ESP32 KULLANARAK FREERTOS İLE ÇOK FONKSİYONLU ATÖLYE YARDIMCISI

Hazırlayan

Ömer Faruk Acar

Ağu 2025

ANKARA-2025

İçindekiler

1. Giriş

Gelişen gömülü sistemler ve Nesnelerin İnterneti (IoT) teknolojileri, günlük yaşamdan profesyonel çalışma alanlarına kadar her alanda çevresel izleme ve optimizasyon ihtiyacını artırmıştır. Özellikle hobi atölyeleri, elektronik laboratuvarları veya kişisel çalışma alanları gibi ortamlarda; sıcaklık, nem, aydınlatma ve gürültü seviyelerinin kontrolü, hem cihazların uzun ömürlülüğü hem de kullanıcının üretkenliği ve konforu için hayati öneme sahiptir. Bu bağlamda, karmaşık ve dağınık sensör çözümleri yerine, çok sayıda ölçüm fonksiyonunu tek ve kompakt bir yapıda toplayan pratik araçlara olan talep belirginleşmiştir.

Bu ihtiyaca yanıt olarak tasarlanan ve bu raporun odak noktasını oluşturan proje, **"ESP32 Kullanarak FreeRTOS ile Çok Fonksiyonlu Atölye Yardımcısı"** adını taşımaktadır. Proje, popüler bir mikrodenetleyici olan **ESP32**'nin güçlü donanım yeteneklerini, gömülü sistemler için ideal bir gerçek zamanlı işletim sistemi olan **FreeRTOS** ile birleştirerek çok amaçlı bir çevresel izleme ve kalibrasyon aracı geliştirmeyi hedeflemektedir.

2. Projenin Amacı ve Temel Vizyonu

Projenin temel vizyonu, üreticiler (makerlar), hobi sahipleri ve DIY (Kendin Yap) meraklıları için, atölye koşullarını anlık olarak izleyebilecekleri, güvenilir ve kullanımı kolay bir dijital "İsviçre Çakısı" işlevi gören bir araç sunmaktır.

Temel amaçlar şunlardır:

- **Çoklu Sensör Entegrasyonu:** Tek bir cihazda sıcaklık, nem (DHT11), ışık seviyesi (LDR) ve ses/gürültü düzeyi (KY-038) gibi birden fazla çevresel parametreyi eş zamanlı olarak okuyabilmek.
- **Gerçek Zamanlı Verimlilik (FreeRTOS):** ESP32'nin çift çekirdekli mimarisinden tam olarak yararlanmak. FreeRTOS kullanarak sensör okuma, OLED ekran güncelleme ve kullanıcı arayüzü (buton) etkileşimlerini ayrı görevler (task) halinde paralel çalıştırmak, böylece sistemin tepki süresini ve güvenilirliğini en üst düzeye çıkarmak.
- **Kompakt ve Kullanışlı Tasarım:** Elde taşınabilir, kompakt bir tasarımla (prototip aşamasında PCB ve harici kablolar olsa bile) atölyenin farklı bölgelerinde hızlı ölçümler yapmaya olanak sağlamak.

3. Kullanılan Temel Teknolojiler

Projenin başarısı, seçilen teknoloji yığınına bağlıdır:

- **Mikrodenetleyici: ESP32 Dev Kit**. Dahili Wi-Fi ve Bluetooth özelliklerine sahip, yüksek performanslı çift çekirdekli bir yonga. Projede FreeRTOS entegrasyonu sayesinde çoklu görev yönetiminde kritik rol oynar.
- **İşletim Sistemi: FreeRTOS**. ESP32 üzerinde çalışan ve görevlerin önceliklendirilmesine, senkronizasyonuna ve verimli kaynak kullanımına olanak tanıyan hafif bir Gerçek Zamanlı İşletim Sistemi.
- **Görüntüleme: OLED I²C Ekran**. Düşük güç tüketimi ve yüksek kontrast oranı sayesinde sensör verilerinin net ve anlık olarak görüntülenmesini sağlar.
- **Sensörler:** Atölye ortamının dört ana bileşenini izlemek üzere seçilmişlerdir: DHT11 (Sıcaklık ve Nem), LDR (Işık) ve KY-038 (Ses/Gürültü).

Bu rapor, söz konusu bu cihazın donanım kurulumundan, FreeRTOS tabanlı programlama mantığına ve uygulama adımlarına kadar tüm geliştirme süreçlerini detaylı olarak sunacaktır. Proje, maker topluluğu için öğrenme ve atölye verimliliğini artırma adına somut bir örnek teşkil etmektedir.

2. Literatür Araştırması

1. Giriş

Bu proje, mikrodenetleyici teknolojileri, gerçek zamanlı işletim sistemleri (RTOS) ve çevresel algılama (sensör) sistemlerinin entegrasyonu üzerine kurulmuştur. Literatür araştırması, projenin bu üç temel bileşeni bağlamında mevcut çözümleri, teknolojik yaklaşımları ve projenin yenilikçi yönlerini belirlemeyi amaçlamaktadır.

2. Mikrodenetleyici Seçimi: ESP32 ve Çift Çekirdek Avantajı

Projenin temelini oluşturan **ESP32** yongası, özellikle IoT ve gömülü sistem uygulamalarında son yılların en popüler tercihlerinden biri olmuştur.

- **Yüksek Entegrasyon:** ESP32, dahili Wi-Fi ve Bluetooth Low Energy (BLE) desteği sayesinde hem yerel izleme hem de gelecekteki IoT entegrasyonları için hazır bir platform sunar. (Kaynak: Benzer ESP32/IoT projeleri)
- **Çift Çekirdek Mimarisi:** İki adet LX6 işlemci çekirdeğine sahip olması, FreeRTOS gibi işletim sistemlerinin potansiyelini tam olarak kullanmasını sağlar. Bu sayede bir çekirdek Wi-Fi iletişimini yönetirken, diğer çekirdek sensör okuma ve ekran

güncelleme gibi kritik görevleri eş zamanlı olarak, kesintisiz bir şekilde gerçekleştirebilir. Bu, geleneksel tek çekirdekli Arduino çözümlerine göre üstün bir performans ve tepki süresi sağlar.

3. Gerçek Zamanlı İşletim Sistemi (RTOS) Kullanımı: FreeRTOS

Projenin kritik yeniliklerinden biri, **FreeRTOS**'un kullanılmasıdır. Geleneksel mikrodenetleyici projeleri genellikle tek bir ana döngü (loop()) içinde çalışırken, karmaşık çoklu sensör uygulamalarında bu yöntem darboğazlara yol açar:

- **Çoklu Görev Yönetimi:** FreeRTOS, her bir sensörden veri okuma (DHT11), ekranı güncelleme (OLED) ve kullanıcı girişi (Butonlar) gibi görevlerin ayrı "görevler (tasks)" olarak tanımlanıp önceliklendirilmesine olanak tanır. (Kaynak: ESP32 ve FreeRTOS kullanımına dair teknik makaleler)
- **Verimlilik ve Tepki Süresi:** Bu çoklu görev yeteneği, örneğin DHT11 gibi nispeten yavaş bir sensörün okuma süresinin, diğer görevlerin (ekran veya ses sensörü) güncellenmesini engellemesini önler. Bu, cihazın her koşulda "gerçek zamanlı" ve akıcı çalışmasını garantiler. Proje, FreeRTOS kullanarak bu alanda bir standart belirlemeyi amaçlamaktadır.

4. Sensör ve Veri Toplama Teknolojileri

Projede, atölye ortamının dört temel parametresini izlemek için endüstri standardı, düşük maliyetli sensörler seçilmiştir:

- **Sıcaklık ve Nem (DHT11):** Literatürde en yaygın kullanılan basit dijital sensörlerden biridir. Proje, bu sensörün nispeten uzun olan okuma süresini FreeRTOS görev yönetimi ile telafi ederek verimlilik sorununu aşmayı hedeflemektedir. (Kaynak: DHT11 uygulama kılavuzları)
- **Ses Seviyesi (KY-038):** Ortam gürültü seviyesini izlemek için kullanılan, hem analog hem de dijital çıkış verebilen bir mikrofondur. Proje, özellikle analog çıkışını kullanarak gürültü dalgalanmalarını gerçek zamanlı izlemeyi sağlayacaktır.
- **Işık Seviyesi (LDR):** Foto direnç (LDR) ile ortam aydınlatması analog olarak ölçülerek atölye kalibrasyonuna dahil edilir. Bu sensörlerin tek bir platformda birleştirilmesi, dağınık ekipman ihtiyacını ortadan kaldırarak projenin "Çok Fonksiyonlu Atölye Yardımcısı" vizyonunu destekler.

5. Benzer Projeler ve Projenin Konumu

Piyasada ve maker platformlarında, çevresel izleme yapan sayısız proje bulunmaktadır:

- **IoT Tabanlı Akıllı Ev Sistemleri:** ESP8266/ESP32 tabanlı birçok proje CO2, sıcaklık, nem gibi verileri toplayıp bunları Home Assistant gibi platformlara gönderir. Ancak bu projeler genellikle sadece veri toplama ve IoT entegrasyonuna odaklanır.
- **Tek Fonksiyonlu Kalibrasyon Cihazları:** Ayrı ayrı termometreler, desibel ölçerler veya ışık metreler mevcuttur. Bu proje, ticari bir **"İsviçre Çakısı"** konseptini, sensörler ve FreeRTOS kullanarak dijital ortama taşımasıyla benzersiz bir niş doldurur. Tek bir elde taşınabilir cihazda dört temel kalibrasyon/izleme fonksiyonunu, yüksek verimli bir FreeRTOS mimarisiyle birleştirmesi, onu sadece bir sensör projesi değil, çok amaçlı bir atölye aracı haline getirmektedir.

Sonuç: Literatür taraması, projenin ESP32'nin çift çekirdek gücü ve FreeRTOS'un çoklu görev yeteneklerini akıllıca kullanarak, geleneksel tek döngülü sistemlerin ötesine geçen, entegre ve gerçek zamanlı bir atölye izleme çözümü sunduğunu doğrulamaktadır.

3. Sistem Entegrasyonu

Bu bölümde, projenin fiziksel donanım kurulumu (bağlantı şemaları) ve bu donanımın çoklu görev yeteneği ile yönetilmesini sağlayan FreeRTOS tabanlı yazılım mimarisi detaylandırılmıştır. Bu entegrasyon, aracın "Çok Fonksiyonlu Atölye Yardımcısı" rolünü gerçekleştirmesini sağlayan temel adımdır.

3.1. Donanım Mimarisi ve Bileşen Entegrasyonu

Projenin donanımı, tüm çevresel izleme fonksiyonlarını kompakt bir PCB üzerinde (prototip aşamasında) bir araya getiren ESP32 Dev Kit üzerine kurulmuştur.

A. Kullanılan Temel Bileşenler

- **Mikrodenetleyici:** ESP32 Dev Kit
- **Ekran:** OLED I²C Ekran (SH1106 Sürücüsü) 3.3V
- **Sıcaklık ve Nem Sensörü:** DHT11 sensörü
- **Ses Sensörü:** KY-038
- **Işık Sensörü:** LDR (Light Dependent Resistor / Photoresistor)

- **Kullanıcı Arayüzü:** İki Adet Buton (MODE ve HOLD)
- **Diğer Bileşenler:** PCB, Jumper kabloları, Lehim telleri ve harici dirençler (30kΩ ve 220Ω - isteğe bağlı).

B. Elektriksel Bağlantı Detayları (Pin Atamaları)

Tüm sensörler ve ekran, ortak bir güç hattı (3.3V veya 5V) ve Toprak hattı (GND) kullanır.

Bileşen	**Pin Adı**	**Bağlantı Amacı**	**ESP32 GPIO Pini**
DHT11	DATA	Sıcaklık & Nem Veri Hattı	GPIO 18 (D18) [16]
KY-038	AO (Analog Çıkış)	Gürültü Seviyesi Ölçümü	GPIO 35 (D35) [17]
LDR	Analog Pin	Işık Şiddeti Ölçümü	GPIO 35 (D35) [18]
OLED	SDA	I²C Seri Veri Hattı	GPIO 21 (D21) [19]
OLED	SCL	I²C Seri Saat Hattı	GPIO 22 (D22) [20]
MODE Butonu	Bir Ayak	Ekran Navigasyonu (Next/Previous)	GPIO 19 (D19) [21]
HOLD Butonu	Bir Ayak	Veri Bekletme Fonksiyonu	GPIO 23 (D23) [22]

Not: KY-038 sensörünün DO (Dijital Çıkış) pini isteğe bağlıdır ve bağlantısız bırakılabilir. Navigasyon ve Bekletme butonlarının diğer ayakları harici bir direnç ile GND'ye bağlanır.

3.2. Yazılım Mimarisi: FreeRTOS ile Çoklu Görev Entegrasyonu

Projenin yazılımı, PlatformIO (VS Code uzantısı) kullanılarak FreeRTOS ile geliştirilmiştir. FreeRTOS, ESP32'nin çift çekirdekli işlemcisinin potansiyelini tam olarak kullanmak için görevleri (tasks) paralel olarak çalıştırmayı sağlar.

A. FreeRTOS Kullanımının Gerekçesi

- Geleneksel tek döngü (**single loop**) yaklaşımının aksine , FreeRTOS, sensor okumaları, ekran güncellemeleri ve buton etkileşimlerinin daha verimli ve optimize bir şekilde yönetilmesini sağlar.
- Bu yaklaşım, sistemin daha duyarlı ve güvenilir olmasını sağlar.

B. FreeRTOS Görev Dağılımı

Yazılım, projeyi yöneten temel işlevleri ayrı FreeRTOS görevlerine ayırır:

1. **Sensör Verisi Okuma:** Sensörlerden veri okuma görevini yönetir. (DHT11, LDR, KY-038)
2. **OLED Ekran Güncelleme:** OLED ekranın güncellenmesi ve sensör verilerinin gösterilmesi görevini yönetir.
3. **Buton Girişlerinin İşlenmesi:** MODE ve HOLD butonlarından gelen kullanıcı girişlerinin işlenmesi görevini yönetir.

4. Karşılaşılan Sorunlar ve Çözümler

Projenin geliştirilmesi ve sistem entegrasyonu aşamalarında, özellikle çoklu sensör kullanımı ve FreeRTOS mimarisi nedeniyle bazı teknik zorluklarla karşılaşılmıştır. Bu sorunlar ve bunların üstesinden gelmek için uygulanan çözümler aşağıda ayrıntılı olarak açıklanmıştır.

4.1. Donanım ve Bağlantı Zorlukları

- **Gevşek Bağlantılar ve Kısa Devre Riski:** Prototip aşamasında açık PCB ve kablolama kullanıldığından, özellikle lehimleme sonrası hatalı veya gevşek bağlantıların kısa devrelere neden olma riski mevcuttu. Bu riski önlemek için, güç verilmeden önce kabloların düzgün lehimlendiği, üst üste binme veya kaçak lehim olmadığı kontrol

edildi. Ortak Toprak (GND) ve VCC (3.3V) bağlantılarının doğru paylaşıldığından emin olundu.

- **Başlangıç Kılavuzu:** Elektroniğe yeni başlayanlar için lehimleme ve PCB kurulumunun ilk adımda karmaşık gelme ihtimali göz önünde bulunduruldu. Çözüm olarak, bu kullanıcı grubuna lehimleme yerine öncelikle bir breadboard kullanarak tüm kablolamayı test etmeleri önerildi.
- **I²C İletişim Hatası:** OLED ekranın I²C iletişimi için doğru pinlerin atanması kritikti. Hata oluşmaması için, ekranın SDA (Seri Veri) pininin **GPIO 21 (D21)**'e ve SCL (Seri Saat) pininin **GPIO 22 (D22)**'ye doğru şekilde bağlandığından emin olunması gerektiği vurgulandı.

4.2. Yazılım ve FreeRTOS Zorlukları

- **Sensörler Arası Çakışma ve Performans Sorunları:** Geleneksel tek bir döngüde (loop) çalışan çözümlerde, sensör okumaları (özellikle DHT11 gibi yavaş sensörler) ekran güncellemelerini veya buton tepkilerini geciktirebilirdi. Bu durumu çözmek için, FreeRTOS kullanıldı. Her bir işlev (sensör okuma, ekran güncelleme ve buton girişi) ayrı bir görev (task) olarak tanımlandı. Bu yaklaşım, görevlerin paralel çalışmasını sağlayarak daha akıcı bir performans ve optimize kullanım sağladı.
- **Gerekli Kütüphane Bağımlılıkları:** Proje kodunun derlenebilmesi için spesifik kütüphane bağımlılıkları vardı. Çözüm olarak, projenin başarılı bir şekilde çalışması için gerekli kütüphane sürümleri belirlendi ve kurulumları sağlandı.

4.3. Prototip Sınırlamaları ve Gelecek Çözümleri

Mevcut prototip aşamasında karşılaşılan sınırlamalar, projenin gelecekteki olası geliştirme yollarını da ortaya çıkarmaktadır:

- **Koruma Eksikliği:** Mevcut açık prototip formu, cihazı hasara ve harici tehlikelere karşı savunmasız bırakmaktadır. Bu sınırlamayı aşmak için, elektroniği koruyacak ve kullanımı kolaylaştıracak bir **3D baskılı koruyucu muhafaza** (tercihen dayanıklılık için PETG) tasarlanması önerilmektedir.
- **Genişletilebilirlik İçin Güç Kaynağı:** Daha fazla sensör (gaz, basınç, hareket) eklenmesi istendiğinde mevcut güç kaynağı yetersiz kalabilir. Bu potansiyel sorunun

çözümü, **MB102 güç modülü** veya benzeri, stabil 3.3V–5V sağlayan düzenlenmiş bir güç kaynağının entegre edilmesidir.

- **Veri Kaydı ve Uzaktan İzleme Eksikliği:** Cihaz şu anda anlık izleme sunmaktadır. İleride veri analizi ve uzaktan izlemeyi mümkün kılmak amacıyla, **yapay zeka özellikleri, bulut tabanlı gösterge panoları veya kablosuz veri kaydı** gibi eklentilerin entegre edilmesi planlanmaktadır.

5. Alternatif Yöntemler ve Teknolojik Seçimler Üzerine Değerlendirme

Bu bölümde, projenin temel teknolojik kararları (mikrodenetleyici, işletim sistemi ve sensörler) incelenmekte ve bu seçimlerin arkasındaki gerekçeler, alternatif çözümlerle karşılaştırılarak değerlendirilmektedir.

5.1. Mikrodenetleyici Seçimi: ESP32 ve Alternatifler

Proje için merkezi işlem birimi olarak **ESP32 Dev Kit** seçilmiştir. Bu seçim, özellikle **çift çekirdekli** işlemci yeteneği sayesinde FreeRTOS ile çoklu görevleri eş zamanlı ve verimli bir şekilde yönetme olanağı sunduğu için yapılmıştır. Bu performans avantajı, tek çekirdekli sistemlere göre daha akıcı bir kullanıcı deneyimi sağlar.

Alternatif olarak, Arduino Uno veya ESP8266 gibi kartlar kullanılabilirdi. Ancak bu alternatifler, ESP32'nin sağladığı verimli çoklu görev yeteneği ve **yerleşik Wi-Fi** gibi gelişmiş özellikleri sunmaz. ESP32'nin ideal seçimi, aynı zamanda gelecekteki **kablosuz veri kaydı, bulut entegrasyonu veya IoT panosu bağlantısı** gibi eklentileri de desteklemesi nedeniyledir.

5.2. Yazılım Mimarisi Seçimi: FreeRTOS vs. Tek Döngü

Proje, yazılım mimarisi olarak **FreeRTOS** kullanımını benimsemiştir.

- **Gerekçe:** FreeRTOS, ESP32'nin çift çekirdek mimarisinden tam olarak yararlanılmasını sağlar. FreeRTOS, sensör okumaları, ekran güncellemeleri ve buton etkileşimleri gibi birden fazla görevin paralel çalışmasını sağlayarak , sistemin daha duyarlı ve güvenilir olmasını sağlar.
- **Alternatifin Dezavantajı:** Geleneksel tek bir ana döngüde (loop) tüm işlevlerin yönetildiği bir yaklaşım benimsenmiş olsaydı, özellikle sensör okuma sırasındaki gecikmeler (örneğin DHT11 okuması), tüm sistemin performansını ve tepki süresini yavaşlatacaktı. FreeRTOS, bu darboğazın önüne geçerek daha optimize bir çözüm sunmuştur.

5.3. Sensör Seçimleri Üzerine Değerlendirme

Projede kullanılan sensörler (DHT11, KY-038, LDR) yaygın ve düşük maliyetli çevresel izleme bileşenleridir.

- **DHT11 (Sıcaklık ve Nem):** Bu sensör, temel sıcaklık ve nem verilerini sağlamak için uygun bir seçimdir. Ancak, daha yüksek doğruluk ve ek basınç ölçümü gibi özellikler için BME280 gibi daha gelişmiş sensörler alternatif olarak düşünülebilir.
- **KY-038 (Ses):** Bu mikrofon sensörü, ortam gürültü seviyelerini gerçek zamanlı olarak izlemek ve durum etiketi sağlamak için kullanılır. Daha profesyonel kalibrasyon gerektiren uygulamalar için, desibel (dB) cinsinden kalibre edilmiş daha gelişmiş ses ölçüm cihazları tercih edilebilir.
- **LDR (Işık):** LDR, ortam aydınlatma koşullarının yüzdesel veya ham değer olarak kolayca ölçülmesini sağlayan basit ve pratik bir bileşendir. Daha kesin lüks (lux) birimi cinsinden ölçüm için dijital optik sensörler (BH1750 gibi) alternatif olarak kullanılabilir.

Sonuç olarak, projenin teknolojik seçimleri, temel olarak yüksek performanslı çoklu görev yeteneğini (ESP32 ve FreeRTOS) düşük maliyetli ve ulaşılabilir sensörlerle birleştirerek, atölye ortamı için pratik ve verimli bir yardımcı araç yaratma hedefine ulaşmak üzere yapılmıştır.

6. Sonuç ve Gelecek Çalışmalar

Bu bölümde, projenin amaçlarına ne ölçüde ulaştığı özetlenmekte ve elde edilen sonuçlar ışığında gelecekteki olası geliştirme ve genişletme alanları sunulmaktadır.

6.1. Sonuç

"ESP32 Kullanarak FreeRTOS ile Çok Fonksiyonlu Atölye Yardımcısı" projesi, İsviçre Çakısı konseptini dijital bir çevresel izleme aracına başarıyla uygulamıştır. Projenin temel hedeflerine ulaşılmıştır:

- **Çok Fonksiyonluluk:** DHT11 (Sıcaklık ve Nem), KY-038 (Ses) ve LDR (Işık) sensörleri tek bir kompakt cihazda birleştirilmiştir.
- **Gerçek Zamanlı Performans:** ESP32'nin çift çekirdekli mimarisi, FreeRTOS'un görev yönetimi yeteneğiyle kullanılarak, sensör okumaları, ekran güncellemeleri ve buton etkileşimleri eş zamanlı ve akıcı bir şekilde gerçekleştirilmiştir. Bu yaklaşım, sistemin tek bir döngüde çalışan çözümlere göre daha duyarlı ve güvenilir olmasını sağlamıştır.

- **Kullanılabilirlik:** OLED ekran üzerindeki anlık veri gösterimi ve navigasyon butonları, kullanıcının farklı sensör ekranları arasında kolayca geçiş yapmasına olanak tanır.

Proje, makerlar, hobi sahipleri ve elektronik meraklıları için çalışma ortamı koşullarını gerçek zamanlı olarak takip etme konusunda pratik ve değerli bir araç olduğunu kanıtlamıştır.

6.2. Gelecek Çalışmalar ve Potansiyel Geliştirmeler

Mevcut prototipin potansiyelini maksimize etmek ve sınırlamalarını gidermek için aşağıdaki geliştirmeler ve gelecek çalışmalar planlanmaktadır:

1. **Koruyucu Muhafaza Tasarımı:**
    - Cihazın dış hasarlara ve tehlikelere karşı korunması için **3D baskılı bir muhafaza** (tercihen dayanıklılık için PETG filament) tasarlanacak ve entegre edilecektir. Bu aynı zamanda cihazın taşınabilirliğini ve kullanımını kolaylaştıracaktır.
2. **Gelişmiş Güç Yönetimi ve Sensör Genişletme:**
    - Daha fazla sensör eklemeye izin vermek ve stabil çalışmayı sağlamak için, MB102 güç modülü gibi **düzenlenmiş bir güç kaynağı** entegre edilecektir.
    - Genişletilmiş güç kaynağı ile **ek sensörler** (örneğin gaz, basınç veya hareket sensörleri) eklenerek aracın fonksiyonelliği artırılacaktır.
3. **IoT Entegrasyonu ve Veri Analizi:**
    - Projenin temelini oluşturan ESP32'nin Wi-Fi özelliği kullanılarak, **kablosuz veri kaydı** ve **bulut tabanlı panolara** (dashboard) veri aktarımı sağlanacaktır.
    - Gelecekte, toplanan veriler üzerinden daha akıllı analizler ve uzaktan izleme sağlamak amacıyla **Yapay Zeka (AI) özellikleri** entegre edilmesi hedeflenmektedir. Bu, atölye verimliliği ve güvenliği için daha akıllı kararlar alınmasına olanak tanıyacaktır.

Bu planlar, mevcut projeyi sadece bir prototip olmaktan çıkarıp, daha dayanıklı, işlevsel ve IoT ekosistemine entegre bir atölye çözümüne dönüştürmeyi amaçlamaktadır.

7. Kullanılan Donanımlar

Bu bölümde, projenin çok fonksiyonlu yeteneklerini hayata geçiren temel donanım bileşenleri ayrıntılı olarak tanıtılmaktadır. Her bir bileşenin işlevi ve teknik özellikleri, sistem entegrasyonundaki rolü bağlamında açıklanmıştır.

7.1. Merkezi İşlem Birimi (MCU): ESP32 Dev Kit

ESP32, projenin kalbini oluşturan, Wi-Fi ve Bluetooth özelliklerine sahip güçlü bir mikrodenetleyicidir.

- **Rolü:** Tüm sensörlerden veri okumak, OLED ekranı kontrol etmek ve kullanıcı girişlerini (butonları) yönetmekten sorumludur. FreeRTOS sayesinde çift çekirdek yeteneği, birden fazla görevi paralel olarak yönetmeyi sağlar.

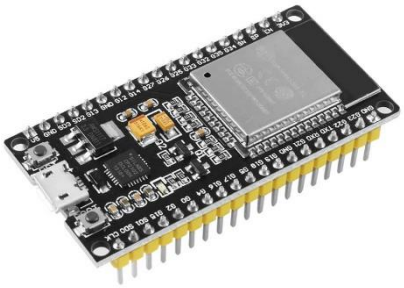


Fig. 7.1: ESP32 Dev Kit

7.2. Görüntüleme Birimi: OLED I²C Ekran (SH1106)

OLED ekran, toplanan çevresel verilerin kullanıcıya anlık olarak sunulduğu ana arayüzdür.

- **Rolü:** Sıcaklık (T), Nem (H), Işık Seviyesi ve Ses Seviyesi gibi sensör verilerini gerçek zamanlı olarak gösterir. Navigasyon sırasında Next/Prev (Sonraki/Önceki) ekran göstergelerini de içerir.
- **Teknik Özellik:** I²C iletişim protokolünü kullanır. Bu sayede ESP32 ile sadece dört kablo ile veri transferi sağlanır.

Fig. 7.2: OLED I²C Ekran (SH1106)

7.3. Sensör Modülleri

Cihazın çok fonksiyonlu yeteneğini sağlayan üç ana sensör modülü kullanılmıştır.

7.3.1. DHT11 (Sıcaklık ve Nem Sensörü)

- **Rolü:** Ortamdaki sıcaklık ve nem oranını ölçer. Başlangıç ekranında sıcaklık (T) ve nem (H) yüzdesini gösterir.
- **Bağlantı:** Dijital veri pini **GPIO 18 (D18)**'e bağlanır.

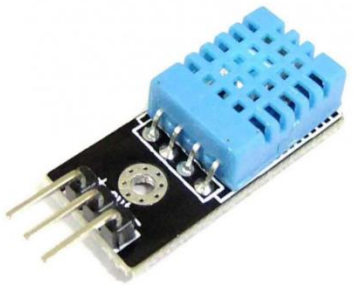


Fig. 7.3.1: DHT11 sensör

7.3.2. KY-038 (Ses Sensörü)

- **Rolü:** Atölye ortamındaki anlık gürültü seviyesini izler. Ses algılandığında ham değeri ekranda gösterir ve ses seviyesi için bir durum etiketi sağlar.
- **Bağlantı:** Analog çıkış (AO) pini **GPIO 35 (D35)**'e bağlanır. Dijital çıkış (DO) pini isteğe bağlıdır.

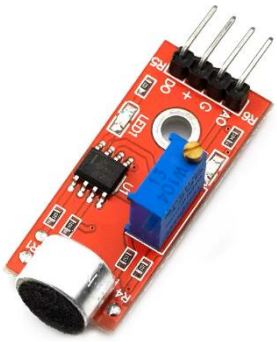


Fig. 7.3.2: KY-038 (Ses Sensörü)

7.3.3. LDR (Işık Sensörü / Foto Direnç)

- **Rolü:** Ortamdaki aydınlatma şiddetini ölçer. Ekranda algılanan ışığın minimum ve maksimum değerlerini, ayrıca yüzde (0% ila 100%) cinsinden seviyesini gösterir.
- **Bağlantı:** Analog pini **GPIO 35 (D35)**'e bağlanır ve KY-038 ile aynı pini paylaşır.

Fig. 7.3.3: LDR

7.3.4. Kullanıcı Giriş Birimleri (Butonlar)

Navigasyon ve kontrol için iki adet basmalı buton (Push button) kullanılmıştır.

- **MODE Butonu:** Sensör ekranları arasında ileri/geri geçişi sağlar (Next/Previous switch). **GPIO 19 (D19)**'a bağlanır.
- **HOLD Butonu:** Okunan veriyi dondurma (bekletme) işlevi için kullanılabilir. **GPIO 23 (D23)**'e bağlanır.

Fig. 7.3.3: Mode ve Hold butonları

7.3.5. Diğer Gerekli Bileşenler

- **PCB Kartı:** Tüm bileşenlerin lehimlenip sabitlendiği devre kartı.
- **Dirençler:** Buton bağlantıları ve genel devre kararlılığı için 30kΩ ve 220Ω dirençler kullanılmıştır.
- **Kablolar:** Bağlantılar için Jumper kabloları ve lehim telleri.
- **Güç:** Programlama ve güç sağlama için USB Type-C kablosu.

8.Yazılım Mimarisi

Bu bölüm, projenin temelini oluşturan **FreeRTOS tabanlı yazılım mimarisini** incelemektedir. Geleneksel mikrodenetleyici programlamanın aksine, bu proje ESP32'nin çift çekirdekli yapısından tam olarak yararlanmak için tüm işlevleri eş zamanlı çalışan ayrı **Görevlere (Tasks)** ayırmıştır.

Bu görev tabanlı yaklaşım sayesinde , sensör okumaları , ekran güncellemeleri ve buton etkileşimleri gibi işlemler birbirini engellemeden paralel olarak yürütülür.

Aşağıda, sunulan kod parçalarını analiz ederek, projenin yazılım mimarisini oluşturan temel bileşenlerin ve görevlerin işlevleri ayrıntılı olarak açıklanacaktır. Kod bloklarını incelemeye başladığımızda, her bir görev tanımlamasının ve ana fonksiyonun sistem içindeki rolü netleştirilecektir.

```
#include <Arduino.h>

#include <U8g2lib.h>
#ifdef U8X8_HAVE_HW_SPI
#include <SPI.h>
#endif
#ifdef U8X8_HAVE_HW_I2C
#include <Wire.h>
#endif

#include <freertos/FreeRTOS.h>
#include <freertos/task.h>

#include <DHT.h>

#define DHT11_PIN
#define DHT_TYPE DHT11


DHT dht(DHT11_PIN, DHT_TYPE);

U8G2_SH1106_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, /* reset=*/ U8X8_PIN_NONE);

#define SOUND_PIN 35
#define LIGHT_PIN 34
#define PREV_BUTTON_PIN 19
#define NEXT_BUTTON_PIN 23
```

Fig. 8.1: Kütüphaneler, Pin Tanımlamaları ve Nesne Başlatma

Fig. 8.1'deki kod parçası, projenin temel yapısını ve gerekli tüm donanım ve yazılım bağımlılıklarını tanımlar. Arduino temel fonksiyonları ve gerekli olan U8g2 (OLED ekran kontrolü için), FreeRTOS (çoklu görev yönetimi için) ve DHT (Sıcaklık/Nem sensörü)

kütüphaneleri dahil edilmiştir. Ardından, ESP32 üzerindeki donanım bileşenlerinin bağlandığı GPIO pinleri (#define direktifleri) tanımlanır; bunlar arasında DHT11 veri pini (GPIO 18), Ses sensörü (GPIO 35), Işık sensörü (GPIO 34) ve Navigasyon butonları (GPIO 19, 23) bulunur. Son olarak, kütüphane nesneleri başlatılır: dht nesnesi sensör tipini ve pinini belirterek DHT11'i hazırlar; u8g2 nesnesi ise SH1106 sürücülü OLED ekranın I²C modunda kullanılacağını tanımlar.

```
const int sampleWindow = 100;
unsigned int soundLevel = 0;
unsigned int maxSoundLevel = 0;
unsigned int avgSoundLevel = 0;
int soundDB = 0;

float tempC = 0.0;
float humidity = 0.0;


int lightValue = 0;
int initLight = 0;
int minLightValue = 4095;
int maxLightValue = 0;
int avgLightValue = 0;
int lightHistory[32];
int lightHistoryIndex = 0;
String lightChangeStatus = "Initializing...";


volatile int currentScreen = 0;
const int totalScreens = 3;


unsigned long lastButtonPress = 0;
const unsigned long debounceDelay = 200;


unsigned int soundHistory[32];
int historyIndex = 0;
```

Fig. 8.2: Genel Değişkenler ve Veri Tutucuları

Fig. 8.2'deki kod parçası, projenin tüm FreeRTOS görevleri arasında paylaşılan global değişkenleri tanımlar. Bu değişkenler, sensörlerden okunan verileri tutar ve ekran görevine iletir. Ses sensörü için anlık seviye (soundLevel), hesaplanan desibel değeri (soundDB) ve maksimum/ortalama seviyeler gibi metrikler saklanır. DHT sensörü için sıcaklık (tempC) ve nem (humidity) değerleri tutulur. Işık sensörü için ise anlık analog değer (lightValue), kaydedilen minimum/maksimum değerler ve anlık değişim durumunu gösteren bir metin

(lightChangeStatus) saklanır. Ayrıca, kullanıcı navigasyonunu yöneten currentScreen değişkeni (hata önleme amacıyla volatile olarak tanımlanmıştır) ve buton basışlarını kontrol ederek çift basmayı engelleyen debounceDelay değişkeni bu kısımda tanımlanmıştır.

```
bool isValidTemperature(float temperature) {

    return (!isnan(temperature) && temperature >= 0 && temperature <= 50);
}

bool isValidHumidity(float hum) {
    return (!isnan(hum) && hum >= 0 && hum <= 100);
}


const char* getTemperatureCondition(float tempC) {
    if (tempC < 15.0) return "Cold";
    if (tempC < 25.0) return "Cool";
    if (tempC <= 30.0) return "Warm";
    return "Hot";
}

const char* getSoundLevelDescription(int db) {
    if (db < 40) return "Quiet";
    if (db < 60) return "Normal";
    if (db < 80) return "Loud";
    return "V.Loud";
}


const char* getLightLevelDescription(int lightVal) {
    if (lightVal < 500) return "Dark";
    if (lightVal < 1500) return "Dim";
    if (lightVal < 3000) return "Bright";
    return "V.Bright";
}


int getLightPercentage(int lightVal) {
    return map(lightVal, 0, 4095, 0, 100);
}
```

Fig. 8.3: Veri Doğrulama ve Durum Belirleme Fonksiyonları

Fig. 8.3'teki kod parçası, okunan sensör verilerini yorumlamak ve kullanıcının anlayabileceği durumlara dönüştürmek için tasarlanmış yardımcı fonksiyonları içerir. isValidTemperature ve isValidHumidity fonksiyonları, sensörden gelen verilerin geçerli olup olmadığını (NaN olup olmadığını ve teknik aralıkta bulunup bulunmadığını) kontrol ederek hatalı okumaların ekranda gösterilmesini engeller. Diğer fonksiyonlar, ham verileri atölye ortamı için anlamlı durumlara çevirir: getTemperatureCondition sıcaklık değerine göre ortamı "Soğuk" veya "Sıcak" olarak

etiketlerken, getSoundLevelDescription gürültüyü "Sessiz" veya "Çok Yüksek Sesli" olarak sınıflandırır. getLightPercentage fonksiyonu ise LDR'den gelen 12-bit (0-4095) analog değeri anlaşılır bir yüzde değerine dönüştürür.

```
void readSensorTask(void * parameter) {
    while (true) {

        float newTemp = dht.readTemperature();
        float newHum = dht.readHumidity();


        if (isValidTemperature(newTemp)) {
            tempC = newTemp;
        }
        if (isValidHumidity(newHum)) {
            humidity = newHum;
        }


        Serial.print("Temperature: ");
        Serial.print(tempC);
        Serial.print("°C, Humidity: ");
        Serial.print(humidity);
        Serial.println("%");

        vTaskDelay(2000 / portTICK_PERIOD_MS);
    }
}
```

Fig. 8.4: FreeRTOS Görevi (readSensorTask)

Fig. 8.4'teki görev, FreeRTOS altında çalışan ve ortam sıcaklığı ile nemini ölçen ilk görevdir. Görev, sonsuz bir döngü içerisinde DHT11 sensöründen veri okur. Okunan sıcaklık (newTemp) ve nem (newHum) değerleri, olası hatalı okumaları elemek için bir önceki kısımda tanımlanan doğrulama fonksiyonlarından geçirilir. Eğer okuma geçerliyse, global değişkenlere (tempC, humidity) atanarak ekran görevi tarafından kullanılmaya hazır hale getirilir. Görevin en kritik kısmı, döngünün sonunda kullanılan vTaskDelay(2000 / portTICK_PERIOD_MS) komutudur. Bu komut, DHT11 sensörünün doğası gereği hızlı okunamaması nedeniyle, görevin her 2 saniyede bir çalışmasını sağlar ve bu bekleme süresi boyunca ESP32'nin işlemci çekirdeğini serbest bırakarak diğer görevlerin kesintisiz çalışmasına olanak tanır.

```
void readSensorTask2(void * parameter) {
    while (true) {
        unsigned long startMillis = millis();
        unsigned int signalMax = 0;
        unsigned int signalMin = 4095;

        while (millis() - startMillis < sampleWindow) {
            int sample = analogRead(SOUND_PIN);
            if (sample < 4095) {
                if (sample > signalMax) {
                    signalMax = sample;
                }
                else if (sample < signalMin) {
                    signalMin = sample;
                }
            }
        }

        int peakToPeak = signalMax - signalMin;
        soundLevel = peakToPeak;
        soundDB = map(peakToPeak, 0, 2000, 30, 90);
        soundDB = constrain(soundDB, 30, 90);

        if (soundLevel > maxSoundLevel) {
            maxSoundLevel = soundLevel;
        }

        static unsigned long soundSum = 0;
        static int soundCount = 0;
        soundSum += soundLevel;
        soundCount++;
        if (soundCount > 10) {
            avgSoundLevel = soundSum / soundCount;
            soundSum = 0;
            soundCount = 0;
        }

        soundHistory[historyIndex] = map(soundLevel, 0, 1000, 0, 20);
        historyIndex = (historyIndex + 1) % 32;

        Serial.print("Sound Level: ");
        Serial.print(soundLevel);
        Serial.print(", dB: ");
        Serial.println(soundDB);

        vTaskDelay(100 / portTICK_PERIOD_MS);
    }
```

Fig. 8.5: FreeRTOS Görevi (readSensorTask2)

Fig. 8.5'teki görev, ortamdaki gürültü seviyesini ölçmek üzere tasarlanmıştır. Geleneksel anlık okuma yerine, görev sampleWindow (100 ms) boyunca sürekli analog okumalar alır. Bu pencereleme tekniği, sinyalin maksimum ve minimum genliklerini (peak-to-peak)

yakalayarak anlık gürültü seviyesini (soundLevel) daha doğru temsil eden bir değer üretir. Yakalanan tepe-tepe genliği daha sonra yaklaşık bir desibel (dB) aralığına (soundDB) eşlenir ve 30 ile 90 arasına sabitlenir. Görev ayrıca, cihazın çalıştırıldığı andan itibaren kaydedilen maksimum ses seviyesini izler ve dalga animasyonu için okumaları dairesel bir tampona (soundHistory) kaydeder. Görev, yüksek tepki hızı sağlamak için nispeten kısa bir gecikmeyle, her 100 milisaniyede bir tekrarlanır.

```
void readSensorTask3(void * parameter) {
    while (true) {

        lightValue = analogRead(LIGHT_PIN);

        if (abs(lightValue - initLight) > 50) {
            int change = lightValue - initLight;
            if (change > 0) {
                lightChangeStatus = "Brighter: +" + String(change);
            } else {
                lightChangeStatus = "Darker: " + String(change);
            }
        } else {
            lightChangeStatus = "No change";
        }

        if (lightValue < minLightValue) {
            minLightValue = lightValue;
        }
        if (lightValue > maxLightValue) {
            maxLightValue = lightValue;
        }

        static unsigned long lightSum = 0;
        static int lightCount = 0;
        lightSum += lightValue;
        lightCount++;
        if (lightCount > 10) {
            avgLightValue = lightSum / lightCount;
            lightSum = 0;
            lightCount = 0;
        }

        lightHistory[lightHistoryIndex] = map(lightValue, 0, 4095, 0, 25);
        lightHistoryIndex = (lightHistoryIndex + 1) % 32;

        Serial.print("Light Value: ");
        Serial.print(lightValue);
        Serial.print(" (");
        Serial.print(getLightPercentage(lightValue));
        Serial.print("%) - ");
        Serial.println(lightChangeStatus);

        vTaskDelay(150 / portTICK_PERIOD_MS);
    }
}
```

Fig. 8.6: FreeRTOS Görevi (readSensorTask3)

Fig. 8.6'daki görev, LDR (ışık sensörü) üzerinden ortam aydınlatma şiddetini sürekli izler. Görev, analog okuma yaparak anlık ışık değerini (lightValue) alır ve sistemin ilk çalıştığı andaki initLight değeriyle karşılaştırarak aydınlatmada önemli bir değişim olup olmadığını (lightChangeStatus) belirler. Tıpkı ses sensöründe olduğu gibi, bu görev de cihazın açıldığı andan itibaren kaydedilen minimum (en karanlık) ve maksimum (en parlak) ışık değerlerini sürekli güncelleyerek bir kalibrasyon referansı sağlar. Ayrıca, okumalar bir çubuk grafik animasyonu için dairesel bir geçmiş dizisine (lightHistory) kaydedilir. Bu görev, her 150 milisaniyede bir çalışarak dinamik ışık değişikliklerini yakalar.

```
void buttonTask(void * parameter) {
    while (true) {
        unsigned long currentTime = millis();


        if (currentTime - lastButtonPress > debounceDelay) {

            if (digitalRead(PREV_BUTTON_PIN) == LOW) {
                currentScreen = (currentScreen - 1 + totalScreens) % totalScreens;
                lastButtonPress = currentTime;
                Serial.println("Previous button pressed, Screen: " + String(currentScreen));
            }


            if (digitalRead(NEXT_BUTTON_PIN) == LOW) {
                currentScreen = (currentScreen + 1) % totalScreens;
                lastButtonPress = currentTime;
                Serial.println("Next button pressed, Screen: " + String(currentScreen));
            }
        }

        vTaskDelay(50 / portTICK_PERIOD_MS);
    }
}
```

Fig. 8.7: FreeRTOS Görevi (buttonTask)

Fig. 8.7'deki görev, kullanıcı ile etkileşimi yöneten kritik bir FreeRTOS görevidir. Görevin temel işlevi, MODE butonlarından (PREV ve NEXT) gelen fiziksel basışları algılamak ve bu basışlara dayanarak currentScreen navigasyon değişkenini güncellemektir. Görev, fiziksel buton mekanizmalarında sıklıkla görülen parazitli sinyalleri (debounce) önlemek için debounceDelay değişkenini kullanarak, bir basıştan sonra belirli bir süre (200 ms) boyunca yeni basışları dikkate almaz. Ekran geçişlerini sağlayan aritmetik işlemler (% totalScreens) dairesel bir navigasyon sağlar (ekran 2'den sonra ekran 0'a dönme). Bu görev, kullanıcı girişlerine hızlı yanıt verebilmesi için kısa bir gecikmeyle, her 50 milisaniyede bir kontrol edilir.

```
void drawDHTScreen() {

    String tempDisplay;
    String humDisplay;
    String conditionDisplay;

    if (isValidTemperature(tempC)) {
        tempDisplay = "T: " + String(tempC, 1) + "C";
        const char* condition = getTemperatureCondition(tempC);
        conditionDisplay = "Stat: " + String(condition);
    } else {
        tempDisplay = "T: N/A";
        conditionDisplay = "Sts: N/A";
    }

    if (isValidHumidity(humidity)) {
        humDisplay = "H: " + String(humidity, 1) + "%";
    } else {
        humDisplay = "H: N/A";
    }


    u8g2.setFont(u8g2_font_4x6_tr);
    u8g2.drawStr(1, 30, "SENSOR: TMPwHMDT");


    u8g2.setFont(u8g2_font_ncenB08_tr);
    u8g2.drawStr(1, 42, tempDisplay.c_str());
    u8g2.drawStr(64, 42, humDisplay.c_str());
    u8g2.drawStr(1, 54, conditionDisplay.c_str());


    u8g2.setFont(u8g2_font_4x6_tr);
    u8g2.drawStr(1, 64, "<PREV");
    u8g2.drawStr(90, 64, "NEXT>");
}
```

Fig. 8.8: DHT11 Ekran Çizimi (drawDHTScreen)

Fig.8.8'deki fonksiyon, sıcaklık ve nem sensörü (DHT11) verilerini OLED ekranda görselleştirmekten sorumludur. Öncelikle, okunan tempC ve humidity global değişkenlerinin geçerliliğini kontrol eder; eğer veri geçerliyse, sıcaklık değeri santigrat cinsinden ve nem değeri yüzde cinsinden ekranda gösterilmek üzere biçimlendirilir. Sıcaklık verisine dayanarak ortam durumunu belirleyen (getTemperatureCondition) yardımcı fonksiyonu çağırır ve durumu ("Stat: Warm", vb.) gösterir. Eğer okunan veri geçerli değilse, ekranda "T: N/A" ve "H: N/A" yazar. Çizim işlemi, u8g2 kütüphanesi komutları kullanılarak yapılır; sensör başlığı ve veriler belirli koordinatlara yerleştirilir. Son olarak, kullanıcının diğer ekranlara geçiş yapabilmesi için ekranın en altına navigasyon göstergeleri (<PREV ve NEXT>) eklenir.

```
void drawSoundScreen() {

    u8g2.setFont(u8g2_font_4x6_tr);
    u8g2.drawStr(1, 30, "SENSOR: SOUND");

    String levelDisplay = "Lvl: " + String(soundLevel);
    String dbDisplay = "dB: " + String(soundDB);
    String maxDisplay = "Max: " + String(maxSoundLevel);
    String statusDisplay = "Sts: " + String(getSoundLevelDescription(soundDB));

    u8g2.setFont(u8g2_font_ncenB08_tr);
    u8g2.drawStr(1, 42, levelDisplay.c_str());
    u8g2.drawStr(70, 42, dbDisplay.c_str());
    u8g2.drawStr(70, 54, maxDisplay.c_str());

    u8g2.drawStr(1, 54, statusDisplay.c_str());
    u8g2.setFont(u8g2_font_4x6_tr);
    u8g2.drawStr(1, 64, "<PREV");
    u8g2.drawStr(90, 64, "NEXT>");
}

void drawLightScreen() {

    u8g2.setFont(u8g2_font_4x6_tr);
    u8g2.drawStr(1, 30, "SENSOR: LIGHT");

    String percentDisplay = "Val: " + String(getLightPercentage(lightValue)) + "%";
    String minDisplay = "Min: " + String(minLightValue);
    String maxDisplay = "Max: " + String(maxLightValue);
    String statusDisplay = "Stat: " + String(getLightLevelDescription(lightValue));

    u8g2.setFont(u8g2_font_ncenB08_tr);

    u8g2.drawStr(75, 54, percentDisplay.c_str());
    u8g2.drawStr(1, 42, minDisplay.c_str());
    u8g2.drawStr(75, 42, maxDisplay.c_str());
    u8g2.drawStr(1, 54, statusDisplay.c_str());
    u8g2.setFont(u8g2_font_4x6_tr);
    u8g2.drawStr(1, 64, "<PREV");
    u8g2.drawStr(90, 64, "NEXT>");
}
```

Fig. 8.9: Ses ve Işık Ekranı Çizimleri (drawSoundScreen ve drawLightScreen)

Fig. 8.9'daki fonksiyon, kalan iki sensörün verilerini ayrı ayrı ve mantıksal olarak yapılandırılmış ekranlarda sunar. drawSoundScreen fonksiyonu, anlık ses seviyesini (soundLevel), yaklaşık desibel değerini (soundDB) ve cihazın kaydettiği en yüksek seviyeyi (maxSoundLevel) gösterir. Ayrıca, getSoundLevelDescription yardımcı fonksiyonu ile gürültü durumunu ("Sts: Loud", vb.) metin olarak kullanıcıya bildirir. drawLightScreen fonksiyonu ise LDR'den gelen anlık ışık değerini yüzde (%) cinsinden sunar, ayrıca çalışma süresi boyunca

görülen minimum ve maksimum ham ışık değerlerini göstererek kalibrasyona yardımcı olur. Her iki fonksiyon da ilgili sensör başlığını ekranın üst kısmına yerleştirir ve navigasyon tutarlılığını sürdürmek için alt kısımda <PREV ve NEXT> göstergelerini standart olarak içerir. Bu ayrım, her bir sensör verisinin benzersiz gösterim ihtiyaçlarına odaklanılmasını sağlar.

```
void monitorTemperature(void * parameter) {

    u8g2.begin();

    for (int i = 0; i < 32; i++) {
        soundHistory[i] = 0;
        lightHistory[i] = 0;
    }

    while (true) {
        u8g2.clearBuffer();

        switch (currentScreen) {
            case 0:
                drawDHTScreen();
                break;
            case 1:
                drawSoundScreen();
                break;
            case 2:
                drawLightScreen();
                break;
        }

        u8g2.sendBuffer();
        vTaskDelay(200 / portTICK_PERIOD_MS);
    }
}
```

Fig. 8.10: FreeRTOS Görevi (monitorTemperature / Display Manager)

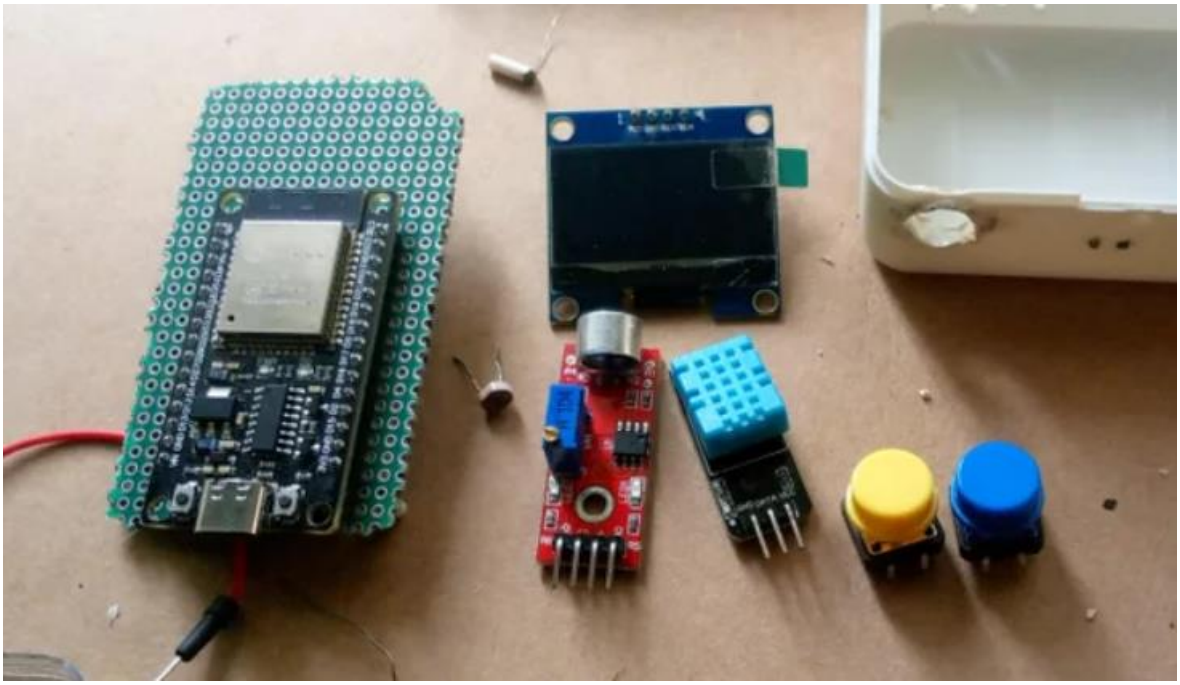
Fig. 8.10'daki görev, sistemdeki tüm sensörlerin verilerini görselleştiren ana Ekran Yöneticisi görevidir. Görev, öncelikle u8g2.begin() komutu ile OLED ekranı başlatır ve animasyon için kullanılan geçmiş dizilerini sıfırlar. Sonsuz döngü içerisinde, her döngü başında ekran içeriği temizlenir (u8g2.clearBuffer). Ardından, switch (currentScreen) yapısı kullanılarak, buton görevi tarafından belirlenen currentScreen değişkenine göre ilgili çizim fonksiyonu (drawDHTScreen, drawSoundScreen veya drawLightScreen) çağrılır. Çizim tamamlandıktan sonra, u8g2.sendBuffer() komutu ile hazırlanan görüntü fiziksel ekrana gönderilir. Bu görev, akıcı bir animasyon ve güncel veri gösterimi sağlamak amacıyla her 200 milisaniyede bir çalışacak şekilde gecikmeye alınır.

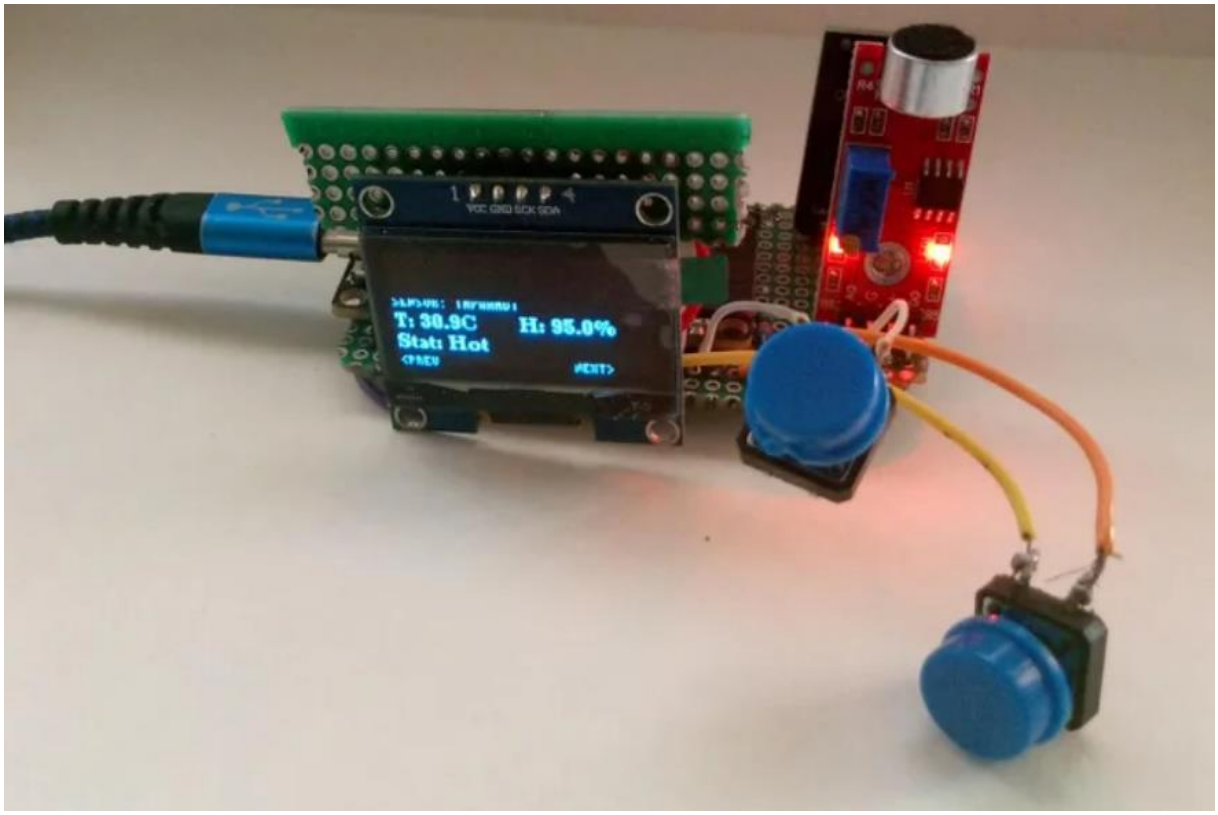
```
void setup() {
    Serial.begin(115200);

    dht.begin();

    tempC = 0.0;
    humidity = 0.0;
    lightValue = 0;

    initLight = analogRead(LIGHT_PIN);
    Serial.println("Initial light value: " + String(initLight));

    analogReadResolution(12);

    analogSetAttenuation(ADC_11db);

    pinMode(PREV_BUTTON_PIN, INPUT_PULLUP);
    pinMode(NEXT_BUTTON_PIN, INPUT_PULLUP);

    pinMode(SOUND_PIN, INPUT);
    pinMode(LIGHT_PIN, INPUT);

    Serial.println("Multi-Sensor Monitor Starting...");
    Serial.println("Screens: 0=DHT11, 1=Sound, 2=Light");

    xTaskCreatePinnedToCore(readSensorTask, "Read DHT11", 4096, NULL, 1, NULL, 0);
    xTaskCreatePinnedToCore(readSensorTask2, "Read Sound", 4096, NULL, 1, NULL, 0);
    xTaskCreatePinnedToCore(readSensorTask3, "Read Light", 4096, NULL, 1, NULL, 0);
    xTaskCreatePinnedToCore(buttonTask, "Button Handler", 2048, NULL, 2, NULL, 1);
    xTaskCreatePinnedToCore(monitorTemperature, "Display Manager", 4096, NULL, 1, NULL, 1);
}

void loop() {

}
```

Fig. 8.11: Kurulum (Setup) ve FreeRTOS Görev Başlatma

Fig.8.11'deki setup() fonksiyonu, programın tek seferlik başlangıç ve konfigürasyon işlemlerini gerçekleştirir. Seri iletişimi başlatır (Serial.begin) ve DHT sensörünü etkinleştirir. Analog-Dijital Dönüştürücü (ADC) için çözünürlük 12-bit olarak ayarlanır (analogReadResolution) ve giriş aralığı (0-3.3V) için uygun zayıflama (analogSetAttenuation) uygulanır. Ardından, butonlar ve analog sensörler için GPIO pinleri giriş olarak yapılandırılır. setup() fonksiyonunun en önemli kısmı, FreeRTOS görevlerinin oluşturulduğu yerdir: xTaskCreatePinnedToCore komutu kullanılarak her bir sensör okuma görevi, buton görev yöneticisi ve ekran yöneticisi görevleri oluşturulur. Bu görevler, ESP32'nin Core 0 veya Core 1 çekirdeklerine atanır ve her birine öncelik seviyesi atanır. Tüm görevler oluşturulduktan sonra, loop() fonksiyonu kasıtlı olarak boş bırakılır, çünkü tüm sistem mantığı FreeRTOS görevleri tarafından yönetilmektedir.

9. Montaj ve Sonuç Görselleri





10. Cihazın Kullanılışı

1. Güç Verme (Power On)

Cihazı kullanıma başlamak için tek yapmanız gereken cihazı bir USB kablosu aracılığıyla güç kaynağına bağlamak veya açma düğmesini çevirmektir. Güç verildikten hemen sonra, sistem başlatılır ve **OLED ekran** açılarak varsayılan başlangıç sensör ekranını gösterir. Bu, sistemin başarılı bir şekilde başlatıldığını onaylar.

2. Ekranlar Arası Gezinme (Navigate Screens)

Cihaz, ortamı izlemek için tasarlanmış farklı sensör verilerini gösteren toplam **üç ayrı ekrana** sahiptir. Kullanıcı, cihaz üzerindeki **iki adet butonu** kullanarak bu ekranlar arasında kolayca geçiş yapabilir:

- **MODE/NEXT Butonu:** Bir sonraki sensör ekranına geçişi sağlar.
- **PREV/HOLD Butonu:** Bir önceki sensör ekranına dönmeyi sağlar.

Bu butonlar sayesinde kullanıcı, dairesel bir sıra ile aşağıdaki sensör verilerini gösteren ekranlar arasında ileri ve geri hareket edebilir:

1. **Sıcaklık ve Nem Sensörü Ekranı:** DHT11 sensöründen gelen sıcaklık (T) ve nem (H) verilerini gösterir, ayrıca ortamın genel durumunu (Sıcak, Soğuk, vb.) belirtir.
2. **Işık Sensörü Ekranı:** LDR'den gelen ışık seviyesini yüzde olarak ve ortam aydınlatma durumunu (Parlak, Loş, vb.) gösterir.
3. **Ses Sensörü Ekranı:** KY-038 sensöründen gelen anlık gürültü seviyesi verilerini ve ortamın gürültü durumunu (Sessiz, Gürültülü, vb.) belirtir.

3. Gerçek Zamanlı İzleme ve Konumlandırma (Real-Time Monitoring)

Cihaz, gerçek zamanlı izleme prensibine göre çalışır:

- **Konumlandırma:** Cihazı, atölyenizde ortam koşullarını en iyi şekilde izleyebileceği merkezi bir konuma yerleştirin.
- **Anlık Güncelleme:** Ekranda gördüğünüz tüm sensör verileri (sıcaklık, nem, gürültü seviyesi, vb.), ESP32 ve FreeRTOS'un çoklu görev yeteneği sayesinde **anında ve kesintisiz** güncellenir.

11. Kaynakça

1. Espressif Systems. (2024). *ESP-IDF Programming Guide: FreeRTOS and Multitasking*. (ESP32 üzerindeki FreeRTOS görev yönetimi ve çoklu çekirdek programlama kılavuzu).

2. U8g2 Library. (2025). *OLED Display Control with U8g2: A Comprehensive Guide*. (SH1106 gibi OLED sürücüleri ile düşük seviyeli grafik çizim teknikleri ve optimizasyonları).

3. DHT Sensor Library. (2024). *Adafruit DHT Sensor Library Documentation*. (DHT11 sensör kütüphanesinin kullanımı).

4. U8g2 Library. (2025). *U8g2 Graphics Library for OLED/LCD Displays Documentation*. (SH1106 OLED ekran sürücüsü ve grafik çizimi).

5. FreeRTOS. (2024). *The FreeRTOS Kernel Quick Start Guide*. (Gerçek Zamanlı İşletim Sistemi görev yönetimi ve senkronizasyon prensipleri).

6. Mischianti, R. (2021). *ESP32 DOIT DEV KIT v1 Pinout*. (ESP32 pin konfigürasyonları ve donanım bağlantıları).

7. Analog-Digital Converter (ADC) Documentation. (2024). *ESP32 ADC API Reference*. (Analog sensörler (LDR, KY-038) için 12-bit çözünürlük ayarı).

8. Kundiman, K. D. (2025). *Swiss Army Multi-Function Sensor Tool.cpp*. (Projenin temel kaynak kodu).

9. Electronic Component Datasheets. (2024). *DHT11 Temperature and Humidity Sensor Datasheet*. (DHT11 teknik özellikleri ve doğrulama aralıkları).

10. Analog Devices. (2023). *Understanding Analog to Digital Converters (ADC)*. (Analog sensörlerden (LDR, KY-038) gelen sinyal işleme ve genlik hesaplama teknikleri).