Flexibele draadloze communicatie met een MCU

EEPROM opent netwerkmogelijkheden voor draadloze MCU's

Gamal Labib (Egypte)

Wanneer je een microcontroller verbindt met een WiFi-netwerk met behulp van de ESP8266, wil je misschien een meer flexibele aanpak dan hard-gecodeerde, vaste WLAN-referenties. In dit artikel demonstreer ik een oplossing met een set voorkeurs-AP netwerken waar je interactief uit kunt kiezen. Daarnaast kunnen we gebruik maken van de WiFi Protected Setup-functie (WPS) die door veel AP's en routers wordt ondersteund.

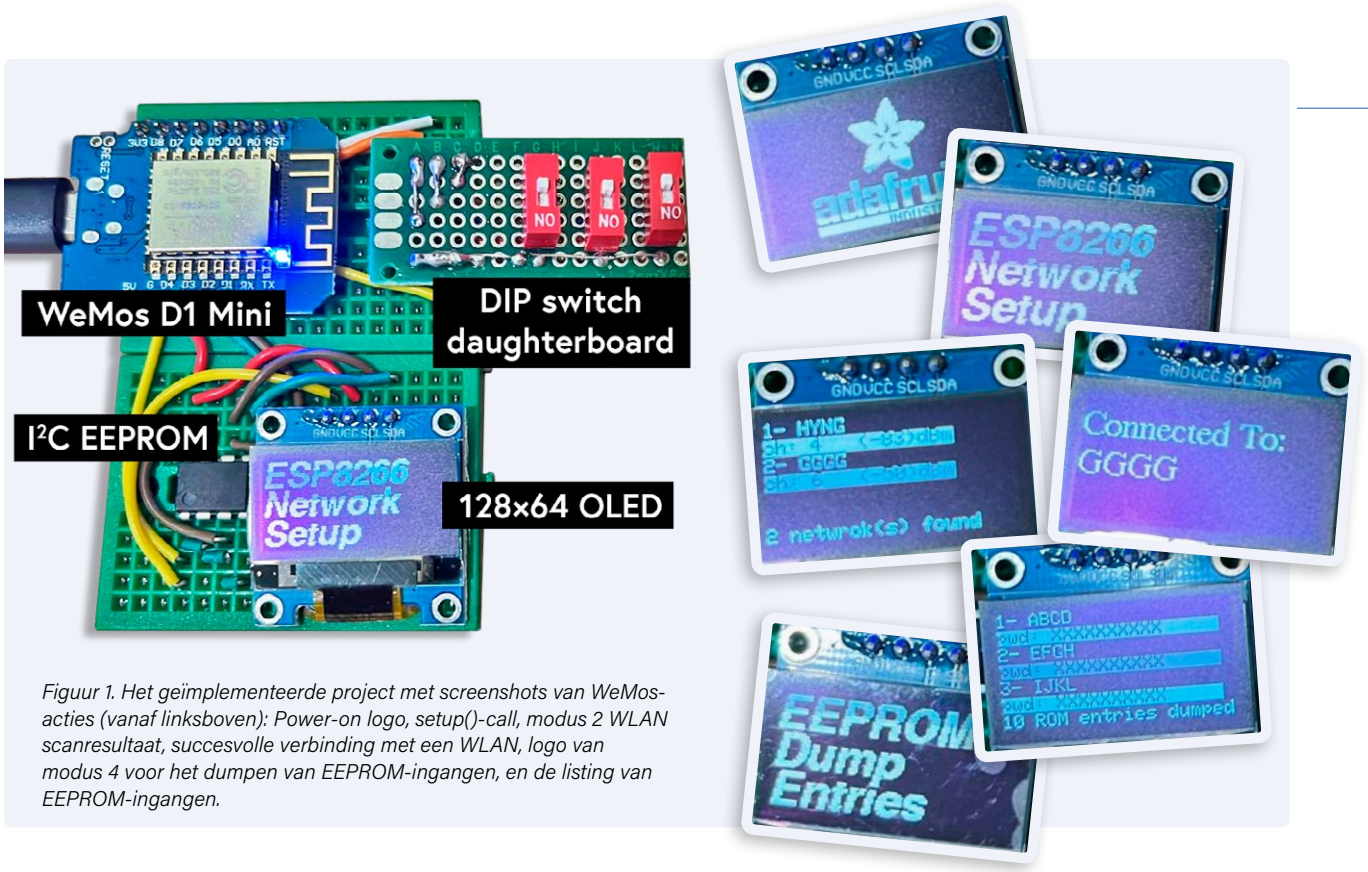
De ESP8266 is een populaire microcontroller (MCU) module die draadloze communicatie ondersteunt en voor diverse IoT-toepassingen wordt gebruikt. De boards op basis van deze module hebben meestal hard-gecodeerde Access Point-referenties, zoals de Service Set Identifier (SSID) en het wachtwoord, in hun Arduino-sketches. In sommige gevallen geven ontwikkelaars ook vaste IP-instellingen op. Deze vaste instellingen kunnen echter problemen veroorzaken wanneer de topologie van het lokale draadloze netwerk (WLAN) verandert. Het is een gebed zonder einde om terug te keren naar de Arduino IDE of een vergelijkbare ontwikkelomgeving alleen maar om de sketches bij te werken op geïmplementeerde MCU-boards, omdat de boards teruggehaald of vervangen moeten worden.

In dit artikel laat ik een oplossing zien die flexibiliteit biedt bij het verbinden van een MCU-board met een WLAN. Waarom gebruiken we, in plaats van een enkele set hard-gecodeerde WLAN-referenties in de sketch van een project, geen set AP-voorkeursnetwerken waaruit interactief gekozen kan worden? Door een interactieve dialoog tot stand te brengen tussen de gebruiker en het MCU-board via een aanraakscherm, webpagina of zoiets wordt het mogelijk om direct nieuwe WLAN-instellingen voor het MCU-board op te geven. Daarnaast kunnen we gebruik maken van de WPS-functie (WiFi Protected Setup) die door veel AP's en routers wordt ondersteund. Met WPS kan een MCU-board naar wens verbinding maken met het voorkeursnetwerk van de gebruiker. Er rijst echter een vraag: moeten we elke keer als het MCU-board opnieuw wordt opgestart, de aanmeldingsstappen van het interactieve dialoogvenster of WPS doorlopen? Het antwoord is nee, zolang we de nieuwe WLAN-instellingen opslaan in een niet-vluchtige geheugen dat toegankelijk is voor de MCU. Gelukkig heeft de ESP8266 een interne EEPROM (electrically erasable programmable read-only memory) die kan worden gemanipuleerd door gebruikerscode om gegevens op te slaan en op te halen. Ik heb deze functie gebruikt om tot tien WLAN-referenties op te slaan die ingesteld zijn op één van de bovengenoemde manieren. Deze aanpak biedt niet alleen netwerkflexibiliteit, maar stelt het MCU-board ook in staat om verbinding te maken met dat WLAN van de opgeslagen 10 netwerken dat de beste dekking biedt, als de ontwikkelaar daarvoor kiest.

Het is echter belangrijk op te merken dat veelvuldig schrijven naar de interne EEPROM niet wordt aanbevolen, omdat de verwachte levensduur van een EEPROM gebaseerd is op het aantal schrijfcycli dat hij ondergaat. Om deze beperking te omzeilen en de aangewezen rol van de MCU te behouden, heb ik een externe EEPROM-chip opgenomen in mijn opstelling, zodat de gebruiker kan onderzoeken of het mogelijk is om hetzelfde te doen met een externe EEPROM. Voor ons WLAN-configuratievoorbeeld hoeven we echter niet veel naar EEPROM te schrijven, dus heb ik gekozen voor de interne EEPROM.

Hardware

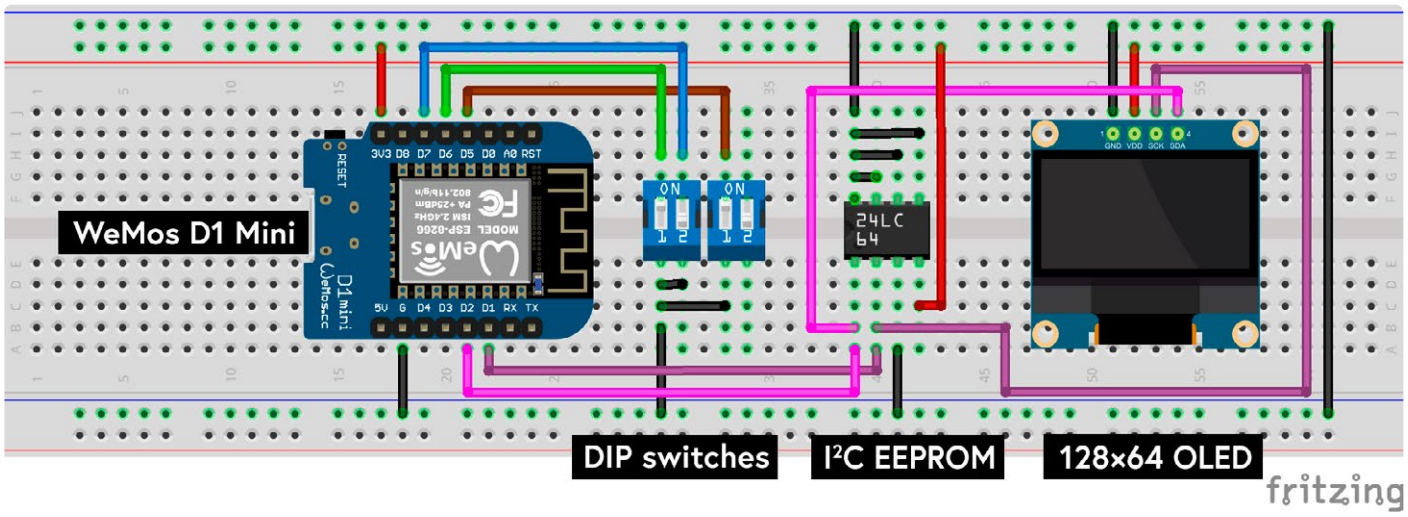
De onderdelenlijst voor dit project is vrij beknopt. Ik heb de WeMos D1 Mini gebruikt, een op een ESP8266 gebaseerd board dat bekend staat om zijn eenvoudige gebruik binnen de Arduino IDE en zijn kleine footprint. Een 0,9"-OLED grafische displaymodule met een resolutie van 128×64 pixels wordt gebruikt om informatie en debugberichten weer te geven. Hij is verbonden met de I²C-bus van het MCU, samen met een externe EEPROM-chip van 8 KB. Om te kiezen uit de acht bedrijfsmodi van de MCU (zie **tabel 1**), heb ik een drietal DIP-schakelaars gebruikt, die zijn aangesloten op drie van de digitale GPIO's van de WeMos (bijvoorbeeld D5, D6 en D7). Omdat de DIP-schakelaars niet op het breadboard passen, heb ik een dochterprint gemaakt om ze aan te sluiten. De breadboard-implementatie



Figuur 1. Het geïmplementeerde project met screenshots van WeMos-acties (vanaf linksboven): Power-on logo, setup()-call, modus 2 WLAN scanresultaat, succesvolle verbinding met een WLAN, logo van modus 4 voor het dumpen van EEPROM-ingangen, en de listing van EEPROM-ingangen.

Tabel 1. DIP-schakelaars voor de diverse WeMos-modi.

Modus	DIP 1	DIP 2	DIP 3	WeMos-actie
1	0	0	0	selectie standaard-WLAN
2	0	0	1	scan beschikbare WLAN's
3	0	1	0	interne EEPROM wissen
4	0	1	1	interne EEPROM dumpen
5	1	0	0	interne EEPROM initialiseren
6	1	0	1	externe EEPROM controleren
7	1	1	0	WPS
8	1	1	1	interactieve WLAN-setup



Figuur 2. Bedrading van het project met behulp van het Fritzing breadboardtool en met een extra vierde DIP-schakelaar voor uitgebreide netwerkopties.

van het project en screenshots van de WeMos in actie zijn te zien in **figuur 1**. Het bedradingsschema van de diverse onderdelen van het project is te zien in **figuur 2**.

Software

Het project legt de nadruk op *best practices* in softwareontwikkeling, zoals goed gestructureerde en herbruikbare code. Om dit te bereiken heb ik de 800 regels tellende sketch opgedeeld in zes aparte bestanden, die elk een specifiek stuk hardware of functionaliteit betreffen (zie **tabel 2**). Deze aanpak maakt de code beter beheersbaar en helpt lezers de complexe code te begrijpen. Het hoofd-sketchbestand richt zich op de WeMos netwerkmodi binnen de `setup()`-functie, waarbij de `loop()`-functie vrij blijft voor de WeMos-applicatiecode. Tijdens de ontwikkelingsfase gebruikte ik `loop()` om het knipperen van de LED's op het board te manipuleren en te controleren of de WeMos correct functioneerde.

EEPROM gebruiken voor WLAN-netwerken

De ESP8266-module heeft 4 MB flashgeheugen, waarvan 4 KB is gereserveerd om een EEPROM te emuleren die normaal gesproken is ingebouwd in Arduino-apparaten. Onze voorbeeldsketch slaat standaard de laatste bruikbare AP-referenties op in de interne EEPROM. Met interne (of externe) niet-vluchtige opslag kan onze sketch bij elke keer opstarten de AP-referenties oproepen na power-up. De EEPROM is ook toegankelijk door middel van functie-aanroepen van de EEPROM-bibliotheek van de Arduino en ik zal deze richting volgen om die netwerkgegevens op te slaan die niet mogen verdwijnen wanneer de module wordt uitgeschakeld. Het belangrijkste concept is om de voorgestelde structuur in **tabel 3** voor WLAN AP-referenties te behouden – zoveel als de interne of externe EEPROM kan bevatten. Elk volgende item in de structuur bevat het SSID of het wachtwoord van het AP in respectievelijk 32 en 64 bytes opslagruimte. Dat is in totaal 96 bytes per AP. Wanneer de gebruiker een nieuwe WLAN toevoegt aan de lijst met voorkeursnetwerken, voegt de code deze structuur toe met de referenties van het SSID en het wachtwoord van het nieuwe netwerk. De gebruiker moet de EEPROM wissen in de eerste run van de projectcode, in **modus 3**, waardoor het einde van de lijst met lege items wordt gemarkeerd. De gebruiker heeft de keuze om de hard-gecodeerde WLAN-referenties toe te passen in modus 1, of om de connectiviteit te controleren met behulp van de referenties in EEPROM, wat in modus 8 gebeurt. Mijn implementatie voor dit project beperkt zich tot dit mechanisme voor maximaal 10 WLAN's, op basis van een vooringestelde constante `MEMCNT` in de code, naast het wissen van de structuur en het initialiseren met een vooraf ingestelde lijst van AP-referenties naar keuze van de gebruiker. De lezer kan de grootte van `MEMCNT` naar behoefte vergroten of verkleinen.
Achterinfo over de toegang tot de EEPROM is te vinden in [2]. Nu ga ik dieper in op de werkingsmodi van de WeMos, waarvoor de DIP-schakelaars telkens moeten worden ingesteld (zoals in tabel 1) en vervolgens de WeMos moet worden herstart om de instelling toe te passen.

Modus 1: standaard WLAN-instellingen (hardgecodeerd)

Een MCU heeft WLAN-referenties nodig, bestaande uit een SSID en een wachtwoord voor het betreffende toegangspunt. Het doorgeven van WLAN-referenties kan hard gecodeerd worden in de sketch die geflashed wordt, of interactief aan de MCU worden doorgegeven wanneer de gebruiker is uitgerust met een toetsenbord en een beeldscherm of een seriële monitor in de IDE die via de seriële poort is verbonden. In dit project gebruik ik de conventionele manier om AP-referenties direct in de sketch vast te leggen.

Tabel 2. De verschillende Arduino-codebestanden en hun functies.

.ino-bestand	Beschrijving	Aantal regels
wemos-d1-mini-network	Globale variabelen, setup netwerkmodus	240
wlan-utils	WLAN-verbindingen manipuleren: WPS-setup, WLAN's scannen/opsommen, connectiviteit controleren	110
tft-utils	TFT-display manipuleren	130
internal-eeprom-utils	EEPROM initialiseren met vijf vooraf ingestelde WLAN-referenties, EEPROM resetten, gegevens lezen/schrijven	180
external-eeprom-utils	Externe EEPROM-connectiviteit en grootte controleren	40
ap-web-setup	Interactieve WLAN-instelling: webserver starten, gedetecteerde WLAN-SSID's tonen, geselecteerde WLAN opslaan in EEPROM	100

Tabel 3. EEPROM-huishoudtabel voor meerdere AP-instellingen.

Grootte item (bytes)	Beschrijving	Type
2	aantal AP-ingangen	integer
32	AP1-SSID	string
64	AP1-wachtwoord	string
.	.	.
.	.	.
.	.	.
32	APn SSID	string
64	APn-wachtwoord	string

Modus 2: WLAN scannen

In deze modus verbreekt de WeMos de verbinding met een eventueel verbonden WLAN om te kunnen scannen naar andere toegankelijke netwerken. Gedetecteerde WLAN's worden opgesomd met hun SSID's en signaalsterkte in decibel (dB). Deze modus geeft de

```
14:34:00.668 -> scan available wlans
14:34:05.788 -> Disconnecting previously connected WiFi
14:34:08.148 -> scan completed
14:34:08.148 -> 1 Networks found
14:34:08.148 -> 1: GGGG (6)  (-63)
14:34:08.268 ->
```

Figuur 3. Berichten in de Arduino IDE seriële monitor – modus 2 scan naar draadloze netwerken.

```
14:35:43.671 -> check external eeprom
14:35:48.751 -> Disconnecting previously connected WiFi
14:35:48.871 -> check external eeprom
14:35:48.871 -> External eeprom isConnected with Status:
14:35:48.871 ->
14:35:48.871 -> TEST: determine size of external eeprom
14:35:48.871 -> 80      FF
14:35:48.871 -> 100     0
14:35:48.871 -> 200     0
14:35:48.871 -> 400     0
14:35:48.871 -> 800     FF
14:35:48.911 -> 1000    AA
14:35:48.911 -> 2000    AA
14:35:48.911 -> external eeprom size: 8192 Bytes
```

Figuur 4. Berichten in de Arduino IDE seriële monitor – modus 6 controleert de externe EEPROM.

gebruiker inzicht in wat het WeMos-board op netwerkgebied te wachten staat en welke modus hij moet kiezen om verbinding te maken met een WLAN. **Figuur 3** toont het resultaat na het opstarten van de WeMos in deze modus, waarbij slechts één WLAN werd gedetecteerd en aansluitend werd verbonden, zoals getoond in figuur 1d.

Modus 3: interne EEPROM wissen

Deze modus zorgt ervoor dat de WeMos zijn interne EEPROM wist als voorbereiding op het samenstellen van een nieuwe lijst met voorkeur-WLAN's. Alle bytes in de EEPROM worden in deze modus gewist naar 0×00.

Modus 4: dump van de interne EEPROM

Deze modus zorgt ervoor dat de WeMos de structuur van de WLAN-referenties, zoals verduidelijkt in tabel 3, uit de interne EEPROM uitleest. Figuren 1e en 1f tonen screenshots van de weergegeven EEPROM entries die momenteel zijn opgeslagen. In deze modus wordt een reeks geformatteerde SSID's en wachtwoorden afgedrukt van de opgeslagen AP-referenties.

Modus 5: initialisatie interne EEPROM

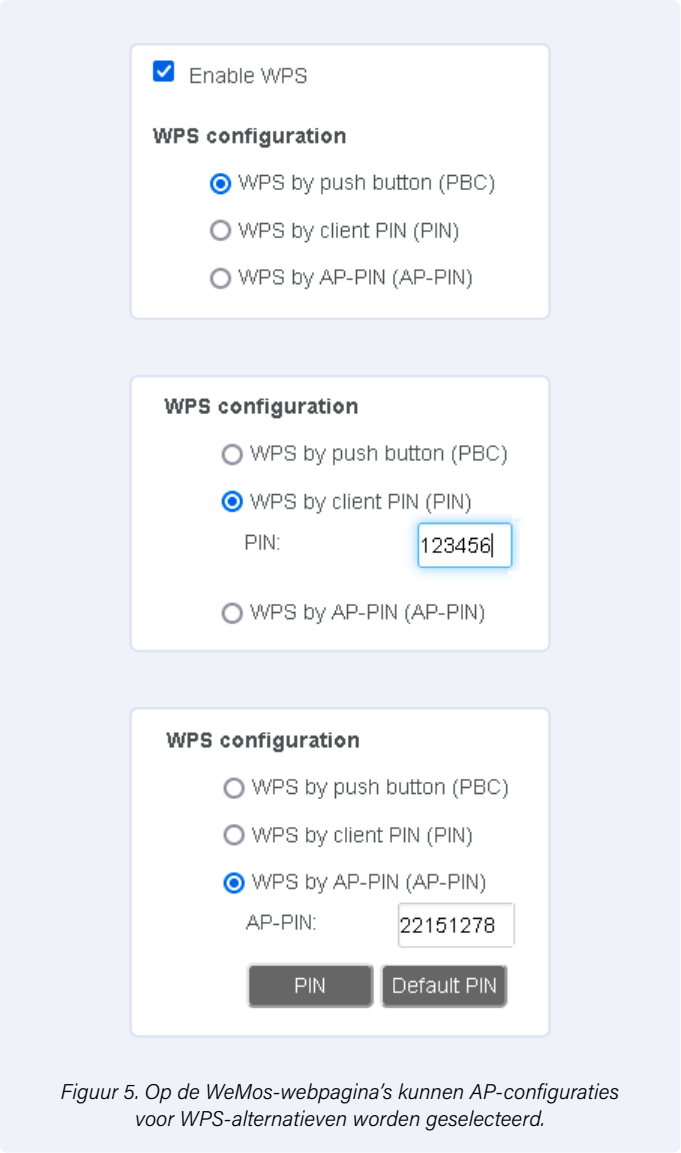
In deze modus kan de gebruiker de referenties van 10 favoriete WLAN's vooraf opslaan in de interne EEPROM. Deze actie is zoiets als de traditionele hardgecodeerde enkelvoudige WLAN-referenties, maar dan tien maal. Na het herstarten in modus 8 zal de WeMos de lijst met referenties voor WLAN-verbindingen controleren. Als het niet lukt om verbinding te maken met een van de vooringestelde WLAN's, dan moet de gebruiker zijn toevlucht nemen tot interactieve of WPS-modi.

Modus 6: controle van de status van de externe EEPROM

In deze modus controleert de WeMos de toestand van de externe EEPROM, in mijn geval een 24LC64 met 8 KB aan boord. **Figuur 4** toont de uitvoer van deze controle in de seriële monitor. Ik heb deze externe chip niet verder bij het project betrokken omdat de interne EEPROM slechts minimaal wordt gebruikt. De lezer kan de I²C-EEPROM-code [3] en -library [4] raadplegen voor integratie in het project.

Modus 7: directe verbinding met WLAN (WPS)

WPS is een functie die wordt ondersteund door moderne AP's en die het proces van het verbinden van draadloze apparaten met een WLAN vereenvoudigt. In plaats van een MCU te voorzien van

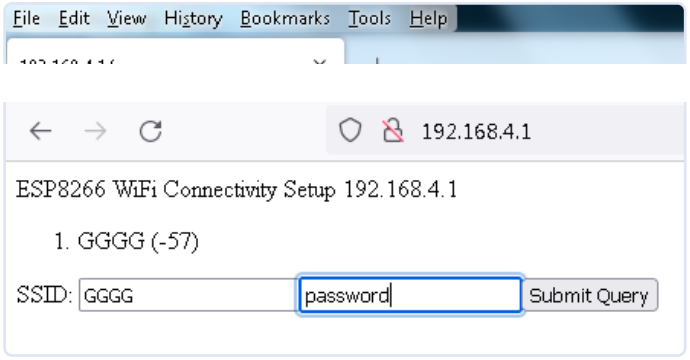


Figuur 5. Op de WeMos-webpagina's kunnen AP-configuraties voor WPS-alternatieven worden geselecteerd.

Figuur 6. WPS-knop en -indicator van een typisch AP.

```
14:37:03.552 -> Disconnecting previously connected WiFi
14:37:03.632 -> Try wps if necessary
14:37:03.632 ->
14:37:03.632 -> Could not connect to WiFi ... state= '7'
14:37:03.672 -> Please press WPS button on your router
14:37:03.672 -> WPS config start
```

Figuur 7. Berichten in de Arduino IDE seriële monitor – modus 7 WPS-setup.



Figuur 8. Modus 8: interactieve WLAN-keuze.

de AP-referenties (zoals SSID en wachtwoord/pre-shared key) kan de MCU de beveiligings-pincode van het AP of zijn eigen vooraf ingestelde pincode verstrekken (zie **figuur 5**). Als alternatief kan een WPS-drukknop worden ingedrukt op het AP voor een directe verbinding met de betreffende MCU (zie **figuur 6**). **Figuur 7** toont het resultaat van het tot stand brengen van een verbinding met behulp van WPS.
In dit project beperk ik de WPS-verbinding tot het AP-drukknopalternatief omdat dit de codering eenvoudiger en algemener maakt om elk AP binnen bereik aan te sluiten, als volgt:

```
bool wpsSuccess = WiFi.beginWPSConfig();
if (wpsSuccess) {
    String newSSID = WiFi.SSID();
    if (newSSID.length() == 0)  { wpsSuccess = false; }
}
```

Het WeMos-board moet zich in de buurt van het gekozen AP bevinden om de AP-parameters in de EEPROM van de module te registreren. Om de module opnieuw te verbinden met hetzelfde AP is het niet langer nodig om op de WPS-knop te drukken, omdat we de module in staat stellen om de opgeslagen AP-parameters op te halen voor hergebruik, op deze manier:

```
WiFi.mode(WIFI_STA);
WiFi.begin(WiFi.SSID().c_str(),WiFi.psk().c_str());
```

Merk op dat de ESP8266-module die het hart vormt van het WeMos-board automatisch de recente succesvolle netwerkgegevens opslaat in de interne EEPROM, dus we moeten onze 'huishoudelijke' structuur, zoals te zien in **tabel 3**, elders in de EEPROM plaatsen om conflicten met de andere mechanismen van de MCU te voorkomen.

Modus 8: interactieve WLAN-instelling

In deze modus controleert de WeMos eerst of de in EEPROM opgeslagen referenties correct zijn. Als het de MCU lukt om verbinding te maken met een AP, eindigt deze modus en neemt de `loop()`-functie het over. Zo niet, dan gaat de MCU naar de interactieve modus, waarin het de WLAN's in zijn omgeving scant en een webserver met IP-adres 192.168.4.1 start om de gedetecteerde AP-SSID's weer te geven. De gebruiker moet de WLAN-adapter van zijn computer instellen op de vaste IP-instellingen voor het privé-netwerk van de WeMos, aangezien de WeMos geen DHCP-service biedt. Een webpagina die lijkt op die in **figuur 8** toont de scanresultaten en stelt de gebruiker in staat om het voorkeursnetwerk te kiezen. De gebruiker klikt dan op de *Submit Query*-knop om de WLAN-referenties op te slaan in EEPROM en een verbinding op te zetten met het geselecteerde WLAN.
EEPROM's kunnen een verschil maken bij het netwerken van draadloze MCU's. In plaats van het hard coderen van WLAN-referenties in de MCU-code, is in dit artikel gebruik gemaakt van de toegang tot een externe of de interne EEPROM van de MCU om dynamisch de referenties van meerdere AP's bij te houden en bij te werken voor acceptabele verbindingen. De broncode bij het artikel is modulair en goed gestructureerd, eenvoudig te begrijpen en biedt herbruikbare code voor de omgang met EEPROM's, WLAN's en voor communicatie met de MCU via het web.

230268-03

Over de auteur

Gamal Labib is al twintig jaar een enthousiast liefhebber van embedded systemen en is momenteel mentor (bij codementor.io). Hij heeft een MEng en een PhD in IT. Naast het schrijven voor technische tijdschriften is hij gasthoogleraar aan Egyptische universiteiten en gecertificeerd IT-consultant.

Vragen of opmerkingen?

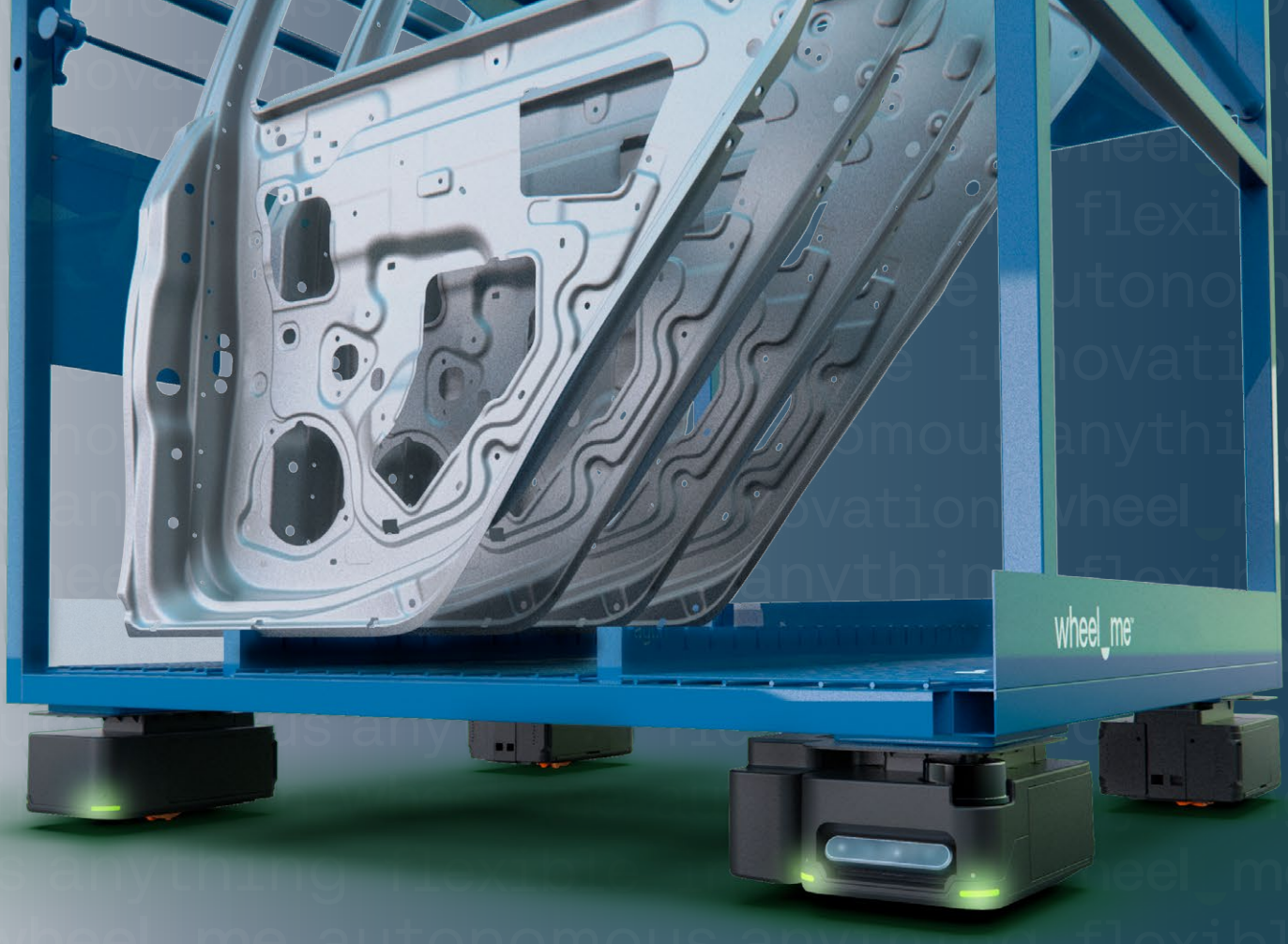
Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via drgamallabib@yahoo.co.uk of naar de redactie van Elektor via redactie@elektor.com.

Gerelateerde producten

- **WeMos D1 mini Pro – ESP8266 based WiFi Module**
  www.elektor.nl/19185
- **0.96" OLED Display (Blue, I2C, 4-Pin)**
  www.elektor.nl/18747
- **Hans Henrik Skovgaard, *Home Appliance Hack-and-IoT Guidebook* (+ FREE ESP8266 Board), Elektor 2022**
  www.elektor.nl/20370

WEBLINKS

[1] Projectpagina bij dit artikel: http://www.elektormagazine.nl/230268-03
[2] Gebruik van de EEPROM met de ESP8266: https://aranacorp.com/en/using-the-eeprom-with-the-esp8266
[3] Arduino met een I2C-EEPROM: https://playground.arduino.cc/Code/I2CEEPROM
[4] Bibliotheek voor I2C-EEPROM: https://github.com/RobTillaart/I2C_EEPROM



Genius by wheel_me

Zorg voor maximale efficiëntie in de intralogistiek met de Genius van wheel.me: Vier aanpasbare robotwielen die autonoom goederenvervoer mogelijk maken tegen een concurrerende prijs.

- Slimme navigatie
- Omnidirectionele beweging
- De nieuwste van de nieuwste sensortechnologie
- Op afstand toegang via de app
- Flexibiliteit om uw laadvermogen aan te passen
- Opladen tijdens het proces

wheel_me
meer details op
www.wheel.me