Personal Assistant

Introducere

Personal assistant este un dashboard inteligent care poate comunica cu dispozitivele tale pentru a afișa informații despre calendar, remindere, vreme, oră, notificări, temperatură / umiditatea din cameră etc. Se poate conecta la telefon pentru a trimite notificări și are un buzzer pentru alarme / remindere. De asemenea, folosește un senzor de lumină pentru a te trezi odată cu răsăritul.

Scopul proiectului este de a avea acces ușor la informațiile utile ție. Proiectul este gândit să stea pe birou și să afișeze diverse lucruri, precum: valori de la senzori, informatii de la telefon, informatii de la un API etc. sau să te notifice de ceva prin intermediul buzzer-ului.

Inspirația pentru proiectul meu a venit în momentul în care am cumpărat un ”Sunrise Alarm Clock” de pe Temu care este configurabil printr-o aplicație de telefon. Mi s-a părut super mișto ideea de a controla un dispozitiv prin internet (nu mi-a plăcut niciodată să setez un ceas cu butoane) și m-am gândit să fac și eu ceva asemănător, dar mult mai configurabil. Am vrut totuși să păstrez funcționalitate de sunrise alarm :)

Cred că un astfel de dashboard este util pentru oricine, deoarece poate fi configurat să afișeze aproape orice folosind un ESP32.

Descriere generală

Proiectul are la bază un ESP32 si un display LCD. Am decis să folosesc un ESP32, deoarece am vrut conexiune la Internet și Bluetooth pentru a putea comunica cu telefonul / un server. Pe lângă aceste dispozitive, proiectul mai conține un buzzer pentru alerte sonore și doi senzori: un senzor pentru temperatură / umiditate și un senzor de lumină pentru detecția răsăritului.

Modul de funcționare este următorul:

- ESP32-ul cere informații din diverse surse: valori de la senzori, notificări de la telefon, lista cu remindere de la un server etc.
- Informațiile primite sunt prelucrate (datele sunt extrase dintr-un pachet Ethernet, valorile de la senzori sunt interpretate etc.)
- Microcontroller-ul actualizează UI-ul și trimite datele la display **sau** acționează buzzer-ul

Hardware Design

Listă de piese:

- 1 x Placă ESP-WROOM-32
- 1 x Display TFT SPI 2.8 inch 240×320 LCD cu touchscreen
- 1 x Senzor digital de lumina BH1750
- 1 x Senzor de Temperatura si Umiditate DHT22
- 1 x Buzzer Pasiv
- Fire de legătură
- 1 x Breadboard 830 puncte

O primă componentă importantă a proiectului meu este display-ul LCD. Acesta comunică cu microcontroller-ul prin protocolul SPI. Înafară de pinii obișnuiți pentru alimentare / SPI, display-ul mai are câțiva pini cu funcționalități specifice:

- Pin-ul **LED** este folosit pentru a controla luminozitatea display-ul. Momentan este conectat la o tensiune constantă de 3.3V, dar ulterior aș vrea să-l conectez la un pin cu PWM
- Pin-ul **DC** este folosit pentru a indica display-ului tipul de date primit. Există date obișnuite, care comunică informații despre starea unui pixel, și există date de comandă, care comunică informații legate de starea întregului display (orientare, sleep etc.)
- Pin-ul **RESET** este folosit pentru a reseta display-ul și a-l reinițializa

Componenta	Pin Componenta	Pin ESP32	Rol
TFT LCD	VCC	3.3V	Alimentare
TFT LCD	GND	GND	Ground
TFT LCD	CS	D5	Linie CS pentru SPI
TFT LCD	RESET	D4	Resetare display
TFT LCD	DC	D2	Selectare mod de comanda / date pentru display (1 - Mod date (coordonate, culoare pixel etc.); 0 - Mod comanda (schimbare orientare, sleep etc.)
TFT LCD	SDI	D23	Linie MOSI pentru SPI
TFT LCD	SCK	D18	Linie SCLK pentru SPI
TFT LCD	LED	3.3V	Luminozitate display

Următoarea componentă este senzorul de lumină BH1750. Senzorul comunică cu microcontroller-ul prin I2C. Singurul lucru de menționat aici este că nu am folosit pinii default de pe ESP32 pentru I2C, deoarece am vrut să realizez conexiunea printr-un singur cablu (dacă foloseam pinii default, trebuia să folosesc 2 fire și eram îngrijorat că o să fie afectat protocolul)

Componenta	Pin Componenta	Pin ESP32	Rol
BH1750	VCC	3.3V	Alimentare
BH1750	GND	GND	Ground
BH1750	SCL	D13	Linie SCL pentru I2C
BH1750	DAT	D14	Linie SDA pentru I2C

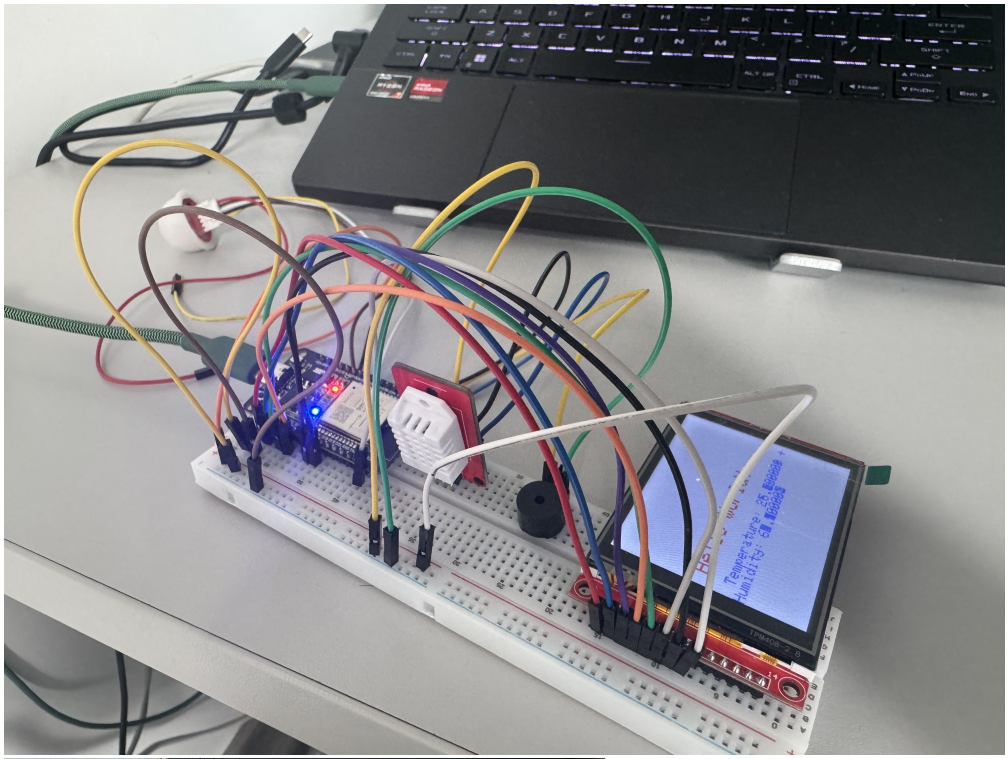
În continuare, trecem la senzorul de temperatură / umiditate DHT22. Senzorul are nevoie de alimentare la 5V, motiv pentru care l-am conectat la pin-ul VIN al microcontroller-ului

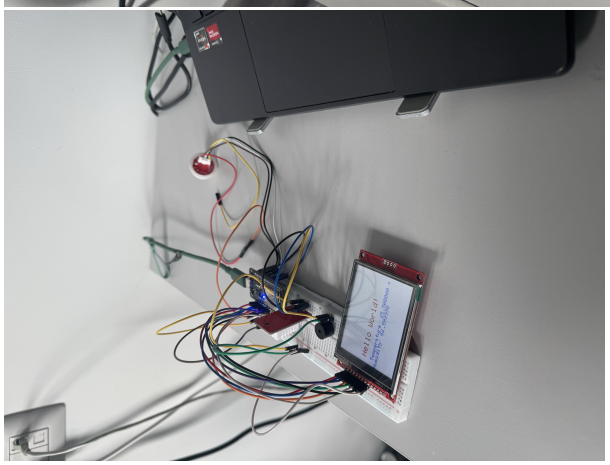
Componenta	Pin Componenta	Pin ESP32	Rol
DHT22	VCC	VIN	Alimentare
DHT22	GND	GND	Ground
DHT22	DAT	D25	Linie de date

Ultima componentă din listă este un buzzer pasiv conectat la un pin cu PWM

Componenta	Pin Componenta	Pin ESP32	Rol
Buzzer Pasiv	+	D32	Generare de frecvențe audio
Buzzer Pasiv	-	GND	Ground

Am asamblat toate componentele și am verificat că funcționează.





Software Design

Implementarea software curentă cuprinde următoarele funcționalități:

- conectarea la serviciile Bluetooth ale telefonului pentru baterie (Battery Service), timp (Current Time Service) și Apple Media (Apple Media Service)
- trimiterea de cereri HTTP către un server REST și parsarea răspunsului pentru a crea lista de remindere
- creerea intefeței grafice pentru afișarea valorilor de la senzori, a datei și timpului, a listei cu remindere și a melodiei redate pe telefon
- Dark Mode / Light Mode care se activează automat în funcție de intensitatea luminii
- activarea buzzer-ului timp de 1 minut la depășirea timpului pentru un reminder
- citirea datelor de la senzorul de temperatură și umiditate
- citirea datelor de la senzorul de lumină

În realizarea proiectului am folosit **PlatformIO** și framework-ul **Arduino**. Am ales PlatformIO, deoarece este foarte ușor să adaugi biblioteci externe și, în plus, am lucrat cu el și la laborator. În ceea ce privește framework-ul, a fost o decizie destul de grea. Inițial am vrut să lucrez cu framework-ul ESP-IDF, dar mi-am dat seama că va fi prea complicat și că mă voi încurca în detalii în loc să implementez funcționalitățile propriu-zise. În schimb, framework-ul Arduino mi-a dat libertatea să vin cu mai multe idei și să implementez funcționalități mai complexe (precum WiFi sau Bluetooth) care să evidențieze potențialul proiectului.

La implementarea proiectului am folosit mai multe biblioteci externe. O parte din ele sunt complet necesare (precum NimBLE sau WiFi), dar pe altele le-am folosit pentru ușurință:

- Arduino.h - Debugging (prin Serial) și setarea modului pinilor (Input/Output)
- TFT_eSPI - Realizarea interfeței grafice. Am ales această bibliotecă, deoarece este foarte ușor de configurat și ocupă puțin spațiu (am încercat și cu LVGL, dar nu aveam spațiu)
- SPI.h - Este folosită de TFT_eSPI pentru gestionarea comunicării prin SPI cu display-ul
- NimBLEDevice - Realizarea conexiunii Bluetooth. Am ales să folosesc NimBLE, deoarece este compatibil cu dispozitive Apple
- time.h - Conversia de la UNIX Timestamp la un format human-readable pentru afișare
- DHT.h - Citirea datelor de la senzorul de temperatură/umiditate. Senzorul pe care l-am ales (DHT22) trimite datele sub formă de semnal digital și folosește un protol special, motiv pentru care am decis că e mai ușor să folosesc o bibliotecă
- Wire.h - Configurare protocolului I2C pentru senzorul de lumină
- BH1750.h - Citirea datelor de la senzorul de lumină
- ezBuzzer.h - Acționarea buzzer-ului. Folosesc această bibliotecă, deoarece permite rularea asincronă a buzzer-ului (fară funcții de delay care să deconecteze Bluetooth-ul)
- WiFi.h - Realizarea conexiunii WiFi
- HTTPClient.h - Crearea și trimiterea cererilor HTTP

Elementul de noutate al proiectului constă în posibiliățile pe care le oferă. Fiind conectat la WiFi și la Bluetooth înseamnă că poți obține orice fel de informații, iar cu ajutorul bilbiotecii TFT_eSPI îți poți crea ușor propria interfață.

Scheletul codului:

Funcție	Rol
batteryStatusCallback	Actualizează variabila *batteryStatus* cu noul nivel al bateriei de la telefon
subscribeToBatteryService	Subscribe la serviciul de Bluetooth pentru baterie. Se primesc notificări când se modifică nivelul bateriei și sunt gestionate de funcția batteryStatusCallback
amsCallBack	Actualizează informațiile despre melodia redată pe telefon (titlu, artist, durată, stare etc.). Pentru a determina tipul informației primite, am urmărit specificațiile din documentația AMS
subscribeToAMS	Subscribe la Apple Media Service
printTime	Afișează (pe Serial) data curentă
getCurrentTime	Sincronizează ceasul microcontroller-ului cu ceasul de la telefon (prin Bluetooth)
printTrack	Afișează informații (pe Serial) despre melodia redată
sensorCallback	Citește valorile senzorilor și modifică modul de afișare (Dark Mode / Light Mode) dacă este nevoie. Funcția este un task FreeRTOS care se apelează automat la 2 secunde
reminderCallback	Verifică lista cu remindere și semnalează pornirea alarmei. Funcția este un task FreeRTOS care se apelează automat la o secundă
httpGETRequest	Creează și trimite cererea HTTP la serverul REST. Întoarce răspunsul de la server
WiFiStationDisconnected	Încearcă reconectarea la WiFi
parseReminderList	Se parsează răspunsul primit de la server și se actulizează lista cu remindere. Răspunsul conține titlul și timpul reminder-ului
drawCustomCentredString	Afișează un text centrat la coordonatele date
drawDividers	Desenează liniile orizontale care delimitează secțiunile interfeței grafice
updateHeaderView	Afișează pe ecran timpul și informațiile de la senzori
updateDateBarView	Afișează pe ecran data curentă
drawRemindersList	Desenează pe ecran lista cu remindere
updateMediaScrubber	Afișează pe ecran titlul melodiei redate pe telefon și desenează o bară care se actualizează cu progresul melodiei
setup	Se inițializează senzorii, display-ul și conexiunea la WiFi și Bluetooth
loop	Se actulizează interfața grafică, se acționează buzzer-ul și se fac periodic cereri la serverul HTTP

Videoclip demo: https://youtube.com/shorts/l6caujJPsVg

Optimizările pe care le-am realizat sunt legate de Bluetooth. Prima optimizare a fost alegerea BLE (Bluetooth Low Energy) în loc de Bluetooth obișnuit. BLE consumă mult mai puțină energie (pe microcontroller, dar și pe telefon) și oferă toate serviciile de care am nevoie. A doua optimizare constă în folosirea funcției de *subscribe* la un serviciu Bluetooth. Folosind subscribe, primesc notificări doar atunci când se modifică datele (de exemplu se schimbă melodia) și nu trebuie să fac polling constant. De asemenea, în cadrul funcțiilor de callback (funcții care se apelează automat când primesc o notificare Bluetooth) am evitat instrucțiunile complexe și doar am setat anumite flag-uri, pe care le-am tratat ulterior în loop.

Link Github: https://github.com/neho2245/proiect_pm

Concluzii

Download

O arhivă (sau mai multe dacă este cazul) cu fişierele obţinute în urma realizării proiectului: surse, scheme, etc. Un fişier README, un ChangeLog, un script de compilare şi copiere automată pe uC crează întotdeauna o impresie bună .

Fişierele se încarcă pe wiki folosind facilitatea **Add Images or other files**. Namespace-ul în care se încarcă fişierele este de tipul **:pm:prj20??:c?** sau **:pm:prj20??:c?:nume_student** (dacă este cazul). **Exemplu:** Dumitru Alin, 331CC → **:pm:prj2009:cc:dumitru_alin**.

Bibliografie/Resurse

Resurse:

- https://developer.apple.com/library/archive/documentation/CoreBluetooth/Reference/AppleMediaService_Reference/Introduction/Introduction.html
- https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Assigned_Numbers/out/en/Assigned_Numbers.pdf
- https://github.com/Bodmer/TFT_eSPI
- https://github.com/h2zero/NimBLE-Arduino
- https://esp32.com/viewtopic.php?t=9875

Export to PDF

From:
https://ocw.cs.pub.ro/courses/ - **CS Open CourseWare**

Permanent link:
https://ocw.cs.pub.ro/courses/pm/prj2026/ciprian.popescu0411/alexandru.nehoianu

Last update: **2026/05/23 17:28**