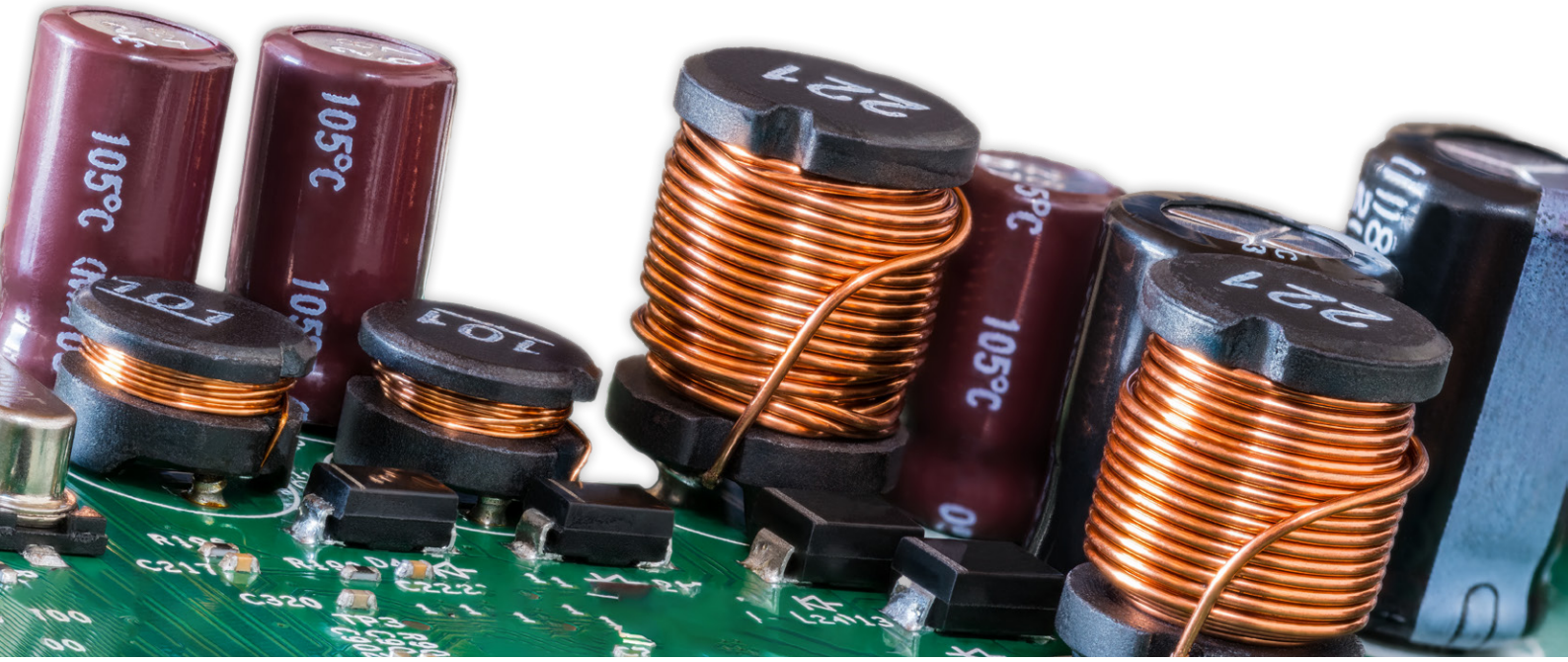
Impedantie-analyzer op basis van een ESP32

simpel, weinig onderdelen en goedkoop!



Volker Ziemann (Zweden)

Een ESP32 als impedantie-analyzer die zelfs de resonantie van een LC-netwerk kan controleren? Ja, dit goedkope controllerboard kan het werk doen, met slechts een handvol externe componenten, en biedt toch een webgebaseerde gebruikersinterface.

We beginnen met wat theorie. De impedantie van een elektronische component $Z = U/I$ is de verhouding tussen de spanning U die over de component valt en de stroom I die er doorheen loopt. De wet van Ohm, geschreven in de vorm $R = U/I$, is een speciaal geval waarbij de weerstand R de eenvoudigste vorm van een impedantie is. Voor een condensator met capaciteit C hangt de impedantie af van de frequentie $\omega = 2\pi f$ en wordt gegeven door $-i/\omega C$; de imaginaire eenheid i is een compacte manier om aan te geven dat de fase van de spanning 90° naijlt op die van de stroom. Evenzo wordt de impedantie van een spoel met inductie L gegeven door $i\omega L$. Ook hier geeft de imaginaire eenheid i aan dat de spanning nu voorijlt op de stroom. We kunnen dus de impedantie van een component bepalen door de frequentie-afhankelijkheid van zijn impedantie en de faserelatie tussen spanning en stroom te onderzoeken.

Deze metingen worden vooral interessant bij meer dan één component. Als een spoel en een condensator parallel worden geschakeld, heeft de kring de hoogste impedantie bij de resonantiefrequentie waarbij de twee wisselstroomweerstanden gelijk zijn. Bij een serieschakeling is de impedantie op dit punt het laagst. Met de Espressif ESP32-impedantie-analyzer kan men nu de frequentie-afhankelijke impedantiekromme

van RLC-netwerken vastleggen en zo hun eigenschappen bepalen. Het enige wat we nodig hebben is een manier om een sinusvormige signaal met constante amplitude te produceren, de frequentie ervan over een liefst zo breed mogelijk bereik te variëren ('sweep'), en dan snel de stroom te meten die door de component(en) loopt – doorgaans aangeduid als *device under test* (DUT). Opmerkelijk genoeg kan de ESP32 microcontroller die ik van Elektor ontving om deel te nemen aan een ontwerpwedstrijd in 2018 de meeste van deze dingen doen. Hij heeft een ingebouwde frequentiegenerator en een DAC die spanningen produceert met frequenties tot een paar honderd kilohertz. Bovendien heeft hij een ADC die spanningen bemonstert met snelheden tot een miljoen keer per seconde, zij het met een beetje valsspelen; maar daar komen we nog op terug. Aangezien de ESP32 slechts één ADC-kanaal op hoge snelheid kan inlezen, en de ADC en DAC onafhankelijk van elkaar werken, kunnen we met de ESP32 alleen de grootte van de impedantie bepalen, maar niet de fase. Meting de door de DUT veroorzaakte faseverschuiving vereist speciale hardware, en zal daarom het onderwerp zijn van een apart artikel.

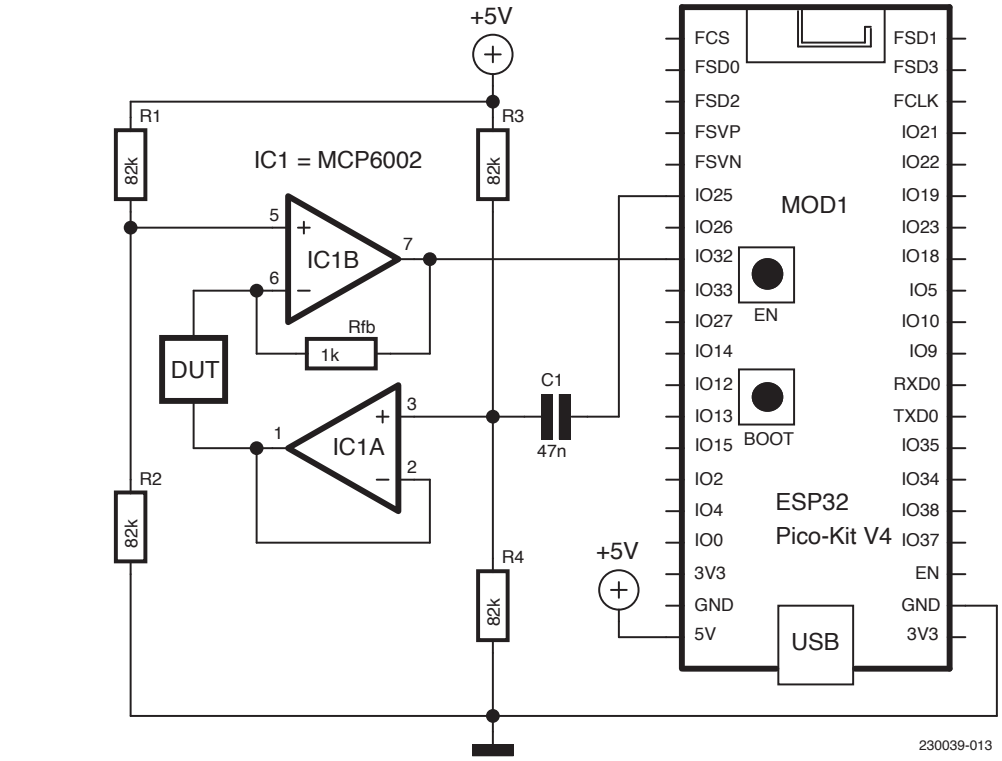
Analoog front-end

We moeten ervoor zorgen dat de amplitude van de DAC-spanning constant is en niet afhangt van de impedantie van de DUT. Dat is de taak van de onderste opamp in **figuur 1**. Deze dient als eenmaal versterkende buffer voor het signaal van pen 25 van de ESP32. Zijn laagohmige uitgang stuurt de DUT aan. De andere aansluiting van de DUT is verbonden met de min-ingang van een tweede opamp, die als transimpedantieversterker is aangesloten. De stroom in diens min-ingang wordt met behulp van de terugkoppelweerstand R_{fb} omgezet in een spanning aan de uitgang, die naar de ADC-pin 32 van de ESP32 wordt gevoerd. Deze ADC kan alleen positieve spanningen verwerken. Daarom wordt het DC-niveau van het signaal met R1 en R2 verschoven naar de helft van de voedingsspanning. Met deze schakeling ziet de DUT een wisselspanning van constante amplitude terwijl we de stroom meten via de ADC.

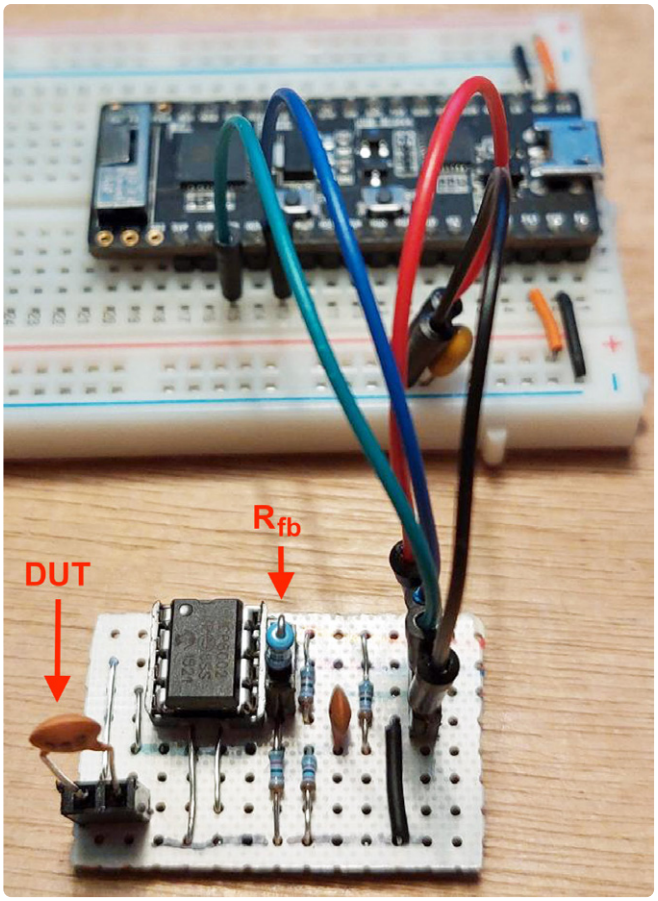
Figuur 2 toont het analoge front-end gemonteerd op een stukje gaatjesprint, met de condensator als DUT linksonder. De rechtopstaande weerstand naast de opamp is de eenvoudig te verwisselen R_{fb}. De vier draden die dit printje verbinden met de ESP32 zijn GND (zwart), 3,3 V (rood), DAC pen 25 (groen), en ADC pen 32 (blauw).

Snelle ADC

De snelle modus van de ADC op de ESP32 maakt deel uit van het I2S-subsysteem dat normaliter wordt gebruikt om digitaal geluid te manipuleren. Bovendien ondersteunt de ESP32 een modus waarbij uitgangswoorden van de ADC in een buffer worden gekopieerd met behulp van directe geheugentoegang (DMA) en de CPU dus niet belast wordt. De ADC is vrijlopend, en DMA leest gegevens met een bepaalde snelheid. Zolang deze snelheid lager is dan de maximale conversiesnelheid van de ADC, die ongeveer 250 kS/s bedraagt, zijn alle monsters 'vers'. Als de snelheid daarentegen te hoog is, worden die samples naar de buffer gekopieerd, wat zichtbaar is als 'trapjes' in de data van de bemonsterde golfvorm.



Figuur 1. Het analoge front-end.



Figuur 2. Om met de ESP32 de impedantie van een DUT te meten, hebben we slechts een paar externe componenten nodig.

De ADC-bemonstering wordt ingesteld door de functie `i2sInit()`, die is afgeleid van de sketch *HiFreq_ADC.ino* die deel uitmaakt van het ESP32-supportpakket voor de Arduino IDE. Het grootste deel van de configuratie bestaat uit het vullen van de structuur `i2s_config` met de gewenste werkingswijze, de bemonsteringsfrequentie, en een paar vlaggen die na flink wat archeologisch onderzoek in [1] zijn gevonden. Voordat we de functie verlaten, stellen we het dempingsniveau zo in dat de volle schaal van de ADC overeenkomt met de voedingsspanning van 3,3 V, kiezen we het ADC-kanaal, en schakelen we de uitgang in.

Frequentiegenerator

De ESP32 heeft een ingebouwd functieblok dat opeenvolgende waarden van een cosinusfunctie 'uitspuugt' met een instelbare snelheid. De uitvoer van dit blok kan, door een aantal configuratiebits goed in te stellen, naar het ingangsregister van de DAC worden geleid, waardoor een cosinusachtige uitgangsspanning ontstaat. Deze frequentiegenerator wordt aangestuurd door rechtstreeks naar de registers `SENS_SAR_DAC_CTRL1_REG` en `SENS_SAR_DAC_CTRL2_REG` te schrijven, hetgeen beschreven is in [1] en meer gedetailleerd in [2]. Volgens de uitleg uit [2] is de functie `cwDACinit()` voorbereid om deze registers te vullen met waarden om de uitgangsfrequentie en -amplitude te configureren. Na het opnemen van header-bestanden die ons in staat stellen mnemotechnische namen te gebruiken, gebruiken we `SET_PERI_REG_MASK(reg,bits)` om `bits` in register `reg` in te stellen. Bijvoorbeeld, het eerste van de volgende commando's

```
SET_PERI_REG_MASK(SENS_SAR_DAC_CTRL1_REG, SENS_SW_TONE_EN);
SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL1_REG, SENS_SW_FSTEP, frequency_step, SENS_SW_FSTEP_S);
```

schakelt de interne snelle frequentiegenerator in. De tweede bepaalt de uitgangsfrequentie door de integer `frequency_step` op te geven, waarvan de waarde wordt afgeleid van een interne klok. De gewenste uitgangsfrequentie volgt een licht aangepaste vergelijking uit [1].

ESP32-sketch, serieel geregeld

Ik programmeerde de ESP32 met de oude versie 1.8.19 van de Arduino IDE, te vinden bij [3], en volgde de instructies om die te installeren. Op het moment van schrijven bood de nieuwe versie 2 van de IDE nog geen ondersteuning voor een later benodigde add-on. Ik moest ook de ESP32-ondersteuningsfuncties van [4] installeren en de installatie-instructies daarvoor volgen.

De volgende sketches zijn gebaseerd op [5], waar een diepgaande bespreking van vele aspecten te vinden is. De bedoeling is om de frequentieopwekking en de acquisitie via de seriële verbinding te besturen op basis van een eenvoudig protocol waarbij alleen strings heen en weer worden gestuurd tussen de ESP32 en een programma op een PC dat seriële communicatie aankan; hieronder vallen Python, Octave en LabView. Het protocol lijkt op het SCPI-protocol dat wordt gebruikt in oscilloscopen en andere meetapparatuur, waarbij een vraagteken een commando aangeeft dat een antwoord van de ESP32 verwacht. Het verzenden van `STATE?` geeft bijvoorbeeld het antwoord `STATE fmin fmax fstep`, waarbij de drie numerieke waarden het bereik en de stapgrootte van de frequentiesweep aangeven.

Nu de verschillende onderdelen van de sketch. Om te beginnen zijn er twee header-bestanden opgenomen – één voor de zelfgebakken ondersteunende functies voor ADC en DAC, de andere om toegang te krijgen tot een flash-gebaseerd SPIFFS-bestandssysteem op de ESP32. Dit laatste dient om kalibratiegegevens op te slaan. Standaardwaarden voor het bereik van de frequentiesweep en andere relevante variabelen zijn ook gespecificeerd. De `setup()`-functie van de sketch initialiseert de seriële communicatie, stelt de uitgangsfrequentie en -amplitude van de DAC in, en leest kalibratiegegevens uit het SPIFFS-bestandssysteem als dat beschikbaar is.

De functie `loop()` controleert met de aanroep van `Serial.available()` of er een request van de hostcomputer is binnengekomen, leest een regel (beëindigd door een `\n` line feed- karakter) en converteert wat is binnengekomen naar de karakter-array `line`. Nu wordt getest of de regel begint met een specifieke string. Als `line` bijvoorbeeld `FREQ 20000` bevat, wordt de numerieke waarde geëxtraheerd met `atoi(&line[5])`. Met de ingebouwde functie `atoi()` wordt de string die op de vijfde positie in de regel begint, omgezet in een geheel getal. Vervolgens wordt die waarde naar de variabele `dac25freq` gekopieerd en wordt de frequentiegenerator met `cwDACinit()` geïnitialiseerd. Op dezelfde wijze zijn alle relevante variabelen toegankelijk vanaf de hostcomputer.

Het meest interessante commando is `SWEEP?` Na ontvangst van dit commando en het declareren van de in deze sectie gebruikte variabelen, laat een for-lus de variabele `f` van `fmin` tot `fmax` lopen met een stapgrootte `fstep`. Binnen de lus wordt eerst de frequentie van de DAC op deze frequentie ingesteld, en na een korte vertraging worden de metingen van de ADC opgehaald met `is2_read()`.

```
for (float f=freqmin; f<freqmax;f+=freqstep) {
  dac25freq=f;
  cwDACinit(dac25freq,dac25scale,0);
```

```
  delay(50);
  i2s_read(I2S_NUM_0, &buffer,     sizeof(buffer), &bytes_
read, 15);
...
}
```

De grootte van het invoerargument `buffer`, gedefinieerd bovenaan de schets, wordt gebruikt om het aantal monsters te bepalen, hier 1024. De functie retourneert ook het aantal `bytes_read`. Aangezien elk monster twee bytes gebruikt, moeten we door twee delen wanneer we de monsters doorlopen om de minimum- en maximumwaarden te bepalen. Aangezien de door de ADC gemeten spanning evenredig is met de stroom door de DUT, gebruiken we deze piek-piek variatie als maat voor de amplitude van de stroom.

Kalibratie

Tijdens het doorlopen van de monsters worden ook de somwaarden `q1` en `q2` opgeteld. Door deze samen met de variabelen `S0`, `S1` en `S2` te gebruiken, krijgen we een rechte lijn op de datapunten, waarvan de parameters nodig zijn voor de kalibratie. De details van het algoritme worden uitgelegd in Appendix B van [5]. Het commando `SAVECALIB` slaat deze parameters op in het permanente SPIFFS-geheugen. Het commando `GETCALIB?` haalt ze op van de PC.
De kalbratieconstanten `calib_slope` en `calib_offset` zijn nodig om verschillende onbekende dempingsfactoren in het systeem in rekening te brengen, bijvoorbeeld ten gevolge van de AC-koppelcondensator aan de ingang van de onderste opamp. Deze dubbelzinnigheid wordt opgelost door het systeem te kalibreren met een weerstand met een bekende waarde als DUT. Alle andere impedanties worden dan bepaald ten opzichte van deze kalbratieweerstand. Idealiter is deze weerstand onafhankelijk van de frequentie, maar met kleine systematische veranderingen wordt rekening gehouden door naast de offset ook de helling als functie van de frequentie te gebruiken. De waarde van de ijkweerstand moet ongeveer dezelfde zijn als die van de terugkoppelweerstand van de transimpedantieversterker, om bij het bereik van de ADC te passen.

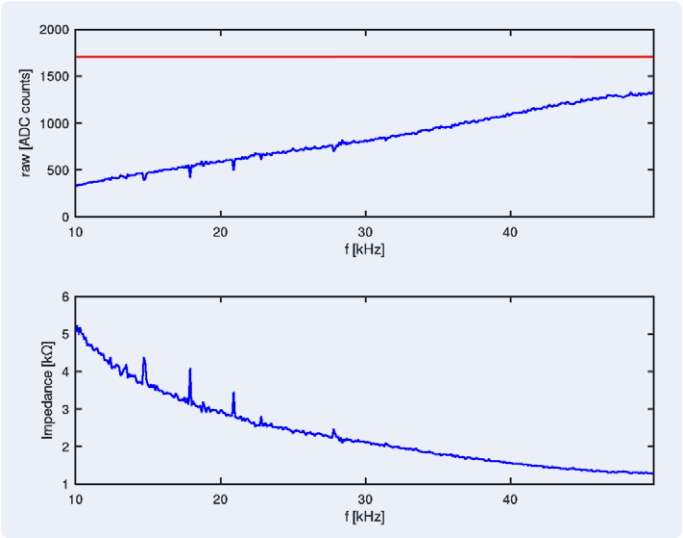
Voordat het systeem wordt gebruikt, moet het SPIFFS-bestandssysteem op de ESP32 worden geïnitialiseerd door een lege map met de naam `data` aan te maken in de directory waar de sketch is opgeslagen. Vervolgens moet de Arduino ESP32 filesystem uploader-plugin van [6] worden geïnstalleerd conform de instructies. Als dat is gebeurd, verschijnt er een nieuw menupunt *ESP32 Sketch Data Upload* in het *Tools*-menu van de Arduino IDE. Nadat je de seriële monitor hebt afgesloten, zal een klik op dat item het SPIFFS-bestandssysteem op de ESP32 aanmaken.

Besturing vanuit Octave of Python

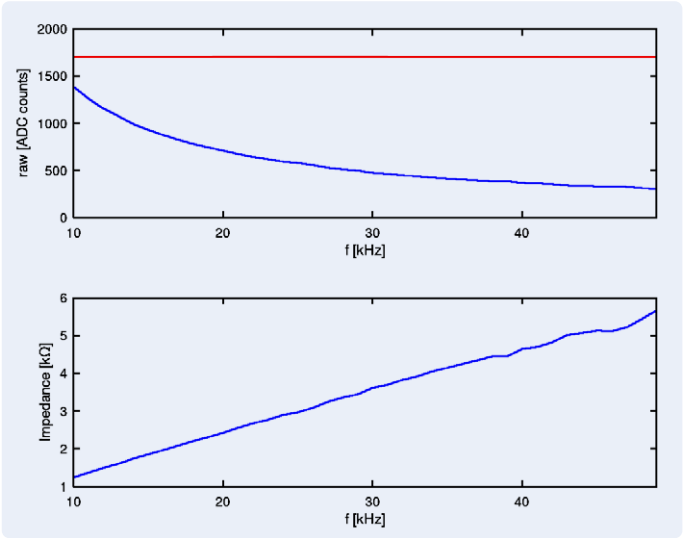
Om de data-acquisitie vanaf de PC te besturen hoeft enkel de seriële poort geopend worden, waarvan de naam verschilt naargelang het besturingssysteem. Deze wordt gewoonlijk `COMn` genoemd onder Windows, `/dev/tty.usbserial-n` op een MAC of `/dev/ttyUSBn` onder Linux. Voor het verzenden van een commando, bijvoorbeeld om de startfrequentie van de sweep in te stellen, hoeft alleen `FMIN 10000` naar de seriële poort te worden geschreven. Om de huidige instellingen te lezen moet `STATE?` worden geschreven, even worden gewacht, en moeten de geretourneerde karakters worden gelezen, tot en met het line feed-karakter `/n`. Het antwoord luidt `STATE fmin fmax fstep`. Het uitvoeren van een frequentiesweep wordt gestart door `SWEEP?` te schrijven en de waarden te ontvangen, steeds één waarde per regel. Zodra alle gegevens beschikbaar zijn, kunnen we de seriële poort sluiten en een plot voorbereiden.
Scripts voor zowel Octave met geïnstalleerde *instrument toolbox* als voor Python met het *python-serial* package zijn opgenomen in het software-archief bij dit artikel. Merk op dat voor de eerste tests zelfs een terminalprogramma zoals Putty kan worden gebruikt.

Gebruik van het systeem

Om het systeem te gebruiken wordt een 1kΩ-kalibratieweerstand als DUT gebruikt en een frequentiesweep uitgevoerd. Door vervolgens het commando `SAVECALIB` uit te voeren, dat standaard is weggecommentarieerd in het begeleidende Octave-script `nwa.m`, worden de kalibratiegegevens op de ESP32 opgeslagen. Als nu het commando `SAVECALIB`



Figuur 3. Ruwe metingen en impedantie bij een condensator als DUT.



Figuur 4. Ruwe metingen en impedantie bij een spoel als DUT.

weer wordt weggecommentarieerd, de weerstand wordt vervangen door een condensator van 2,2 nF, en `nwa.m` weer wordt uitgevoerd, verschijnt de correct gekalibreerde plot van **figuur 3**. Het toont de verwachte omgekeerde evenredigheid met de frequentie, aan de hand waarvan we de capaciteit berekenen met het Octave-commando:

```
capacitance_nF= mean(1./(2*pi*Zabs*1e3.*xx*1e3))*1e9
```

op basis van de formule:

$C = 1/(2\pi f Z_{abs})$

Het gemiddelde over alle beschikbare datapunten wordt berekend met de ingebouwde functie `mean()`. De machten van tien zijn nodig om rekening te houden met de kilo's in kΩ en kHz en de nano's in nF. Bijvoorbeeld, `1e9` aan het eind converteert de capaciteit van farad naar nanofarad.
Herhaling van de sweep met de condensator vervangen door een spoel van 22 mH levert de grafieken van **figuur 4** op. Zoals verwacht neemt de impedantie van een spoel (onderste grafiek) lineair toe met de frequentie, zodat de zelfinductie kan worden bepaald aan de hand van de formule:

$L = Z/2\pi f_{abs}$

of geschreven in Octave met:

```
inductance_mH= mean(1e3*Zabs./(2*pi*xx*1e3))*1e3
```

die ook het gemiddelde neemt over alle datapunten met `mean()`. Ook hier zijn de machten van tien nodig om rekening te houden met de kilo's in kΩ en kHz en de milli's in mH.

Webgebaseerde gebruikersinterface

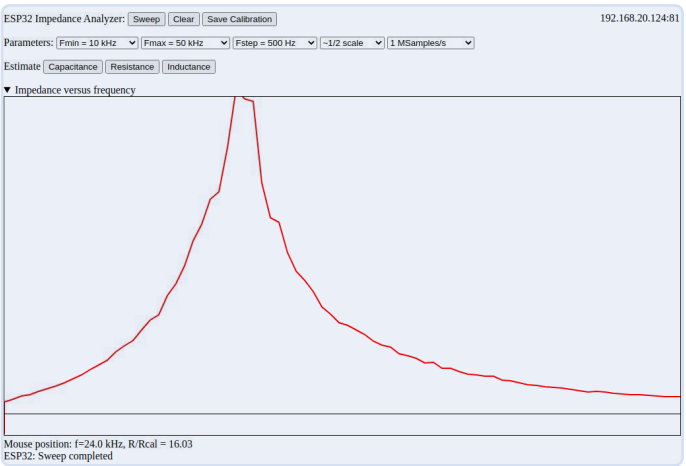
Tot nu toe hebben we de impedantiemeting bestuurd vanuit Octave of Python, maar nu gaan we een webbrowser gebruiken om de metingen te besturen en weer te geven. Daarom wordt de ESP32 geconfigureerd om een webserver te draaien die de webpagina van **figuur 5** produceert. Zodra de browser deze webpagina ontvangt, opent embedded JavaScript-code een tweede communicatiekanaal naar de ESP32, gebaseerd op websockets. Hier vervult een websocket de rol van de seriële verbinding die wordt gebruikt om tekstberichten heen en weer te sturen. Deze berichten worden verstuurd door op de knoppen bovenaan de webpagina te klikken. Daarmee starten we een frequentiesweep, wissen de weergegeven curves en slaan de kalibratiegegevens op. Op de regel daaronder kunnen de parameters die de ESP32 besturen, waaronder het bereik van de sweep en de bemonsteringssnelheid van de ADC, via keuzemenu's worden aangepast. De knoppen op de derde regel berekenen de capaciteit, weerstand en zelfinductie en geven het resultaat weer op de statusregels onder de grafiek. De onderste regel geeft de van de ESP32 ontvangen informatie weer. In de grafiek is de horizontale as de frequentie-as met het bereik dat is opgegeven in de menu's bovenaan de pagina. De verticale as toont Z_{abs} ten opzichte van de kalibratieweerstand, die wordt weergegeven als horizontale blauwe lijn. Een klik op de curve toont de frequentie en Z_{abs} in de statusregel.

Sketch op de ESP32

Na het aanpassen van de WiFi-netwerknaam (SSID) en het wachtwoord moeten de support-bestanden voor WiFi, websocket en webserver worden opgenomen, waarna we de webserver `server2` configureren om te luisteren op netwerkpoort 80, en de `websocket` om te communiceren via poort 81. De tekstberichten die tussen de browsers worden verzonden zijn JSON-geformatteerd, in de vorm `{"INFO":"Yada yada"}`. Het parsen van deze berichten en het construeren van meer complexe berichten wordt afgehandeld door de `ArduinoJson`-bibliotheek, terwijl ondersteuning voor de ADC en DAC wordt geboden door *ESP32_I2Sconfig.h*, zoals eerder.

```
const char* ssid     = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";
#include <WiFi.h>
#include <WebSocketsServer.h>
WebSocketsServer webSocket = WebSocketsServer(81);
#include <WebServer.h>
#include <SPIFFS.h>
WebServer server2(80);
#include <ArduinoJson.h>
#include "ESP32_I2Sconfig.h"
```

Na het specificeren van verschillende variabelen worden helperfuncties gedefinieerd, waarvan `webSocketEvent()` de belangrijkste is. Deze wordt aangeroepen wanneer een bericht van de browser binnenkomt. Als het een JSON-geformatteerd bericht is, haalt het volgende codefragment het commando `cmd` en de waarde `val` eruit met behulp van functies uit de `ArduinoJson`-bibliotheek. Afhankelijk van het commando worden dan de juiste acties geactiveerd. Als bijvoorbeeld het commando `SWEEP` wordt ontvangen, rapporteert hij terug aan de browser met `sendMSG()` en zet hij de variabele `mmode` op één, die in de hoofdlus wordt geïnterpreteerd. Aangezien `webSocketEvent()` asynchroon wordt aangeroepen, onderbreekt hij andere activiteiten en dit moet zo kort mogelijk worden gehouden; daarom wordt de



Figuur 5. De ESP32-geproduceerde webpagina toont een resonantiepiek in de impedantiecurve bij paralelschakeling van een 2,2nF-condensator, een 22mH-spoel en een33kΩ-weerstand.

sweep uitgesteld tot het hoofdprogramma. Aan de andere kant gebruikt het instellen van de startfrequentie van de sweep `freqmin` niet veel CPU-cycli en wordt dit binnen `webSocketEvent()` afgehandeld.

```
DynamicJsonDocument root(300);
deserializeJson(root,payload);
const char *cmd = root["cmd"];
const long val = root["val"];
if (strstr(cmd,"SWEEP")) {
  sendMSG("INFO","ESP32: Received Sweep command");
  mmode=1;
} else if (strstr(cmd,"FMIN")) {
  freqmin=val;
  Serial.printf("FreqMin = %g\n",freqmin);
} else

...
```

Een reeks commando's die lijken op de eerder gebruikte wordt op dezelfde manier behandeld.
Het volgende codefragment uit de `setup()`-functie maakt eerst verbinding met het WLAN met de eerder verstrekte referenties. Het drukt puntjes af via de seriële lijn tot de verbinding tot stand gekomen is, en drukt dan het IP-nummer af, start de websocket-communicatie, en registreert `webSocketEvent` om berichten af te handelen die via de websocket binnenkomen. Merk op dat de seriële lijn niet strikt noodzakelijk is in dit deel, maar het is niettemin geruststellend om te zien wat er gebeurt op de ESP32, vooral tijdens de ontwikkeling van het systeem.

```
WiFi.begin(ssid, password);
while(WiFi.status() != WL_CONNECTED)
Serial.print("\nConnected to ");
Serial.print(ssid);
Serial.print(" with IP address: ");
Serial.println(WiFi.localIP());
webSocket.begin();
webSocket.onEvent(webSocketEvent);
```

Vervolgens controleren we, nog steeds in `setup()`, of het SPIFFS-bestandssysteem bestaat, net als in het eerste deel, en laden we de kalibratiegegevens daaruit. Nu bevat het SPIFFS-bestandssysteem echter ook het bestand *esp32_impedance_analyzer.html*, dat de webpagina beschrijft. Na het starten van `server2` wordt dit HTML-bestand geregistreerd om standaard te worden afgeleverd aan verbonden browsers, zoals aangegeven door het eerste argument in `server2.serveStatic()`:

```
server2.begin();
server2.serveStatic("/",SPIFFS, "/esp32_impedance_
analyzer.html");
```

De rest van de `setup()`-functie lijkt grotendeels op de eerdere sketch. In de functie `loop()` wordt eerst alles met betrekking tot de http- en websocket-servers afgehandeld, voordat de variabele `info_available` wordt gecontroleerd. Deze wordt geset door `sendMSG()` om aan te geven dat de `info_buffer` een JSON-geformatteerd bericht bevat, dat onmiddellijk naar de browser wordt verzonden met een aanroep van `webSocket.sendTXT()`.

```
server2.handleClient();   // handle http server
webSocket.loop();         // handle websocket server
if (info_available==1) {
  info_available=0;
    webSocket.sendTXT(websock_num,
info_buffer,strlen(info_buffer));
}
```

Dan wordt de waarde van `mmode` gecontroleerd en wordt de juiste actie gestart. Zo voert `mmode==1` de frequentiesweep uit met code die sterk lijkt op de eerder gebruikte code. Hier wordt alleen een JSON-geformatteerd bericht met de naam `WF0` (voor golfvorm, waveform) opgebouwd en opgeslagen in de variabele `doc` met:

```
doc["WF0"][nn-1]=floor((512/16)*Z);
```

waarbij de variabele `nn` over de frequentiepunten loopt en `Z` de absolute waarde is van de impedantie bij die frequentie. Merk op dat de variabele wordt geschaald voor 512 pixels die het bereik van nul tot 16 kΩ bestrijken, en dat de waarden worden omgezet in een integer met `floor()`. Nadat de lus is voltooid, wordt de golfvorm naar de browser verzonden met:

```
serializeJson(doc,out);
webSocket.sendTXT(websock_num,out,strlen(out));
sendMSG("INFO","ESP32: Sweep completed");
```

en kondigt aan dat de sweep klaar is met `sendMSG()`. De andere waarden van `mmode` slaan de kalibratie op en rapporteren de geschatte capaciteit, weerstand of zelfinductie aan de browser.
Het bestand *esp32_impedance_analyzer.html* beschrijft de webpagina en bevat de JavaScript-code die hem tot leven brengt.

De webpagina

De meeste interactieve webpagina's gebruiken `<style>`-tags om algemene aspecten te beschrijven van hoe dingen op de pagina verschijnen, gevolgd door HTML-opdrachten die de webpagina beschrijven, en JavaScript-instructies om de pagina interactief te maken.

De volgende stijl-instructie bovenaan *esp32_impedance_analyzer.html* zorgt ervoor dat er een rand rond het displaygebied wordt getekend en dat twee weergegeven curven in rood en zwart worden weergegeven. De laatste instructie zorgt ervoor dat het IP-nummer aan de rechterkant van de webpagina wordt getoond:

```
<style>
#displayarea { border: 1px solid black; }
#trace0 { fill: none; stroke: red; stroke-width: 2px;}
#trace1 { fill: none; stroke: black; stroke-width: 1px;}
#ip
</style>
```

Het belangrijkste deel van de beschrijving van de webpagina staat tussen de tags `<BODY>` en `</BODY>`. Helemaal bovenaan staat de beschrijving van de knoppen. De definitie van de eerste knop staat tussen de `button`-tags en luidt:

```
<button id="sweep" type="button"
onclick="sweep();">Sweep</button>
```

Hiermee wordt `Sweep` op de knop weergegeven, en een klik voert de JavaScript-functie `sweep()` uit. Dergelijke acties gekoppeld aan een gebeurtenis worden gewoonlijk *callback-functies* genoemd. Door aan de knop een ID toe te kennen, in dit geval `Sweep`, kunnen we later de eigenschappen ervan wijzigen, bijvoorbeeld de weergegeven tekst of de te activeren actie. Het definiëren van de andere knoppen gaat op dezelfde manier.

Het keuzemenu op de tweede regel heeeft een iets andere syntaxis. De definitie van dit menu staat tussen `SELECT`-tags. Als een van de items is geselecteerd, roept het `setDacFreqMin(this.value)` aan, waarbij `this.value` de waarde is die is opgegeven in de verschillende `OPTION`-tags. De `OPTGROUP`-tags zijn slechts decoratief.

```
<SELECT onchange="setDacFreqMin(this.value);">
<OPTGROUP label="Sweep start frequency">
<OPTION value="1000">Fmin = 1 kHz</OPTION>
<OPTION value="2000">Fmin = 2 kHz</OPTION>
<OPTION value="5000">Fmin = 5 kHz</OPTION>
<OPTION selected="selected" value="10000">
    Fmin = 10 kHz</OPTION>
<OPTION value="20000">Fmin = 20 kHz</OPTION>
<OPTION value="50000">Fmin = 50 kHz</OPTION>
  </OPTGROUP>
</SELECT>
```

Ten slotte is het gebied om de impedantie weer te geven een *Scalable Vector Graphic* (SVG) dat een gespecificeerde grootte van 1024 × 512 pixels heeft en twee `path`'s weergeeft met ID `trace0` en `trace1`. Met de `id` kunnen we ze later wijzigen. De eigenschap `d` beschrijft de golfvorm. Initialiseren gebeurt door de cursor te verplaatsen naar pixel (0,200) met `M0 200`. Later wordt `d` opnieuw gedefinieerd met de golfvorm `WF0` die binnenkomt van de ESP32.

```
<svg id="displayarea" width="1024px" height="512px">
<path id="trace0" d="M0 200" />
<path id="trace1" d="M0 200" />
</svg>
```

Tenslotte worden de twee statusregels gedefinieerd met:

```
<div id="status">Status window</div>
<div id="reply">Reply from ESP32</div>
```

Ze worden benoemd via `id`, waarmee later de weergegeven tekst kan worden gewijzigd. Evenzo wordt het gebied rechtsboven voor het IP-nummer `ip` genoemd.

JavaScript

Alle JavaScript-opdrachten worden omsloten door `SCRIPT`-tags. Na de openingstag worden verschillende variabelen gedefinieerd, en wordt het IP-nummer van de ESP32 bepaald met:

```
var ipaddr=location.hostname + ":81";
document.getElementById('ip').innerHTML=ipaddr;
```

Hier wordt het poortnummer van de websocket gekopeld aan de ESP32 en onmiddellijk wordt de tekst van de tag, gelabeld `ip`, bijgewerkt. De ruimte om tekst weer te geven wordt geadresseerd door de langere constructie in de tweede regel, waarbij `document` verwijst naar de webpagina zelf. Het deel na de punt geeft toegang tot het benoemde element `ip`, en `innerHTML` verwijst naar de weergegeven tekst die wordt gewijzigd om `ipaddr` te tonen. Deze constructie wordt veel gebruikt om toegang te krijgen tot benoemde elementen en om hun eigenschappen te wijzigen. De functies `toStatus()` en `toReply()` volgen dit schema om tekst te kopiëren naar de statusregels onderaan de webpagina.
Nu het adres van de websocket op de ESP32 bekend is, wordt deze geopend met `new WebSocket()` en vervolgens worden callback-functies gekoppeld aan de events `onopen`, `onclose` en diverse andere. Vaak wordt alleen een kort bericht naar de JavaScript-console gestuurd met `console.log()`. De console is toegankelijk via de *Developer tools* van de browser of met `Ctrl-Shift-I` bij Chromium.

```
var websock = new WebSocket('ws://' + ipaddr);
websock.onopen = function(evt)
    ;
```

De interessantste callback-functie reageert op binnenkomende berichten van de ESP32. De enigszins ingekorte functie `websock.onmessage()` ontleedt eerst het binnenkomende JSON-geformatteerde bericht, dat wordt opgeslagen in `event.data` en plaatst de commando/waarde-paren in de structuur `stuff`. Als `stuff` het commando `INFO` bevat, wordt de bijbehorende waarde opgeslagen in `val` en weergegeven op de webpagina met `toReply()`. Als het commando `WF0` is, bevat het de golfvorm met de impedantiegegevens. Het aantal aankomende datapunten wordt bepaald met `val.length` en gebruikt om de horizontale as zo aan te passen dat alle beschikbare 1024 pixels worden benut. Vervolgens wordt de eigenschap `d` van het pad geïnitialiseerd, en er worden punten aan toegevoegd, één voor elk item in `val`. Aangezien pixel (0,0) linksboven is, moet de verticale positie worden omgekeerd met behulp van `512-val[i]`. Tenslotte wordt de zwarte referentielijn met de kalibratieweerstand weergegeven.

```
websock.onmessage=function(event) {
var stuff=JSON.parse(event.data);
var val=stuff["INFO"]; // info
if (val != undefined)
var val=stuff["WF0"]; // waveform0
if (val != undefined) {
nstep=Math.floor(0.5+1024/val.length)
pixmax=nstep*val.length;
```

```
var d="M0 511";
for (i=0; i<val.length; i++)

document.getElementById('trace0').setAttribute('d',d);
d="M0 480 L1024 480";
document.getElementById('trace1').setAttribute('d',d);
}
}
```

De rest van het JavaScript bestaat voornamelijk uit callback-functies naar knoppen en menu's. De functie `sweep()`, geactiveerd door de corresponderende knop, luidt:

```
function sweep() {
websock.send(JSON.stringify({ "cmd" : "SWEEP", "val" :
-1 }));
}
```

De functie `JSON.stringify()` verpakt zijn argument in een correct geformatteerd bericht en `websocket.send()` stuurt het naar de ESP32. Alle andere callback-functies volgen hetzelfde schema.
Net voor het einde van het JavaScript-gedeelte wordt `showCoordinates()` gedefinieerd om de frequentie en impedantie weer te geven die overeenkomen met de pixel op het grafiekgebied waarop is geklikt. Deze functie krijgt een callback van een `mousedown`-event door een `addEventListener()` te koppelen aan `displayarea`:

```
document.getElementById("displayarea")
.addEventListener('mousedown', showCoordinates, false);
```

Deze functie maakt het op een handige manier mogelijk om direct frequenties en de bijbehorende impedanties te bepalen uit de weergegeven curve.
In figuur 5 is te zien dat het resonantiegedrag van een RLC-parallelkring die wij als DUT hebben gebruikt, een resonantiepiek vertoont bij 24 kHz, waarbij de impedantie groter is dan 16 kΩ.
De firmware en andere bestanden kunnen gratis worden gedownload van [7]. Nu is de impedantie-analyzer zelfstandig in die zin dat er geen besturingsprogramma nodig is. Alle communicatie vindt plaats tussen de ESP32 en een browser, zelfs die op een smartphone.

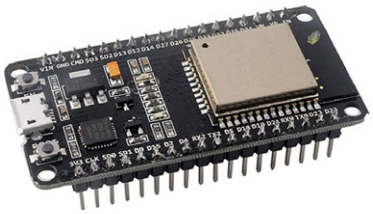
230039-03

Vragen of opmerkingen?

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

Over de auteur

De belangstelling van Volker Ziemanns voor elektronica begon met de 40 W Edwin-versterker (Elektor, midden jaren '70), maar hij sloeg een andere weg in, studeerde natuurkunde en werkte sindsdien met deeltjesversnellers – bij SLAC in de VS, CERN in Genève en nu in Uppsala, Zweden. Aangezien elektronica een belangrijke rol speelt bij de besturing en gegevensverwerving in versnellers, kwam zijn vroege belangstelling vor elektronica hem gedurende zijn hele carrière van pas. Hij doceert nu aan de Universiteit van Uppsala. Een van zijn boeken over data-acquisitie met Arduino en Raspberry Pi raakt aan het onderwerp van dit artikel.



Gerelateerde producten

- **ESP32-DevKitC-32D (SKU 18701)**
  www.elektor.nl/18701
- **OWON HDS1021M-N 1-ch Oscilloscope (20 MHz) + Multimeter (SKU 18778)**
  www.elektor.nl/18778

WEBLINKS

[1] ESP32 Technical Reference Manual (Version 4.7), te vinden op :
https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf
[2] Uitleg ESP32 cosinus-golfvormgenerator: https://github.com/krzychb/dac-cosine
[3] Arduino-website: https://www.arduino.cc
[4] Arduino Support Functions: https://github.com/espressif/arduino-esp32
[5] V. Ziemann, A Hands-on Course in Sensors Using the Arduino and Raspberry Pi, 2nd ed., CRC Press, Boca Raton, 2023:
[6] ESP32fs-plugin: https://github.com/me-no-dev/arduino-esp32fs-plugin
[7] Broncode voor dit project op GitHub: https://tinyurl.com/4awvvemc