer Networks*, IEEE, s.834–841.

[60] **Tanenbaum, A.S. ve Woodhull, A.S.** (1997). *Operating systems: design and implementation*, cilt 68, Prentice Hall Englewood Cliffs.

[61] **Brown, S.D.**, **Francis, R.J.**, **Rose, J. ve Vranesic, Z.G.** (2012). *Field-programmable gate arrays*, cilt180, Springer Science & Business Media.

[62] **Hartenstein, R.W.** (1987). *Hardware description languages*, North-Holland.

[63] **Chang, W.**, (2001), Embedded configurable logic ASIC, uS Patent 6,260,087.

[64] **Xilinx**, `https://www.xilinx.com/products/intellectual-property.html`, alındığı tarih: 14.04.2019.

[65] **Lampret, D.**, **Chen, C.M.**, **Mlinar, M.**, **Rydberg, J.**, **Ziv-Av, M.**, **Ziomkowski, C.**, **McGary, G.**, **Gardner, B.**, **Mathur, R. ve Bolado, M.** (2003). Openrisc 1000 architecture manual, *Description of assembler mnemonics and other for OR1200*.

[66] **Lampret, D.** (2001). OpenRISC 1200 IP core specification, *September June*.

[67] **Herveille, R.** *ve diğerleri* (2002). WISHBONE system-on-chip (SoC) interconnection architecture for portable IP cores, *OpenCores Organization*.

[68] **Gaisler**, `https://www.gaisler.com/doc/leon3_product_sheet.pdf`, alındığı tarih: 14.04.2019.

[69] **Gaisler**, `https://www.gaisler.com/products/grlib/grlib.pdf`, alındığı tarih: 14.04.2019.

[70] **Frank, R.**, **Bronzi, W.**, **Castignani, G. ve Engel, T.** (2014). Bluetooth Low Energy: An alternative technology for VANET applications, *2014 11th annual conference on wireless on-demand network systems and services (WONS)*, IEEE, s.104–107.

[71] **Decuir, J.** (2014). Introducing Bluetooth Smart: Part 1: A look at both classic and new technologies., *IEEE Consumer Electronics Magazine*, *3*(1), 12–18.

[72] **Fraden, J.** (2004). *Handbook of modern sensors: physics, designs, and applications*, Springer Science & Business Media.

[73] **Digilent**, https://reference.digilentinc.com/reference/programmable-logic/atlys/reference-manual, alındığı tarih: 16.04.2019.

[74] **Xilinx**, https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools.html, alındığı tarih: 14.04.2019.

[75] **Official, D.**, (2008), Adept2, https://store.digilentinc.com/digilent-adept-2-download-only/, [Online; accessed 19-January-2019].

[76] **OpenCores**, https://opencores.org/projects/orpsoc, alındığı tarih: 10.04.2019.

[77] **Akçay, L.**, **Tükel, M. ve Örs, S.B.** (2016). Implementation of an OpenRISC based SoC and Linux Kernel installation on FPGA, *2016 24th Signal Processing and Communication Application Conference (SIU)*, s.1969–1972.

[78] **Akcay, L.**, **Tukel, M. ve Ors, B.** (2017). Design and implementation of an OpenRISC system-on-chip with an encryption peripheral, *2017 European Conference on Circuit Theory and Design (ECCTD)*, s.1–4.

[79] **Gaisler**, https://www.gaisler.com/doc/bcc.pdf, alındığı tarih: 23.04.2019.

[80] **Gaisler**, https://www.gaisler.com/doc/grmon-eval/grmon3.pdf, alındığı tarih: 23.04.2019.

[81] **Laopoulos, T.**, **Neofotistos, P.**, **Kosmatopoulos, C. ve Nikolaidis, S.** (2003). Measurement of current variations for the estimation of software-related power consumption [embedded processing circuits], *IEEE Transactions on instrumentation and measurement*, *52*(4), 1206–1212.

[82] **Tiwari, V.**, **Malik, S. ve Wolfe, A.** (1994). Power analysis of embedded software: a first step towards software power minimization, *Proceedings of the 1994 IEEE/ACM international conference on Computer-aided design*, IEEE Computer Society Press, s.384–390.

[83] **Xian, C.**, **Cai, L. ve Lu, Y.H.** (2007). Power measurement of software programs on computers with multiple I/O components, *IEEE Transactions on Instrumentation and Measurement*, *56*(5), 2079–2086.

[84] **Nakutis, Z.** (2009). Embedded systems power consumption measurement methods overview, *MATAVIMAI*, *2*(44), 29–35.

[85] **Becker, J.**, **Huebner, M. ve Ullmann, M.** (2003). Power estimation and power measurement of Xilinx Virtex FPGAs: trade-offs and limitations, *16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings.*, s.283–288.

[86] **Bara, L.**, **Boncalo, O. ve Marcu, M.** (2015). Hardware support for performance measurements and energy estimation of OpenRISC processor, *2015 IEEE 10th Jubilee International Symposium on Applied Computational Intelligence and Informatics*, IEEE, s.399–404.

[87] **Technology, L.**, `https://www.analog.com/media/en/technical-documentation/data-sheets/2481fd.pdf`, alındığı tarih: 30.06.2019.

[88] **Gustafsson, J.**, **Betts, A.**, **Ermedahl, A. ve Lisper, B.** (2010). The Mälardalen WCET benchmarks: Past, present and future, *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[89] **Guangzhou, H.** (2011). Information Technology Co., Ltd, *HC-06 Module Data Sheet*.

[90] **Gaisler**, `https://www.gaisler.com/doc/leon-linux-overview.pdf`, alındığı tarih: 20.04.2019.

[91] **Gaisler**, `https://www.gaisler.com/index.php/downloads/linux`, alındığı tarih: 23.04.2019.

[92] **Gaisler**, `https://www.gaisler.com/anonftp/bcc/bin/`, alındığı tarih: 23.04.2019.

ÖZGEÇMİŞ

Ad Soyad: Mehmet Onur DEMİRTÜRK

Doğum Tarihi ve Yeri: 27.10.1991 - Mulhouse(F)

E-Posta: demirturk16@itu.edu.tr

Adres: İTÜ Elektronik ve Haberleşme Mühendisliği Bölümü, 34469, Maslak/İstanbul

ÖĞRENİM DURUMU:

- **Lisans:** 2014, Hacettepe Üniversitesi,Mühendislik Fakültesi, Elektrik-Elektronik Mühendisliği Bölümü
- **Y. Lisans:** 2019,İstanbul Teknik Üniversitesi, Elektronik ve Haberleşme Mühendisliği A.B.D, Elektronik Programı.