Fire Alarm

Autor: Cosmin-Răzvan Vancea (333CA)

Introducere

- Proiectul constă în realizarea unei alarme de incendiu care anunță utilizatorul pe telefon în cazul în care detectează fum sau flăcări.
- Scopul proiectului este de a detecta incendii, în cazuri extreme un astfel de sistem putând fi un factor decisiv în salvarea de bunuri și vieți.

Descriere generală

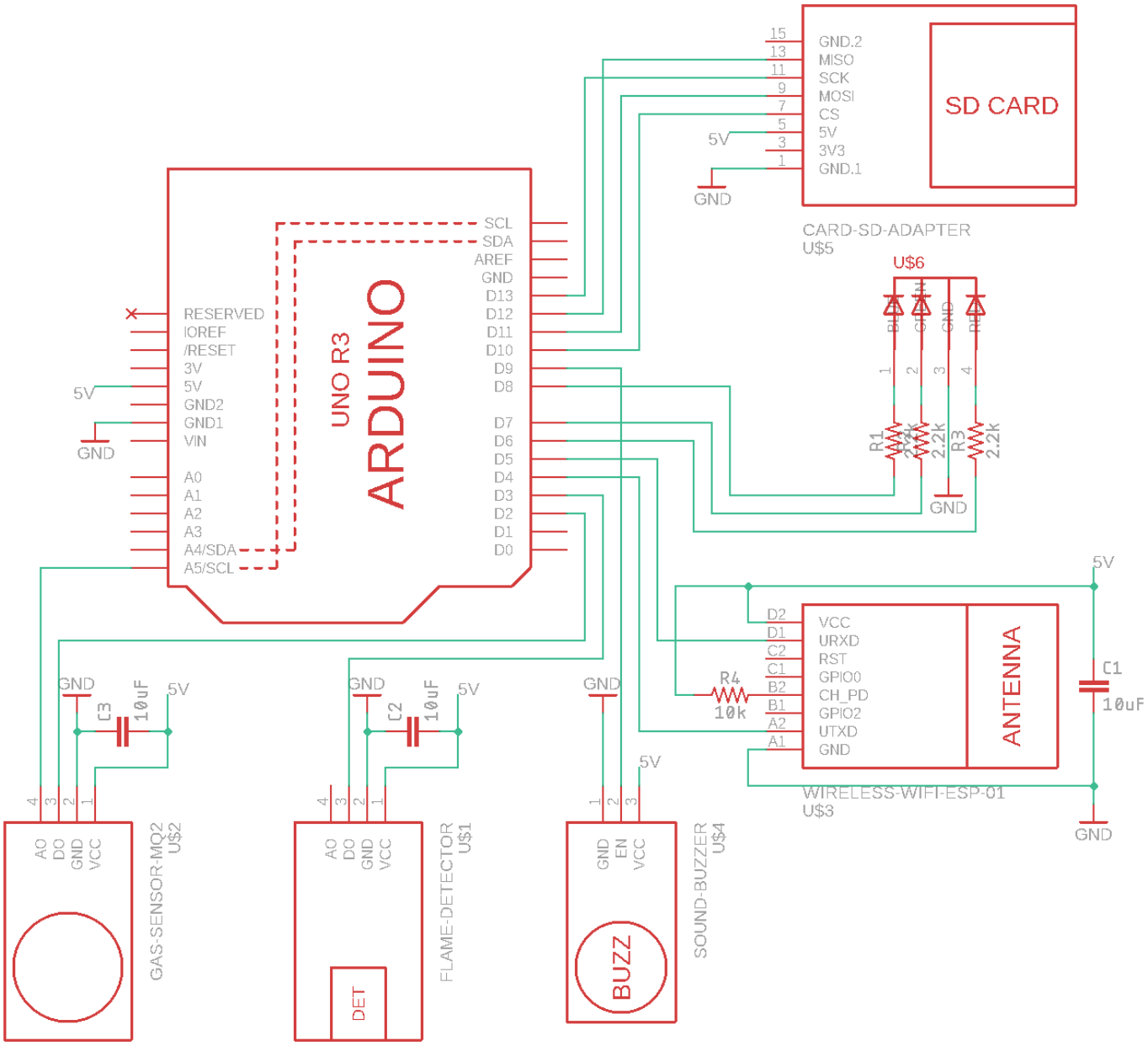
- La inițializare, placa Arduino citește de pe un SD Card următoarele date de configurare:
  1. rețeaua Wi-Fi la care se conectează
  2. serverul la care va trimite datele
- Utilizatorul este notificat prin culoarea LED-ului dacă datele de configurare au fost validate sau nu.
- Placa Arduino citește datele primite de la senzorii de fum si flăcări.
- În cazul în care cel puțin unul dintre senzori indică un incendiu, se va trimite un semnal către server cu ajutorul modulului Wi-Fi și se va activa alarma sonoră (buzzer).
- Când serverul primește un semnal de la alarmă, acesta notifică mai departe telefonul proprietarului și/sau alte dispozitive.

Hardware Design

Componente

1. Arduino UNO
2. Senzor de gaze MQ-2
3. Senzor de flacără IR
4. Modul Wi-Fi ESP-01
5. Modul SD Card
6. Modul buzzer activ
7. Led RGB
8. Rezistențe (3x 2.2k + 1x 10k)
9. Condensatoare (3x 10uF)
10. Breadboard
11. Mini breadboard
12. Fire tată-tată (3 seturi)
13. Fire mamă-tată (1 set)

Schema electrică



Software Design
Informații generale

Proiectul este compus din două componente software:

1. software-ul pentru Arduino: responsabil cu citirea senzorilor și transmiterea datelor către server
2. software-ul pentru server: responsabil cu prelucrarea și stocarea datelor primite, urmată de notificarea dispozitivelor clienților

Codul Arduino a fost dezvoltat în Visual Micro.

Biblioteci

Pentru realizarea proiectului am utilizat următoarele biblioteci 3rd-party:

- SDLib: permite accesarea cardului SD unde este stocat fișierul de configurare
- IniFile: permite citirea fișierului `/fire-alarm/net-config.ini` de pe cardul SD

În plus am dezvoltat următoarele două mini-biblioteci pentru a ușura interacțiunea cu dispozitivele conectate:

- ESP8266: abstractizează comenzile date plăcii Wi-Fi ESP-01. În mod normal, ESP-01 primește comenzi de forma `AT+<cmd> <parametri>` și returnează un răspuns pe interfața serială. Biblioteca are rolul de a ascunde aceste detalii în spatele unei interfețe high-level (clasă C++), astfel nu va trebui să lucrez direct cu interfața serială. În plus, biblioteca oferă suport minimalist pentru trimiterea de cereri HTTP.
- AsyncADC: permite inițializarea unei conversii ADC fără să se aștepte rezultatul. Se poate utiliza în cadrul întreruperilor deoarece timpul de execuție este foarte scurt (doar se face o scriere într-un registru de control). Biblioteca abstractizează lucrul cu regiștrii de control ai ADC-ului (`ADCSRA` și `ADMUX`).

Funcții implementate

Mai jos sunt detaliate cele mai importante funcții din fișierul sursă principal: fire-alarm.ino.

Funcții de **`setup()`**:

- `InitSDCard`: inițializează modulul pentru SD Card și citește datele de configurare din fișierul `/fire-alarm/net-config.ini`
- `InitWiFi`: așteaptă inițializarea modulului Wi-Fi și se conectează la AP-ul (router wireless) specificat în fișierul de configurare de către utilizator
- `BlinkForever`: în cazul în care se detectează o eroare la oricare pas de inițializare descris anterior, se apelează această funcție care blochează execuția codului și va seta LED-ul să emită o anumită culoare care semnalează eroarea întalnită.
- `SetLEDColour`: setează culoarea LED-ului RGB. Acest LED este folosit pentru a semnala starea în care se află aparatul (ex: verde = inițializat + stare OK de funcționare)

În plus, în faza de `setup` se activează întreruperile pe liniile `INT0` (senzor fum) și `INT1` (senzor flăcări) și se atașează câte un ISR care va fi executat la schimbarea valorilor.

Funcții de **`loop()`**:

- `PostDataToServer`: se reconectează la AP (în cazul în care s-a pierdut conexiunea) și se postează un mesaj HTTP la endpoint-ul specificat în fișierul de configurare. Dacă nu se poate trimite mesajul, atunci se va semnala eroarea prin LED și se va reîncerca transmiterea pachetului la fiecare 10 secunde până se reușește. Spre deosebire de o eroare de inițializare, aici execuția codului nu se mai oprește.
- `SendNotification`: se generează body-ul cererii HTTP POST în funcție de datele citite de la senzori, apoi se apelează `PostDataToServer` pentru a trimite cererea.

Pe langă semnalizarea prin LED, dispozitivul va scrie pe interfața serială mesaje de logging la fiecare pas important (în special: inițializare SD Card/Wi-Fi și trimiterea de mesaje HTTP). Acest lucru ajută la depanarea problemelor.

Rutine pentru **întreruperi**:

- `GasInterrupt`: apelată la schimbarea valorii de pe pinul digital asociat senzorului de fum. Dacă noua valoare este `LOW` înseamnă că senzorul a detectat fum și se va inițializa `AsyncADC`-ul definit anterior pentru a porni citirea de pe pinul analog asociat. Se va seta o variabilă globală pentru a marca evenimentul.
- `FlameInterrupt`: apelată la schimbarea valorii de pe pinul digital asociat senzorului de flăcări. Dacă noua valoare este `LOW` înseamnă că senzorul a detectat flăcări și se va marca faptul că s-au detectat flăcări într-o variabilă globală
  - *notă:* din păcate modulul pentru senzorul de flăcări folosit de mine nu are și un pin analog de pe

care să pot citi intensitatea.

De asemenea, la detectarea de fum sau flăcări se va activa alarma sonoră. Aceasta se oprește când ambii senzori nu mai detectează nimic suspect.

După cum se observă, în rutinele de întreruperi nu se întamplă nimic excepțional, maxim se scrie în 1-2 variabile/regiștri. Codul care se ocupă cu notificarea se află în `loop`. Aici la fiecare *X* secunde se verifică dacă senzorii au detectat ceva (se interoghează variabilele globale scrise în ISR), iar în caz afirmativ, se va citi valoarea senzorului de fum prin `AsyncADC` și se vor trimite măsurătorile la server. Motivul pentru care am optat să implementez și să folosesc `AsyncADC` în loc de un simplu `analogRead` în `loop` este că de la momentul în care este declanșată întreruperea până la momentul în care se face polling în `loop` pot trece multe secunde/minute, iar astfel senzorul poate să îmi returneze valori normale în momentul când eu le citesc în `loop`. Folosind `AsyncADC` mă asigur că în `loop` voi avea acces la datele care erau citite acum *Y* secunde când s-a declanșat întreruperea.

Fișierul de configurare

Utilizatorul dispozitivului poate specifica datele de configurare fără a fi nevoit sa recompileze software-ul – doar trebuie să modifice un fișier de configurare de pe un SD Card! Fișierul trebuie plasat pe card la calea: `/fire-alarm/net-config.ini`. Exemplu:

net-config.ini

```
[wifi]
ssid=HUAWEI-abcd                          ; nume rețea wireless
pass=my-extremely-hard-to-guess-password  ; parolă wireless

[server]
host=fire-alarm.example.com               ; adresa serverului
destinație
port=7331                                 ; portul serverului
destinație
endpoint=/api/v1/measurement/add          ; calea către endpoint-ul
unde se vor posta măsurătorile
guid=744A2B95-D130-40A1-94E3-D8719FEDCCCD ; identificator unic pentru
fiecare dispozitiv "Fire Alarm"
```

Optimizări de memorie

Arduino UNO dispune de două spații de memorie diferite: 2K memorie SRAM și 32K memorie flash. Memoria SRAM mică limitează foarte mult dimensiunea datelor cu care pot lucra (inclusiv string-uri → acestea sunt stocate în SRAM, deci în cel mai bun caz aș putea avea maxim ~2000 de caractere fără să mai iau în considerare stack-ul și restul datelor din program).

Pentru a ocoli această limită, am ales să stochez datele constante (eg: string-uri) în memoria flash folosind keyword-ul PROGMEM oferit de AVR și să le aduc în SRAM doar când am nevoie să le prelucrez (eg: formatarea unei cereri HTTP/unui mesaj de eroare). Acest trick îmi permite să:

1. am string-uri de dimensiune mare în program (eg: cereri HTTP neformatate)
2. am mesaje de eroare foarte explicite
3. reduc la minim utilizarea SRAM-ului

În plus, majoritatea funcțiilor care acceptă string-uri constante implementate de mine au două forme: (vezi ESP8266.h)

1. una care accepta string-uri din SRAM
2. una care acceptă string-uri din FLASH

Server

Software-ul pentru server este realizat în Flask. Practic este un web server care:

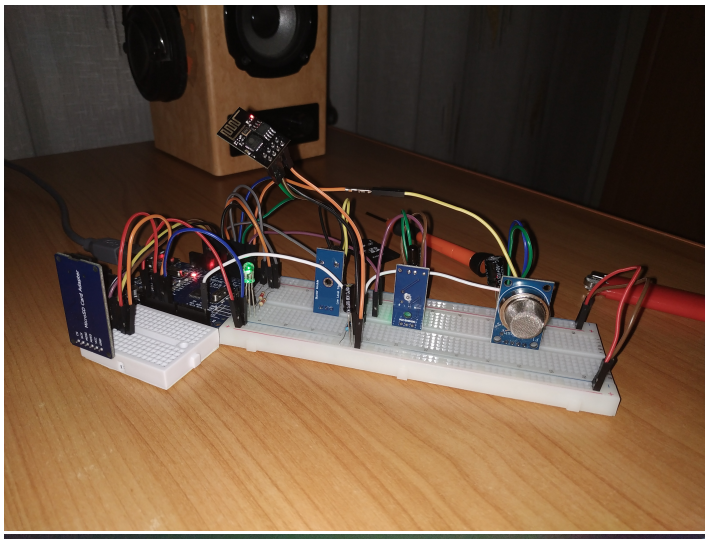
1. expune un endpoint la care dispozitivul Arduino trimite măsurătorile împreuna cu GUID-ul său (fiecare dispozitiv Fire Alarm este identificat printr-un GUID specificat în fișierul de configurare)
2. oferă utilizatorului o interfață web prin care poate citi măsurătorile senzorilor săi (trebuie să cunoască GUID-ul)

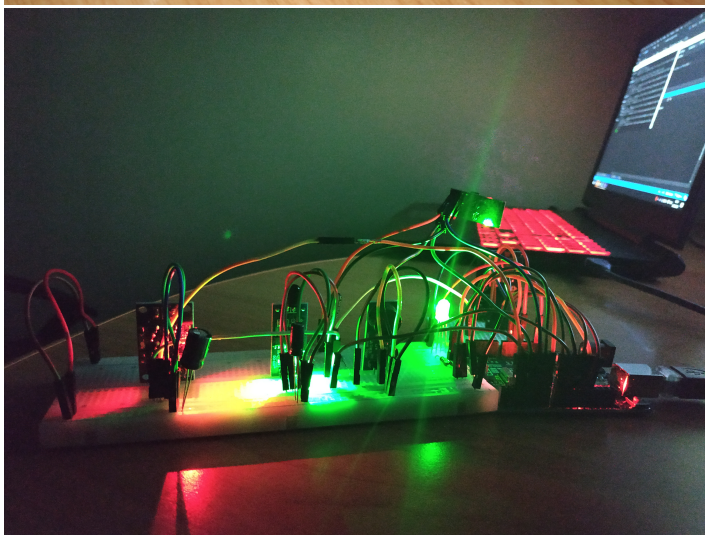
Notificarea pe telefon se face prin serviciul PushBullet. În interfața web, proprietarul poate asocia un cont de PushBullet dispozitivelor sale. Fiecare măsurătoare care depășește valorile normale va fi trimisă ca notificare contului PushBullet asociat.

Detaliile de implementare nu intră în materia cursului.

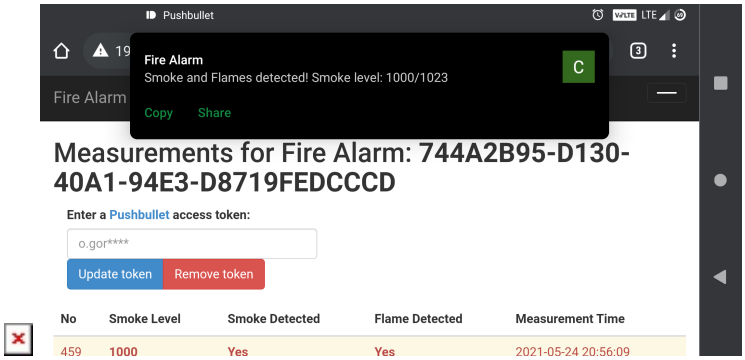
Rezultate obținute

Poze dispozitiv





Interfața cu utilizatorul



Videoclip de prezentare

Concluzii

A fost o experiență productivă deoarece proiectul ales de mine a trebuit să fie gândit din două perspective diferite care trebuie să se întâlnească într-un punct comun: un sistem local ce colectează date de la senzori și un sistem remote ce primește aceste date, le prelucrează și le distribuie mai departe. Pe lângă implementarea efectivă, a fost nevoie de multă documentare în prealabil, iar astfel m-am familiarizat cu căutarea de informații în documentația oficială.

Download

- Cod sursă (arhivă)
- Cod sursă (GitHub)

Jurnal

- 19 aprilie 2021 - sosire componente
- 23 aprilie 2021 - realizat ansambul hardware
- 24 aprilie 2021 - creată pagina proiectului
- 03 mai 2021 - implementat software-ul pentru Arduino
- 06 mai 2021 - implementat software-ul pentru Server
- 24 mai 2021 - completată pagina proiectului

Bibliografie/Resurse

- ATmega328P Datasheet (ADC & Timers)
- ESP8266 AT Instruction Set
- Arduino SoftwareSerial
- Arduino SD Library
- Arduino SD Card Best Practices
- Arduino PROGMEM
- Flask Documentation
- PushBullet API Documentation

Export to PDF