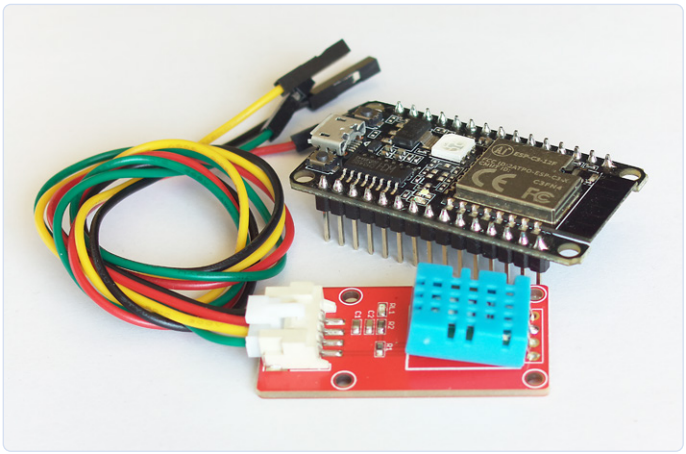
Bluetooth Low Energy met ESP32-C3 en ESP32

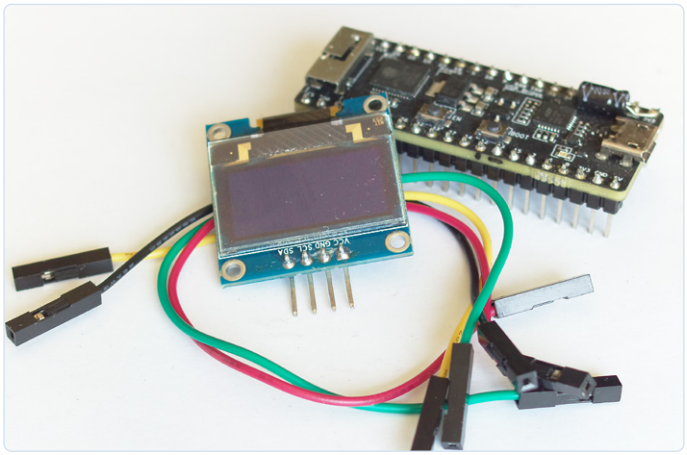
het hoeft niet altijd WiFi te zijn!

Mathias Claußen (Elektor)

In tegenstelling tot de ESP8266 is de ESP32-C3 uitgerust met een Bluetooth Low Energy RF-communicatie. Als je slechts kleine hoeveelheden data over korte afstanden hoeft te versturen, is dit een energiezuinig alternatief voor WiFi. We demonstreren dit hier met een klein project: een temperatuur/vochtigheidssensor met een ESP32-C3 zendt zijn gegevens naar een ESP32 met een klein OLED-display.



Figuur 1. Alles wat je nodig hebt om een sensor-node te maken…

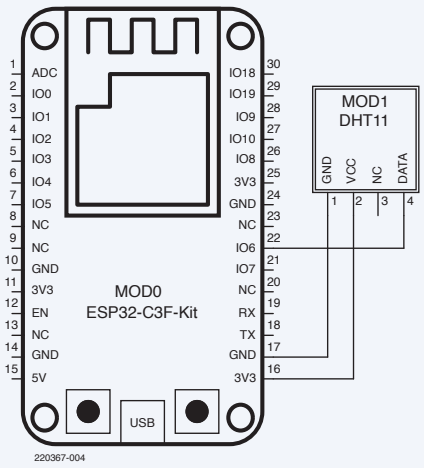


Figuur 2. …en het bijbehorende display.

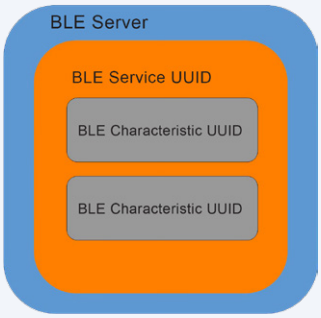
De ESP32-C3 [1] met zijn RISC-V core en zijn bijzonder goede prijs/kwaliteit-verhouding kan beschouwd worden als de opvolger van de ESP8266-MCU van Espressif. Een van de voordelen van deze nieuwe chip is de geïntegreerde Bluetooth Low Energy (BLE) communicatiemogelijkheid. BLE maakt het mogelijk om gegevens uit te wisselen tussen eindtoestellen over een korte afstand en is zeer energiezuinig. Deze communicatiestandaard is ideaal voor een breed scala van toepassingen, waarbij kleine hoeveelheden gegevens over een korte afstand moeten worden verzonden. Hoofdtelefoons, microfoons, headsets en zelfs horloges maken gebruik van BLE om verbinding te maken met verschillende eindapparaten (meestal smartphones).
Maar waarom BLE en niet gewoon WiFi? Als het gaat om het periodiek verzenden van gegevens over korte afstanden, is WiFi niet bepaald zuinig. Daar komt nog bij dat WiFi bedoeld is voor gebruik met een access point binnen een Ethernet-netwerk. Ter vergelijking: een klein batterijgevoed ESP32-C3-apparaat dat via BLE communiceert, kan veel langer op één batterij werken.
Dit project leidt je door de eerste stappen van BLE-communicatie. De opstelling maakt gebruik van een vochtigheids- en temperatuursensor aangesloten op een ESP32-C3, die de gemeten waarden naar een ESP32-module stuurt waar ze op een klein OLED-display worden weergegeven.

De bestanddelen

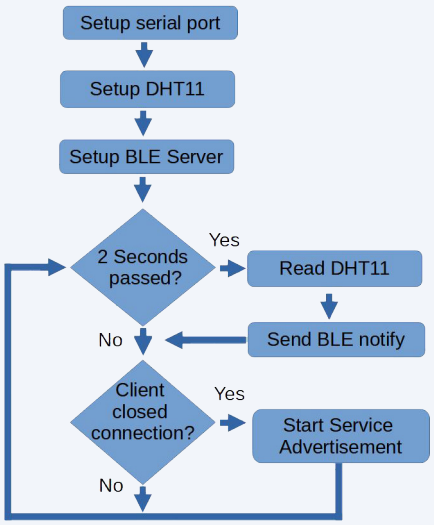
Voor dit project hebben we standaardcomponenten gebruikt die je in de Elektor-shop kunt bestellen. Ze zijn niets bijzonders, misschien



Figuur 3. Het schema van de sensor-node.



Figuur 4. Structuur van de BLE-servers en de UUID's.



Figuur 5. Stroomdiagram voor de sensor-node.

heb je ze zelfs al in de onderdelenla liggen. De sensor-node bevat alleen een DHT11-sensor die zowel de temperatuur als de vochtigheid meet. Deze zit, samen met veel andere nuttige randapparatuur, in de Raspberry Pi Pico Experimenting Bundle. De ESP32-C3 controller wordt geleverd in de vorm van het ESP32-C3-12F Kit Development Board (beide verkrijgbaar in de Elektor-shop – zie **kader**). Alle gebruikte componenten zijn te zien in **figuur 1**.

Voor het display gebruiken we een ESP32-PICO-Kit V4 controller en een klein 0,96" OLED-display (zie kader), plus niet meer dan vier jumperdraden voor de bedrading. Een WeMos LoLin ESP32 met een geïntegreerd OLED-display is ook bruikbaar, maar dan moeten de pintoewijzingen van het display worden gewijzigd. De onderdelen voor het display zijn te zien in **figuur 2**.

BLE-datapakketten

Het communicatieprotocol dat wordt gebruikt om gegevens te verzenden via Bluetooth Low Energy is niet compatibel met het vroegere Bluetooth Classic-protocol. Bij BLE zijn er in principe servers en clients, die beide in staat zijn om datapakketten uit te wisselen met behulp van Attribute Protocols (ATT) en Generic Attribution Profiles (GATT). GATT voorziet in een reeks diensten en kenmerken die procedures en attributen bevatten. Een attribuut kan bijvoorbeeld een sensorwaarde vertegenwoordigen. Elk van de attributen wordt geadresseerd door een UUID, die door de ontwikkelaar kan worden toegekend. De attributen worden op hun beurt verpakt in diensten, een of meer per server, die op hun beurt ook een UUID hebben. Een voorbeeld van een dienst is de levering van een dataset met sensorwaarden (temperatuur, vochtigheid enzovoort).

Bij GATT geschiedt de toegangsautorisatie per verbinding, dat wil zeggen dat er geen onderscheid wordt gemaakt welk eindapparaat de verbinding tot stand brengt, zolang de parameters en sleutels het maar mogelijk maken dat die verbinding tot stand komt.

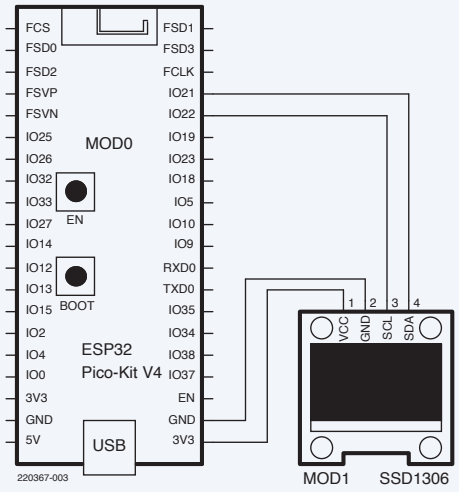
Deze sterk vereenvoudigde voorstelling van GATT zou moeten volstaan voor dit project. Dit is immers alleen bedoeld als een introductie tot het werken met BLE, dus er zijn hier geen beveiligingsmechanismen geïmplementeerd. Meer informatie over BLE is te vinden op de webpagina van Bluetooth SIG [2] of bij het Elektor Webinar over BLE Android Apps [3]. Nu kunnen we aan de slag met het opzetten van onze BLE-server en -client.

BLE-server

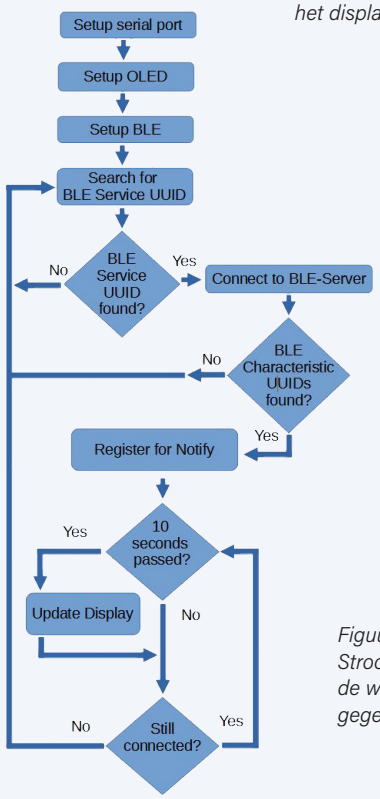
Voor we met de software beginnen, werpen we eerst een blik op de hardware. De aansluiting van de DHT11 op de ESP32-C3 is te zien in **figuur 3**. VCC wordt verbonden met 3,3 V, GND met massa en het DATA-signaal van de sensor met IO06 van de ESP32-C3.

De structuur van de BLE-server is te zien in **figuur 4**. Hij bestaat uit lagen, waarbij de server zelf de buitenste laag vormt. Daarop volgen de services, waarvan er in dit geval slechts één is, met de UUID *91bad492-b950-4226-aa2b-4ede9fa42f59*. In de service zijn de kenmerken opgenomen die zullen worden verstrekt. Een daarvan heeft de UUID *cba1d466-344c-4be3-ab3f-189f80dd7518* voor de temperatuur in graden Celsius (°C); de andere UUID *ca73b3ba-39f6-4ab3-91ae-186dc9577d99* is voor de vochtigheidswaarden. Elk van deze kenmerken heeft een waarde en een beschrijving. Deze beschrijving heeft weer een UUID en is hier ingesteld op *0x2902* – waarbij deze waarde aangeeft dat het om een karakteristiekbeschrijving gaat. Als je hier meer over wilt weten, raadpleeg dan de "Bluetooth low energy characteristics, a beginner's tutorial" van Nordic Semiconductor [4].

In onze software worden eerst alle onderdelen van de BLE-server geconfigureerd. Vervolgens wordt elke twee seconden een nieuwe meetwaarde door de DHT11 ingelezen en als BLE-notificatie (*Notify*) naar aangesloten apparaten gestuurd. Dit is een push-bericht; de client hoeft niet te bevestigen dat het bericht is aangekomen. Het stroomdiagram is te zien in **figuur 5**.



Figuur 6. Schema van het displaygedeelte.



Figuur 7. Stroomdiagram voor de weergave van de gegevens.

Een ding dat opvalt in de code zijn de aanroepen van `delay(5);` na de `notify();`-aanroepen. Bijvoorbeeld:

```
dht11HumidityCharacteristics.setValue(String(event.
   relative_humidity).c_str());
dht11HumidityCharacteristics.notify();
delay(5);
```

Dit is om te voorkomen dat er de BLE-stack overvuld wordt. Helaas kan dit, afhankelijk van de versie van de Arduino-omgeving voor de ESP32 en ESP32-C3, er toch toe leiden dat de BLE-stack onbedoeld de geest geeft.
Zelfs na een `Disconnect` roepen we een vertraging van 500 ms op voor de verwerking van de BLE stack. Overigens heeft dit (helaas) invloed op zowel de ESP32 als de ESP32-C3, aangezien beide van huis uit dezelfde BLE-stack gebruiken.

Een alternatief voor de standaard Bluetooth Classic- en BLE-stack (BlueDroid based Stack) [5] is de Apache MyNewt NimBLE [6] die alleen gebruikt kan worden voor puur BLE (niet Classic Bluetooth) toepassingen in het Arduino framework [7]. We zullen deze stack nu gebruiken voor de client-unit, die in ons geval bestaat uit een ESP32 met een OLED-display.

BLE-client

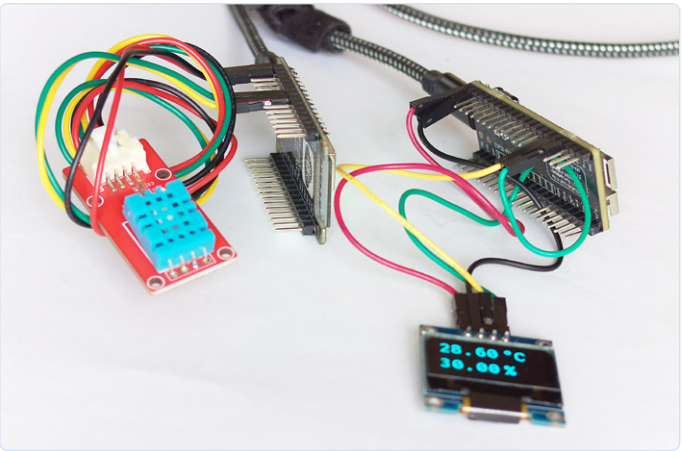
Net als bij de server bekijken we eerst de client-hardware, daarna de software. Hier hebben we een ESP32-PICO-Kit V4 gebruikt, waarop een ESP32-module is gemonteerd om de rekenkracht te leveren. Hoewel hun namen op elkaar lijken, zijn deze twee modules niet nauw verwant. De ESP32 heeft twee Xtensa LX6-processorkernen [8], terwijl de ESP32-C3 één enkele op RISC-V gebaseerde kern gebruikt (RV32IMC). Met niet meer dan vier draden wordt het OLED-display aangesloten op de ESP32 PICO-kit. De VCC van het display wordt verbonden met 3,3 V en de GND van het display met de GND van de ESP32 PICO-kit. Dan komen de I²C-verbindingen SDA en SCL. SDA wordt verbonden met GPIO21 en SCL met GPIO22 van de ESP32-PICO-kit. Het geheel is getekend in **figuur 6**.
Een stroomdiagram van het volledige proces is te zien in **figuur 7**. Na het opstarten van de ESP32 worden de seriële interface en de OLED geïnitialiseerd, gevolgd door de BLE-stack. Dan begint de vijf seconden durende zoektocht naar nieuwe BLE-servers.
Als een server wordt gevonden, wordt als antwoord de functie `onResult` in de klasse `Configured_AdvertisedDeviceCallbacks` aangeroepen. Hier wordt bepaald of de server een service aanbiedt met de UUID *91bad492-b950-4226-aa2b-4ede9fa42f59*. Indien de server een service met de passende UUID heeft, wordt het zoeken naar een nieuwe BLE-server gestaakt en wordt een verbinding met de gevonden BLE-server tot stand gebracht. Daarna wordt de service van de server gecontroleerd op twee UUID's voor de twee kenmerken *cba1d466-344c-4be3-ab3f-189f80dd7518* en *ca73b3ba-39f6-4ab3-91ae-186dc9577d99*. Dit zijn de twee kenmerken van UUID's die door de server worden gebruikt om de temperatuur- en vochtigheidsmetingen te identificeren. Indien de kenmerken beide beschikbaar zijn, wordt de verbinding gehandhaafd. Indien geen geschikte BLE-server of -service wordt gevonden, wordt een nieuwe zoekactie gestart.

Eerst controleert de client of de twee UUID's voor temperatuur en vochtigheid ook meldingen kunnen versturen (*Notifies*). De volgende code (het voorbeeld hier is alleen voor de vochtigheid metingen) wordt gebruikt om dit te bereiken:

```
pRemoteHumCharacteristic =
pRemoteService->getCharacteristic(humUUID);
…
    if (pRemoteHumCharacteristic != nullptr) {
      if(true==pRemoteHumCharacteristic->canNotify()){
        pRemoteHumCharacteristic->
        registerForNotify(NewHumNotify);
      }
…
}
```

Als er meldingen kunnen worden verzonden, wordt daarvoor een callback ingesteld. Telkens wanneer een nieuwe melding binnenkomt,

wordt de functie `NewHumNotify` aangeroepen voor de vochtigheid. Een notificatie kan maximaal 20 bytes aan gebruikersgegevens bevatten, die vrij kunnen worden toegewezen en van elk gekozen formaat mogen zijn. In onze software stuurt de BLE-server de notificatiewaarde als een volledig geformatteerde leesbare string. Om ze op het display weer te geven, hoeven ze alleen nog maar op de juiste manier te worden voorbereid.
Zodra er verbinding is met de BLE-server wordt het OLED-display eenvoudig elke 10 s geactualiseerd om de laatste waarden te tonen zodra deze worden ontvangen. De voltooide BLE-server en BLE-client zijn te zien in **figuur 8**.



Figuur 8. De BLE-server en -client in actie.

BLE en de ESP32/ESP32-C3

De BLE-toepassing die hier als voorbeeld wordt beschreven, is niet bijzonder geavanceerd. De bedoeling is om de principes te demonstreren, zodat je vertrouwd raakt met de basis en voldoende vertrouwen krijgt om deze materie verder te verkennen. BLE biedt een energiebesparende optie voor datatransport over korte afstand zonder de infrastructuur van een WiFi-netwerk. .

220267-03

Vragen of opmerkingen?

Hebt u vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via mathias.claussen@elektor.com of naar de redactie van Elektor via redactie@elektor.com.

GERELATEERDE PRODUCTEN

- **Raspberry Pi Pico Experimenting Bundle**
  www.elektor.nl/19834
- **ESP32-PICO-Kit V4 (SKU 18423)**
  www.elektor.nl/18423
- **WeMos Lolin ESP32 OLED Display Module for Arduino (SKU 18575)**
  www.elektor.nl/18575
- **ESP-C3-12F-Kit Development Board with built-in 4 MB Flash (SKU 19855)**
  www.elektor.nl/19855
- **Seeed Studio Grove DHT11 Temperature and Humidity sensor (SKU 20020)**
  www.elektor.nl/20020
- **0.96" OLED-Display (blue, I²C, 4 pins) (SKU 18747)**
  www.elektor.nl/18747
- ***Develop your own Bluetooth Low Energy Applications* (Book, SKU 20200)**
  www.elektor.nl/20200
- ***Develop your own Bluetooth Low Energy Applications* (E-Book, SKU 20201)**
  www.elektor.nl/20201

WEBLINKS

[1] Mathias Claußen, "Aan de slag met de ESP32-C3 RISC-V MCU": www.elektormagazine.nl/news/aan-de-slag-met-de-esp32-c3-risc-v-mcu
[2] Bluetooth SIG: "Intro to Bluetooth GAP (GATT)": www.bluetooth.com/bluetooth-resources/intro-to-bluetooth-gap-gatt/
[3] C. Valens, "Rapid Prototyping Bluetooth Low Energy Android Apps Using MIT App Inventor," Elektor.TV, June 2021: ww.youtube.com/watch?v=Jxv9h0nHIBA&t=2930s
[4] Nordic Semiconductor, "Bluetooth low energy Characteristics, a beginner's tutorial": https://bit.ly/3sCEVzR
[5] ESP32 Bluetooth Classic- en BLE-stack: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/index.html
[6] ESP32 NimBLE-stack: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/bluetooth/nimble/index.html
[7] NimBLE-Arduino: https://github.com/h2zero/NimBLE-Arduino
[8] ESP32 LX6 Core: https://en.wikipedia.org/wiki/Tensilica