



Projecten voor het werkveld 2

Circutor Energy Meter

Graduaat in Internet of Things

Gysen Bart

Academiejaar 2022-2023

Campus : Kempen

VOORWOORD

Voor dit project wou ik graag wat meer de focus leggen op software. Zeker omdat ik hier nog in wil groeien leek me dit een goeie keuzen. Bij het vorige project heb ik vooral gewerkt met een microcontroller. Om de focus op software te leggen hebben we met dit project gewerkt met Embedded Linux. Dit was voor mij nog een onbekend terrein, in het begin heeft het toch wat moeite gekost om te begrijpen wat er allemaal bij komt kijken. Het was een heel leuk project, ik heb hier heel veel bijgeleerd en mijn interesses hierin zijn sterk gegroeid. Daarvoor zou ik heel graag Docent Peter Van Grieken willen bedanken, hij heeft hier ook heel veel tijd ingestoken en heeft er mij enorm hard bij geholpen.

INHOUDSOPGAVE

VOORWOORD	2
INHOUDSOPGAVE.....	3
OVERZICHT FIGUREN	5
INLEIDING	7
1 HET PROJECT	8
1.1 Omschrijving.....	8
2 COMMUNICATIE PROTOCOLLEN	9
2.1 Universal Serial Bus	10
2.2 RS-485	11
2.2.1 Werking	11
2.2.2 Netwerktopologie	12
2.3 Modbus/RTU	12
2.3.1 Modbus RTU vs ASCII	13
2.3.2 Werking	13
2.3.3 Modbus PDU (Protocol Data Unit)	14
2.4 MQTT (Message Queuing Telemetry Transport)	15
2.4.1 Werking	15
2.4.2 QoS (Quality of Service)	16
3 COMPONENTEN	17
3.1 Energie meter Circutor CVM-1D.....	17
3.2 USB to RS485 converter	18
3.2.1 FT232RL	18
3.2.2 MAX485	19
3.3 USB to TTL converter	19
3.4 BeagleBone Black (SBC).....	20
3.5 Home Assistant	21
3.5.1 Node-RED	22
3.5.2 InfluxDB	23
3.5.3 Grafana	24
3.5.4 Verbinden componenten	25
4 CROSS-COMPILING	26
5 VOORBEREIDING	27
5.1 Debian Virtual Machine	27
5.2 Voorbereiding BeagleBone Black.....	27
5.3 GIT	28
6 TESTEN EN PROTOTYPES.....	29
6.1 Hello World! In Cross-Compiling	30
6.2 Test Serial-port	32
6.2.1 Openserial(serialdev).....	32
6.2.2 Write & read.....	33
6.2.3 Compare data.....	34
6.3 Modbus data Circutor CVM-1D.....	35
6.4 Test modbus frame sending	36
6.5 File-splitting & register definition	39
6.6 Paho MQTT C-library	40
6.7 Prototype	44
6.7.1 Data filtering	44
6.7.1.1 ReadAddress	44

6.7.1.2	ConvertToRegisterUnits	45
6.7.2	MQTT Topic & Payload	46
6.7.2.1	Struct.....	46
6.7.2.2	ReadModbus_CircutorData_MqttTopicPayload	46
6.7.3	Main	47
7	FINAL CODE!	51
8	DATA MONITORING	54
8.1	Node-RED koppeling	54
8.1.1	Network node "mqtt in"	55
8.1.2	Function node.....	57
8.1.3	Storage node influxdb out.....	58
8.2	Influxdb	59
8.3	Grafana	60

OVERZICHT FIGUREN

Figuur 1 Blokschema	8
Figuur 2 Voorbeeld energie meter monitoring	8
Figuur 3 USB topologie	10
Figuur 4 Bekabeling voor RS-485	11
Figuur 5 RS-485 Netwerk topologie	12
Figuur 6 Modbus functiecodes	14
Figuur 7 Modbus Frame.....	14
Figuur 8 Circutor CVM-1D.....	17
Figuur 9 Circutor CVM-1D schema	17
Figuur 10 USB to RS485 converter	18
Figuur 11 FT232RL IC	18
Figuur 12 MAX485 IC.....	19
Figuur 13 USB to TTL.....	19
Figuur 14 BeagleBone Black	20
Figuur 15 Node-RED voorbeeld	22
Figuur 16 InfluxDB voorbeeld	23
Figuur 17 Grafana voorbeeld	24
Figuur 18 Verbinding componenten	25
Figuur 19 Native & cross compiling	26
Figuur 20 Hello World! code.....	30
Figuur 21 BBTest Hello World!.....	Error! Bookmark not defined.
Figuur 22 Hello World! Remote debugging step 1	31
Figuur 23 Hello World! Remote debugging step2	31
Figuur 24 openserial(serialdev)	32
Figuur 25 openserial(char *devicename)	33
Figuur 26 Write data.....	34
Figuur 27 Read data	34
Figuur 28 Test case	34
Figuur 29 Output serial test.....	35
Figuur 30 Register addresses energiemeter	35
Figuur 31 Voorbeeld Modbus data	36
Figuur 32 ReadAddress create frame	37
Figuur 33 Send frame & debug	37
Figuur 34 Read frame	38
Figuur 35 Main frametest	38
Figuur 36 Frametest output	39
Figuur 37 Register & registernames energy meter	39
Figuur 38 Main file-splitting & registers definition	40
Figuur 39 paho linker project.....	41
Figuur 40 Output paho test.....	43
Figuur 41 Client paho test	43
Figuur 42 Filtered data.....	44
Figuur 43 ConvertToRegisterUnits	45
Figuur 44 Struct mqtt	46
Figuur 45 Read data & convert.....	47
Figuur 46 Combine topics	47
Figuur 47 Defines energy meter	47
Figuur 48 Main part one prototype	47
Figuur 49 Main part two prototype.....	48
Figuur 50 Main part three paho struct	48
Figuur 51 Main part four sending data.....	49
Figuur 52 Output prototype	49
Figuur 53 MQTT client prototype	50
Figuur 54 Final read & send loop	Error! Bookmark not defined.
Figuur 55 Output final code	53

Figuur 56 Node-RED nodes.....	54
Figuur 57 Network node mqtt-in	55
Figuur 58 mqtt node server settings	55
Figuur 59 mqtt node subscribe to topic.....	56
Figuur 60 Function node.....	57
Figuur 61 function node str to int	57
Figuur 62 Storage node influxdb out.....	58
Figuur 63 Storage node influxdb out db settings.....	58
Figuur 64 Storage node influxdb out settings.....	59
Figuur 65 InfluxDB create DB	59
Figuur 66 Grafana add source	60
Figuur 67 Grafana select data source	60
Figuur 68 Grafana data source config 1	61
Figuur 69 Grafana data source config 2	61
Figuur 70 Grafana add new dashboard	62
Figuur 71 Grafana add new panel	62
Figuur 72 Grafana edit panel	63
Figuur 73 Grafana edit panel stats.....	63
Figuur 74 Grafana stats edit 1	64
Figuur 75 Grafana stats edit 2	64
Figuur 76 Grafana	65

INLEIDING

Projecten voor het werkveld is een vak om je kennis te laten maken met het werkveld. De stappen die worden gemaakt in de praktijk om een project tot een goed einde te brengen worden hier gesimuleerd.

Omdat we in de richting Internet of Things zitten was het de grootste doelstelling om met een Cloud te werken. Met enkel de Cloud hebben we natuurlijk niet genoeg materiaal om een heel project van te maken. In deze richting leren we ook veel over elektronica en vooral over de verschillende soorten communicatie technologieën, dit mocht dus niet ontbreken in het project. Voor het "brein" van het project hebben we gekozen om niet met een microcontroller maar wel met een Single Board Computer te werken. Dit biedt meer mogelijkheden. Om het wat uitdagender te maken is er beslist dat we het programmeren van de software remote en cross-platform gingen doen. Uiteindelijk hebben we dan beslist dat we een energie meter gaan uitlezen via Modbus. De data die van de energie meter komt gaan we dan doorsturen naar de Cloud. Met die data moesten we natuurlijk iets kunnen doen. Het zou mooi zijn om de data te kunnen visualiseren. Hier hebben we gekozen voor dat met Home Assistant te doen. Home Assistant is een open source platform voor centrale aansturingen van slimme apparaten en domotica. De reden dat we voor Home Assistant hebben gekozen is omdat we hier verschillende tools in kunnen integreren die we nodig hebben en die het gemakkelijker maken om alles te kunnen visualiseren.

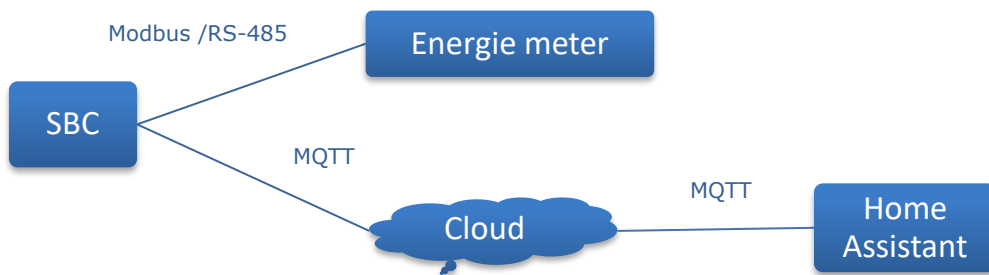
Het project kan gebruikt worden om uw energieverbruik te monitoren, acties te ondernemen bij hoog energieverbruik of om rapporten op te maken over uw verbruik. Het kan u helpen om bewuster om te gaan met energie wat in deze tijd meer en meer belangrijk wordt.

1 HET PROJECT

Het doel van dit project is gericht naar een organisatie of in ons geval een realistische simulatie hiervan. Dit hebben we gedaan door te brainstormen naar een haalbare oplossing, met als vereiste werken met de Cloud.

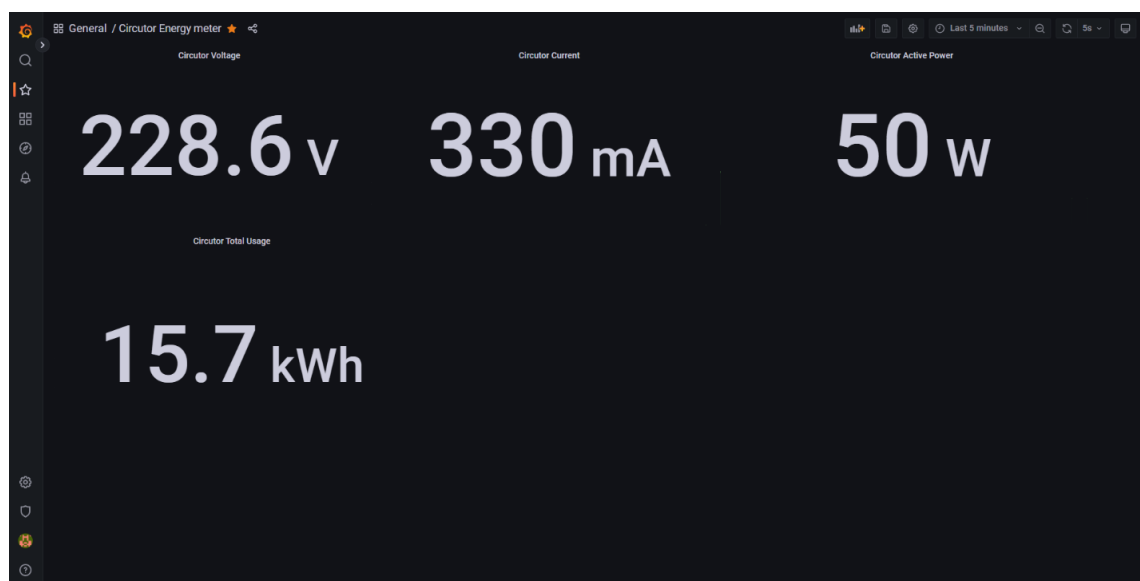
1.1 Omschrijving

Het project dat we gemaakt hebben is een energie meter die we vanop afstand kunnen uitlezen om dan de data te monitoren op een beeldscherm. Door gebruik te maken van een Single Board Computer en Embedded Linux, die ook gekoppeld is aan het internet kunnen we de energie meter uitlezen en de data naar de Cloud sturen. Het uitlezen gebeurt via een seriële Modbus netwerk en de data wordt verstuurd via het MQTT protocol naar de Cloud. Om de energie meter te monitoren maken we gebruik van Home Assistant. Via Home Assistant wordt de data opgevraagd uit de Cloud en in een database gezet. De database is gekoppeld aan een virtualisatie tool. Wanneer data in de database terecht komt wordt dit ook automatisch doorgestuurd naar de virtualisatie tool. Zo hebben we een real-time monitoringssysteem van de energie meter. Het debuggen van de software gebeurt remote over het netwerk. In het project gaan we ook gebruik maken van cross-Compiling.



Figuur 1 Blokschema

In de afbeeldingen hieronder en het blokschema hierboven kan je terug vinden hoe dit project is opgesteld en een voorbeeld van het monitoringssysteem.



Figuur 2 Voorbeeld energie meter monitoring

2 COMMUNICATIE PROTOCOLLEN

Binnen de telecommunicatie is een communicatieprotocol een set van regels en afspraken voor de representatie van data, signalering, authenticatie en foutdetectie, nodig voor het verzenden van informatie over een communicatiemedium. De communicatieprotocollen voor digitale computernetwerken hebben vele eigenschappen bedoeld om er voor te zorgen dat er betrouwbare data-uitwisseling kan plaatsvinden over een onbetrouwbaar communicatiekanaal of medium. Een communicatieprotocol is eigenlijk het volgen van bepaalde regels, zodat een systeem goed kan communiceren en daardoor informatie uit kan wisselen. Deze regels worden in een norm of standaard vastgelegd. Een communicatieprotocol voor netwerkcomponenten wordt een netwerkprotocol genoemd.

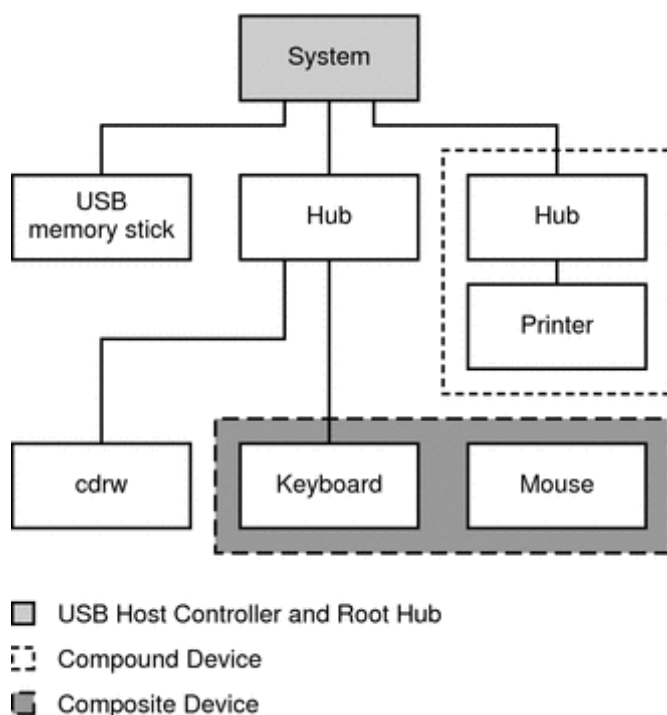
In dit hoofdstuk gaan we de protocollen bespreken die ik heb gebruikt in het project. In totaal heb ik er 4 gebruikt:

- **USB (Universal Serial Bus)**
Gebruiken we voor de aansluiting van de energie meter. De USB wordt aangesloten op de SBC en gebruiken we voor een RS-485 omvormer.
- **RS-485**
Dit is de fysieke aansluiting op de energie meter. Dit hebben we nodig omdat Modbus/RTU over een RS-485 aansluiting werkt.
- **Modbus/RTU**
Dit is het communicatie protocol dat we gebruiken om de energie meter uit te lezen.
- **MQTT**
Dit is het netwerkprotocol dat we gaan gebruiken om data naar de Cloud te sturen.

2.1 Universal Serial Bus

Universal serial bus of kortweg USB is een standaard voor de aansluiting van randapparatuur op computers. Het vervangt de langzamere parallelle en seriële poorten, voornamelijk doordat de snelheid van gegevensoverdracht met USB veel groter is. De standaard is vastgesteld door Intel en protocolversie 1.0 werd in 1996 geïntroduceerd. Met behulp van de gestandaardiseerde USB-poort konden hardware fabrikanten producten maken die compatibel waren met zowel pc's als Macs. Naast randapparatuur voor computers worden verschillende typen microcontrollers voorzien van een USB-poort, bijvoorbeeld om hun toepassing binnen computers en embedded systems te vereenvoudigen. Een bijkomend voordeel van USB is, dat deze de stroomvoorziening van de aangesloten randapparatuur kan verzorgen. Ook kan USB-apparatuur aangesloten worden zonder de computer te hoeven herstarten — dit wordt hotplugging genoemd. Hoewel in de naam het woord bus voorkomt, is USB strikt genomen geen bus, omdat er zonder hub maar één apparaat per poort aangesloten kan worden.

Een USB-systeem bestaat uit een host met een of meer downstream-poorten en meerdere randapparatuur, die een gelaagde stertopologie vormen. Er kunnen extra USB-hubs worden aangesloten waardoor maximaal vijf lagen mogelijk zijn. Een USB-host kan meerdere controllers hebben, elk met een of meer poorten. Er kunnen maximaal 127 apparaten worden aangesloten op één hostcontroller. USB-apparaten zijn in serie verbonden via hubs. De hub die in de hostcontroller is ingebouwd, wordt de root hub genoemd.



Figuur 3 USB topologie

2.2 RS-485

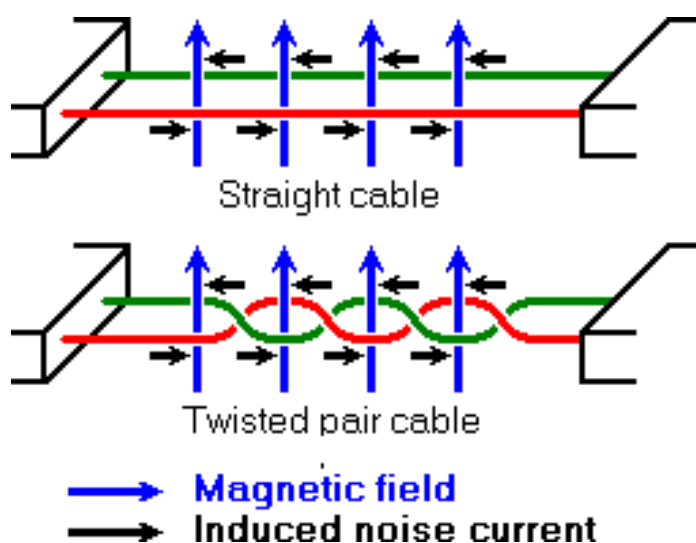
RS-485 is een seriële interface voor telecommunicatie zoals het beter bekende RS-232. RS-485 onderscheidt zich van RS-232 door een grotere kabellengte, minder gevoeligheid voor storingen en het gebruik van een busstructuur. Vanwege deze voordelen wordt deze verbinding veel gebruikt als veldbus in de industrie.

2.2.1 Werking

RS-485 tweedraads is in tegenstelling tot RS-232 half-duplex, dit wil zeggen dat zenden en ontvangen gebeurt over dezelfde signaallijnen maar nooit tegelijkertijd. Er wordt dus telkens omgeschakeld tussen zenden en ontvangen. RS-485 vierdraads is (net als RS-422) wel full-duplex, dit wil zeggen dat zenden en ontvangen gebeurt over aparte signaallijnen en tegelijkertijd kan plaatsvinden.

De vierdraads-verbinding bestaat uit 4 signaallijnen, een TxA, TxB en RxA, RxB. Aan alle vier de uiteinden bevindt zich een afsluitweerstand. Naast de signaallijnen is er een grondlijn.

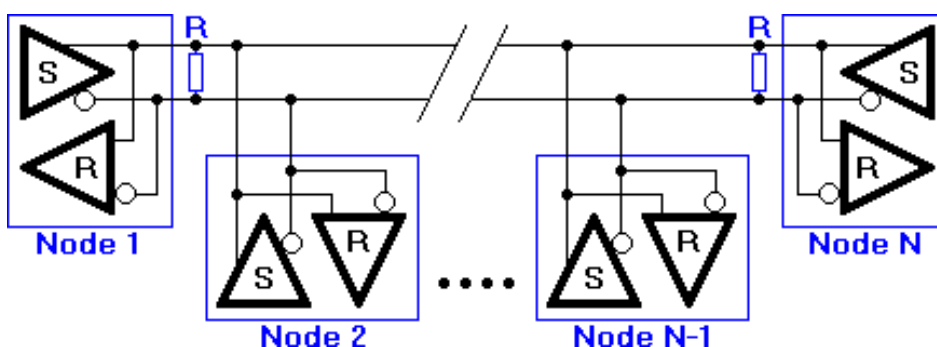
De tweedraads-connectie bestaat uit 2 signaallijnen, een A(-) en een B(+) lijn. In sommige gevallen kan dit ook omgekeerd zijn, waardoor de kabels van A naar B aangesloten moeten worden. A is het geïnverteerde signaal van B, waar B inactief hoog is en A inactief laag. De RS-485 ontvanger vergelijkt het spanningsverschil tussen beide lijnen, in plaats van de absolute spanning op één signaallijn. Dit werkt zeer goed en voorkomt het bestaan van aardlussen, een belangrijke bron van communicatieproblemen. Het beste resultaat wordt behaald als de A(-) en B(-) lijnen getwist zijn.



Figuur 4 Bekabeling voor RS-485

2.2.2 Netwerktopologie

RS-485 is als enige geschikt voor netwerken van meerdere zenders en ontvangers in hetzelfde netwerk. Wanneer de standaard RS-485 ontvangers gebruikt worden met een ingangsweerstand van 12 k Ω , dan is het mogelijk om 32 apparaten aan te sluiten op het netwerk. Op dit moment beschikbare RS-485 inputs met hoge ingangsweerstand maken het mogelijk dat dit aantal uitgebreid wordt tot 256. RS-485 repeaters zijn ook beschikbaar waarmee het mogelijk is het aantal gekoppelde systemen uit te breiden tot duizenden, over een afstand van vele kilometers. En dat alles met een interface die geen intelligente netwerk hardware nodig heeft, de implementatie aan de software zijde is niet veel moeilijker dan met RS-232. Dit is de reden waarom RS-485 zo populair is bij computers, PLC's, microcontrollers en intelligente sensoren in wetenschappelijke en technische applicaties.



Figuur 5 RS-485 Netwerk topologie

2.3 Modbus/RTU

Modbus is een datacommunicatieprotocol dat oorspronkelijk in 1979 door Modicon (nu Schneider Electric) werd gepubliceerd voor gebruik met zijn programmeerbare logische controllers (PLC's). Modbus is een standaard communicatieprotocol geworden en is nu een algemeen beschikbaar middel om industriële elektronische apparaten aan te sluiten.

Modbus is populair in industriële omgevingen omdat het openlijk wordt gepubliceerd en royaltyvrij is. Het is ontwikkeld voor industriële toepassingen, is relatief eenvoudig te implementeren en te onderhouden in vergelijking met andere normen en legt weinig beperkingen op aan het formaat van de te verzenden gegevens.

Het Modbus-protocol maakt gebruik van seriële communicatielijnen, Ethernet of de internetprotocolsuite als transport laag. Modbus ondersteunt communicatie van en naar meerdere apparaten die op dezelfde kabel of hetzelfde Ethernet-netwerk zijn aangesloten. Er kan bijvoorbeeld een apparaat zijn dat de temperatuur meet en een ander apparaat om vochtigheid te meten dat op dezelfde kabel is aangesloten, beide metingen communiceren met dezelfde computer, via Modbus.

2.3.1 Modbus RTU vs ASCII

Er zijn twee verschillende seriële transmissiemodi gedefinieerd: de RTU-modus en de ASCII-modus. Het definieert de bit inhoud van berichtvelden die serieel op de lijn worden verzonden. Het bepaalt hoe informatie in de berichtvelden wordt verpakt en gedecodeerd. De transmissiemodus (en seriële poortparameters) moeten hetzelfde zijn voor alle apparaten op een Modbus Seriële lijn. Hoewel de ASCII-modus in sommige specifieke toepassingen vereist is, kan interoperabiliteit tussen Modbus-apparaten alleen worden bereikt als elk apparaat dezelfde transmissiemodus heeft: alle apparaten moeten de RTU-modus implementeren. De ASCII-transmissiemodus is een optie. Apparaten moeten door de gebruikers worden ingesteld op de gewenste transmissiemodus, RTU of ASCII. De standaardinstelling moet de RTU-modus zijn.

2.3.2 Werking

Elk apparaat dat communiceert op een Modbus krijgt een uniek device adres. In Modbus RTU, Modbus ASCII en Modbus Plus (die allemaal RS-485 single-cable multi-drop netwerken zijn), mag alleen een master device commando's sturen. Alle andere apparaten zijn "slaves" die reageren op verzoeken en commando's.

Modbus-commando's kunnen een Modbus-apparaat instrueren:

- om waarden in zijn registers aan te passen die zijn opgenomen in de coil- en holdingsregisters.
- een IO-poort lezen.
- het apparaat opdracht geven om een of meer waarden terug te sturen die zijn opgenomen in de coil- en holdingregisters

Een Modbus-commando bevat het Modbus-adres van het apparaat waarvoor het is bedoeld (1 tot 247). Alleen het geadresseerde apparaat reageert en handelt op de opdracht, ook al kunnen andere apparaten deze ontvangen (een uitzondering zijn specifieke uitzendbare opdrachten die naar knooppunt 0 worden verzonden, waarop wordt gereageerd maar niet wordt erkend). Alle Modbus-opdrachten bevatten controle informatie zodat de ontvanger transmissiefouten kan detecteren.

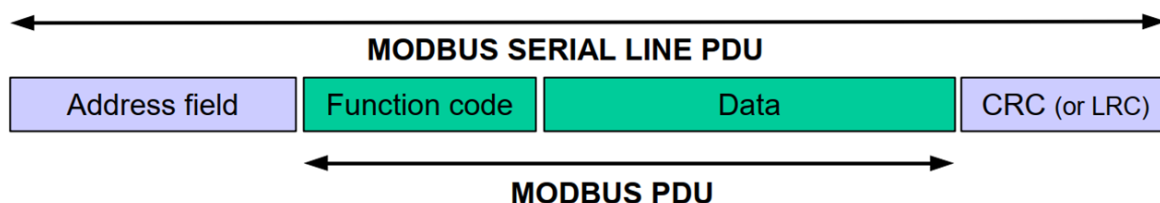
2.3.3 Modbus PDU (Protocol Data Unit)

De PDU en de code die deze verwerkt, vormen de kern van de Modbus Application Protocol Specification. Deze specificatie definieert het formaat van de PDU, de verschillende gegevensconcepten die door het protocol worden gebruikt, het gebruik van functiecodes om toegang te krijgen tot die gegevens en de specifieke implementatie en beperkingen van elke functiecode. Hieronder een afbeelding van de functiecodes.

				Function Codes		
				code	Sub code	(hex)
Data Access	Bit access	Physical Discrete Inputs	Read Discrete Inputs	02		02
		Internal Bits Or Physical coils	Read Coils	01		01
			Write Single Coil	05		05
			Write Multiple Coils	15		0F
	16 bits access	Physical Input Registers	Read Input Register	04		04
		Internal Registers Or Physical Output Registers	Read Holding Registers	03		03
			Write Single Register	06		06
			Write Multiple Registers	16		10
			Read/Write Multiple Registers	23		17
			Mask Write Register	22		16
			Read FIFO queue	24		18
	File record access	Read File record		20		14
		Write File record		21		15
	Diagnostics		Read Exception status	07		07
			Diagnostic	08	00-18,20	08
			Get Com event counter	11		0B
			Get Com Event Log	12		0C
			Report Server ID	17		11
			Read device Identification	43	14	2B
Other		Encapsulated Interface Transport		43	13,14	2B
		CANopen General Reference		43	13	2B

Figuur 6 Modbus functiecodes

Het Modbus PDU-formaat wordt gedefinieerd als een functiecode gevolgd door een bijbehorende set gegevens. De grootte en inhoud van deze gegevens worden bepaald door de functiecode en de volledige PDU (functiecode en gegevens) mag niet groter zijn dan 253 bytes. Elke functiecode heeft een specifiek gedrag dat slaves flexibel kunnen implementeren op basis van hun gewenste applicatiegedrag. De PDU-specificatie definieert kernbegrippen voor gegevenstoegang en manipulatie; een slave kan echter gegevens verwerken op een manier die niet expliciet in de specificatie is gedefinieerd.



Figuur 7 Modbus Frame

- **Address field:**
Elke slave in een netwerk heeft een eigen adresnummer variërend van 1 tot 247.
- **Function code:**
De functiecode in het verzoek vertelt het geadresseerde slave-apparaat wat voor soort actie moet worden uitgevoerd.
- **Data:**
Data in het verzoek toont welk coil- en holdingsregisters we willen bereiken. Bij antwoord van de slave zit hier de opgevraagde data in.
- **CRC:**
CRC staat voor Cyclic Reduncany check. CRC zijn twee bytes die worden toegevoegd aan het einde van elk Modbus bericht voor foutcontrole. Elke byte in het bericht wordt verzonden om de CRC te berekenen. Het ontvangende apparaat berekent ook de CRC en vergelijkt deze met de CRC van het verzendende apparaat. Als zelfs maar één bit in het bericht onjuist wordt ontvangen, zullen CRC's anders zijn en zal er een fout optreden.

2.4 MQTT (Message Queuing Telemetry Transport)

MQTT protocol (Message Queuing Telemetry Transport) is een gemeenschappelijke taal waardoor sensoren, actuatoren en machines met elkaar kunnen communiceren. Het is een lichtgewicht publish en subscribe systeem. Het protocol is ontwikkeld als eenvoudig systeem om met een lage bandbreedte data over te dragen. Dankzij deze eigenschappen is MQTT erg geschikt om te gebruiken binnen Internet of Things applicaties. MQTT wordt gebruikt in verschillende industrieën, zoals de auto-industrie, productie, telecommunicatie, olie en gas. MQTT maakt gebruik van de volgende basis begrippen: Publish/Subscribe, Messages, Topics en Broker.

De broker is de centrale spil voor alle verzonden berichten. De betrokken apparaten communiceren alleen met de broker en kennen elkaar onderling verder niet. Ze hoeven de ip-adressen en de technische details van de andere deelnemers dus niet te weten. Het is de taak van de broker om berichten te accepteren en aan de juiste ontvanger door te geven.

2.4.1 Werking

Als een sensor, bijvoorbeeld een microcontroller met een temperatuursensor, wil communiceren via MQTT, moet hij eerst verbinding maken met een broker. Bij het MQTT-protocol is poort 1883 gereserveerd voor onversleutelde en poort 8883 voor versleutelde communicatie. MQTT is, anders dan bijvoorbeeld HTTP, een statusbehoudend protocol. Een verbinding kan dus ook blijven bestaan als er geen gegevens worden verzonden.

Clients kunnen berichten versturen naar de broker, de term hiervoor is **PUBLISH**. Voor het versturen hebben we twee dingen nodig, een **TOPIC** en de **PAYLOAD**.

- **TOPIC**
Een topic lijkt qua opbouw op een Unix-bestandspad, het is een soort mappen structuur. De secties worden gescheiden door een /. Bijvoorbeeld:
ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualVoltage
- **PAYLOAD**
De payload is de inhoud van het bericht.

Een client dat berichten wil ontvangen, maakt verbinding met de broker en abonneert zich op een of meerdere TOPICS. De term hiervoor is **SUBSCRIBE**.

Voor het opvragen van meer waarden zijn er twee jokertekens (# en +), die dus niet in de naam van een topic voor kunnen komen.

- **#**
Met dit jokerteken worden alle berichten op de lagere niveaus aangevraagd.
`ProjectenVoorHetWerkveld/CircutorEnergyMeter/#`
Hier abonneren we op elk topic dat valt onder
`"ProjectenVoorHetWerkveld/CircutorEnergyMeter"`
- **+**
is het teken voor een niveau:
`house/rooms+/sensors/temperature`
is een abonnement op alle temperaturen. Een MQTT-compatibele radiatorthermostaat kan zo bijvoorbeeld alle sensorwaarden van een huis opvragen en daar op reageren.

2.4.2 QoS (Quality of Service)

Quality of Service (QoS) in MQTT is een overeenkomst tussen verzender en ontvanger over de garantie van het afleveren van een bericht.

Er zijn drie niveaus van QoS:

1. maximaal één keer
2. minstens één keer
3. precies één keer

Wanneer een client publiceert naar een broker, bepaalt de client het QoS-niveau voor dat bericht. Wanneer de broker het bericht naar een geabonneerde client stuurt, wordt de QoS die door de eerste client voor dat bericht is ingesteld, opnieuw gebruikt. Dus in feite bepaalt de oorspronkelijke uitgever van een bericht de QoS van het bericht helemaal tot aan de uiteindelijke ontvangers.

- **QoS Level 0 - maximaal één keer**
Dit is de eenvoudigste, laagste overhead methode om een bericht te verzenden. De klant publiceert gewoon het bericht en er is geen bevestiging door de broker.
- **QoS Niveau 1 - minstens één keer**
Deze methode garandeert dat het bericht met succes wordt overgedragen aan de broker. De broker stuurt een bevestiging terug naar de afzender, maar in het geval dat de bevestiging verloren gaat, zal de afzender zich niet realiseren dat het bericht is doorgekomen, dus zal het bericht opnieuw worden verzonden. De client zal opnieuw verzenden totdat hij de bevestiging van de broker krijgt. Dit betekent dat verzending gegarandeerd is, hoewel het bericht de broker meer dan eens kan bereiken.
- **QoS Level 2 - precies één keer**
Dit is het hoogste serviceniveau, waarbij er een opeenvolging van vier berichten is tussen de afzender en de ontvanger, een soort handshake om te bevestigen dat het hoofdbericht is verzonden en dat de bevestiging is ontvangen. Wanneer de handshake is voltooid, weten zowel de afzender als de ontvanger zeker dat het bericht precies één keer is verzonden.

3 COMPONENTEN

In de omschrijving van het project heb ik al reeds uitgelegd uit welke componenten dit project bestaat. In dit hoofdstuk gaan we wat dieper in de details van de componenten en de mogelijkheden die ze bieden. Voor het testen van dit project zijn er ook nog een aantal componenten gebruikt die hier aan bod komen. In dit project hebben we de volgende componenten gebruikt:

- Circutor CVM-1D (Energie meter)
- USB to RS485 converter
- USB to TTL
- Beaglebone Black (SBC)
- Home Assistant

3.1 Energie meter Circutor CVM-1D

Een energie meter is een apparaat waarmee het energieverbruik kan geregistreerd worden. Er bestaan verschillende soorten energie meters, zo zijn er gas meters, water meters en elektriciteit 's meters. In dit project hebben we gebruik gemaakt van een elektriciteit 's meter. Ook hier zijn er verschillende soorten in te vinden. Je hebt de meters van de netbeheerders, die bestaan uit analoge en digitale meters. Er bestaan ook plug-in meters, dat is een stekker waar je een apparaat kan inpluggen en zo het verbruik ervan kan meten. Dan zijn er ook nog de modulaire energie meters. Die kan je in je elektriciteitskast inbouwen om heel de installatie te monitoren of een enkele kring. Deze bestaan in mono-fase of in 3-fase, afhankelijk van wat je installatie is of waar je het voor wilt gebruiken.

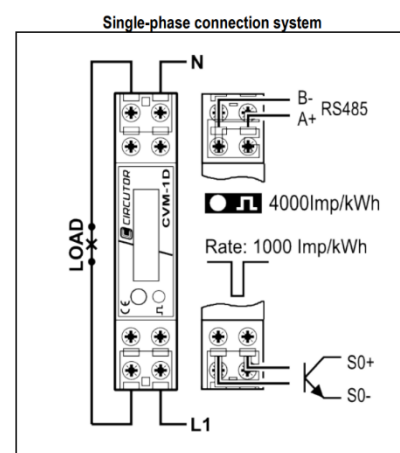
De meter die we in dit project hebben gebruikt is de Circutor CVM-1D, dat is een mono-fase modulaire energie meter. Deze meter kan gemonitord worden via Modbus/RTU. Het voordeel hiervan is dat je in combinatie met een domotica of smarthome systeem toestellen kan laten uitschakelen wanneer het verbruik te hoog is. Of in combinatie met zonnepanelen toestellen kan laten aanschakelen om de opgewekte energie te verbruiken. Hiernaast een afbeelding en een aansluitschema.



Figuur 8 Circutor CVM-1D

De belangrijkste kenmerken van de energiemeter zijn:

- Monofasige circuits tot 32A.
- 24 actuele, maximale en minimale elektrische variabelen.
- Modbus/RTU (RS-485).
- Zescijferig LCD-display.
- Programmeerbaar alarm of impulsuitgangen



Figuur 9 Circutor CVM-1D schema

3.2 USB to RS-485 converter

Een USB to RS-485 converter is een component dat een computer of een ander apparaat met een USB-poort in staat stelt te communiceren met apparaten die het RS-485 seriële communicatieprotocol gebruiken. De converter bevat aan één einde een USB-connector en aan de andere einde een RS-485-connector. Dit maakt het mogelijk voor apparaten die RS-485 gebruiken om te communiceren met apparaten die USB gebruiken, zoals computers of andere controllers. Dit kan ook langere kabelafstanden mogelijk maken, omdat RS-485 ondersteuning biedt voor langere kabelafstanden dan USB. De converter die we hebben gebruikt is van GmM, hieronder een afbeelding ervan.



Figuur 10 USB to RS485 converter

De converter bestaat uit twee belangrijke IC's, de **FT232RL** en de **MAX485**.

3.2.1 FT232RL

De FT232RL is een kleine IC die een brug vormt tussen een USB-verbinding en een seriële UART (universele asynchrone ontvanger / zender) interface. Hierdoor is er communicatie mogelijk tussen een USB-ondersteund apparaat, zoals een computer of smartphone, en een apparaat dat een seriële interface gebruikt, zoals een microcontroller, GPS-module of ander elektronisch apparaat.

De IC bevat een USB-controller, een klokgenerator en een USB-ontvanger. De USB-controller regelt de communicatie tussen het USB-ondersteunde apparaat en de IC, terwijl de klokgenerator het kloksignaal genereert dat nodig is voor de seriële communicatie. De USB-ontvanger converteert de gegevens tussen het USB- en seriële formaat. De IC kan worden verbonden met het doelapparaat met een eenvoudige UART-interface en worden aangedreven door de USB. Hieronder een afbeelding van de IC.

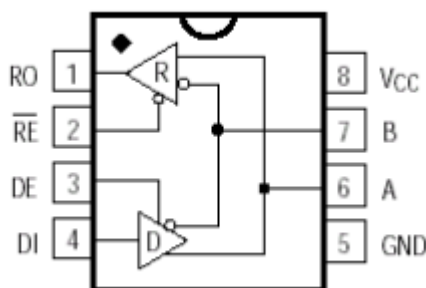


Figuur 11 FT232RL IC

3.2.2 MAX485

De MAX485 is een RS-485 transceiver IC gemaakt door Maxim Integrated. Het wordt gebruikt om signalen om te zetten van TTL-niveau (transistor-transistor logica) naar RS-485-niveau en omgekeerd.

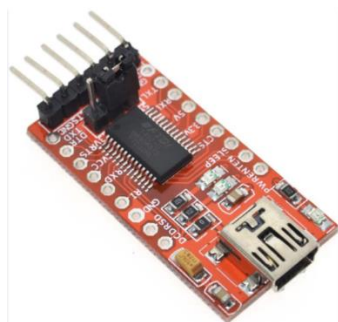
De MAX485 IC heeft twee ingangen en twee uitgangen die corresponderen met de twee lijnen in een RS-485-netwerk, genaamd "A" en "B" (vaak aangeduid als "Data +" en "Data -"). De ingangen van de IC worden aangesloten op de uitgangen van een TTL-compatibele microcontroller of andere uitgangsschakeling. De IC zorgt vervolgens voor de conversie van deze TTL-niveausignalen naar RS-485-niveausignalen die op de A en B-uitgangen van de IC beschikbaar zijn voor transmissie over een RS-485-netwerk. Wanneer een RS-485-sigitaal ontvangen wordt door de IC, wordt het ook geconverteerd van RS-485-niveau naar TTL-niveau en beschikbaar gesteld aan de ingangen van de microcontroller of andere ingangsschakeling. De IC is ook uitgerust met een driver-enable-ingang die de transmissie en ontvangst van de IC kan activeren of deactiveren. Dit maakt het mogelijk om meerdere transceivers op hetzelfde RS-485-netwerk te gebruiken zonder dat ze elkaar storen.



Figuur 12 MAX485 IC

3.3 USB to TTL converter

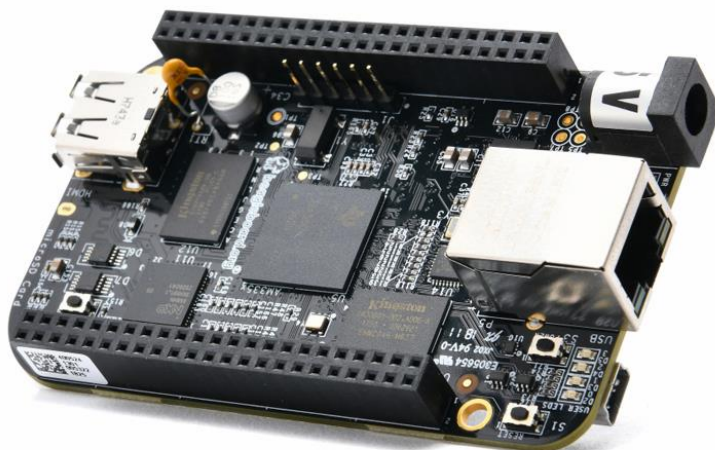
Een USB-naar-TTL-converter is een apparaat dat USB-signalen omzet in TTL-signalen (Transistor-Transistor Logic). Dit maakt het mogelijk om een computer te verbinden met een apparaat dat alleen TTL-signalen begrijpt, zoals sommige microcontroller-boards. Deze converter wordt vooral gebruikt voor te testen en om signalen op te vangen en zo te controleren. Hieronder een afbeelding van de converter.



Figuur 13 USB to TTL

3.4 BeagleBone Black (SBC)

SBC is een afkorting voor "Single Board Computer". Dit is een computer die op één enkel bord is gebouwd, in tegenstelling tot traditionele computers die uit verschillende borden en componenten bestaan. SBC's zijn vaak klein en energiezuinig en worden vaak gebruikt in embedded systemen, robots, drones, etc.



Figuur 14 BeagleBone Black

De BeagleBone Black is een single-board computer die speciaal ontworpen is voor hobbyisten, onderwijsinstellingen en ontwikkelaars die snel en gemakkelijk toegang willen tot de kracht van een microprocessor. Het is gebaseerd op de Texas Instruments AM3358 System-on-Chip (SoC) met een 1GHz ARM Cortex-A8 processor, 512MB DDR3 RAM en 4GB on-board eMMC opslag. Hiermee biedt de BeagleBone Black genoeg rekenkracht voor veelgebruikte toepassingen zoals web-ontwikkeling, robotica, automatisering, en het aansturen van sensoren en actuatoren.

De BeagleBone Black beschikt over diverse interfaces zoals USB, Ethernet, HDMI en seriële poorten, waarmee de computer verbonden kan worden met een breed scala aan randapparatuur en displays. Daarnaast heeft het een microSD-kaartslot waarmee gebruikers bestanden en software kunnen opslaan en laden. Verder zijn er diverse uitbreidingsmogelijkheden zoals pinheaders voor GPIO, I2C en SPI, waarmee een groot aantal sensoren en actuatoren aangesloten kunnen worden. Hierdoor kan de BeagleBone Black eenvoudig worden ingezet voor vele doeleinden.

De BeagleBone Black is ontworpen met open source software en hardware, wat betekent dat de ontwerpen voor de hardware en software vrij beschikbaar zijn en dat gebruikers zelf aanpassingen kunnen maken. Hierdoor kunnen gebruikers de BeagleBone Black aanpassen voor hun specifieke toepassingen. De BeagleBone Black wordt geleverd met een Linux-distributie genaamd Angstrom, maar ondersteunt ook andere besturingssystemen zoals Ubuntu en Android.

In het project hebben we gebruik gemaakt van het besturingssysteem Debian Buster IoT. Het is een versie van het Debian besturingssysteem die geoptimaliseerd is voor gebruik op Internet of Things (IoT) apparaten.

3.5 Home Assistant

Home Assistant is een open source platform voor centrale aansturing slimme apparaten en domotica, met focus op vrijheid en privacy. De software kan zowel als losse software, binnen een container of met een bijbehorend besturingssysteem gebruikt worden. Home Assistant is vervolgens toegankelijk via een web interface en mobiele applicaties voor Android en iOS. Het platform kan onafhankelijk van een internetverbinding of externe diensten werken en wordt daarom vaak aangewezen als een privacy vriendelijk alternatief voor andere domoticaplatformen.

Home Assistant kan via een breed scala aan integrations, gemaakt door vrijwilligers, communiceren met gangbare domoticaplatformen van onder andere Google, IKEA, Amazon, Apple (HomeKit), Signify (Philips Hue), Tuya en meer. Home Assistant fungeert dan ook als een "hub", van waaruit de gebruiker slimme logica, dashboards en automatisering kan programmeren zonder afhankelijk te zijn van verschillende applicaties van verschillende leveranciers van domotica hardware. Home Assistant biedt verder ondersteuning voor verschillende draadloze communicatieprotocollen zoals Bluetooth, Z-Wave en ZigBee. Verder kunnen slimme meters en controlesystemen van bijvoorbeeld zonnepanelen geïntegreerd worden om zo de gebruiker een live overzicht van energieverbruik te bieden, en op basis van verandering in deze data geautomatiseerd actie te ondernemen.

Home Assistant wordt openbaar ontwikkeld en bijdragen komen van vrijwilligers. Tussen september 2013 en juli 2022 hebben in totaal meer dan 2880 personen bijgedragen aan het open source project dat geleid wordt door Paulus Schoutsen. Hiervan werkt een subgroep van 20 tot 30 mensen intensief mee aan de doorontwikkeling van Home Assistant.

In het project gebruiken we Home Assistant om de energie meter te monitoren. Hierin maken we gebruik van enkel add-ons waaronder:

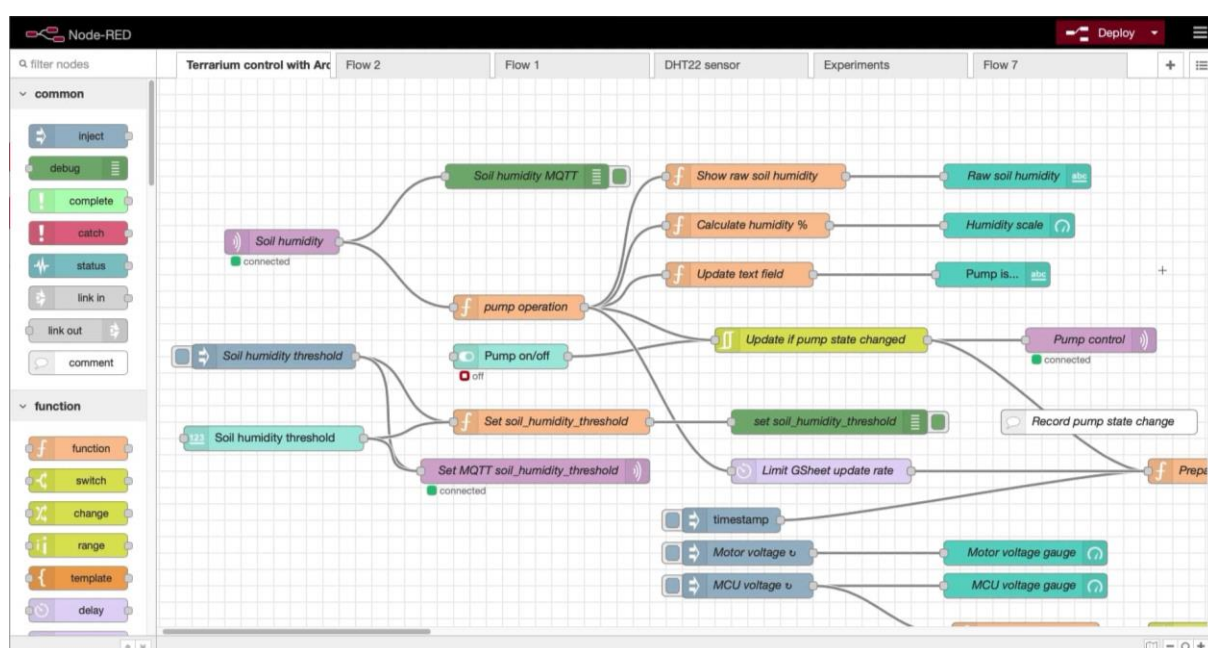
- Node-RED
- InfluxDB
- Grafana

3.5.1 Node-RED

Node-RED is een open-source visuele programmeertool waarmee hardware-apparaten, APIs en online diensten aan elkaar gekoppeld kunnen worden. Het stelt gebruikers in staat om stromen te maken, of reeksen van acties, met behulp van een drag-n-drop-interface. Deze stromen kunnen worden gebruikt om apparaten aan te sturen, gegevens op te halen en te verwerken, en berichten en meldingen te versturen. Node-RED is gebouwd op Node.js en wordt vaak gebruikt in Internet of Things (IoT)- en domotica-projecten.

Node-RED gebruiken we om de data uit de Cloud te halen om vervolgens in een database te steken.

De afbeelding hieronder illustreert een voorbeeld van een node-RED programmatie.



Figuur 15 Node-RED voorbeeld

3.5.2 InfluxDB

InfluxDB is een opensource tijdreeks-database die speciaal is ontworpen voor het opslaan, verwerken en analyseren van tijdreeksgegevens. Het is een NoSQL-database die gebaseerd is op de structuur van columnfamilies, waardoor het efficiënt is in het verwerken van grote hoeveelheden tijdreeksgegevens. InfluxDB biedt veel flexibiliteit in het opslaan van gegevens, zoals het toevoegen van tags en velden aan metingen, het creëren van series en het bewerken van data. Het wordt vaak gebruikt in combinatie met sensoren, IoT-apparaten, netwerkapparatuur en systemen voor het verzamelen van logboeken, prestatie- en meetgegevens. Daarnaast biedt InfluxDB een eenvoudige querytaal, genaamd InfluxQL, waarmee gebruikers gemakkelijk gegevens kunnen opvragen en analyseren. InfluxDB is ook goed te integreren met andere systemen en tools, zoals Grafana voor het visualiseren van gegevens.

InfluxDB gebruiken we om de data van de energie meter te stockeren.



Figuur 16 InfluxDB voorbeeld

3.5.3 Grafana

Grafana is een open-source platform voor visuele monitoring. Het maakt gebruik van een query-taal om gegevens uit verschillende bronnen te halen, zoals Prometheus, InfluxDB, Graphite, Elasticsearch en andere. Deze gegevens worden vervolgens weergegeven in mooie, interactieve grafieken en dashboards. Met Grafana kunt u bijvoorbeeld de prestaties van uw servers volgen, de gezondheid van uw netwerk controleren en de gebruiksstatistieken van uw applicaties bekijken. U kunt ook alert regels instellen zodat u op de hoogte wordt gebracht als er iets misgaat.

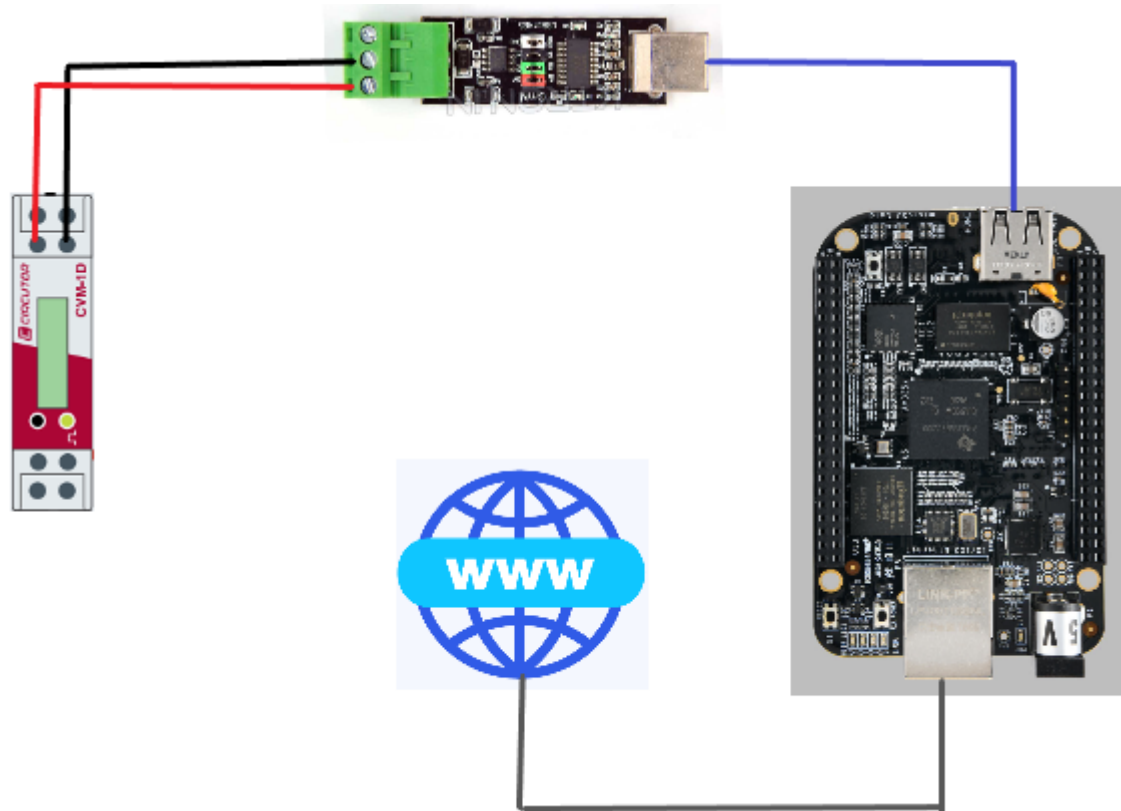
Grafana biedt ook een groot aantal plug-ins en integraties, waardoor het gemakkelijk is om gegevens vanuit verschillende bronnen te verzamelen en te analyseren. En met de ingebouwde samenwerking en deelmogelijkheden kunt u eenvoudig dashboards delen met collega's en klanten. Hieronder een voorbeeld van zo een dashboard.



Figuur 17 Grafana voorbeeld

3.5.4 Verbinding componenten

De elektronische componenten verbinden is noodzakelijk om een werkende te hebben. Door de componenten te verbinden, kunnen ze communiceren en samenwerken om de gewenste functie uit te voeren. In de afbeelding hieronder kan je zien hoe ze verbonden zijn.

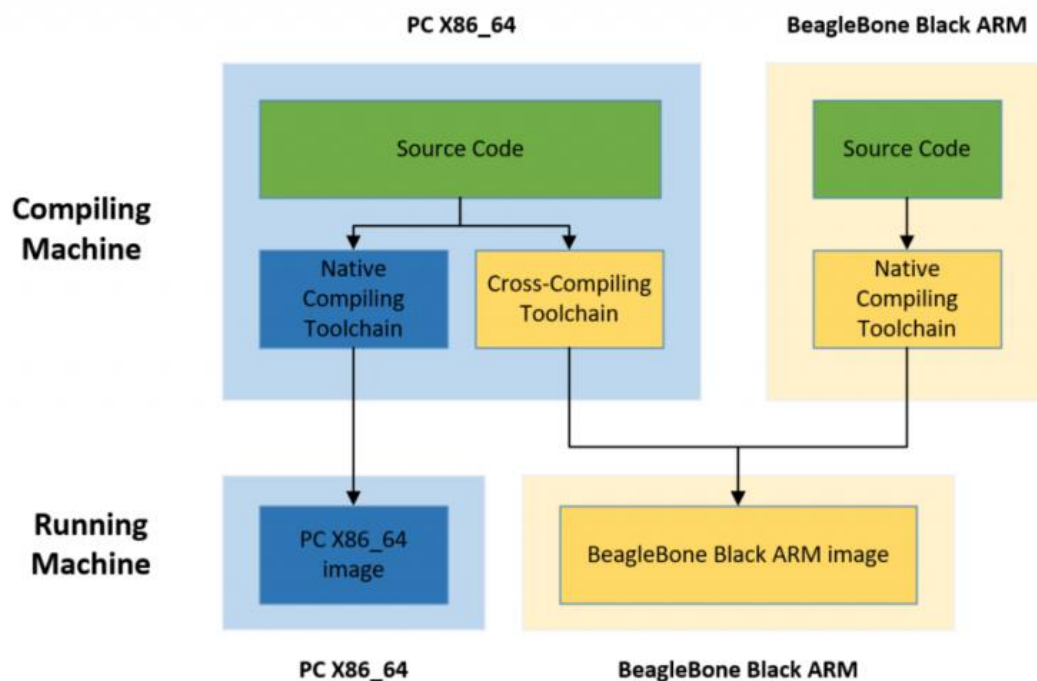


Figuur 18 Verbinding componenten

4 CROSS-COMPILING

Kort uitgelegd is compileren een programma/software dat is geschreven in een programmeertaal “vertalen” naar computertaal.

Cross-compiling is het proces waarbij software wordt gecompileerd voor een ander platform dan het platform waarop de compilatie plaatsvindt. Dit betekent dat de software wordt gecompileerd voor een ander besturingssysteem of processorarchitectuur dan waarop de compilatie plaatsvindt. In ons geval is het een programma dat wordt gecompileerd op een systeem met de x64 architectuur naar een ARM architectuur.



Figuur 19 Native & cross compiling

Zoals je op de bovenstaande afbeelding kan zien, zijn er 2 soorten toolchains:

- Native-Compiling Toolchain:**
 Een native compiling toolchain is een set van tools die worden gebruikt om software te compileren voor hetzelfde platform waarop de software zal worden uitgevoerd.
- Cross-Compiling Toolchain:**
 Een cross-compile toolchain is een set van tools die je in staat stelt om code te bouwen voor een platform (target) op een ander platform (host). De toolchain omvat vaak een cross-compiler, die een versie van de compiler is die op de host draait, maar code genereert voor de target, evenals andere tools zoals een linker en assembler die ook zijn afgestemd op het target platform.

In dit project maken we gebruik van de Cross-Compiling Toolchain.

5 VOORBEREIDING

Zoals in het hoofdstuk Cross-Compiling al uitgelegd, gaan we gebruik maken van de Cross-Compiling Toolchain. Voor dat we hier mee aan de slag kunnen gaan moeten we eerst een aantal voorbereidingen maken.

- Installeren van een Debian Virtual Machine.
- Voorbereiden van de BeagleBone Black.
- Aanmaken GIT

5.1 Debian Virtual Machine

Voor het Cross-Compiling gebruiken we een laptop die draait op een x64 processor architectuur. Omdat het besturingssysteem van de BeagleBone een Debian versie is gaan we op de laptop een Debian VM installeren. Van daaruit gaan we dan alle prototypes bouwen en de uiteindelijke software voor het project. Alles gebeurt hier ook remote, de BeagleBone is dus niet fysiek gekoppeld met de laptop.

De software die ik gebruikt heb voor de VM aan te maken is VirtualBox. Het installatie proces ga ik niet in detail bespreken omdat dit nogal veel is. Hieronder staat een link met alle stappen met screenshots die moeten genomen worden om een Debian VM aan te maken.

<https://linuxopsys.com/topics/install-debian-on-virtualbox>

5.2 Voorbereiding BeagleBone Black

Voor de BeagleBone klaar te maken voor Cross-Compiling zijn wel een aantal stappen nodig. Zo moeten we:

- **Het besturingssysteem installeren:**
Voor het installeren van het besturingssysteem heb ik een SD kaart gebruikt en heb ik de stappen gevolgd die in de onderstaande link staan, dit is de officiële website van Beagleboard.
<https://beagleboard.org/getting-started>
- **De Cross-Compiler Toolchain installeren**
- **Eclipse IDE installeren**
- **Configureren van remote debugging in Eclipse IDE**

Voor het installeren van de Cross-Compiler Toolchain, Eclipse IDE en het configureren van remote debugging heb ik een video gebruikt die je in de link hieronder kan terug vinden.

[Installatie-video](#)

5.3 GIT

Voor het bouwen van de prototypes en het altijd en overal ter beschikking te hebben heb ik gebruik gemaakt van GIT en Github. GIT is een gratis, open-source versiebeheersysteem dat is ontworpen om samenwerking en efficiëntie in softwareontwikkeling te verbeteren. Het maakt het mogelijk om wijzigingen in software of andere bestanden bij te houden en eerdere versies te herstellen.

Git houdt bijvoorbeeld bij wie welke wijziging heeft aangebracht, wanneer deze is aangebracht, en waarom deze is aangebracht. Dit maakt het gemakkelijk om bij te houden wie verantwoordelijk is voor welke code en problemen op te lossen als er problemen optreden. Git biedt ook de mogelijkheid om branches aan te maken, waardoor meerdere ontwikkelaars tegelijkertijd aan hetzelfde project kunnen werken zonder dat dit leidt tot conflicten.

Git is een van de meest populaire versiebeheersystemen die wordt gebruikt voor softwareontwikkeling en wordt vaak gebruikt in combinatie met GitHub, een platform waar softwareontwikkelaars projecten kunnen delen, samenwerken en bijdragen.

6 TESTEN EN PROTOTYPES

Een prototype is een voorbeeld of model van een product of ontwerp. Het kan worden gebruikt om het concept te testen of te demonstreren, of om feedback te verzamelen van gebruikers of andere belanghebbenden. Prototypes kunnen fysiek of digitaal zijn en kunnen op verschillende niveaus van afwerking variëren, van ruwe schetsen tot bijna afgewerkte producten.

Stap voor stap gaan we prototypes bouwen en testen uitvoeren zodat als deze geslaagd zijn we er verder op kunnen bouwen. Dit zorgt ervoor dat we de final code zo gemakkelijk mogelijk kunnen bouwen en zonder bugs in. De broncode van elk prototype of test is beschikbaar via Github, de link staat bij elke kop.

1. Hello World! In Cross-Compiling.
2. Test Serial-port.
3. Modbus data Circutor CVM-1D.
4. Test modbus frame sending
5. File-splitting & register definition
6. Paho MQTT C-library
7. Prototype

Het volledige project inclusief al de testen, het prototype en de final code is terug te vinden op github.

<https://github.com/bartpassat/EnergyMeterProject.git>

6.1 Hello World! In Cross-Compiling

[Link](#) naar Github.

De eerste code die ik gemaakt heb is Hello World!. Ik heb deze code als eerst gemaakt om te controleren dat de Cross-Compiling en remote debugging juist geconfigureerd is en werkt.

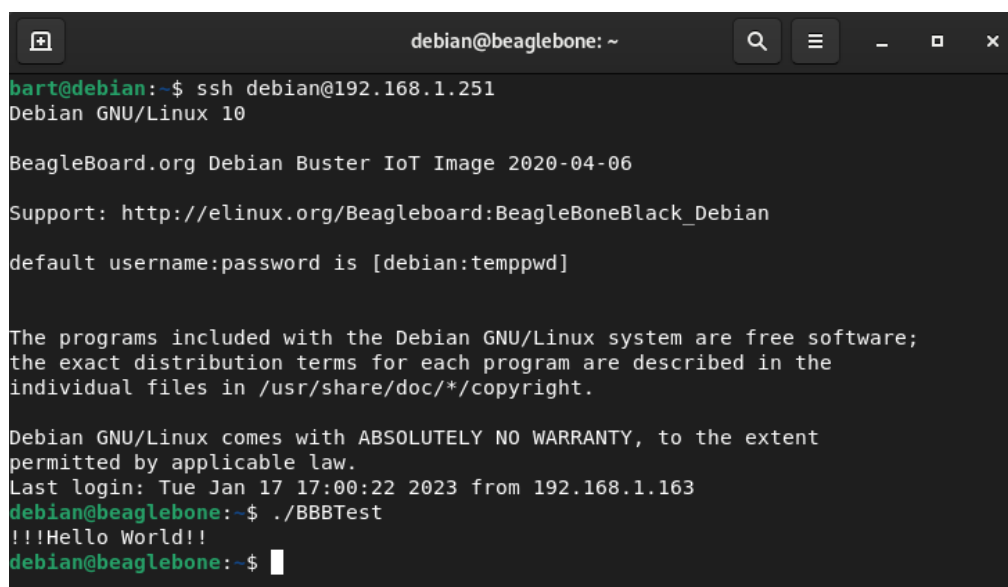
```

 9 #include <iostream>
10 using namespace std;
11
12 int main() {
13     int x = 5;
14     x++;
15     cout << "!!!Hello World!!" << endl; // prints !!!Hello World!!!
16     return 0;
17 }
18

```

Figuur 20 Hello World! code

In de bovenstaande afbeelding kan je zien dat ik voor deze test twee elementen heb gebruikt. Met `cout << "!!!Hello World!!" << endl;` op regel 15 ga ik de tekst "!!!Hello World!!" printen in de linux terminal. Hiermee kan ik zonder te debuggen controleren in de terminal van de BeagleBone of dat de Cross-Compiling werkt. We moeten alleen verbinding maken met de BeagleBone en het bestand BBBTest uitvoeren. Dit is het uitvoerbaar bestand van onze code. Dat kan je zien in de afbeelding hieronder.



```

debian@beaglebone: ~
bart@debian:~$ ssh debian@192.168.1.251
Debian GNU/Linux 10

BeagleBoard.org Debian Buster IoT Image 2020-04-06

Support: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

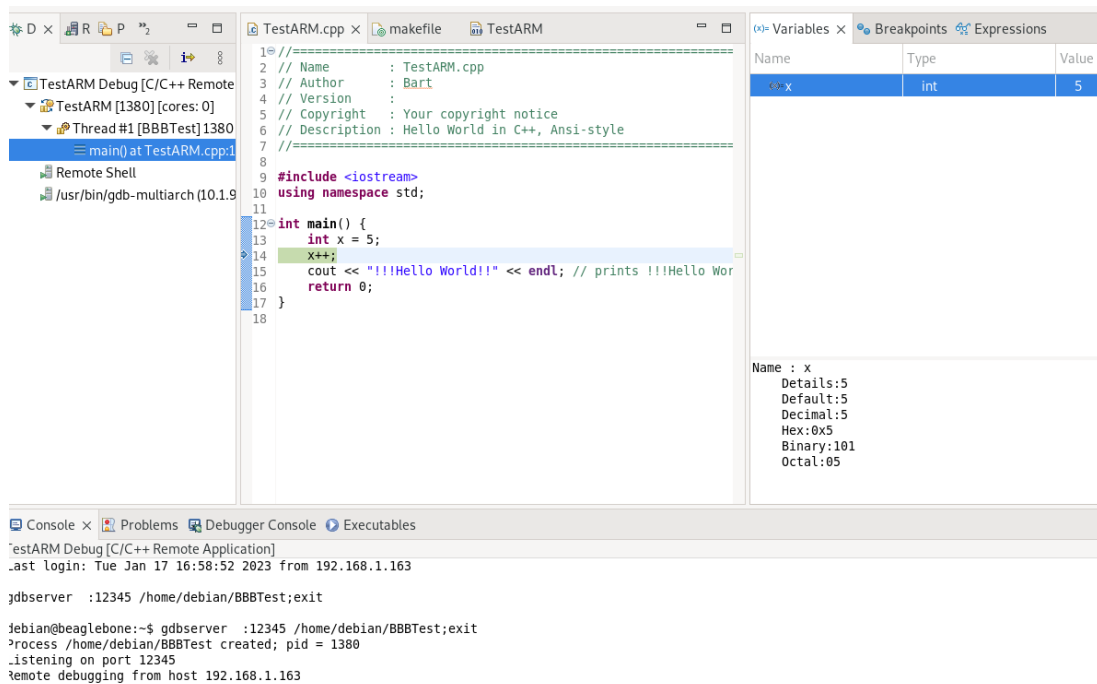
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Jan 17 17:00:22 2023 from 192.168.1.163
debian@beaglebone:~$ ./BBBTest
!!!Hello World!!
debian@beaglebone:~$

```

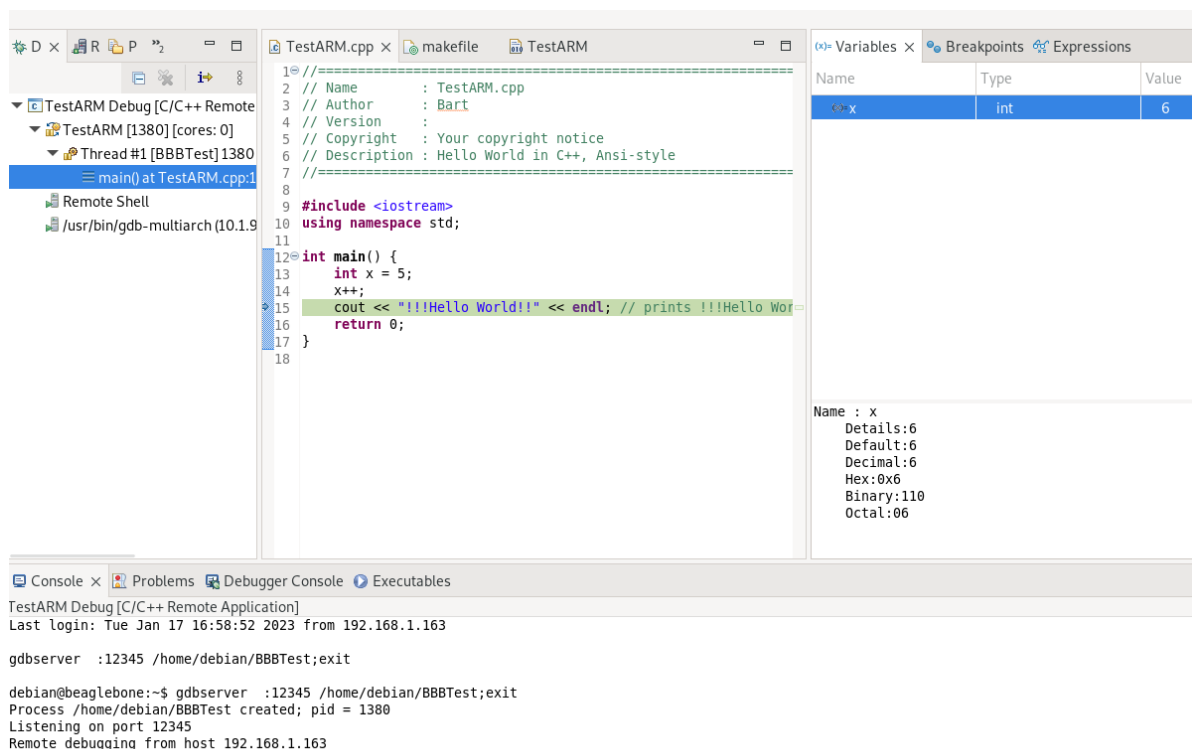
Figuur 21 Output Hello World!

Op regel 13 heb ik de variabele **x** heb aangemaakt waar ik de waarde **5** aan heb toegewezen. Op regel 14 ga ik deze incrementeren. Dit heb ik gedaan om het remote debuggen te controleren.



Figuur 22 Hello World! Remote debugging step 1

In bovenstaande afbeelding kan je in het tabblad Console zien dat we remote aan het debuggen zijn. In de file TestARM.cpp zijn we op regel 15, in de rechtse kolom onder Variables zien we `x` staan met de waarde 5. In de afbeelding hieronder zijn we voorbij de stap met het incrementeren en als we nu naar de variabele `x` kijken zien we dat de waarde nu 6 is. Het remote debuggen werkt!



Figuur 23 Hello World! Remote debugging step2

6.2 Test Serial-port

[Link](#) naar github.

Zoals uitgelegd in het hoofdstuk [communicatie protocollen](#), gebeurt het uitlezen van data van de energiemeter via USB. Voor dat we hiermee kunnen beginnen gaan we eerst controleren of er communicatie is via de USB poort. Voor dit te kunnen doen gaan we de [USB to TTL converter](#) gebruiken. Hier gaan we simpelweg RX met TX (loopback) verbinden, zo kunnen we de data dat we versturen terug gaan opvangen en controleren. Als deze test geslaagd is gaan we deze stukjes code gebruiken in de verdere prototypes en de final code.

6.2.1 Openserial(serialdev)

Voor data te kunnen sturen via USB wordt de seriële poort (USB) geopend. Bij het openen van de seriële poort krijgen we van het systeem een file descriptor toegewezen. Een file descriptor is een waarde die door het operating system wordt gebruikt om een file te identificeren. Dit kunnen we dan gebruiken om dat bepaald bestand uit te lezen of er iets in te schrijven. Het openen gaan we doen met de functie **openserial(serialdev)**. Deze functie geven we de locatie van het bestand, dat doen we door de array **serialdev[]** (regel 111). daarna gaat deze functie ons de file descriptor (fd) geven (regel 116).

```

111     char serialdev[] = "/dev/ttyUSB0";
112
113     puts(serialdev);
114     puts ("Open port");
115
116     fd = openserial(serialdev);
117
118     if (!fd) {
119         fprintf(stderr, "Error while initializing %s.\n", serialdev);
120         return 1;
121     }
122

```

Figuur 24 openserial(serialdev)

Als de functie niks terug geeft wilt het zeggen dat er iets is mis gelopen met het openen. Dat wordt gecontroleerd met een if-statement (regel 118). Als er iets is misgelopen gaan we een foutcode op de terminal printen (regel 119) en het programma afbreken (regel 120).

Als we in de openserial() functie gaan kunnen we nog een aantal zaken aanpassen als we willen, we kunnen onder andere:

- De baudrate aanpassen.
- De karakter grote aanpassen.
- Stop bits selecteren.
- Pariteit aan of uit zetten.
- Receiver aan of uit zetten.

Baudrate aanpassen doen we met de functie cfsetospeed().

<https://linux.die.net/man/3/cfsetospeed>

Al de rest gaan we aanpassen met de C_flags.

<https://man7.org/linux/man-pages/man0/termios.h.0p.html>


```

34  int openserial(char *devicename)
35  {
36      int fd;
37      struct termios tty;
38
39      if ((fd = open(devicename, O_RDWR | O_NOCTTY | O_NONBLOCK)) == -1)
40      {
41          perror("openserial(): open()");
42          return 0;
43      }
44
45      if (tcgetattr(fd, &oldterminfo) == -1) {
46          perror("openserial(): tcgetattr()");
47          return 0;
48      }
49
50      tty = oldterminfo;
51
52      cfsetospeed (&tty, B115200);
53      cfsetispeed (&tty, B115200);
54
55      tty.c_cflag = (tty.c_cflag & ~CSIZE) | CS8;    // 8-bit chars
56      // disable IGNBRK for mismatched speed tests; otherwise receive break
57      // as \000 chars
58      tty.c_iflag &= ~IGNBRK;    // disable break processing
59      tty.c_lflag = 0;    // no signaling chars, no echo,
60                          // no canonical processing
61      tty.c_oflag = 0;    // no remapping, no delays
62      tty.c_cc[VMIN] = 0;    // read doesn't block
63      tty.c_cc[VTIME] = 5;    // 0.5 seconds read timeout
64      tty.c_iflag &= ~(IXON | IXOFF | IXANY); // shut off xon/xoff ctrl
65      tty.c_cflag |= (CLOCAL | CREAD); // ignore modem controls,
66                          // enable reading
67      tty.c_cflag &= ~(PARENB | PARODD);    // shut off parity
68      tty.c_cflag &= ~CSTOPB;
69      tty.c_cflag &= ~CRTSCTS;
70
71      if (tcsetattr (fd, TCSANOW, &tty) != 0)
72      {
73
74          return 1;
75      }
76
77      return fd;
78  }

```

*Figuur 25 openserial(char *devicename)*

6.2.2 Write & read

Het volgende wat we gaan doen is data via de seriële poort versturen en terug uitlezen. Om dit proces te controleren gaan we telkens printen wat we gaan doen in de terminal. Het printen gaan we doen met de functie **puts()**. Het versturen doen we met de functie

write(fd, msg, sizeof(msg))). Deze functie moeten we de file descriptor (fd) meegeven, de data (msg) en de grote van de data.

```
123     puts("Send data over TX");
124     unsigned char msg[] = {"Data RX/TX test passed"};
125     write(fd, msg, sizeof(msg));
126
```

Figuur 26 Write data

Het volgende dat er dan wordt gedaan is de data terug uitlezen.

```
127     puts ("wait 1 second");
128     sleep (1);
129
130     puts("Read data from RX buffer");
131     char read_buf [256];
132     int n = read(fd, &read_buf, sizeof(read_buf));
133
134     puts("Bytes recieved:");
135     puts(read_buf);
136
```

Figuur 27 Read data

Met **sleep(1)** regel(128) wachten we eerst 1 seconden. We creëren daarna een buffer (regel 131) om de data in op te slaan. Daarna gaan we de data uitlezen met de functie **read(fd, &read_buf, sizeof(read_buf))** (regel 132). De uitgelezen data wordt in de buffer geplaatst. Daarna wordt de data geprint in de terminal.

6.2.3 Compare data

Na eerst data verzonden te hebben en daarna uitgelezen te hebben gaan we de laatste stap in de test doen, het vergelijken van de data. Omdat we een loopback hebben gecreëerd moet de data die we verzonden hebben ook terug komen. Om dit te controleren gaan we gebruik maken van de functie:

memcmp(read_buf, "Data RX/TX test passed", 22)

Hiermee gaan we de read_buf vergelijken met de data "Data RX/TX test passed" het nummer "22" geeft aan hoeveel karakters we willen vergelijken.

```
139     if (memcmp(read_buf, "Data RX/TX test passed", 22))
140     {
141         error = 1;
142         puts ("Data RX/TX test FAILED");
143     }
144
145     if (error)
146     {
147         puts("***** SERAIL PORT TEST FAILED *****");
148     }
149     else
150     {
151         puts("serial port OK");
152     }
153
154     closeserial(fd);
155     return 0;
```

Figuur 28 Test case

Als de test niet geslaagd is (de functie **memcmp** geeft dan een andere waarde als 0)

gaan we de variabele **error** op 1 zetten. Die gaan we daarna controleren in een if-statement en zo nodig een error printen in de terminal. Als ze wel geslaagd is gaan we de file descriptor terug sluiten met de functie **closeserial(fd)**. En het programma afbreken. Als we dit uitvoeren krijgen we het volgende op de terminal:

```

debian@beaglebone:~$ ./BBBTest
/dev/ttyUSB0
Open port
Send data over TX
wait 1 second
Read data from RX buffer
Bytes recieved:
Data RX/TX test passed
Compare RX with TX data
serial port OK
debian@beaglebone:~$

```

Figuur 29 Output serial test

6.3 Modbus data Circutor CVM-1D

In [Modbus/PDU](#) heb ik uitgelegd hoe het Modbus data model eruit ziet en hoe ze te gebruiken. Voor de energiemeter uit te lezen moeten we weten welke registers aan te spreken om de gewenste data te kunnen krijgen. Dit kunnen we terug vinden in de [datasheet](#) van de energiemeter. In de afbeelding hieronder kan je bij **Instantaneous** een deel van de adressen van de registers terug vinden.

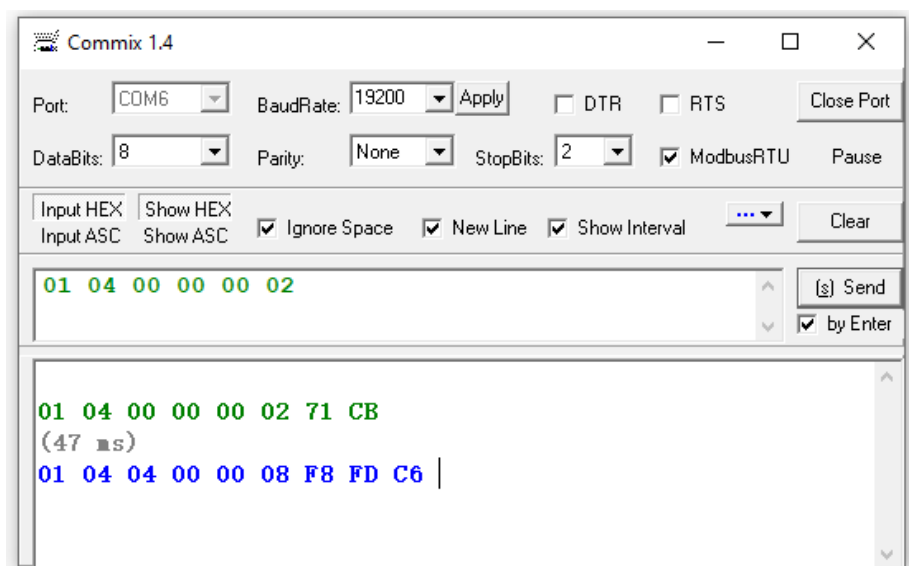
6.- Modbus/RTU memory map

Parameters	Symbol	Var	Instantaneous	Maximum	Minimum	Units
Voltage	V	1	0000-0001	0032-0033	0044-0045	V x10
Current	A	2	0002-0003	0034-0035	0046-0047	A x100
Active Power	kW	3	0004-0005	0036-0037	0048-0049	± kW x100
Reactive Power (L/C)	kvar	4	0006-0007	0038-0039	004A-004B	± kvar x100
Inductive Reactive Power	kvarL	5	0008-0009	003A-003B	004C-004D	± kvarL x100
Capacitive Inductive Power	kvarC	6	000A-000B	003C-003D	004E-004F	± kvarC x100
Apparent power	kVA	7	000C-000D	003E-003F	0050-0051	± kVA x100
Power Factor	PF	8	000E-000F	0040-0041	0052-0053	PFx100
Maximum Demand	kW / A	9	0010-0011	0042-0043	0054-0055	kW / A x100
Active energy	kW·h	10	0012-0013	-	-	kW·h x100
Inductive Reactive Energy	kvarL·h	11	0014-0015	-	-	kvarL·h x100
Capacitive Reactive Energy	kvarC·h	12	0016-0017	-	-	kvarC·h x100
Reactive Energy (L/C)	kvar·h	13	0018-0019	-	-	kvar·h x100

Figuur 30 Register addresses energiemeter

Voor dit te onderzoeken en te testen heb ik het volgende commando gebruikt als voorbeeld. 01 04 00 00 00 02 + CRC. Met dit commando gaan we de actuele spanning uitlezen.

- **01**: Device address.
- **04**: Read input register.
- **00 00**: Met deze twee bytes geven we weer welk register we willen bereiken.
- **00 02**: Met deze twee bytes geven we weer hoeveel bytes we willen uitlezen.



Figuur 31 Voorbeeld Modbus data

Met dit voorbeeld kunnen we de data analyseren. De eerste byte is zoals bij het versturen het device address. De tweede byte is ook zoals bij het versturen de functie code. De derde byte die geeft aan uit hoeveel bytes de data bestaat. De data die we uiteindelijk verwachte moet uit één getal bestaan. Om dit te verwezenlijken gaan we de eerstvolgende "bruikbare" byte (08) vermenigvuldigen met 256. Dit doen we omdat dit de **high byte** is. Dan tellen we de **low byte** (F8) er bij op. Deze uitkomst (2296) vermenigvuldigen we met 10 omdat de datasheet dit zo aangeeft, dat kunnen we zien bij Units. De uitkomst hiervan is 22960, dit zou de voltage moeten zijn. Omdat ik dit verwarrend vind om mee te werken heb ik dit anders gedaan, in plaats van te vermenigvuldigen ga ik het delen door 10. De uitkomst is dan 229,6 Volt, dit vind ik veel interessanter om mee te werken omdat dit leesbaarder is. In totaal ontvangen we 9 bytes, dit is ook handig om te weten om het frame te kunne controleren verder in de ontwikkeling.

6.4 Test modbus frame sending

[Link](#) naar Github.

Nu dat we weten dat de seriële poort werkt en we het Modbus frame kennen gaan we een stuk code maken dat het frame kan verzenden en ontvangen.

Voor dit te doen is er een nieuwe functie toegevoegd:

```
170 void ReadAddress(int l_fd, unsigned char l_slaveaddress, unsigned int l_address, unsigned char l_debug)
```

Deze functie moeten we de file descriptor bezorgen (l_fd), het slave adres (l_slaveaddress), het register adres (l_address) en als we het proces willen monitoren of niet (l_debug). Het monitoren is in het begin belangrijk omdat we dan weten wat er gebeurt.

In onderstaande afbeelding is te zien dat deze functie eerst het frame zal aanmaken met het slave adres en het register adres dat we de functie hebben geven, het zal dan ook automatisch de CRC berekenen en toevoegen aan het frame.

```

170 void ReadAddress(int l_fd, unsigned char l_slaveaddress, unsigned int l_address, unsigned char l_debug)
171 {
172     unsigned char l_counter;
173     unsigned short l_crc16;
174     unsigned char l_modbusframe[8];
175     l_modbusframe[0] = l_slaveaddress; // Modbus slave address
176     l_modbusframe[1] = 0x04;           // Function code
177     l_modbusframe[2] = 0x00;           // Hi byte of register
178     l_modbusframe[3] = l_address;      // Lo byte of register
179     l_modbusframe[4] = 0x00;
180     l_modbusframe[5] = 0x02;           // All registers we need are 16-bit
181
182     l_crc16 = CRC16(l_modbusframe, 6);
183     l_modbusframe[6] = l_crc16;        // Lo byte of CRC16
184     l_modbusframe[7] = l_crc16 >> 8; // Hi byte of CRC16
185 }

```

Figuur 32 ReadAddress create frame

Na dit gedaan te hebben zal de functie het frame verzenden naar de file descriptor. Het zal erna ook de functie **tcdrain(l_fd)** aanroepen. Deze functie zorgt er voor dat alle data die naar de file descriptor is geschreven ook daadwerkelijk wordt verzonden. Als er gekozen is voor de "debug" optie, dan zal het frame door de for-loop worden geprint in de terminal.

```

187 write(l_fd, l_modbusframe, 8); // Send modbus frame
188 tcdrain(l_fd);                // wait until all bytes have been sent
189
190 if (l_debug)
191 {
192
193     printf("Modbus frame send: ");
194     for (l_counter=0; l_counter<7; l_counter++)
195     {
196         printf("%#02x, ", l_modbusframe[l_counter]);
197     }
198     printf("%#02x.\n", l_modbusframe[7]);
199 }

```

Figuur 33 Send frame & debug

Het frame is aangemaakt en het is verstuurd, nu gaan we de ontvangen data in lezen, controleren en printen in de terminal. Het lezen doen we door de **read** functie, deze functie zal ook het aantal bytes dat zijn ontvangen terug geven, die gaan we in de variabele **n** zetten (regel 205). Eerder bij het analyseren van het frame hebben we gezien dat we 9 bytes moeten ontvangen, dit gaan we nu controleren. Als de variabele **n** niet gelijk is aan 9 (regel 207) wilt het zeggen dat er iets is misgelopen, we gaan dan een foutcode sturen. Het versturen van de error wordt alleen maar gedaan als de debug functie geactiveerd is.

```

204     sleep(1);
205     int n = read(l_fd, RawRxData, MODBUSRXBUFFERSIZE);
206
207     if (n!=9)
208     {
209         if (l_debug)
210         {
211             printf("[MODBUS ERROR] Nr of bytes received: %d (should be 9).\n", n);
212         }
213         valid=0;
214     }
215     else
216     {
217         if (l_debug)
218         {
219             printf("[MODBUS] Frame received: ");
220             for (l_Counter=0; l_Counter<8; l_Counter++)
221             {
222                 printf("#02x, ", RawRxData[l_Counter]);
223             }
224             printf("#02x. \n", RawRxData[8]);
225         }
226     }
227 }

```

Figuur 34 Read frame

Als het frame wel correct is en we gebruiken de debug functie dan zal het ontvangen frame worden getoond in de terminal.

In de main hebben we deze functie toegevoegd en alle parameters meegegeven.

```

230 int main (int argc, char *argv[])
231 {
232     int fd;
233     char serialdev[] = "/dev/ttyUSB0";
234
235     puts(serialdev);
236     puts ("Open port");
237
238     fd = openserial(serialdev);
239
240     if (!fd) {
241         fprintf(stderr, "Error while initializing %s.\n", serialdev);
242         return 1;
243     }
244
245     ReadAddress(fd, 0x01, 0x00, 1);
246     closeserial(fd);
247     return 0;
248 }

```

Figuur 35 Main frametest

In onderstaande afbeelding is de output van de frametest in de terminal te zien.

```

debian@beaglebone: ~
debian@beaglebone:~$ ./BBBTest
/dev/ttyUSB0
Open port
Modbus frame send: 0x1, 0x4, 00, 00, 00, 0x2, 0x71, 0xcb.
[MODBUS] Frame received: 0x1, 0x4, 0x4, 00, 00, 0x8, 0xe4, 0xfc, 0xf.
debian@beaglebone:~$

```

Figuur 36 Frametest output

6.5 File-splitting & register definition

[Link](#) naar Github.

Met al de functies die zijn aangemaakt of toegevoegd is het handiger om dit op te splitsen in verschillende files. Voor de seriële poort is een aparte file aangemaakt waar de functies daarvan in staan, hetzelfde heb ik gedaan voor alles van Modbus. Dan heb ik nog een aparte file aangemaakt om alle registers van de energie meter te definiëren en ook de register namen. Die gaan we later nog nodig hebben om het uitlezen van verschillende registers te vergemakkelijken. De register namen kunnen we dan ook samen met de data op de terminal tonen om te weten over welke data het gaat.

```

14 static int Circutor_CVM_1D_Address = 0x01;
15
16 static int Registers_Circutor_CVM_1D[] = {
17     0x0000, 0x0032, 0x0044,
18     0x0002, 0x0034, 0x0046,
19     0x0004, 0x0036, 0x0048,
20     0x0006, 0x0038, 0x004A,
21     0x0008, 0x003A, 0x004C,
22     0x000A, 0x003C, 0x004E,
23     0x000C, 0x003E, 0x0050,
24     0x000E, 0x0040, 0x0052,
25     0x0010, 0x0042, 0x0054,
26     0x0012, 0x0014, 0x0016, 0x0018, 0x001A, 0x001C, 0x001E, 0x0020,
27     0x0022, 0x0024, 0x0026, 0x0028, 0x002A, 0x002C, 0x002E, 0x0030
28 };
29 };
30
31 static u_int8_t RegisterNames_Circutor_CVM_1D[][50] = {
32     "Actual Voltage", "Maximum Voltage", "Minimum Voltage",
33     "Actual Current", "Maximum Current", "Minimum Current",
34     "Actual Active Power", "Maximum Active Power", "Minimum Active Power",
35     "Actual Reactive Power", "Maximum Reactive Power", "Minimum Reactive Power",
36     "Actual Inductive Reactive Power", "Maximum Inductive Reactive Power", "Minimum Inductive Reactive Power",
37     "Actual Capacitive Inductive Power", "Maximum Capacitive Inductive Power", "Minimum Capacitive Inductive Power",
38     "Actual Apparent power", "Maximum Apparent power", "Minimum Apparent power",
39     "Actual Power Factor", "Maximum Power Factor", "Minimum Power Factor",
40     "Actual Maximum Demand", "Maximum Maximum Demand", "Minimum Maximum Demand",
41     "Active energy", "Inductive Reactive Energy", "Capacitive Reactive Energy", "Reactive Energy", "Partial Active Energy",
42     "Partial Inductive Reactive Energy", "Partial Capacitive Reactive Energy", "Partial Reactive Energy", "Generated Active Energy",
43     "Generated Inductive Reactive Energy", "Generated Capacitive Reactive Energy", "Generated Total Reactive Energy",
44     "Partial Generated Active Energy", "Partial Generated Inductive Reactive Energy", "Generated Capacitive Reactive Energy",
45     "Partial Generated Total Reactive Energy"
46 };

```

Figuur 37 Register & registernames energy meter

Ik heb dit ook zo opgebouwd dat de registers en de register namen op dezelfde posities in de twee array's staan, zo kunnen we dit later combineren.

In de main file dat is afgebeeld kan je zien dat ik de functie **ReadAddress** als parameters de variabele van het device address en de tweede positie van het register array heb gegeven. Deze positie komt overeen met "Maximum Voltage".

```

32 int main (int argc, char *argv[])
33 {
34     int fd;
35     int error = 0;
36     char serialdev[] = "/dev/ttyUSB0";
37
38     puts(serialdev);
39     puts ("Open port");
40
41     fd = openserial(serialdev);
42
43     if (!fd) {
44         fprintf(stderr, "Error while initializing %s.\n", serialdev);
45         return 1;
46     }
47
48     ReadAddress(fd, Circutor_CVM_1D_Address, Registers_Circutor_CVM_1D[1], 1);
49     closeserial(fd);
50     return 0;
51 }

```

Figuur 38 Main file-splitting & registers definition

6.6 Paho MQTT C-library

[Link](#) naar Github.

Zoals ik eerder al heb uitgelegd in de [omschrijving](#), gaan we gebruik maken van MQTT om data naar de Cloud te sturen. Om dit te kunnen doen heb ik gebruik gemaakt van de Paho MQTT C-library.

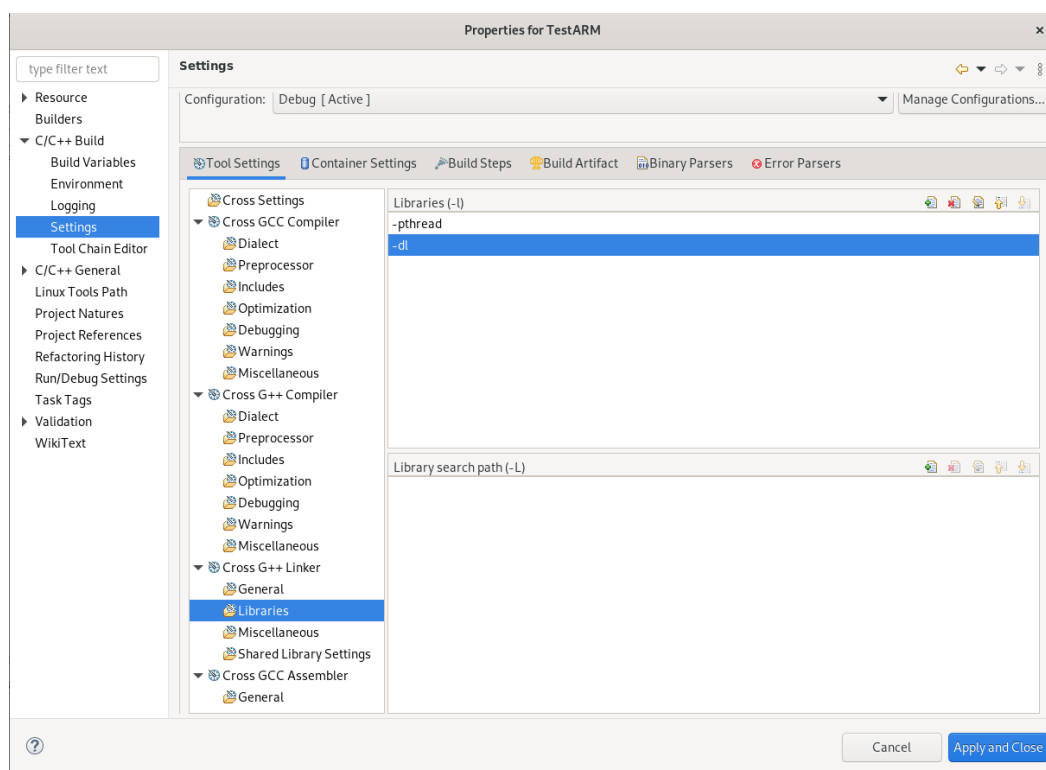
De Paho MQTT C-library is een open-source client implementatie van het MQTT (Message Queuing Telemetry Transport) protocol, geschreven in de programmeertaal C. Het is ontworpen om te werken op een breed scala van besturingssystemen en platforms. De library biedt een eenvoudige API voor het verbinden met MQTT-brokers, subscriben op topics, publishen van messages en kan gemakkelijk worden geïntegreerd in C/C++-applicaties.

De library is terug te vinden op Github: <https://github.com/eclipse/paho.mqtt.c>

We kunnen deze library gebruiken op verschillende manieren, meer daarover is terug te vinden op de Github pagina. In dit project gaan we de library uiteindelijk toevoegen aan het project zelf maar we gaan dit eerst testen in een afzonderlijk project. Hier zijn een paar stappen voor nodig.

1. Ik heb een nieuw project gestart en alle .c en .h files in de map src van paho met uitzondering van de samples gekopieerd naar de src map in het nieuwe project.

2. Het volgende dat we moeten doen is -pthread en -dl toevoegen aan de linker zoals afgebeeld:



Figuur 39 paho linker project

3. Nu moeten we MQTTAsync.c, MQTTAsyncUtils en MQTTVersion.c exclude from build. Dit doen we door rechter muisknop op de file en dan bij resource configuration exclude from build en select all.
4. Nu moeten we de file "VersionInfo.h.in openen en de inhoud veranderen in:

```
#ifndef VERSIONINFO_H
#define VERSIONINFO_H

#define BUILD_TIMESTAMP "2023"
#define CLIENT_VERSION "V1.0"

#endif /* VERSIONINFO_H */
```

Daarna moet de file worden hernoemt naar "VersionInfo.h".

5. Het laatste wat we doen is de code vermeld in de link aan de main file van het project toevoegen.
<https://eclipse.github.io/paho.mqtt.c/MQTTClient/html/pubsync.html>
 Het enige dat ik heb veranderd is de link naar de broker veranderd in test.mosquitto.org.

```
17 #define ADDRESS "tcp://test.mosquitto.org:1883"
```

De main gaat er als volgend uitzien maar dan met het address aangepast.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "MQTTClient.h"

#define ADDRESS      "tcp://mqtt.eclipseprojects.io:1883"
#define CLIENTID     "ExampleClientPub"
#define TOPIC        "MQTT Examples"
#define PAYLOAD      "Hello World!"
#define QOS          1
#define TIMEOUT      10000L

int main(int argc, char* argv[])
{
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    int rc;

    if ((rc = MQTTClient_create(&client, ADDRESS, CLIENTID,
                              MQTTCLIENT_PERSISTENCE_NONE, NULL)) != MQTTCLIENT_SUCCESS)
    {
        printf("Failed to create client, return code %d\n", rc);
        exit(EXIT_FAILURE);
    }

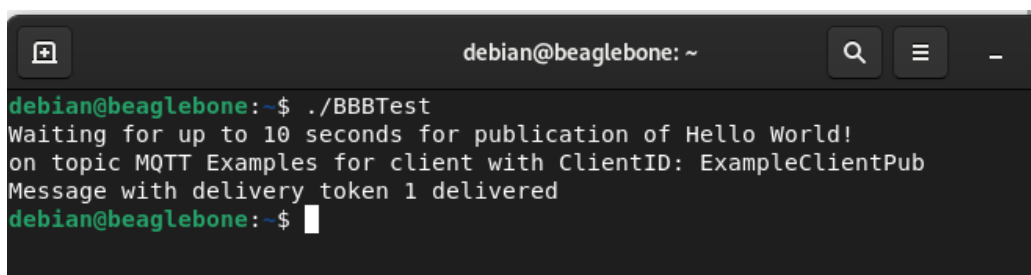
    conn_opts.keepAliveInterval = 20;
    conn_opts.cleansession = 1;
    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS)
    {
        printf("Failed to connect, return code %d\n", rc);
        exit(EXIT_FAILURE);
    }

    pubmsg.payload = PAYLOAD;
    pubmsg.payloadlen = (int)strlen(PAYLOAD);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    if ((rc = MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token)) != MQTTCLIENT_SUCCESS)
    {
        printf("Failed to publish message, return code %d\n", rc);
        exit(EXIT_FAILURE);
    }

    printf("Waiting for up to %d seconds for publication of %s\n"
           "on topic %s for client with ClientID: %s\n",
           (int)(TIMEOUT/1000), PAYLOAD, TOPIC, CLIENTID);
    rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
    printf("Message with delivery token %d delivered\n", token);

    if ((rc = MQTTClient_disconnect(client, 10000)) != MQTTCLIENT_SUCCESS)
        printf("Failed to disconnect, return code %d\n", rc);
    MQTTClient_destroy(&client);
    return rc;
}
```

Voor het uitvoeren hebben we ook nog een client nodig om met de broker te verbinden en de data te kunnen controleren. Ik maak gebruik van mqtt.fx. Als we deze code uitvoeren krijgen we de volgende output in de terminal.



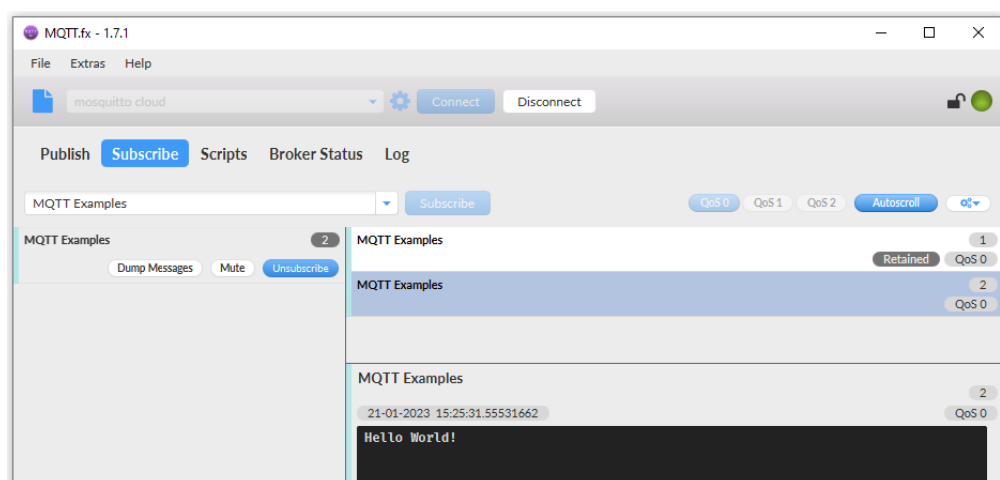
```

debian@beaglebone: ~
debian@beaglebone:~$ ./BBBTest
Waiting for up to 10 seconds for publication of Hello World!
on topic MQTT Examples for client with ClientID: ExampleClientPub
Message with delivery token 1 delivered
debian@beaglebone:~$

```

Figuur 40 Output paho test

Aan deze output kunnen we zien dat het publiceren van data succesvol is verlopen. Nu gaan we enkel nog de client raadplegen om te zien of die data ook wel correct is. In onderstaande afbeelding kan je zien dat de data "Hello World" is. Nu kunnen we data versturen en controleren.



Figuur 41 Client paho test

6.7 Prototype

[Link](#) naar Github.

Na al deze tests hebben we eigenlijk zo goed als al de basis om een prototype te bouwen. In dit prototype gaan het register "Actual voltage" van de energie meter uitlezen. Dan gaan we het Modbus frame filteren zodat enkel nog de data overblijft, die gaan we dan proberen te versturen met MQTT.

6.7.1 Data filtering

Bij het monitoren van de energie meter hebben we niet het hele Modbus frame nodig. We gaan het frame dus moeten filteren zodat er alleen de gevraagde data nog overblijft. In de [analyse](#) van de Modbus data hebben we gezien welke bytes we juist nodig hebben en hoe we ze moeten converteren totdat we de exacte data hebben. Hiervoor heb ik twee dingen gedaan:

- Functie **ReadAddress** aangepast.
- Functie **ConvertToRegisterUnits** aangemaakt.

6.7.1.1 ReadAddress

De functie had oorspronkelijk als doel om een frame te sturen en het ontvangen frame in te lezen. Om deze data te filteren heb ik een nieuwe variabele aangemaakt

```

143     // filtering the data that we need and combine it.
144     int FilteredData = ((RawRxData[5]*256)+RawRxData[6]);
145
146     return FilteredData;
147 }
```

Figuur 42 Filtered data

RawRxData is de array waar het uitgelezen frame in staat we weten dat we byte 6 en 7 nodig hebben en we gaan dan deze twee bytes samenvoegen. In de analyse had ik de berekening hiervoor al uitgelegd. De data is gefilterd en we geven deze terug.

6.7.1.2 ConvertToRegisterUnits

In de register units zijn er twee verschillen in het converteren van de data:

- Voltage, alleen hierbij moet het getal door 10 gedeeld worden.
- Bij al de rest moet het getal gedeeld worden door 100.

Voor dit te doen heb ik een aparte functie aangemaakt (Modbus_Circutor_CVM_1D_Registers.c), deze functie moet als parameters de data hebben en het type dat we willen converteren, verder geeft de functie de geconverteerde data terug.

```
23 float ConvertToRegisterUnits(int data, int type)
24 {
25     float ConvertedData;
26     if (type == VOLTAGE)
27     {
28         ConvertedData = (float) data;
29         ConvertedData = (ConvertedData/10);
30     }
31
32     else
33     {
34         ConvertedData = (float) data;
35         ConvertedData = (ConvertedData/100);
36     }
37
38     return ConvertedData;
39 }
```

Figuur 43 ConvertToRegisterUnits

6.7.2 MQTT Topic & Payload

We hebben nu de data gefilterd en we kunnen de data ook omvormen naar de eenheden die we nodig hebben. De volgende stap is het klaar maken van de Topics en de payload. We starten eerst met het topic, deze ziet er als volgt uit:

"ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualCurrent"

Het bestaat uit twee delen, het eerste deel is het vaste deel dat niet verandert:

"ProjectenVoorHetWerkveld/CircutorEnergyMeter"

Het tweede deel moet verandere naargelang welk register we uitlezen, dat kan het voltage zijn of de stroom...

6.7.2.1 Struct

Ik ben gestart met het aanmaken van een struct. Dit zorgt ervoor dat de variabele die ik hier voor ga gebruiken gebundeld zijn en verduidelijkt.

```
55 struct mqtt
56 {
57     unsigned char Topic[100];
58     unsigned char TopicPartOne[45];
59     unsigned char TopicPartTwo[50];
60     float Payload;
61     int SizeTopicPartOne;
62     int SizeRegisterArray;
63 };
--
```

Figuur 44 Struct mqtt

- **Topic:** Dit is het samengevoegde topic dat wordt verzonden.
- **TopicPartOne:** Dit is het vaste deel.
- **TopicPartTwo:** Dit is het variabele deel.
- **Payload:** Dit is de data die we gaan verzenden.

6.7.2.2 ReadModbus_CircutorData_MqttTopicPayload

Voor het topic maken voor het register dat wordt gevraagd heb ik ook een aparte functie aangemaakt.

```
49 void ReadModbus_CircutorData_MqttTopicPayload(int type, int fd, struct mqtt *data)
```

De functie verwacht 3 parameters:

- **Type:** Dit geeft aan welk register we willen uitlezen, hierdoor kan zowel de register en de register benaming worden opgeroepen.
- **Fd:** file descriptor
- ***data:** de pointer naar de struct mqtt.

In deze functie wordt de functie `ReadAddress` opgeroepen om de data te ontvangen van het type dat is doorgegeven, daarna gaan we de functie `ConvertToRegisterUnits` oproepen met hetzelfde type en wordt de geconverteerde data in de variabele `Payload` van de struct `mqtt` gestoken.

```
58
59     ReceivedData = ReadAddress(fd, Circutor_CVM_1D_Address, Registers_Circutor_CVM_1D[type], 0);
60     data->Payload = ConvertToRegisterUnits(ReceivedData, type);
61
```

Figuur 45 Read data & convert

Het volgende dat we gaan doen is de register benaming van het type dat was doorgegeven kopiëren naar `TopicPartTwo`. Daarna gaan we met twee for loops topic part one en two samenvoegen in een lokale array om die daarna te kunnen kopiëren naar de Topic in de struct.

```
62     strcpy(data->TopicPartTwo, RegisterNames_Circutor_CVM_1D[type]);
63
64     for (i = 0; i < data->SizeTopicPartOne; i++)
65     {
66         Topic[i] = data->TopicPartOne[i];
67     }
68     for (i = 0; i < data->SizeRegisterArray; i++)
69     {
70         Topic[data->SizeTopicPartOne + i] = data->TopicPartTwo[i];
71     }
72
73     strcpy(data->Topic, Topic);
74
75
76 }
```

Figuur 46 Combine topics

Als we deze functie aanroepen moeten we de functie het type register dat we willen uitlezen meegeven, hiervoor heb ik globale defines aangemaakt in de header file.

```
50 //defines for reading the energy meter
51 #define VOLTAGE 0
52 #define CURRENT 3
53 #define ACTIVEPOWER 6
54 #define ACTIVEENERGIE 27
```

Figuur 47 Defines energy meter

6.7.3 Main

In de main file gaan we het paho voorbeeld gebruiken. We gaan hier een paar dingen aan toevoegen. We beginnen met de eigen struct voor mqtt toe te voegen en het vaste gedeelte van het topic in `TopicPartOne` te kopiëren. Verder maken we nog de variabele aan voor de file descriptor en de locatie van het bestand van de usb.

```
37 int main (int argc, char *argv[])
38 {
39     // In this struct we put all the modbus data that we want to send through mqtt.
40     struct mqtt mqttdata;
41     // Topic part one we will always use in every topic because it is a part of the energy meter.
42     strcpy(mqttdata.TopicPartOne, "ProjectenVoorHetWerkveld/CircutorEnergyMeter/");
43     // Used for the serial connection.
44     int fd;
45     char serialdev[] = "/dev/ttyUSB0";
46 }
```

Figuur 48 Main part one prototype

Dan heb ik bij het if-statement bij het connecteren naar de broker een else toegevoegd om te printen dat we zijn geconnecteerd met de broker. Daarna ga ik de seriële poort openen en de energie meter uitlezen. Nu heb ik als voorbeeld de amperages genomen. Daarna gaan we de poort terug sluiten.

```

73 // Connecting to the mqtt broker.
74 if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS)
75 {
76     printf("Failed to connect, return code %d\n", rc);
77     exit(EXIT_FAILURE);
78 }
79 else
80 {
81     printf("MQTT Client connected to the Mosquitto broker.\n\n");
82 }
83
84 // Open the serial port, read the modbus data and close the serial port.
85 fd = openserial(serialdev);
86
87 if (!fd)
88 {
89     fprintf(stderr, "Error while initializing %s.\n", serialdev);
90 }
91
92
93 ReadModbus_CircutorData_MqttTopicPayload(CURRENT, fd, &mqttdata);
94 closeserial(fd);
95

```

Figuur 49 Main part two prototype

Daarna gaan we struct variabele "pubmsg.payload" van de paho library geheugen toewijzen (regel 97), dit heb ik gedaan omdat ik anders een error krijg. Bij het uitlezen van de energie meter is de data al volledig geconverteerd en ook omgevormd naar het type dat we hebben opgevraagd. Die data staat nu in de struct variabele "mqttdata.Payload". Die data moeten we nog kopiëren naar de struct variabele "pubmsg.payload" van de paho library (regel 98).

```

96 // Publish message
97 pubmsg.payload = (char *) malloc(20);
98 sprintf(pubmsg.payload, "%f", mqttdata.Payload);
99 pubmsg.payloadlen = (int)strlen(pubmsg.payload);
100 pubmsg.qos = QOS;
101 pubmsg.retained = 0;

```

Figuur 50 Main part three paho struct

Als laatste heb ik nog bij de functie "MQTTClient_publishMessage" de struct variabele "mqttdata.Topic" gegeven (regel 102). Hetzelfde heb ik nog gedaan bij het printen van het proces (regel 112).

```

102     if ((rc = MQTTClient_publishMessage(client, mqttdata.Topic, &pubmsg, &token)) != MQTTCLIENT_SUCCESS)
103     {
104         printf("Failed to publish message, return code %d\n", rc);
105         exit(EXIT_FAILURE);
106     }
107     else
108     {
109         printf("Message published.\n");
110     }
111
112     printf("Waiting for up to %d seconds for publication of %s\n"
113           "on topic %s for client with ClientID: %s\n",
114           (int)(TIMEOUT/1000), pubmsg.payload, mqttdata.Topic, CLIENTID);
115
116     rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
117     printf("Message with delivery token %d delivered.\n\n", token);
118
119
120     // Disconnect the client from the broker.
121     if ((rc = MQTTClient_disconnect(client, 10000)) != MQTTCLIENT_SUCCESS)
122     {
123         printf("Failed to disconnect, return code %d\n", rc);
124     }
125     MQTTClient_destroy(&client);
126     return rc;

```

Figuur 51 Main part four sending data

Als we deze code runnen dan krijgen we in de terminal de volgende output.



```

debian@beaglebone: ~
debian@beaglebone:~$ ./BBBTest
MQTT Client created.

MQTT Client connected to the Mosquitto broker.

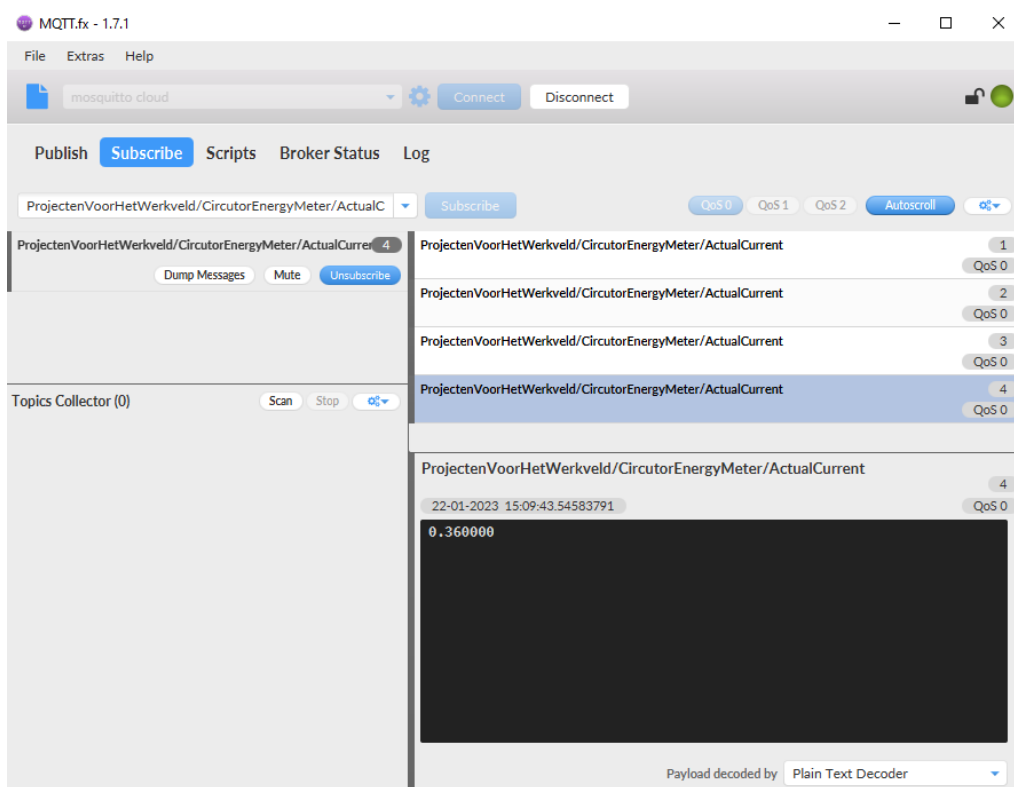
Message published.
Waiting for up to 10 seconds for publication of 0.360000
on topic ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualCurrent
for client with ClientID: Circutor
Message with delivery token 1 delivered.

debian@beaglebone:~$

```

Figuur 52 Output prototype

Als laatste gaan we de data nog controleren in de mqtt client.



Figuur 53 MQTT client prototype

Dit is nu een werkend prototype, dit is de start naar de final code. Alles wat we hierin hebben gedaan ga ik gebruiken mits een paar aanpassingen.

7 FINAL CODE!

[Link](#) naar Github.

Met een werkend prototype kunnen we beginnen werken aan de final code. Het belangrijkste is om te weten wat we juist willen bereiken met de final code. Voor mij was het volgende belangrijk:

- Het programma moet blijven draaien totdat ik het afbreek.
- Ik wil vier zaken verzenden:
 - Spanning
 - Stroom
 - Elektrisch vermogen
 - Totale verbruik van de meter
- Ik wil elk onderdeel om de twee seconden versturen.

Om het programma te laten draaien heb ik een **while(1)** loop gebruikt. Het programma zal enkel uit de loop geraken als het wordt afgebroken, handmatig of door een fout. Om de vier verschillende zaken te verzenden ga ik het proces om de energie meter uit te lezen tot het verzenden van de message vier keer opnieuw uitvoeren. Enkel het type bij het uitlezen zal veranderen, al de rest blijft zich herhalen.

Omdat alles zich blijft herhalen heb ik voor het versturen van de message een aparte functie aangemaakt "void MQTT_SendMessage(struct mqtt *data);". Deze functie zal de data versturen, we moeten de functie enkel de struct waar het topic en de payload instaat geven. Voor deze functie heb ik ook een nieuwe file aangemaakt "MQTT_Custom.c". hier heb ik ook nog een functie ingebouwd om de MQTT client aan te maken en om te connecteren met de broker "void MQTT_init()". In de while loop heb ik nog een kleine statemachine ingebouwd met een switch statement. Dit doe ik zodat ik kan wisselen tussen de zaken die ik wil verzenden. Hiervoor gebruik in de define's die ik in de tests al aangemaakt had, te zien in **Figuur 47 Defines energy meter**. Als laatst gebruik ik de **sleep(2)** functie om twee seconden te wachten.

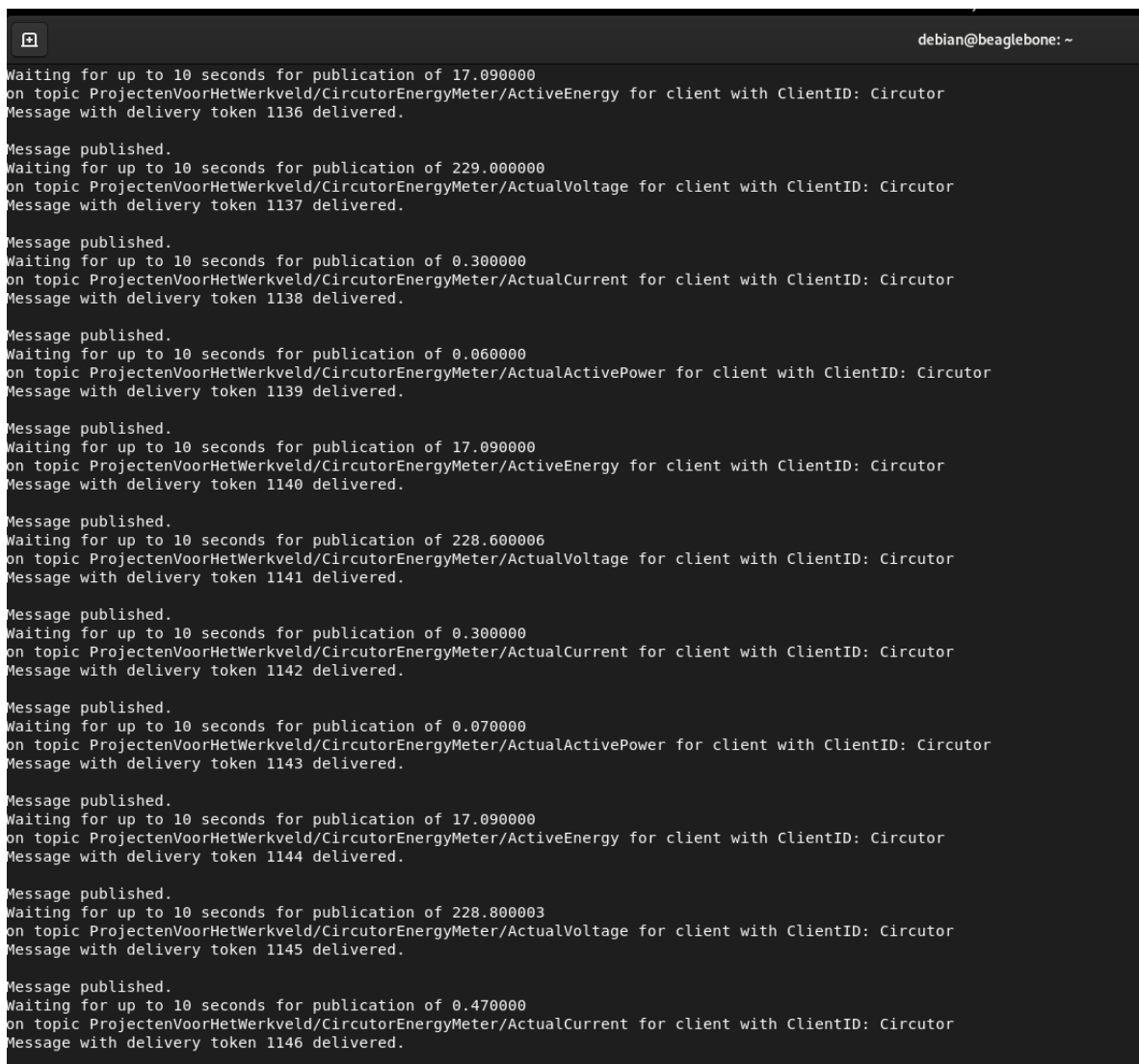
```

44     while(1)
45     {
46         // Open the serial port, read the modbus data and close the serial port.
47         fd = openserial(serialdev);
48
49         if (!fd)
50         {
51             fprintf(stderr, "Error while initializing %s.\n", serialdev);
52         }
53
54         switch(unit_state)
55         {
56             case VOLTAGE:
57                 Readmodbus_CircutorData_MqttTopicPayload(VOLTAGE, fd, &mqttdata);
58                 MQTT_SendMessage(&mqttdata);
59                 unit_state = CURRENT;
60                 sleep(2);
61                 break;
62
63             case CURRENT:
64                 Readmodbus_CircutorData_MqttTopicPayload(CURRENT, fd, &mqttdata);
65                 MQTT_SendMessage(&mqttdata);
66                 unit_state = ACTIVEPOWER;
67                 sleep(2);
68                 break;
69
70             case ACTIVEPOWER:
71                 Readmodbus_CircutorData_MqttTopicPayload(ACTIVEPOWER, fd, &mqttdata);
72                 MQTT_SendMessage(&mqttdata);
73                 unit_state = ACTIVEENERGIE;
74                 sleep(2);
75                 break;
76
77             case ACTIVEENERGIE:
78                 Readmodbus_CircutorData_MqttTopicPayload(ACTIVEENERGIE, fd, &mqttdata);
79                 MQTT_SendMessage(&mqttdata);
80                 unit_state = VOLTAGE;
81                 sleep(2);
82                 break;
83
84         }
85
86         closeserial(fd);
87
88     }
89 }

```

Figuur 54 switch-state final code

Als we het programma laten draaien geeft de terminal ook de output van wat er allemaal gebeurt.

A terminal window with a dark background and light text. The title bar at the top right says 'debian@beaglebone: ~'. The terminal output shows a repeating pattern of waiting for a message to be published, publishing it to a specific MQTT topic, and receiving a delivery token. The topics are 'ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActiveEnergy', 'ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualVoltage', and 'ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualCurrent'. The ClientID is 'Circutor'. The delivery tokens range from 1136 to 1146.

```
Waiting for up to 10 seconds for publication of 17.090000
on topic ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActiveEnergy for client with ClientID: Circutor
Message with delivery token 1136 delivered.

Message published.
Waiting for up to 10 seconds for publication of 229.000000
on topic ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualVoltage for client with ClientID: Circutor
Message with delivery token 1137 delivered.

Message published.
Waiting for up to 10 seconds for publication of 0.300000
on topic ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualCurrent for client with ClientID: Circutor
Message with delivery token 1138 delivered.

Message published.
Waiting for up to 10 seconds for publication of 0.060000
on topic ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualActivePower for client with ClientID: Circutor
Message with delivery token 1139 delivered.

Message published.
Waiting for up to 10 seconds for publication of 17.090000
on topic ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActiveEnergy for client with ClientID: Circutor
Message with delivery token 1140 delivered.

Message published.
Waiting for up to 10 seconds for publication of 228.600006
on topic ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualVoltage for client with ClientID: Circutor
Message with delivery token 1141 delivered.

Message published.
Waiting for up to 10 seconds for publication of 0.300000
on topic ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualCurrent for client with ClientID: Circutor
Message with delivery token 1142 delivered.

Message published.
Waiting for up to 10 seconds for publication of 0.070000
on topic ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualActivePower for client with ClientID: Circutor
Message with delivery token 1143 delivered.

Message published.
Waiting for up to 10 seconds for publication of 17.090000
on topic ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActiveEnergy for client with ClientID: Circutor
Message with delivery token 1144 delivered.

Message published.
Waiting for up to 10 seconds for publication of 228.800003
on topic ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualVoltage for client with ClientID: Circutor
Message with delivery token 1145 delivered.

Message published.
Waiting for up to 10 seconds for publication of 0.470000
on topic ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualCurrent for client with ClientID: Circutor
Message with delivery token 1146 delivered.
```

Figuur 55 Output final code

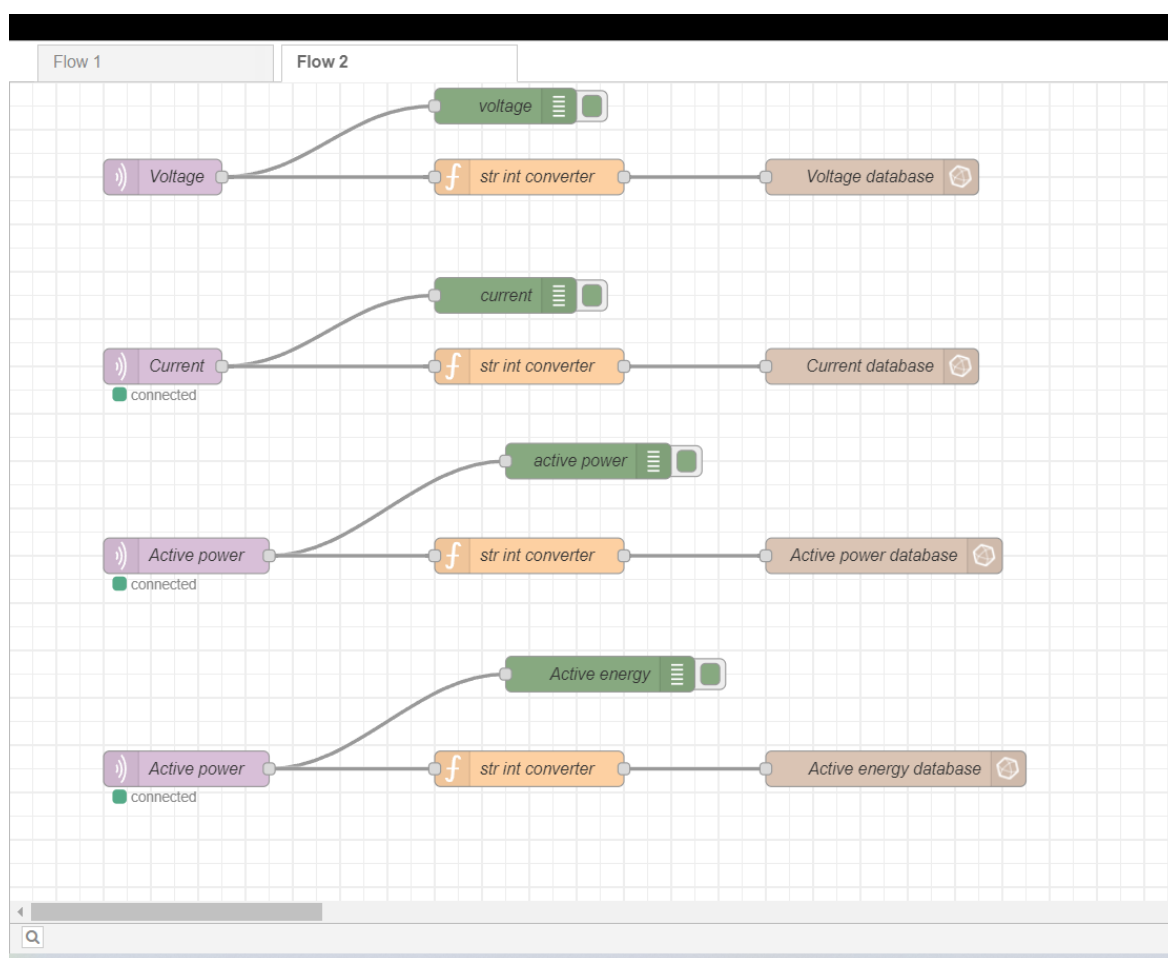
8 DATA MONITORING

Het programma om de data te verzenden naar de Cloud is klaar. Als het programma wordt uitgevoerd zal het de data blijven sturen, nu moeten we die data nog gaan opvangen en monitoren. Zoals ik eerder al heb uitgelegd in de [omschrijving](#) gaan we [home assistant](#) hiervoor gebruiken. We hebben hier drie zaken voor nodig:

- [Node-RED](#), voor de koppeling tussen Cloud en database.
- [InfluxDB](#), de database voor het stockeren van de data.
- [Grafana](#), voor het visualiseren van de data.

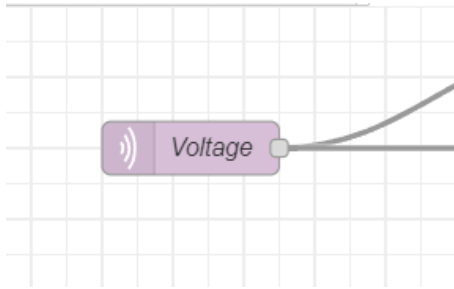
8.1 Node-RED koppeling

Voor elk onderdeel heb ik een blok gemaakt waar we via een network node "mqtt-in" verbinding gaan maken met de Cloud broker en gaan subscriben op het topic dat we voor het onderdeel nodig hebben. Het volgende dat ik gebruikt heb is een string to int converter, de data dat in het programma wordt verstuurd is een string maar we willen een integer in de database. Als laatste heb ik de verbinding gemaakt met de database, dat gebeurt door een storage node "influxdb out".



Figuur 56 Node-RED nodes

8.1.1 Network node “mqtt in”



Figuur 57 Network node mqtt-in

Met deze node kan een verbinding worden gemaakt met de broker en we kunnen publishen of subscriben op een topic. Als we deze node hebben toegevoegd moeten we eerst de connectie maken met de broker. In onderstaande afbeelding zijn de instellingen hiervoor zichtbaar. In het vak server geven we het adres en de poort van de broker in. We gaan automatisch connecteren en we hebben de versie ingesteld.

Edit mqtt in node > Edit mqtt-broker node

Delete Cancel Update

Properties

Name Mosquitto cloud

Connection Security Messages

Server test.mosquitto.org Port 1883

☒ Connect automatically

☐ Use TLS

Protocol MQTT V3.1.1

Client ID Leave blank for auto generated

Keep Alive 60

Session ☒ Use clean session

Enabled 4 nodes use this config On all flows

Figuur 58 mqtt node server settings

Na dit gedaan te hebben kunnen we de server selecteren waar we mee willen verbinden. Het topic waar we op willen subscriben geven we in het vak topic in. Meer is er niet nodig om verbinding te maken en de data op te vangen. De laatste stap gaan we voor elk onderdeel herhalen maar we veranderen telkens het laatste deel van het topic met:

- /ActualVoltage
- /ActualCurrent
- /ActualPower
- /ActiveEnergy

Edit mqtt in node

Delete Cancel Done

Properties

Server Mosquitto cloud

Action Subscribe to single topic

Topic ProjectenVoorHetWerkveld/CircutorEnergyMeter/ActualVoltage

QoS 0

Output auto-detect (string or buffer)

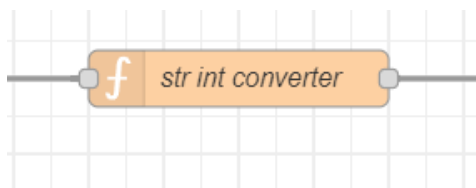
This option is depreciated. Please use the new auto-detect mode.

Name Voltage

Enabled

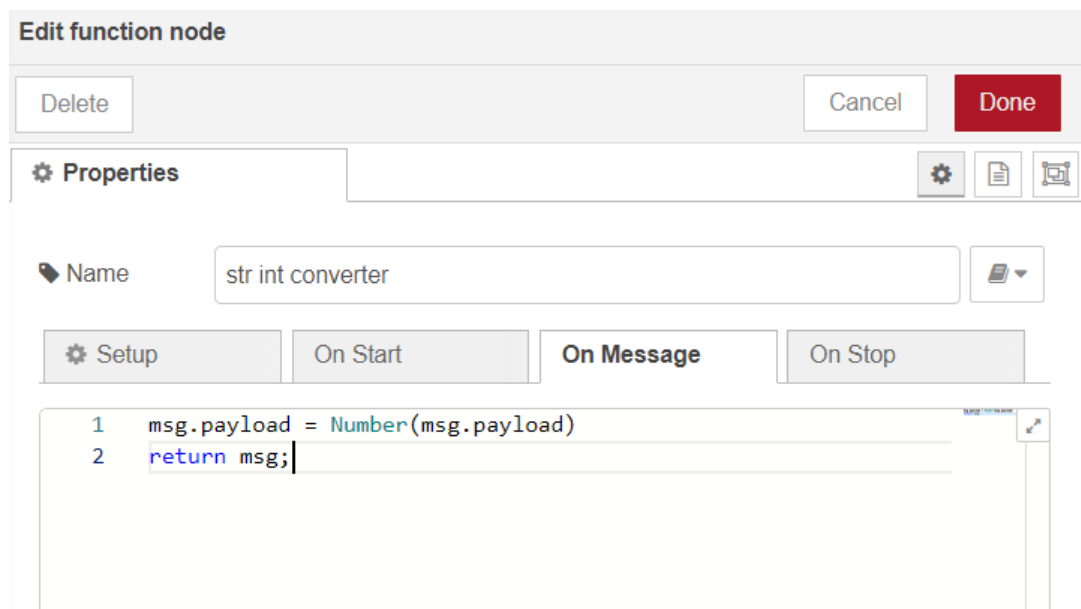
Figuur 59 mqtt node subscribe to topic

8.1.2 Function node



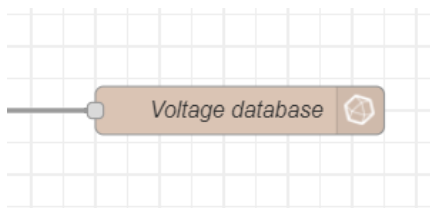
Figuur 60 Function node

Voor het converteren van string naar int gaan we gebruik maken van een function node. De function node laat het toe dat JavaScript code kan worden uitgevoerd op de "messages" die door de node gaan. De message is hier een object "msg". Om de inhoud hiervan te krijgen gebruiken we de regel "msg.payload". Het enige wat we hier mee gaan doen is een type cast naar int (`msg.payload = Number(msg.payload)`). Daarna gaan we het object terug sturen met `return`. Dit is alles wat we nodig hebben om de string te converteren.



Figuur 61 function node str to int

8.1.3 Storage node influxdb out



Figuur 62 Storage node influxdb out

Deze node zorgt ervoor dat de binnenkomende data in de database wordt gezet. Voor we hier gebruik van kunnen maken moet er eerst een database worden aangemaakt, hoe dit verloopt leg ik nog uit in InfluxDB. We hebben vier databases aangemaakt, één voor de voltage, één voor de stroom, één voor het vermogen en één voor het totale verbruik.

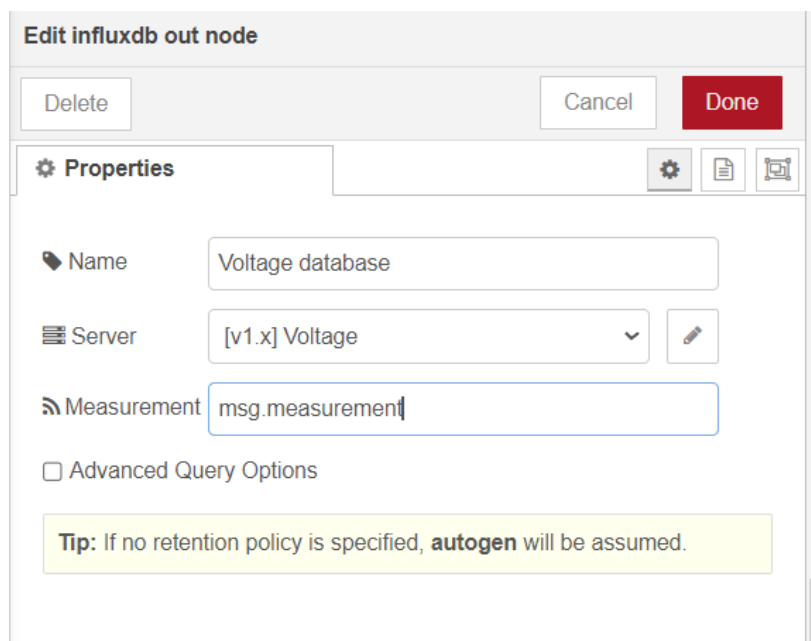
Bij de instellingen van de node gaan we bij Host "localhost" ingeven, de database draait op het zelfde systeem, de poort is 8086. Daarna moeten we de database ingeven en onze inlog gegevens. Deze stap moeten we voor elke database herhalen.

 A screenshot of the 'Edit influxdb out node' settings form. The form has a title bar 'Edit influxdb out node > Edit influxdb node' and buttons for 'Delete', 'Cancel', and 'Update'. Below the title bar is a 'Properties' section with a gear icon and a document icon. The form contains the following fields:

- Name:** Voltage
- Version:** 1.x (dropdown menu)
- Host:** localhost
- Port:** 8086
- Database:** Voltage
- Username:** Bart
- Password:** (masked with dots)
- Enable secure (SSL/TLS) connection:** (checkbox, unchecked)

Figuur 63 Storage node influxdb out db settings

Als we alle databases hebben geconfigureerd is de laatste stap om elke node toe te wijzen aan die database. Eerst moeten we de node een benaming geven, daarna moeten we de database toewijzen en als laatste moeten we de message geven. In de afbeelding hieronder het voorbeeld ervan.



Edit influxdb out node

Delete Cancel Done

Properties

Name Voltage database

Server [v1.x] Voltage

Measurement msg.measurement

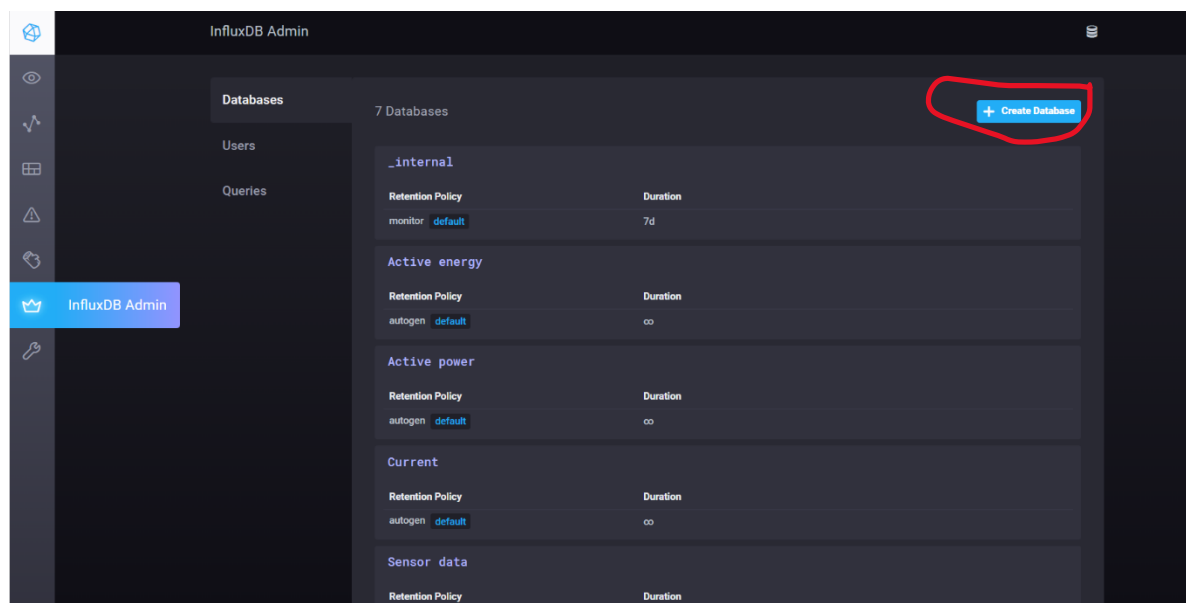
☐ Advanced Query Options

Tip: If no retention policy is specified, **autogen** will be assumed.

Figuur 64 Storage node influxdb out settings

8.2 Influxdb

Voor elk topic hebben we een aparte database nodig. Een database in influxdb is heel gemakkelijk aan te maken. Daarvoor moeten we naar het tabblad InfluxDB Admin gaan. Dan op Create Database klikken, de benaming ingeven en de database is aangemaakt. De database is dan ook direct bruikbaar.

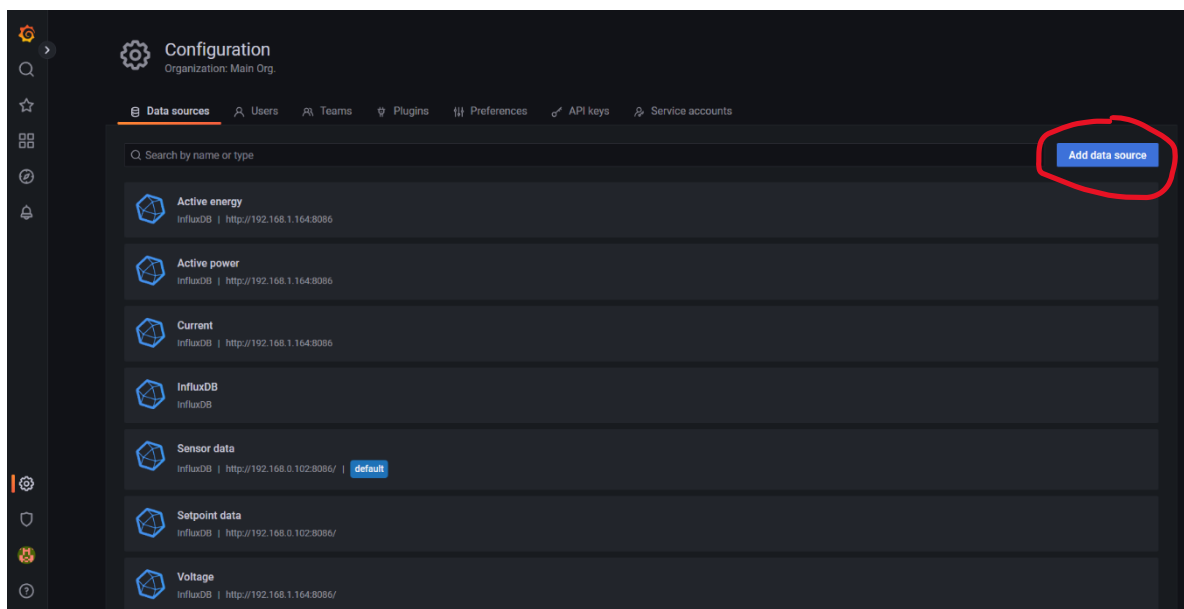


Figuur 65 InfluxDB create DB

8.3 Grafana

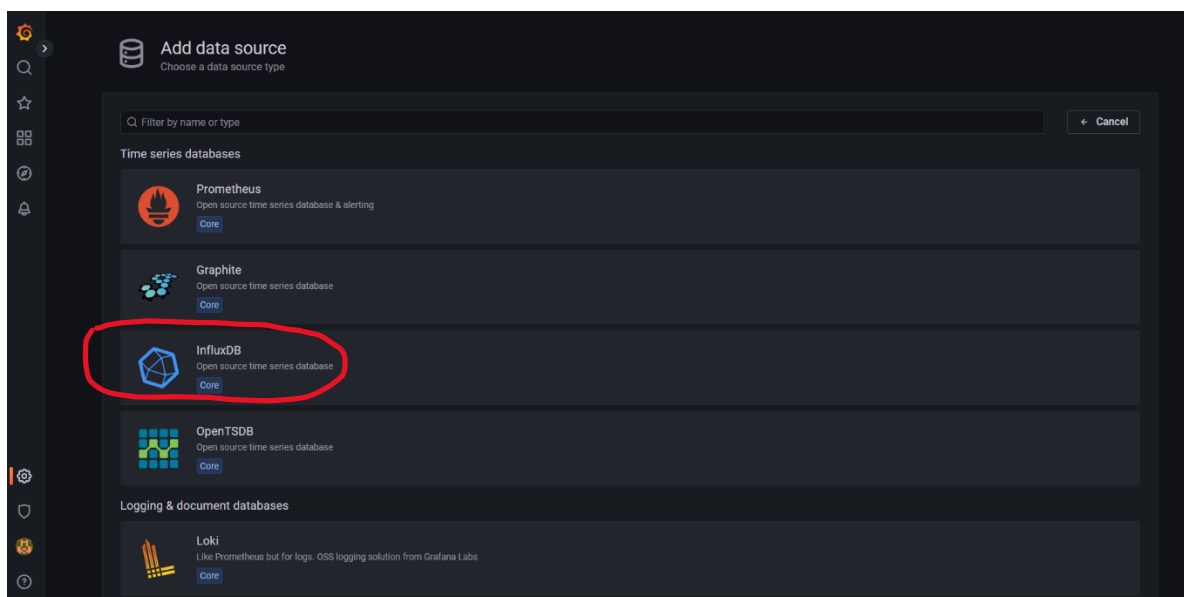
Grafana is iets ingewikkelder om te configureren. Hier moeten we eerst een InfluxDB data source configureren voor elke database. Daarna moeten we de resources toevoegen aan een dashboard en zo instellen zodat we de gewenste visualisatie hebben.

Om te beginnen moeten naar het tabblad configuration gaan. En klikken op Add data source.



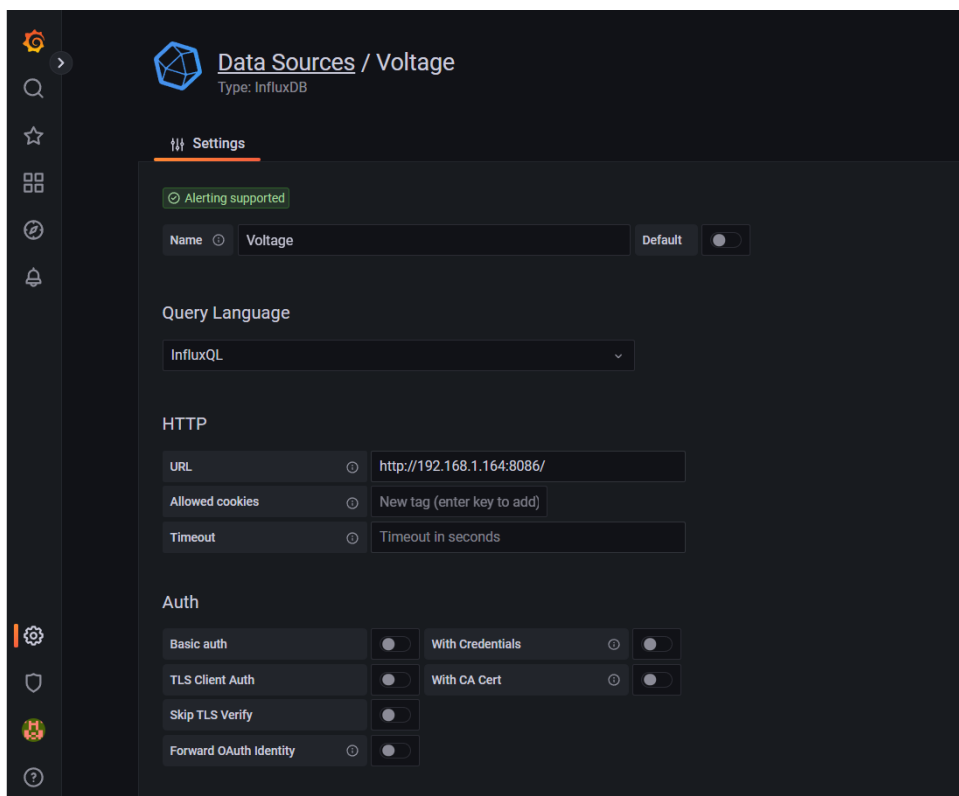
Figuur 66 Grafana add source

Daarna selecteren we InfluxDB.



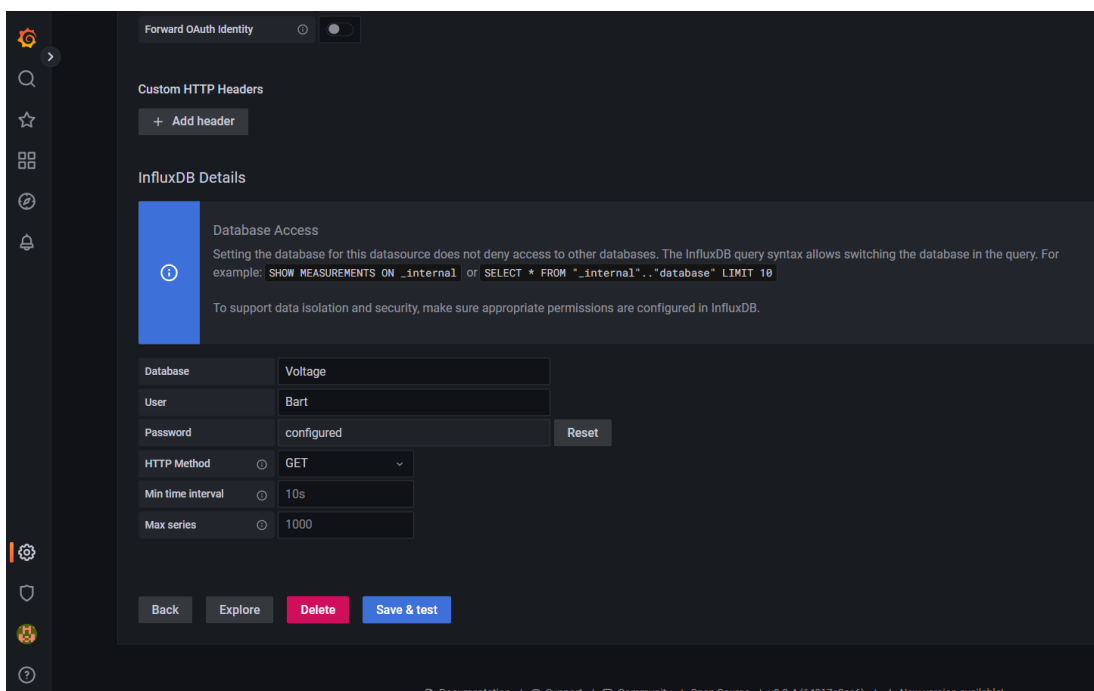
Figuur 67 Grafana select data source

Als volgende moeten we de naam ingeven en het adres van de InfluxDB server.



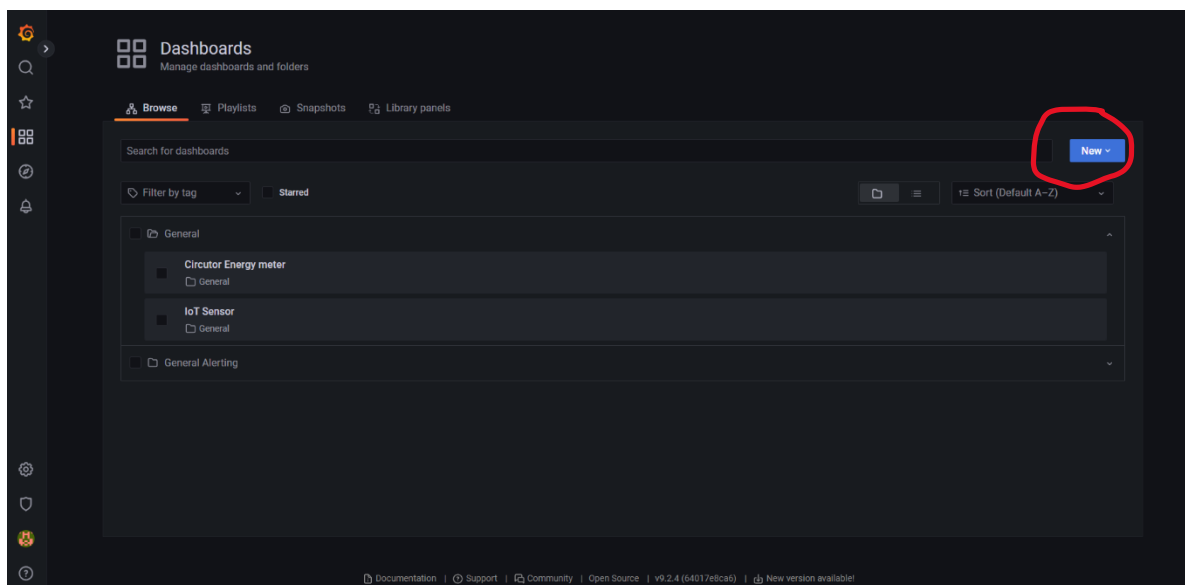
Figuur 68 Grafana data source config 1

Daarna moeten we de database naam ingeven, en gebruiker gegevens. Bij HTTP Method geven we GET in. Dan drukken op Save & Test.



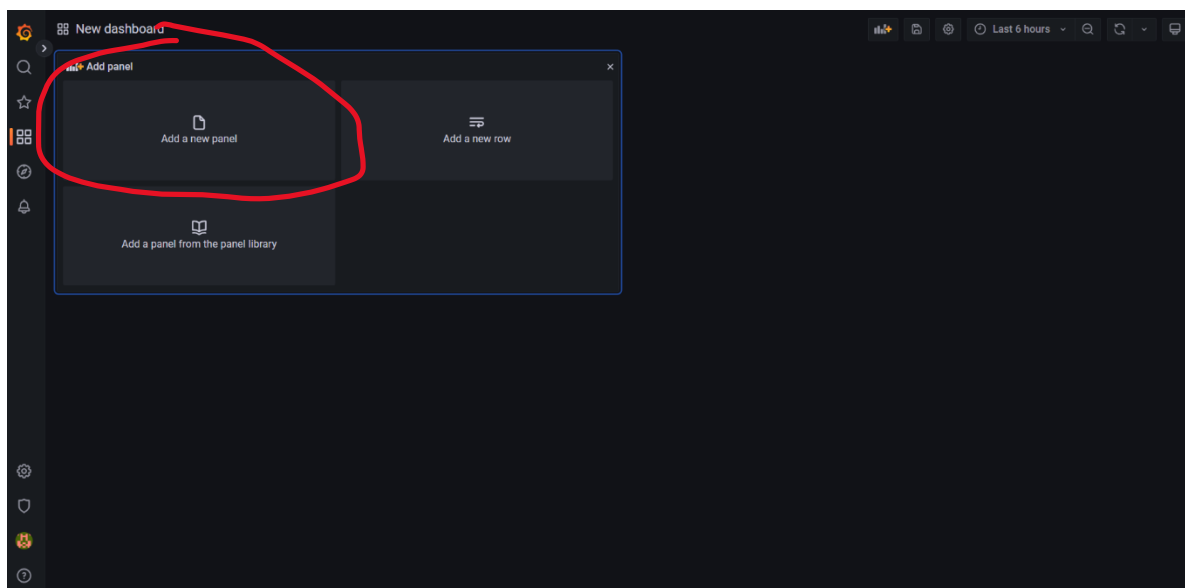
Figuur 69 Grafana data source config 2

De data sources zijn nu succesvol toegevoegd. De laatste stap is een dashboard aanmaken en de data sources daarin toevoegen. Daarvoor moeten we naar het tabblad dashboard gaan en op de knop New klikken.



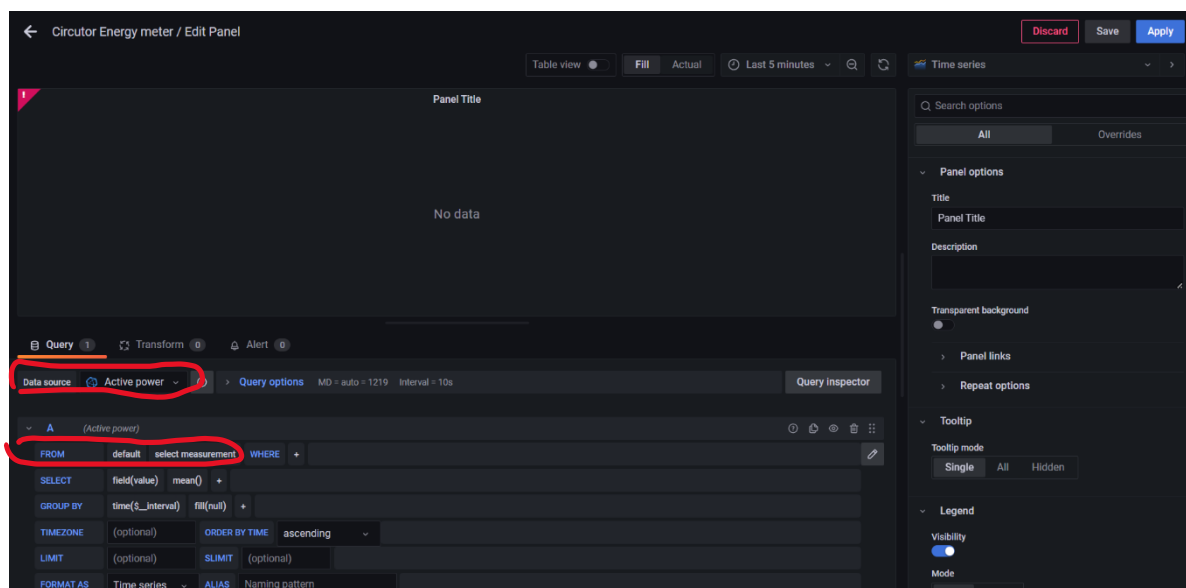
Figuur 70 Grafana add new dashboard

Het volgende is een nieuw panel toevoegen.



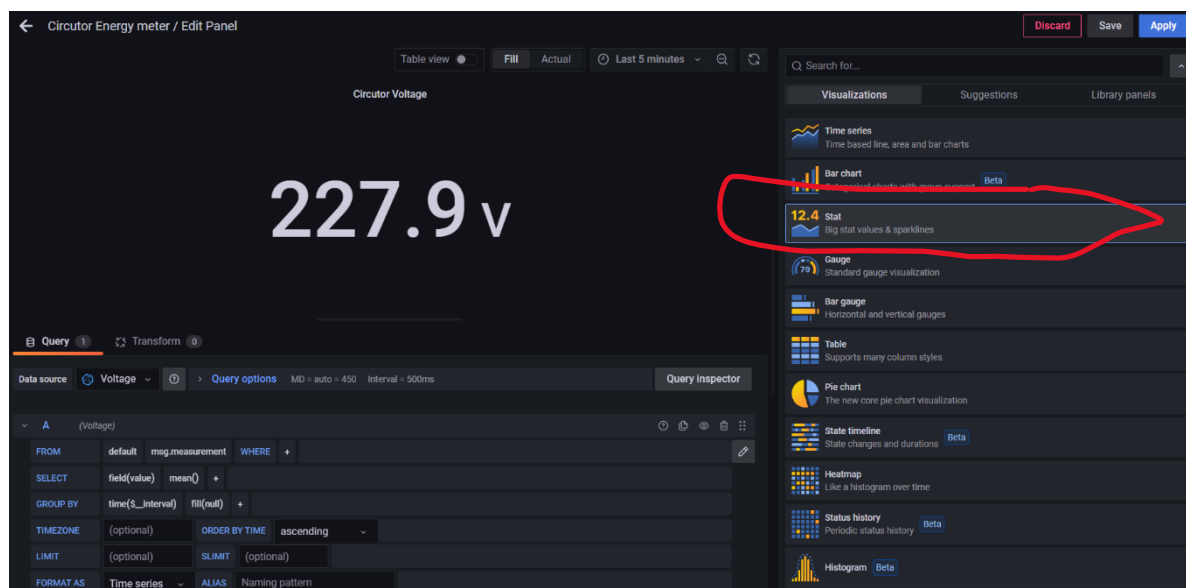
Figuur 71 Grafana add new panel

Dan moeten we bij Data source de gewenste database selecteren. Bij FROM moeten we bij select measurement "msg.measurement" selecteren.



Figuur 72 Grafana edit panel

Daarna kunnen we het type visualisatie selecteren rechts vanboven. Ik heb gekozen voor Stats.



Figuur 73 Grafana edit panel stats

Daarna geven we bij Title de benaming in. En dan heb ik de optie Transparent background geslecteerd, dit oogt mooi. Bij Value options heb ik gekozen voor "Calculate", bij Calculation kies ik voor "last" en bij Fields voor "Numeric Fields"

12.4 Stat

Search options

All Overrides

Title
Circutor Voltage

Description

Transparent background
☒

> Panel links

> Repeat options

Value options

Show
Calculate a single value per column or series or show each row
Calculate All values

Calculation
Choose a reducer function / calculation
Last * x v

Fields
Select the fields that should be included in the panel
Numeric Fields v

Figuur 74 Grafana stats edit 1

En als laatste kunnen we bij standard options de gewenst unit selecteren, in het voorbeeld is dat Volt(V), Decimals 1. Dit is bij elk panel anders.

12.4 Stat

Search options

All Overrides

Standard options

Unit
Volt (V) v

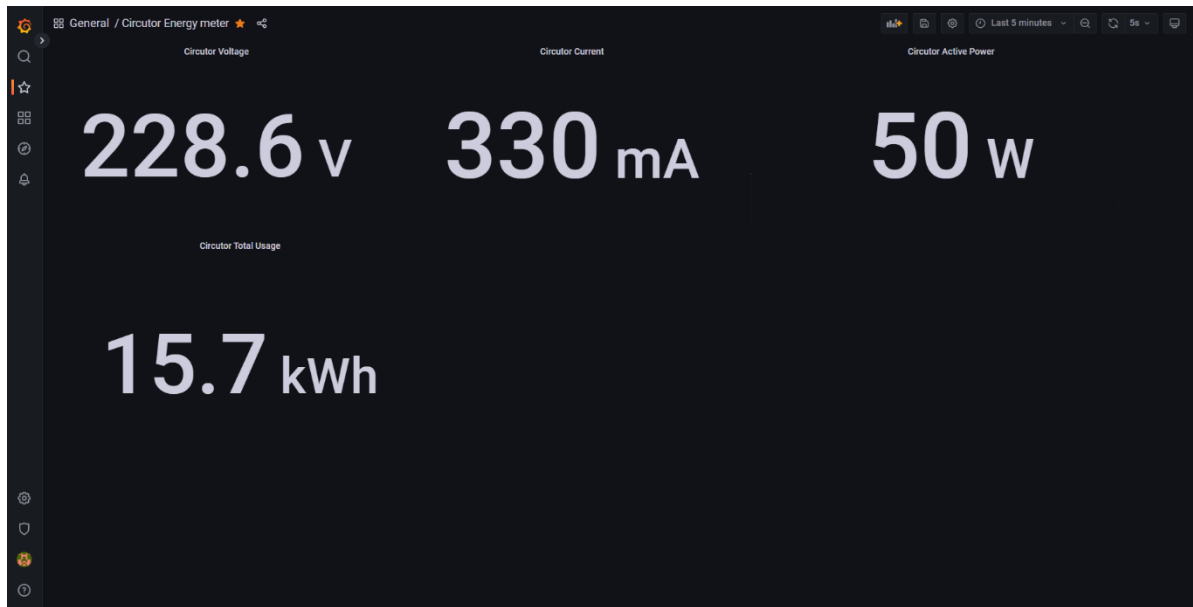
Min
Leave empty to calculate based on all values
auto

Max
Leave empty to calculate based on all values
auto

Decimals
1

Figuur 75 Grafana stats edit 2

Na dit gedaan te hebben voor elk item zijn we klaar met het dashboard. Dit was ook het laatste onderdeel van het project. In de afbeelding hieronder het eindresultaat.



Figuur 76 Grafana