|  |  |
| --- | --- |
|  | |
| Projecten van het werkveld  STM32 MCU met FreeRTOS & LwIP | |
| Gysen Bart | Graduaat in Internet of Things |
|  |
| Academiejaar 2022-2023  Campus : Kempen |



Voorwoord

Met trots presenteer ik u dit verslag dat de resultaten en ervaringen weergeeft van mijn project. Ik vond het belangrijk om met een voor mij nieuw soort microcontroller te werken namelijk een STM32. Dit project bood me de gelegenheid om mijn kennis verder uit te breiden en mij voor te bereiden op het professionele leven.

Als student ben ik altijd gedreven geweest om nieuwe mogelijkheden te verkennen en uitdagingen buiten mijn comfortzone aan te gaan. Daarom koos ik ervoor om te werken met een microcontroller die ik nog niet eerder had gebruikt. Ik geloofde dat dit mijn begrip van embedded systemen en elektronica in het algemeen zou vergroten.

Gedurende dit project ben ik geconfronteerd met complexe problemen en onverwachte obstakels. Het kostte tijd en moeite om vertrouwd te raken met de microcontroller, zijn configuratie tool en drivers. Maar dankzij mijn vastberadenheid en doorzettingsvermogen kon ik deze obstakels overwinnen en mijn kennis enorm vergroten.

Dit project bood me niet alleen technische kennis, maar ook persoonlijke groei. Ik ontwikkelde mijn probleemoplossende vaardigheden, leerde beter organiseren, en verbeterde mijn vermogen om zelfstandig te werken. Deze vaardigheden zijn van onschatbare waarde in mijn voorbereiding op het professionele leven.

Ik wil graag van deze gelegenheid gebruik maken om mijn oprechte dank uit te spreken aan Peter Van Grieken, die mij enorm veel heeft bijgeleerd en ontzettend veel tijd en moeite in mijn project heeft gestoken. Zijn begeleiding en ondersteuning waren onmisbaar tijdens mijn ontdekkingsreis. Ik ben hem zeer dankbaar voor zijn geduld, expertise en toewijding.

Inhoudsopgave

[Voorwoord 3](#_Toc137034884)

[Inhoudsopgave 4](#_Toc137034885)

[Overzicht figuren 5](#_Toc137034886)

[Inleiding 6](#_Toc137034887)

[1 Het project 7](#_Toc137034888)

[2 Hardware Componenten 8](#_Toc137034889)

[2.1 NUCLEO-F429ZI 8](#_Toc137034890)

[2.1.1 STM32F429ZI 9](#_Toc137034891)

[2.2 Rotary encoder 12](#_Toc137034892)

[2.3 OLED Display 13](#_Toc137034893)

[2.4 Relais module 13](#_Toc137034894)

[3 Software componenten 14](#_Toc137034895)

[3.1 STM HAL drivers 14](#_Toc137034896)

[3.2 CubeMX 15](#_Toc137034897)

[3.3 FreeRTOS 16](#_Toc137034898)

[3.3.1 GPOS vs RTOS 16](#_Toc137034899)

[3.3.2 Bare-Metal vs RTOS 17](#_Toc137034900)

[3.4 LwIP 18](#_Toc137034901)

[3.5 MQTT 19](#_Toc137034902)

[3.6 Home Assistant 20](#_Toc137034903)

[3.7 Node-Red 20](#_Toc137034904)

[4 Deelprojecten 22](#_Toc137034905)

[4.1 Rotary encoder 22](#_Toc137034906)

[4.1.1 Configuratie 23](#_Toc137034907)

[4.1.2 Test 24](#_Toc137034908)

[4.1.3 Denderen 24](#_Toc137034909)

[4.2 OLED display 26](#_Toc137034910)

[4.2.1 SSD1306 driver library 26](#_Toc137034911)

[4.2.1.1 Ssd1306\_conf\_template.h 26](#_Toc137034912)

[4.2.1.2 Ssd1306.c 26](#_Toc137034913)

[4.2.2 Onderzoek 30](#_Toc137034914)

[4.2.3 Configuratie 31](#_Toc137034915)

[4.3 LwIP & MQTT 32](#_Toc137034916)

[4.3.1 LwIP 32](#_Toc137034917)

[4.3.2 MQTT 33](#_Toc137034918)

[4.4 FreeRTOS 37](#_Toc137034919)

[4.4.1 Configuratie 37](#_Toc137034920)

[4.4.2 Global setters & getters 37](#_Toc137034921)

[4.4.3 Queue 39](#_Toc137034922)

[4.4.3.1 Queue interrupt routines 39](#_Toc137034923)

[4.4.4 Tasks 40](#_Toc137034924)

[4.4.4.1 DefaultTask 40](#_Toc137034925)

[4.4.4.2 MqttTask 41](#_Toc137034926)

[4.4.4.3 OledTask 43](#_Toc137034927)

[4.4.4.4 RotaryTask 44](#_Toc137034928)

Overzicht figuren

[Figuur 1 Project 7](#_Toc137034929)

[Figuur 2 Nucleo-F429ZI 8](#_Toc137034930)

[Figuur 3 Nucleo-F429ZI overview 9](#_Toc137034931)

[Figuur 4 Blokschema STM32F429ZI 11](#_Toc137034932)

[Figuur 5 incrementele encoder 12](file:///D:\Bart\School\IoT%20'22%20-%20'23\Projecten%20van%20het%20werkveld\Verslag%20&%20ppt\Verslag%20projecten%20van%20het%20werkveld.docx#_Toc137034933)

[Figuur 6 Rotary encoder mechanisch voorbeeld 12](#_Toc137034934)

[Figuur 7 OLED display 13](#_Toc137034935)

[Figuur 8 Relais module 13](#_Toc137034936)

[Figuur 9 Voorbeeld CubeMX 15](#_Toc137034937)

[Figuur 10 General Purpose Operating System 16](#_Toc137034938)

[Figuur 11 Real-Time Operating System 17](#_Toc137034939)

[Figuur 12 Super Loop 17](#_Toc137034940)

[Figuur 13 RTOS Task loops 18](#_Toc137034941)

[Figuur 14 Voorbeeld Node-Red 21](#_Toc137034942)

[Figuur 15 Signaal rotary links 22](#_Toc137034943)

[Figuur 16 Signaal rotary rechts 22](#_Toc137034944)

[Figuur 17 Rotary GPIOs CubeMX 23](#_Toc137034945)

[Figuur 18 Uitlezen rotary interrupt 24](#_Toc137034946)

[Figuur 19 Schakeldender scope 25](#_Toc137034947)

[Figuur 20 External interrupt Rotary 25](#_Toc137034948)

[Figuur 21 ssd1306.c hardware functions 27](#_Toc137034949)

[Figuur 22 ssd1306\_SetCursor 27](#_Toc137034950)

[Figuur 23 ssd1306\_WriteString 28](#_Toc137034951)

[Figuur 24 ssd1306\_WriteChar 28](#_Toc137034952)

[Figuur 25 ssd1306\_DrawPixel 29](#_Toc137034953)

[Figuur 26 ssd1306\_Line 29](#_Toc137034954)

[Figuur 27 ssd1306\_UpdateScreen 30](#_Toc137034955)

[Figuur 28 SPI modi 30](file:///D:\Bart\School\IoT%20'22%20-%20'23\Projecten%20van%20het%20werkveld\Verslag%20&%20ppt\Verslag%20projecten%20van%20het%20werkveld.docx#_Toc137034956)

[Figuur 29 SPI modi analyzer 31](#_Toc137034957)

[Figuur 30 Config GPIO SPI 31](#_Toc137034958)

[Figuur 31 CubeMX SPI 32](#_Toc137034959)

[Figuur 32 CubeMX LwIP Static IP 33](#_Toc137034960)

[Figuur 33 CubeMX LwIP PHY 33](#_Toc137034961)

[Figuur 34 mqtt\_client\_new 33](#_Toc137034962)

[Figuur 35 mqtt\_client\_info 34](#_Toc137034963)

[Figuur 36 mqtt\_init 34](#_Toc137034964)

[Figuur 37 mqtt\_connection\_cb 35](#_Toc137034965)

[Figuur 38 mqtt\_incoming\_publisch\_cb 35](#_Toc137034966)

[Figuur 39 mqtt\_incoming\_data\_cb 36](#_Toc137034967)

[Figuur 40 mqtt publisch\_message 36](#_Toc137034968)

[Figuur 41CubeMX RTOS memory 37](#_Toc137034969)

[Figuur 42 CubeMX RTOS tasks en queue 37](#_Toc137034970)

[Figuur 43 getter-setter rot value 38](#_Toc137034971)

[Figuur 44 getter-setter ETH state 38](#_Toc137034972)

[Figuur 45 mutex handles 39](#_Toc137034973)

[Figuur 46 mutex create 39](#_Toc137034974)

[Figuur 47 queue create 39](#_Toc137034975)

[Figuur 48 queue put value rotary encoder 39](#_Toc137034976)

[Figuur 49 queue put value reset rotary encoder 40](#_Toc137034977)

[Figuur 50 create tasks 40](#_Toc137034978)

[Figuur 51 DefaultTask 41](#_Toc137034979)

[Figuur 52 MqttTask 42](#_Toc137034980)

[Figuur 53 OledTask 43](#_Toc137034981)

[Figuur 54 RotaryTask 44](#_Toc137034982)

Inleiding

Dit verslag beschrijft het project waarin we FreeRTOS en LwIP hebben geïntegreerd op een STM32-microcontrollerWe hebben FreeRTOS, een real-time besturingssysteem, en LwIP, een lichtgewicht TCP./IP-stack, geïntegreerd op een STM32F4 microcontroller. Dit stelt ons in staat om de verschillende componenten als taken te plannen en prioriteren, en biedt netwerkfunctionaliteit via TCP/IP-protocollen.

In dit verslag behandelen we de configuratie, implementatie van taken, intercommunicatie, netwerkinterface-configuratie en functionaliteitstests. We bespreken ook obstakels en oplossingen die we tijdens het ontwikkelproces zijn tegengekomen.

De integratie van FreeRTOS en LwIP biedt mogelijkheden voor geavanceerde embedded systemen met netwerkfunctionaliteit. Dit verslag geeft een overzicht van het project, documenteert de stappen en overwegingen die tot een succesvolle integratie hebben geleid.

# Het project

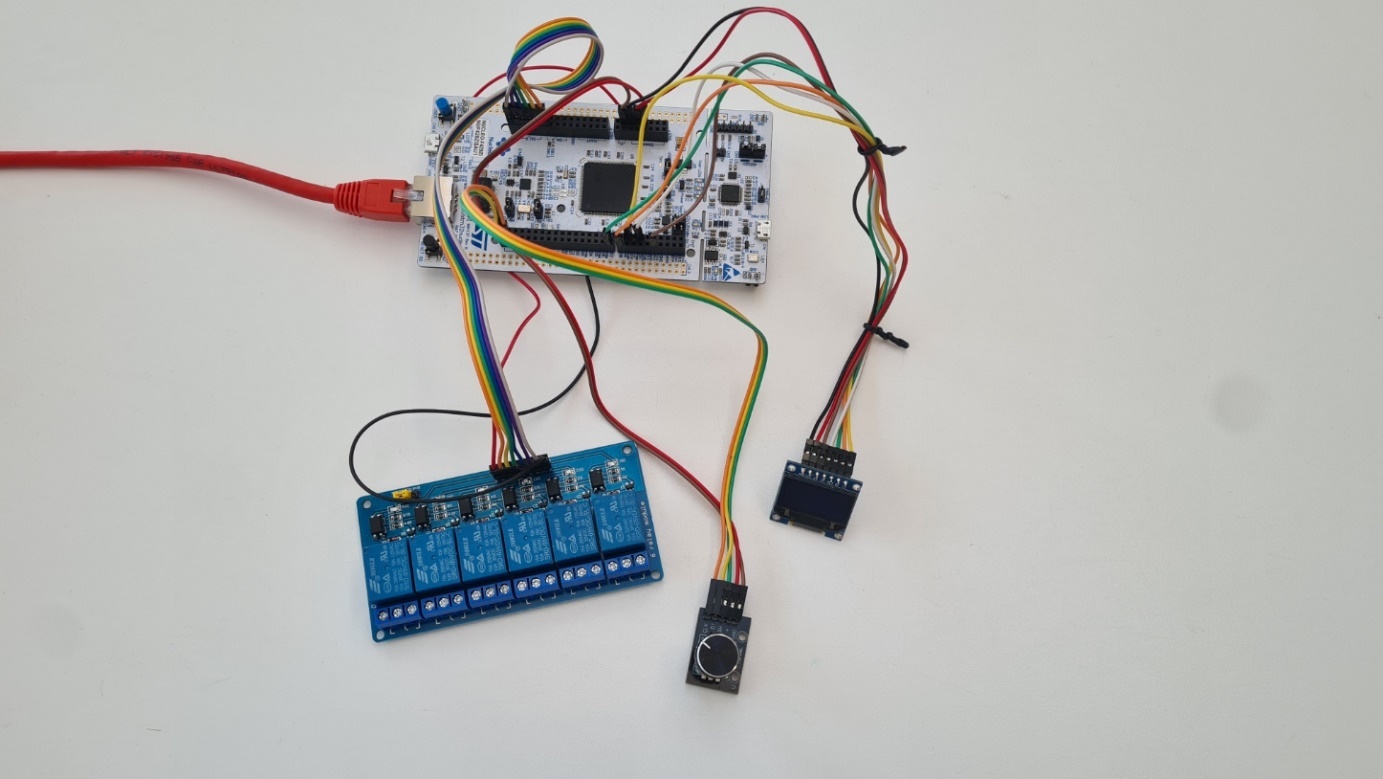
In het project is gebruik gemaakt van de devkit NUCLEO-F429ZI, FreeRTOS, LwIP, een MQTT-broker in Home Assistant, de Home Assistant app, een klein OLED-display, een rotary encoder en GPIO’s met externe relais.

In dit project lag de focus meer op de achterliggende software dan op het “visuele en tastbare”. Uiteindelijk kan het de status van de rotary encoder weergeven op het OLED-display, deze informatie versturen naar de MQTT-broker en deze weergeven in Home Assistant. Verder kan het nog met behulp van Home Assistant ook externe relais schakelen. Deze relais kunnen bijvoorbeeld worden gebruikt om verlichting, apparaten of andere elektrische componenten in en uit te schakelen. Het aansturen van de relais via de Home Assistant-app biedt de mogelijkheid om op afstand apparaten te bedienen en automatiseringsscenario's te creëren.

De project doelstellingen zijn:

* Gebruik maken van een 32-bit microcontroller met ethernet
* Uitlezen van een rotary encoder
* Aansturen van een OLED-display
* Gebruik maken van FreeRTOS
* Gebruik maken van Lwip & MQTT
* Aansturen van externe relais via GPIO
* Aansturen van de relais via Home Assistant

De meeste doelstellingen van dit project heb ik eerst afzonderlijk onderzocht, gemaakt en uitgebreid getest. Ik ga in dit verslag ook alles per ‘deelproject’ toelichten. Aan het einde van het project is alles samengebracht en gefinetuned waar nodig. Het totaal plaatje kan je in onderstaande afbeelding terug vinden.



Figuur Project

# Hardware Componenten

In dit hoofdstuk zal ik de verschillende hardware componenten van het project toelichten. Deze componenten vormen de bouwstenen van het project. Door een dieper inzicht te krijgen in hun functionaliteit, zult u een beter begrip krijgen van het project als geheel.

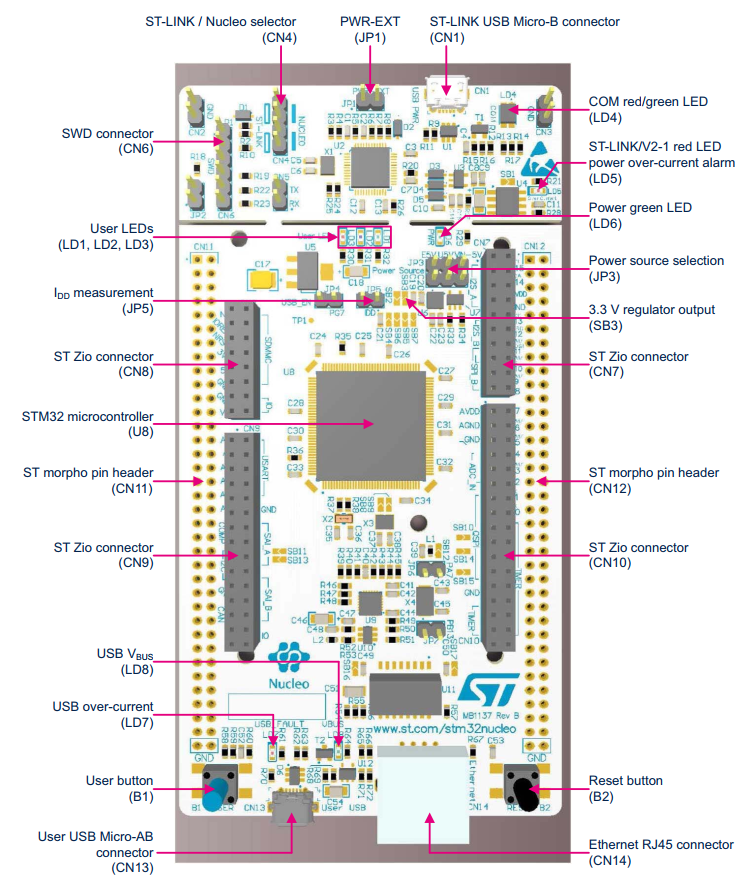
## NUCLEO-F429ZI



Figuur Nucleo-F429ZI

In het project maken we gebruik van de Nucleo-F429ZI. Dat is een STM32 Nucleo-144 Development Board dat gebruik maakt van de STM32F429ZI microcontroller dat is ontworpen en geproduceerd door STMicroelectronics. Het maakt deel uit van hun Nucleo-familie van development boards, die bedoeld zijn om ontwikkelaars, studenten en hobbyisten een eenvoudige en betaalbare manier te bieden om projecten te prototypen en te evalueren. Het beschikt over ST Zio headers wat het mogelijk maakt om arduino shields te gebruiken, ook beschikt het over de ST Morpho headers, die maken het mogelijk om alle pinnen te gebruiken. Het bord beschikt over de volgende functies:

* Interne ST-LINK/V2-1 debugger
* Flexibele bordvoeding 5V van ST-LINK/V2-1 USB VBUS
* Externe voedingsbronnen 5V en 7 tot 12V op ST Zio of ST morpho connectoren
* USB OTG of full-speed apparaat met micro-AB connector
* IEEE-802.3-2002-compatibele Ethernet-connector
* Drie user-LED's
* Twee drukknoppen - USER en RESET
* 32.768kHz Kristaloscillator - LSE kristal



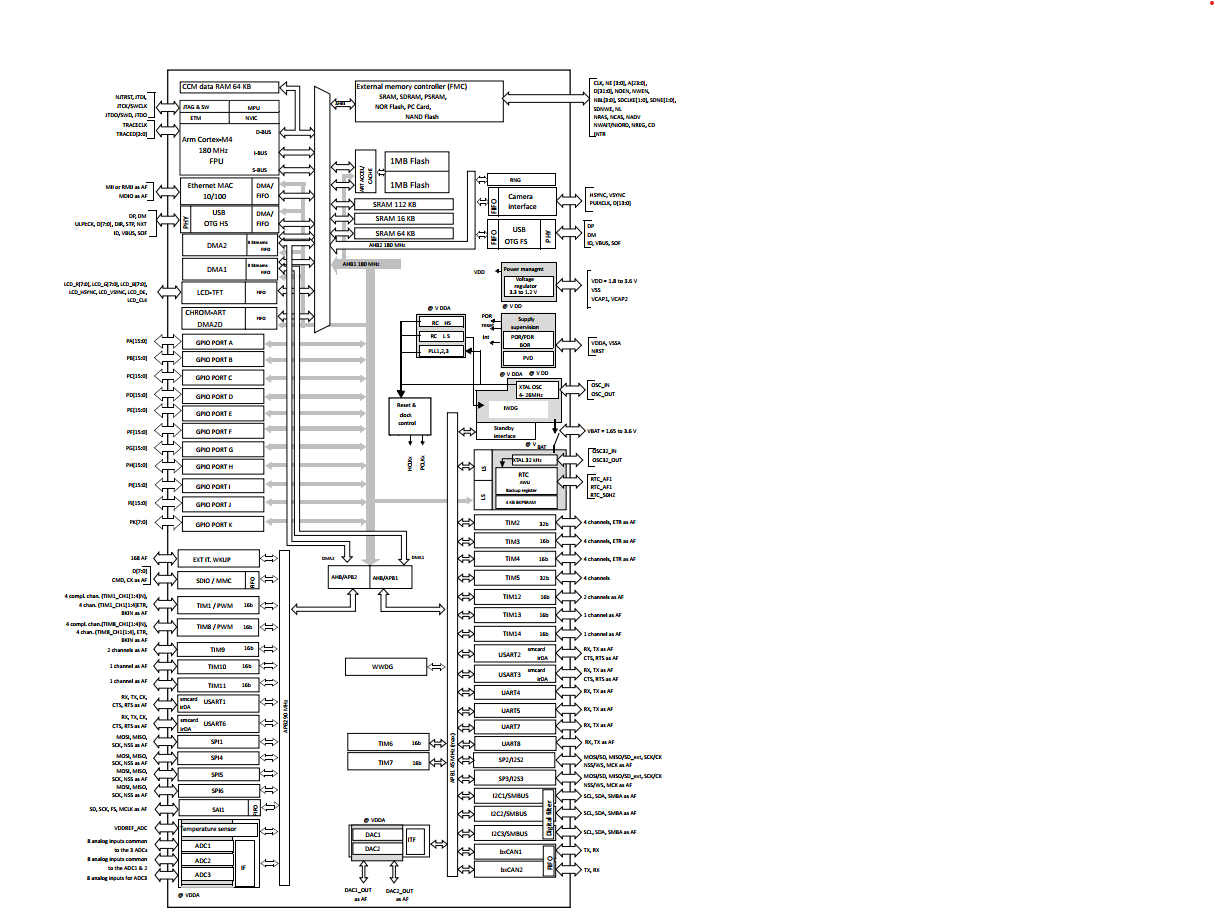
Figuur Nucleo-F429ZI overview

### STM32F429ZI

De STM32F429ZI is een microcontroller unit (MCU) uit de STM32F4-serie van STMicroelectronics. Het is gebaseerd op de ARM Cortex-M4 core en biedt verschillende functies die nuttig zijn voor embedded toepassingen. Hier zijn enkele belangrijke kenmerken:

* **Hoge prestaties**: De STM32F429ZI werkt op een hoge 180Mhz frequentie, wat zorgt voor krachtige verwerking.
* **Flashgeheugen**: Er is tot 2MB flashgeheugen aanwezig. Hierdoor is er voldoende opslagruimte aanwezig voor code en gegevens.
* **RAM**: De controller is uitgerust met maximaal 256 KB aan RAM, wat ruimte biedt voor variabelen, stack en opslag van runtimegegevens.
* **Peripherals**: De controller biedt een uitgebreide set aan peripherals, waaronder meerdere USART, SPI, I2C en CAN interfaces, USB OTG, Ethernet MAC en SDIO interfaces.
* **Analoge functies**: De STM32F429ZI bevat een 12-bits analog-to-digital coverter (ADC) met maximaal 24 kanalen. Het bevat ook een digital-to-analog converter (DAC) met maximaal 2 kanalen voor het genereren van bijvoorbeeld audio signalen.
* **Timers**: De controller bevat tot twaalf 16-bit waaronder twee moter-control PWM, verder zijn er nog twee 32-bit en twee watchdogtimers.
* **LCD-controller**: Er is een geïntegreerde LCD-controller aanwezig waardoor het geschikt is voor toepassingen die grafische gebruikersinterfaces vereisen.
* **Snelle connectiviteit**: De MCU ondersteunt snelle communicatie via USB 2.0 Full-Speed en High-Speed (met On-The-Go ondersteuning), Ethernet MAC en verschillende seriële communicatie-interfaces.
* **Digital Signal Processing (DSP) functies**: De Cortex-M4 kern bevat een hardware floating-point unit (FPU) en DSP-instructies, waardoor efficiënte signaalverwerking en wiskundige bewerkingen mogelijk zijn.
* **Ondersteuning voor Real-Time Operating System (RTOS)**: De MCU is compatibel met verschillende RTOS-oplossingen, zoals FreeRTOS, waardoor ontwikkelaars complexe en multitasking-toepassingen kunnen bouwen.
* **Ontwikkelingsondersteuning**: STMicroelectronics biedt een uitgebreid ondersteuning, inclusief ontwikkelingsboards, software-librarys en uitgebreide documentatie.

Deze kenmerken maken de STM32F429ZI een krachtige en veelzijdige microcontroller die geschikt is voor veel toepassingen, waaronder industriële regelsystemen, consumentenelektronica, medische apparaten en Internet of Things (IoT) toepassingen.



Figuur Blokschema STM32F429ZI

## KY-040 ROTARY ENCODER MODÜLÜ - DEVELOPMENT MODULE - Ulutaş ElektronikRotary encoder

Een rotary encoder is een elektronisch toestel met een roterende as, dat pulsen genereert als de as verdraait. Er bestaan ook versies die pulsen genereren langs een translatie-as.

Er zijn in principe twee soorten encoders:

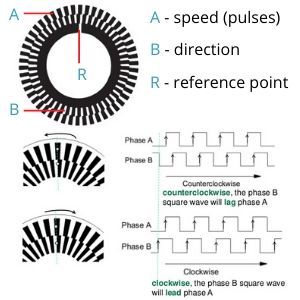
* incrementele encoders
* absoluut encoders

In dit project gaan we enkel gebruik maken van een incrementele encoder.

Figuur incrementele encoder

Een incrementele encoder geeft een simpele puls trein. De pulsen worden geteld en omgezet naar een rotatiehoek (of een translatie-afstand). Door twee puls treinen te gebruiken (A, B) die 90 graden in fase zijn verschoven, kan de elektronica van de encoder ook de richting van de beweging afleiden uit de puls treinen. Daarnaast is er vaak nog een derde signaal dat een enkele puls geeft op 0 graden (Z of C). In veel encoders is van deze signalen ook de inverse aanwezig (A, /A, B, /B, Z, /Z).

In onderstaande afbeelding is te zien dat wanneer we tegen de klok draaien Phase A hoog wordt terwijl Phase B laag blijft. Als we met de klok meedraaien is het andersom, hier gaat eerst Phase B hoog worden en daarna pas Phase A. Door deze signalen uit te lezen in de software kunnen we de richting bepalen.



Figuur Rotary encoder mechanisch voorbeeld

## OLED Display

OLED is Organic Light Emitting Diode die licht uitzendt als reactie op een elektrische stroom. Een OLED-scherm werkt zonder achtergrondverlichting, zodat het diepe zwartniveaus kan weergeven. Het is klein van formaat en licht in gewicht dan Liquid Crystal Displays.

Een 128x64 OLED-display is een eenvoudig dot matrix grafisch display. Het heeft 128 kolommen en 64 rijen waardoor het in totaal 128x64 = 8192 pixels weergeeft. Door deze pixel's led aan / uit te zetten, kunnen we een grafisch beeld van elke vorm erop weergeven. Het OLED-display wordt aangedreven door een SSD1306 driver IC. SSD1306 is een CMOS OLED-driver met controller voor een dot-matrix OLED-display. Door het gebruik van de SSD1306-driver is het aantal benodigde externe componenten en het stroomverbruik verminderd.

Het OLED-display dat in dit project gebruikt wordt werkt zowel via I2C als SPI.



Figuur OLED display

## Relais module

Omdat veel elektrische systemen meer vermogen nodig hebben dan een microcontroller kan leveren, kan de relaismodule in veel verschillende toepassingen worden gebruikt. Relaismodules zijn eenvoudigweg printplaten die een of meer relais bevatten. Ze zijn er in verschillende vormen en maten, maar zijn meestal rechthoekig met 2, 4, 6 of 8 relais erop gemonteerd, soms zelfs tot wel 16 relais. Relaismodules bevatten ook andere componenten dan de relais zelf. Waaronder indicatie-LED's, beschermingsdiodes, transistors, weerstanden en andere onderdelen.



Figuur Relais module

# Software componenten

In dit hoofdstuk zal ik de verschillende software componenten die gebruikt zijn in het project toelichten. Deze componenten vormen de bouwstenen van de software in het project.

## STM HAL drivers

De HAL-drivers zijn ontworpen om een uitgebreide set API's te bieden en gemakkelijk te kunnen communiceren met de bovenste applicatielagen.

Elke driver bestaat uit een reeks functies die de meest voorkomende peripherals dekken. De ontwikkeling van elke driver wordt gestuurd door een gemeenschappelijke API die de driverstructuur, de functies en de parameterbenamingen standaardiseert.

De HAL-drivers omvatten een set drivermodules, waarbij elke module is gekoppeld aan een zelfstandige peripheral. In sommige gevallen kan de module echter gekoppeld zijn aan een functionele mode van een peripheral. Als voorbeeld zijn er meerdere modules beschikbaar voor de USART peripheral: de UART-drivermodule, de USART-drivermodule, de SMARTCARD-drivermodule en de IRDA-drivermodule.

De belangrijkste kenmerken van HAL zijn:

* Een draagbare set API's die geschikt is voor verschillende microcontrollers en die zowel gemeenschappelijke peripherals als uitbreidings-API's voor specifieke peripherals dekken.
* Drie programmeermodellen voor API's: polling, interrupt en DMA.
* De API's zijn compatibel met RTOS:  
  + Volledig reintegratie van API's.
  + Systematisch gebruik van time-outs in polling-modus.
* Ondersteuning van meerdere instanties van peripherals, waardoor   
  gelijktijdige API-aanroepen mogelijk zijn voor meerdere instanties van een bepaalde peripheral (bijvoorbeeld USART1, USART2...).
* Alle HAL-API's implementeren het mechanisme van user-callbackfuncties:  
  + De HAL Init/DeInit-functies kunnen user-callbackfuncties aanroepen om peripheral initialisatie/de-initialisatie (klok, GPIO's, interrupt, DMA) uit te voeren.
  + Interrupts.
  + Error events.
* Objectvergrendelingsmechanisme: veilige toegang tot hardware om meerdere onbedoelde toegangen tot gedeelde bronnen te voorkomen.
* Time-out wordt gebruikt voor alle blokkerende processen: de time-out kan een eenvoudige teller zijn of een tijdbasis.

## CubeMX

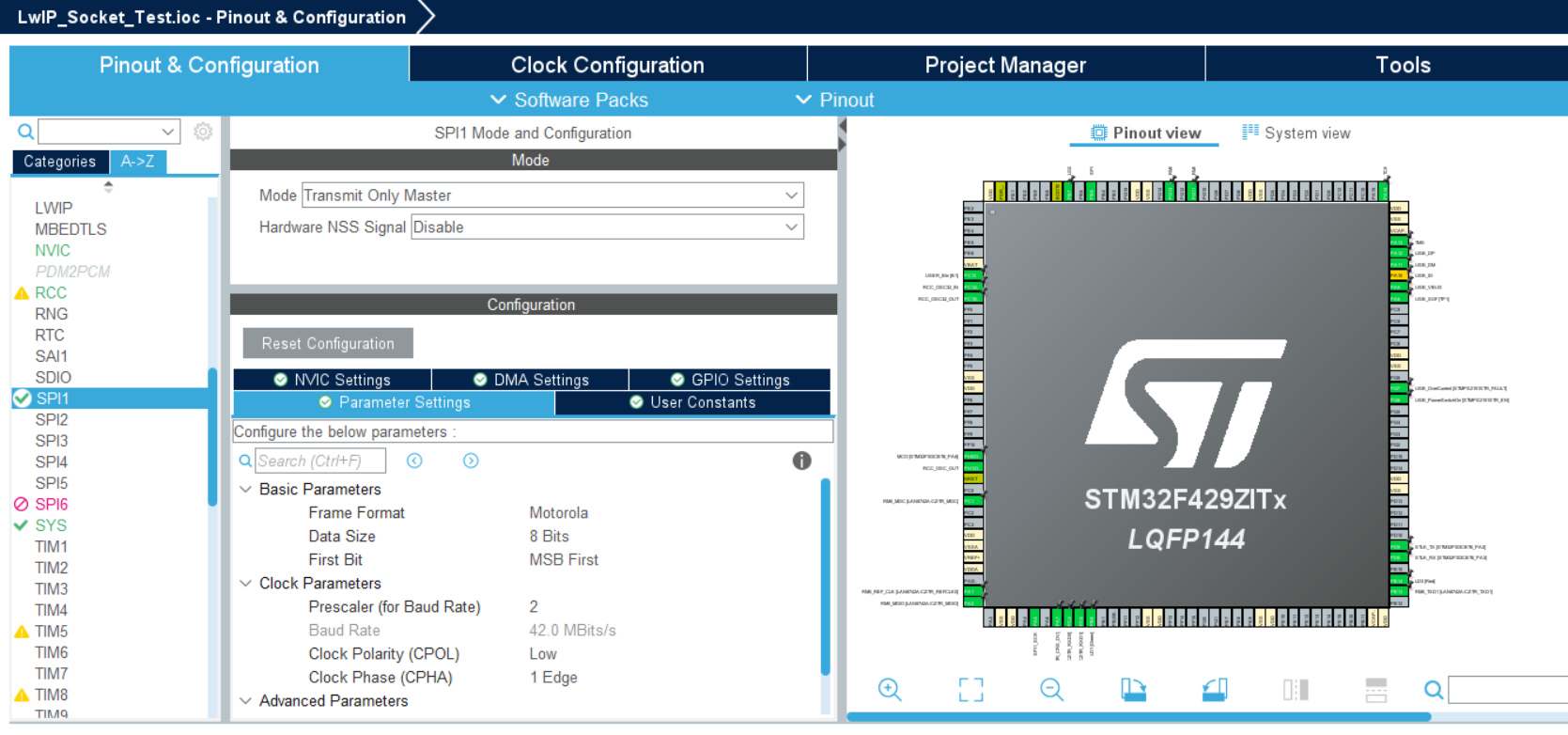
STM32CubeMX is een grafische tool die een zeer eenvoudige configuratie van STM32-microcontrollers en microprocessors mogelijk maakt, evenals het genereren van de bijbehorende initialisatie C-code voor de Arm Cortex-M-kern of een gedeeltelijke Linux-apparaatboom voor Arm Cortex-A-kern, via een stapsgewijs proces.

De eerste stap bestaat uit het selecteren van een STMicrolectronics STM32-microcontroller, microprocessor of een ontwikkelingsplatform dat overeenkomt met de vereiste set randapparatuur, of een voorbeeld dat op een specifiek ontwikkelingsplatform wordt uitgevoerd.

Voor microprocessors maakt de tweede stap het mogelijk om de GPIO's en de klokopstelling voor het hele systeem te configureren en om randapparatuur interactief toe te wijzen aan de Arm Cortex-M of aan de Cortex-A-wereld. Specifieke hulpprogramma's, zoals DDR-configuratie en -afstemming, maken het gemakkelijk om aan de slag te gaan met STM32-microprocessors. Voor Cortex-M core bevat de configuratie extra stappen die precies vergelijkbaar zijn met die beschreven voor microcontrollers.

Voor microcontrollers en microprocessor Arm Cortex-M bestaat de tweede stap uit het configureren van elke vereiste embedded software dankzij een pinout-conflictoplosser, een helper voor het instellen van een klokstructuur, een energieverbruikscalculator en een hulpprogramma dat de randapparatuur (zoals GPIO of USART) en de middleware-stacks (zoals USB of TCP / IP) configureert.

De standaard software en middleware stacks kunnen worden uitgebreid dankzij verbeterde STM32Cube uitbreidingspakketten. De partnerpakketten van STMicrolectronics of STMicrolectronics kunnen rechtstreeks worden gedownload van een speciale pakketbeheerder die beschikbaar is binnen STM32CubeMX, terwijl de andere pakketten vanaf een lokale schijf kunnen worden geïnstalleerd.



Figuur Voorbeeld CubeMX

## FreeRTOS

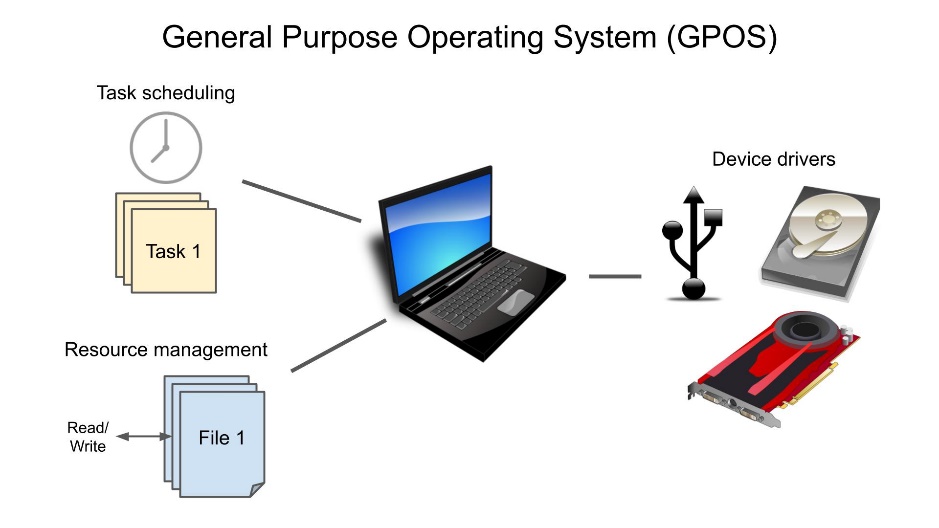
Een real-time besturingssysteem (RTOS) is meestal een lichtgewicht besturingssysteem dat multi-threaded applicaties draait en real-time deadlines kan halen. Met deadlines duiden we een niveau van determinisme aan wat betekent dat we kunnen uitzoeken wanneer bepaalde taken worden uitgevoerd voorafgaand aan de runtime. De meeste RTOS'en bevatten een scheduler, resourcemanagement en device drivers. In dit project wordt er gebruik gemaakt van FreeRTOS. FreeRTOS is lang niet het enige RTOS systeem, zo bestaan er nog:

* VxWorks
* QNX
* RTLinux
* embOS
* zephyrOS

### GPOS vs RTOS

**General Purpose Operating System (GPOS)**

Vaak, als we denken aan "besturingssystemen", denken we aan dingen als Windows, macOS en Linux. Dit zijn voorbeelden van besturingssystemen voor algemeen gebruik (GPOS). Meestal zijn deze besturingssystemen ontworpen voor gebruikersinteractie en bieden ze een soort interface, of dat nu een opdrachtregelinterface (CLI) of een grafische gebruikersinterface (GUI) is. Ze zijn ook ontworpen om meerdere toepassingen uit te voeren, vaak met multithreading, en bieden andere voordelen, zoals resource- / bestandsbeheer en drivers. Omdat gebruikersinteractie meestal de belangrijkste focus is voor GPOS-ontwerp, is enige latentie acceptabel (zolang gebruikers geen lange vertragingstijd opmerken). Als gevolg hiervan zijn exacte taakdeadlines moeilijk (zo niet onmogelijk) te voorspellen in een GPOS.

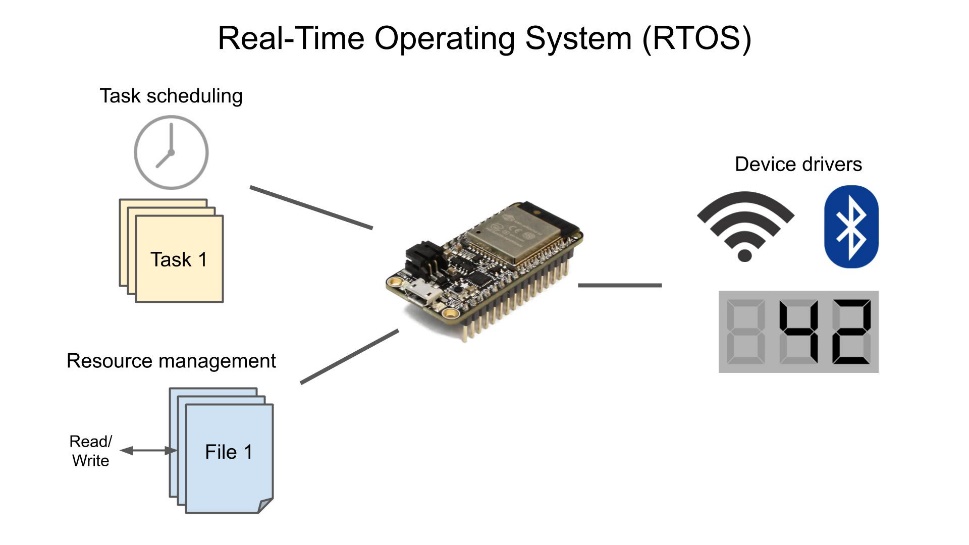


Figuur 10 General Purpose Operating System

**Real-Time Operating System (RTOS)**

De meeste RTOS'en zijn ontworpen om op microcontrollers te werken. Als gevolg hiervan kunnen ze vaak afzien van complexe gebruikersinterfaces om een paar taken tegelijk uit te voeren zonder gebruikersinvoer te hoeven verwerken. Bovendien kunnen toepassingen die zijn geschreven voor microcontrollers strikte timingdeadlines hebben die moeten worden gehaald, zoals het afvuren van een bougie om de zoveel milliseconden, het besturen van een medisch apparaat om iemand in leven te houden en het afvuren van stuwraketten op een satelliet op een precies tijdstip om in een baan om de aarde te blijven.

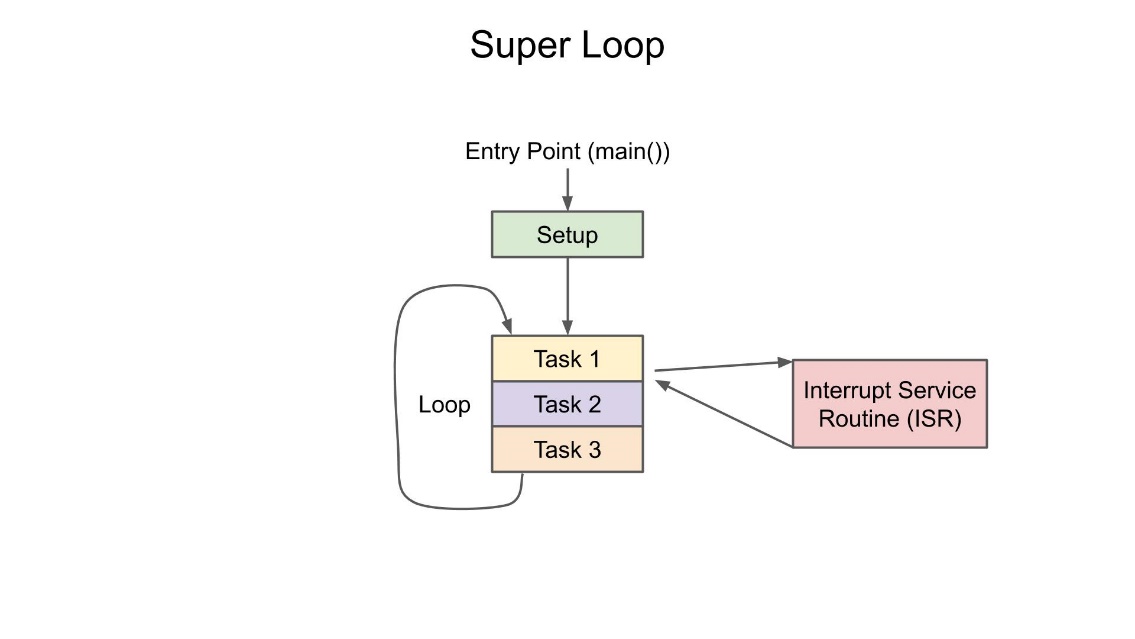
Veel RTOS'en worden ook geleverd met resource management libraries, zoals de mogelijkheid om bestanden te lezen en ernaar te schrijven, evenals low-level drivers, zoals Wifi- en Bluetooth-stacks en LCD-drivers.



Figuur 11 Real-Time Operating System

### Bare-Metal vs RTOS

**Bare-Metal**

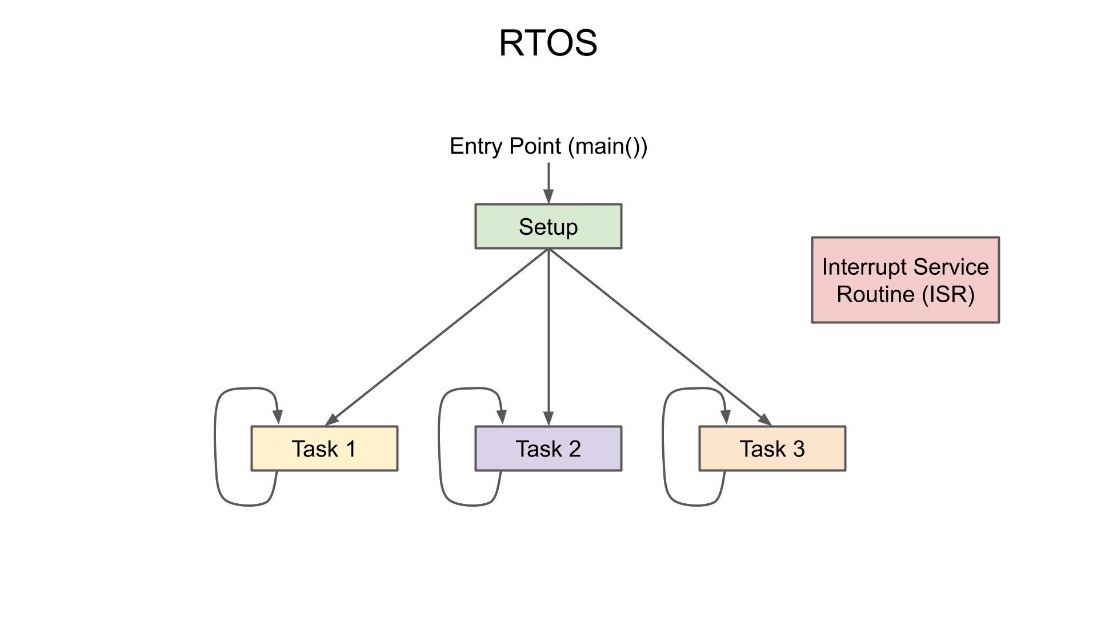
In een microcontroller maken we gebruik van aparte taken zoals bijvoorbeeld het aansturen van een led, een statemachine, uitlezen van een buffer, etc. Vooral in kleinere en meer beperkte microcontrollers wordt dit geprogrammeerd in Bare-Metal. Hierbij wordt er geen gebruik gemaakt van een operating system maar wordt alles achter elkaar, elke keer opnieuw uitgevoerd (Super-Loop). 

Figuur Super Loop

In feite is Bare-Metal nog steeds een van de meest populaire manieren om een microcontroller te programmeren, omdat het gemakkelijk te implementeren en gemakkelijk te debuggen is. Er kan zelfs gebruik worden gemaakt van interrupts, die zorgen ervoor dat het programma stopt en willekeurige code uitvoert wanneer een externe gebeurtenis optreedt (zoals een timer die afloopt of een knop die wordt ingedrukt).

**RTOS**

Het probleem gaat zich voordoen wanneer er zoveel taken wordt toegevoegd dat sommige deadlines beginnen te missen of voorkomen dat andere functies werken. Dit is waar een RTOS kan helpen. In plaats van alles achter elkaar uit te voeren, kun je in alles “tegelijkertijd” uitvoeren. RTOS zorgt ervoor dat er verschillende taken naast elkaar kunnen draaien zodat de taken elkaar niet kunnen ophouden. Er kan nog steeds gebruik gemaakt worden van interrupts. De werking is vergelijkbaar met die van de Bare-Metal manier, de taken worden gestopt, interrupt wordt uitgevoerd en de taken gaan weer verder.



Figuur RTOS Task loops

## LwIP

Het verbinden van embedded microcontrollers met het internet is een cruciale taak in moderne toepassingen. De ingebouwde controllers, met name in consumentenapparaten en wearables, vereisen tegenwoordig een internet verbinding. Dit is belangrijk omdat de apparaten slimmer worden. Verbinding maken met internet is niet alleen cruciaal, maar ook een flinke taak. De typische TCP/IP-stack is te resource-intensief voor bare-metals om te implementeren. Microcontrollers kunnen nooit TCP / IP-stack implementeren vanwege weinig geheugen, RAM en rekenkracht, dus ze hebben iets lichtgewichts nodig om dezelfde functionaliteit te emuleren.

Light-Weight Internet Protocol (LwIP) is een kleine onafhankelijke implementatie van de TCP/IP-stack voor embedded systemen. Oorspronkelijk ontwikkeld door Adam Dunkels aan het Swedish Institute of Computer Science (SICS), is LwIP een gratis, open-source TCP/IP-stack die nu bij Savannah wordt onderhouden. Verschillende gerenommeerde fabrikanten van embedded systemen hebben LwIP geadopteerd, zoals Texas Instruments, Altera Corporation, Xilinx, STMicroelectronics, Freescale en Analog Devices, voor netwerkdrivers van hun embedded apparaten. Het protocol wordt zelfs gebruikt door Arduino Ethernet Shield en WiFi-ontwikkelborden zoals ESP32 en ESP8266 voor het implementeren van netwerkfuncties. Elke hardware fabrikant kan het protocol gebruiken met het besturingssysteem of met de driver op zijn platform als een open protocol onder de gewijzigde BSD-licentie. Het protocol kan ook worden gebruikt in toepassingen op voorwaarde dat het onderliggende systeem de LwIP-poort ondersteunt.

Met LwIP kan een volwaardige TCP/IP-stack op embedded systemen worden geïmplementeerd terwijl er minimale systeembronnen en geheugen worden verbruikt. De stack is zelfs aangepast aan Arduino-libraries. De stack vereist slechts 40kb flash-geheugen en verbruikt slechts tientallen Kbytes RAM tijdens runtime. Zelfs een op maat gemaakte API is beschikbaar voor de stack die geen gegevens hoeft te kopiëren.

LwIP biedt de basisfunctionaliteit voor het opzetten van netwerkverbindingen, het verzenden en ontvangen van gegevens via IP-netwerken en het implementeren van verschillende netwerkprotocollen zoals:

* Internet Protocol (IP)
* Internet Control Message Protocol (ICMP)
* Neighbor Discovery (ND)
* Multicast listener discovery for IPv6 (MLD)
* User Datagram Protocol (UDP)
* Transmission Control Protocol (TCP)
* Internet Group Management Protocol (IGMP)
* Address Resolution Protocol  (ARP)
* Point-to-Point Protocol (PPP)

Verder beschikt het ook nog over drie API’s:

* low-level "core" / "callback" of "raw" API
* [netconn API](https://lwip.fandom.com/wiki/Netconn_API)
* [socket API](https://lwip.fandom.com/wiki/Socket_API)

## MQTT

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.

## 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.

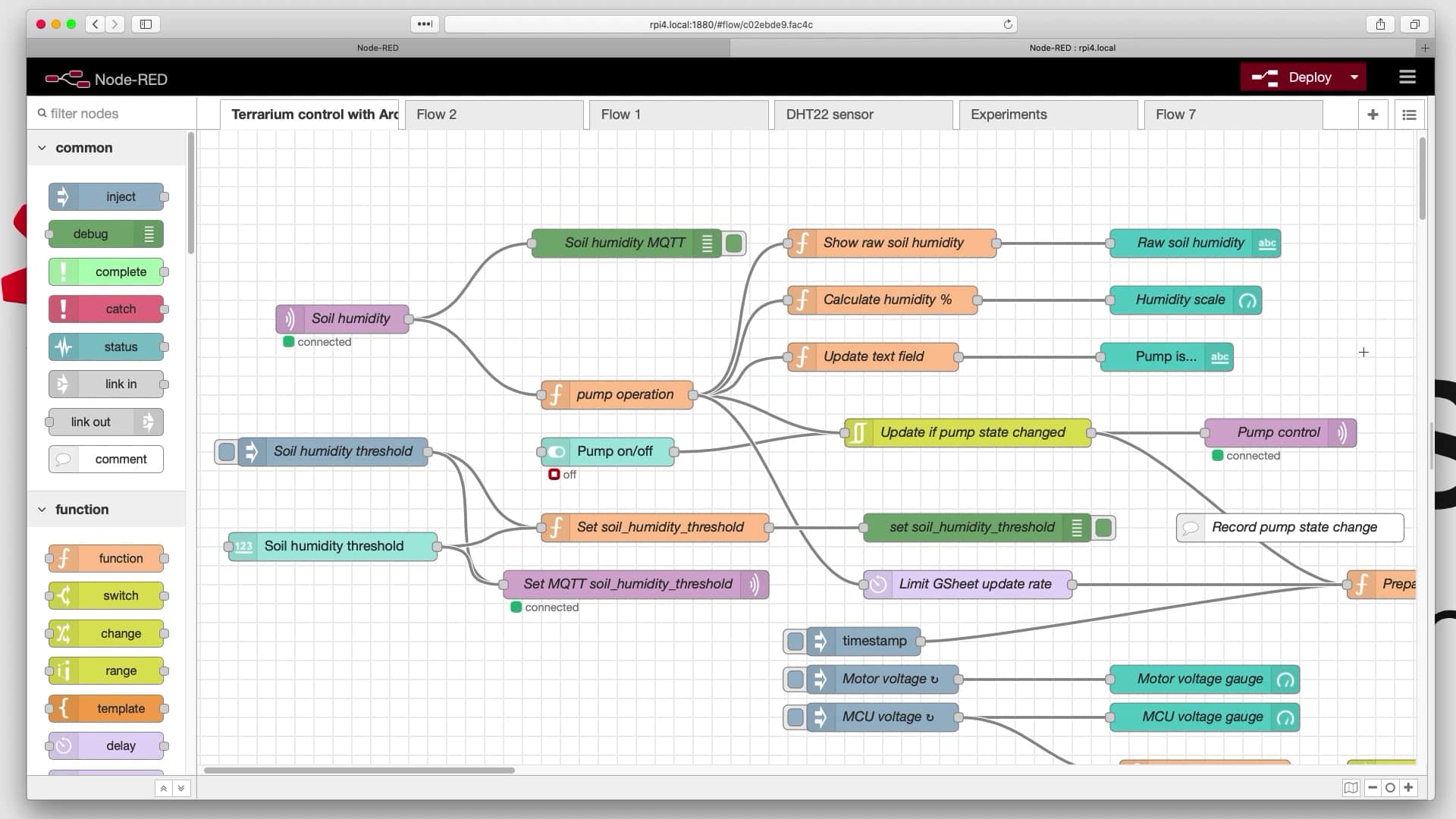
Home Assistant wordt gebruikt om de externe relais aan te sturen, ook wordt in het dashboard de stand van de rotary encoder weergegeven.

## 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 ik om de data uit de mqtt broker te halen en te sturen

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



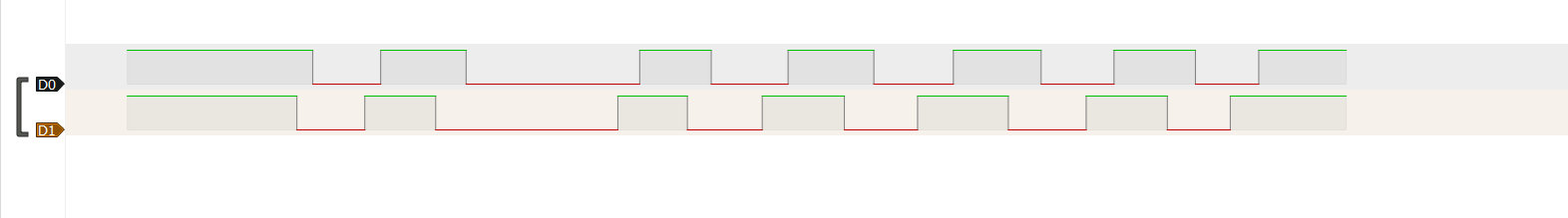
Figuur Voorbeeld Node-Red

# Deelprojecten

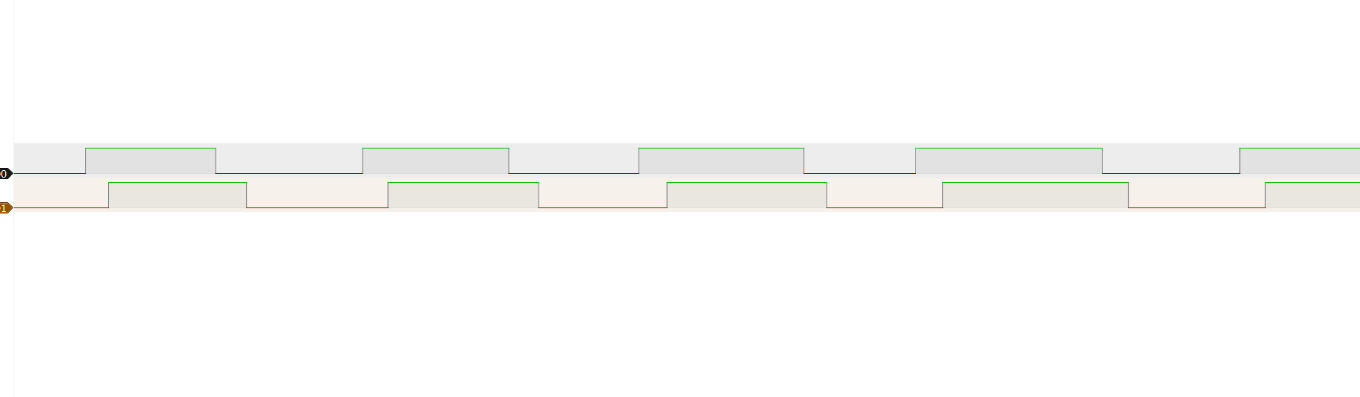
Omdat dit project een beetje complex ging worden heb ik gebruik gemaakt van deelprojecten. Dit was handig omdat ik dan het overzicht kon houden over de verschillende onderdelen van het project. Ook als er iets zou mislopen zijn de andere onderdelen nog een beetje beschermd. Ook geeft dit als voordeel dat de onderdelen gemakkelijk worden aangepast aan de noden van de andere deelprojecten.

## Rotary encoder

Zoals in [2.2 Rotary encoder](#_Rotary_encoder) al is uitgelegd, geeft de encoder twee signalen die we moeten uitlezen. Voor de zekerheid heb ik dit nog gecontroleerd met een logic analyzer.



Figuur Signaal rotary links

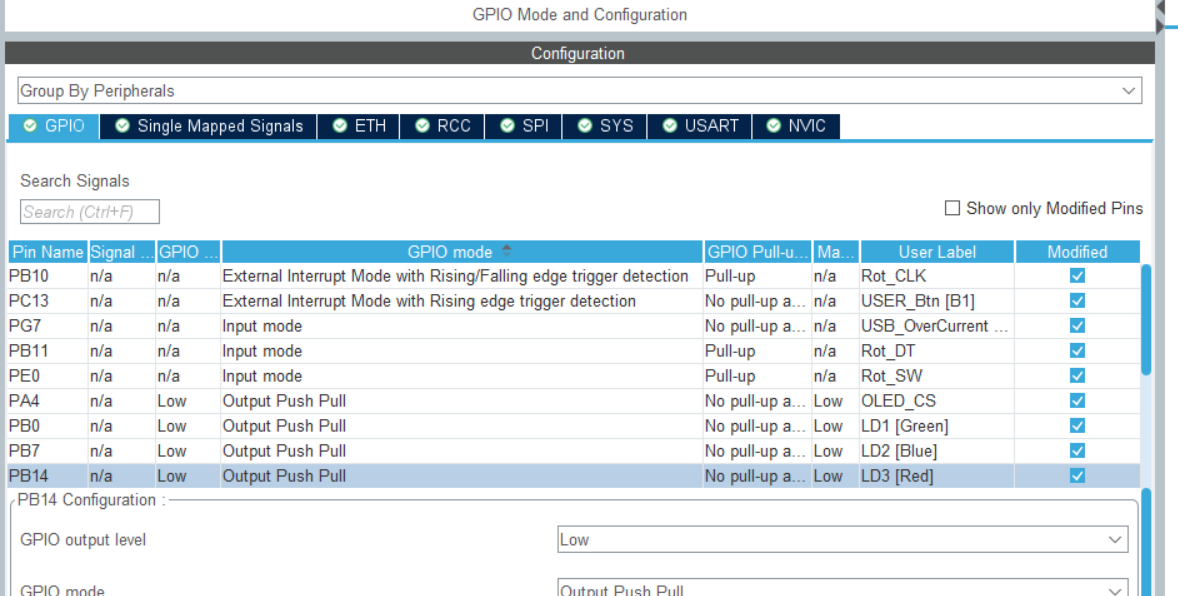


Figuur Signaal rotary rechts

Zoals we in bovenstaande afbeeldingen kunnen zien klopt het inderdaad dat de signalen 90° verschillen. De rotary encoder dat ik gebruik hebben de pinnen DT (data) en CLK (clock). Het eerste signaal is CLK, het tweede DT. Als ik naar links draai ijlt DT voor op CLK, naar rechts ijlt DT na op CLK.

Dit gaat allemaal zeer snel, om op die snelheid te reageren ga ik gebruik maken van interrupts. Voor het wat duidelijk te houden ga ik een interrupt laten triggeren wanneer er een verandering van signaal is op CLK, daarna ga ik de stand van DT controleren om de richting te bepalen. Als we de richting hebben bepaald gaan we aan de hand van de richting een variabele optellen of aftellen.

### Configuratie

Voor dat er code voor kan geschreven worden moeten we eerst de inputs configureren. Dat doen we in CubeMX. 

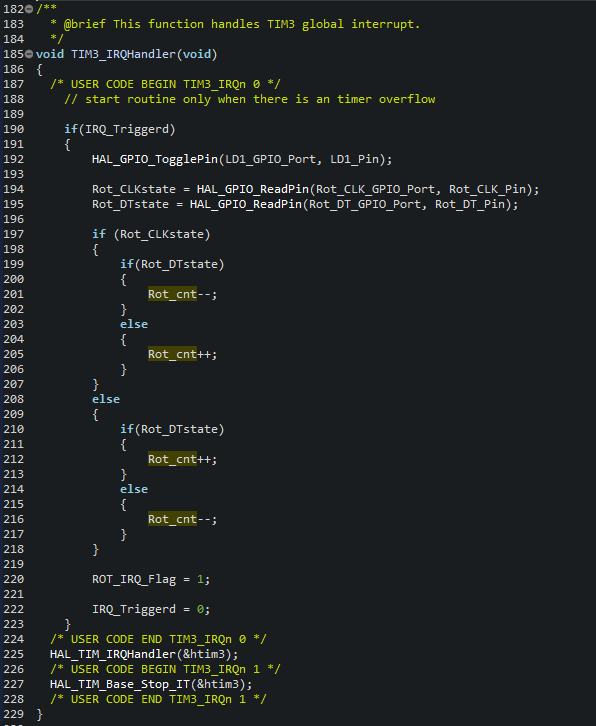
Figuur Rotary GPIOs CubeMX

De volgende pinnen heb ik geconfigureerd:

* **PB10 -> Rot\_CLK**: Dit heb ik ingesteld als ext interrupt op rising en falling edge, dit zorgt ervoor dat bij elke verandering de interrupt wordt getriggerd.
* **PB11 -> Rot\_DT**: Dit is ingesteld als een gewone input.
* **PE0 -> Rot\_SW**: Ook dit is ingesteld als gewone input.

### Test

Voor dit te testen heb ik code geschreven aan de hand van de uit te lezen signalen dat ik eerder al heb uitgelegd. We gaan de richting bepalen door eerst het CLK signaal te lezen omdat daarop wordt getriggerd, dit doen we door Rot\_CLKstate. Eens dat gebeurd is gaan we aan de hand van die waarde het DT signaal uitlezen (Rot\_DTstate), hier gaan we de richting in bepalen.

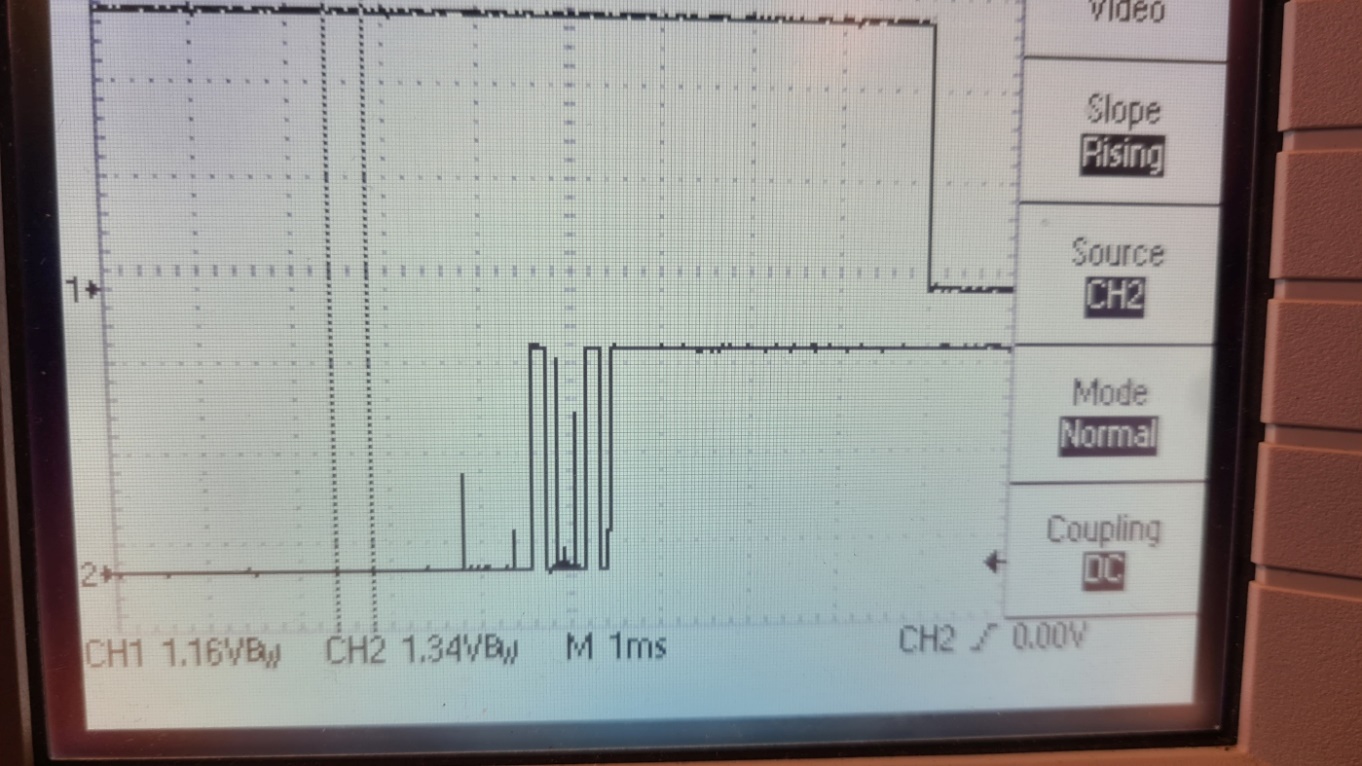


Figuur Uitlezen rotary interrupt

### Denderen

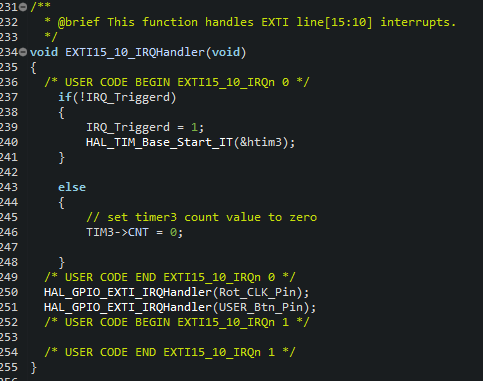
Bij de test heb ik ondervonden dat de variabele Rot\_cnt inorm schommelde in waarde’s. Door dit te debuggen viel het op dat de interrupt meermaals werd getriggerd door één positie te veranderen.

De rotary encoder geeft of schakelt een mechanisch contact. Wordt er contact gemaakt, dan wordt in werkelijkheid het contact binnen een paar microseconden een aantal malen gemaakt en verbroken. Dit gaf de problemen in de interrupt, elke keer als het signaal veranderde werd de interrupt getriggerd. Dit wordt schakel dender of bouncing genoemd. Met de oscilloscope werd dit ook duidelijk.



Figuur Schakeldender scope

Om dit probleem op te lossen moeten we ervoor zorgen dat als de interrupt wordt getriggerd er een overbrugging is totdat de dender is verdwenen. Dit heb ik gedaan door gebruik te maken van timer in output compare mode. De timer zal geactiveerd worden wanneer er een interrupt plaatsvindt en zijn teller zal blijven tellen tot deze gelijk is aan de ingestelde counter period. Wat er dan voor zorgt dat de timer een interrupt triggerd en daar worden dan de signalen uitgelezen, erna wordt de timer gestopt zodat deze niet opnieuw de interrupt doet triggeren. Als er in de “overbrugging” periode een nieuwe interrupt wordt getriggerd zal de counter value van de timer worden gerest. Dit zorgt ervoor dat ik vanaf nu altijd correcte waardes heb.



Figuur External interrupt Rotary

## OLED display

Zoals ik in [2.3 OLED-display](#_OLED_Display) al had aangehaald kunnen we dit scherm op twee manieren gebruiken, I2C en SPI. Ik heb voor SPI gekozen. Ik ben ook opzoek gegaan naar een geschikte library voor de ssd1306 driver om deze te kunnen implementeren om het scherm te sturen.

### SSD1306 driver library

De library heb ik gevonden op github en kan via de volgende link gedownload worden. <https://github.com/afiskon/stm32-ssd1306>

#### Ssd1306\_conf\_template.h

De driver moet eerst geconfigureerd worden in de file ssd1306\_conf\_template.h na de configuratie dient \_template te worden verwijderd. De driver kan gebruikt worden met zowel SPI en I2C. Welke we gaan gebruiken dient geselecteerd te worden. De driver is ook getest op de volgende microcontrollers:

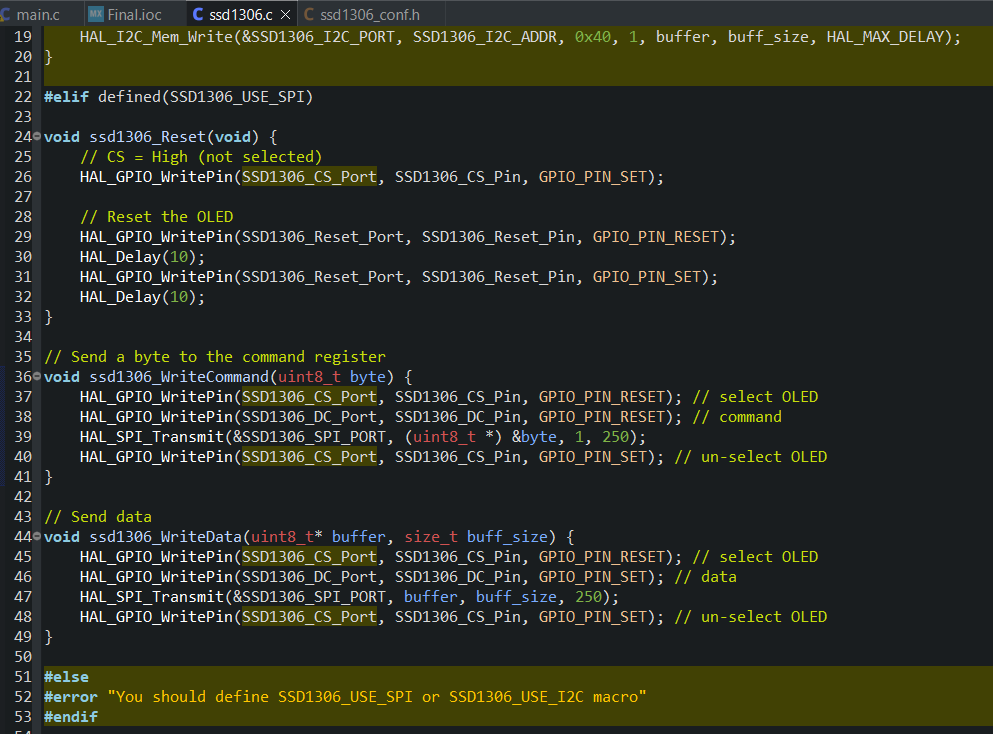
* STM32F0
* STM32F1
* STM32F3
* STM32F4
* STM32F7
* STM32L0
* STM32L1
* STM32L4
* STM32H7
* STM32G0

Hier geldt hetzelfde, wat we gaan gebruiken moeten we selecteren. Het volgende dat dient te gebeuren is het instellen van de SPI port en de bijhorende GPIO’s. In de template staan een aantal define’s i.v.m. SPI configuratie, wat hier dient te gebeuren zijn de benamingen te kopieren naar de desbetreffende pins die worden geconfigureerd via CubeMX dit zijn we in [4.2.3 Configuratie](#_Configuratie).

#### Ssd1306.c

Voor het gebruik van de driver zijn er drie functies die met de hardware gaan praten. Het is ook enkel in deze functies dat SPI transmitting wordt gebruikt.

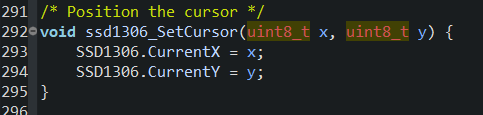
* **void ssd1306\_Reset(void)**Deze functie gaat de driver resetten, dit gebeurt door middel van gpio’s.
* **void ssd1306\_WriteCommand(uint8\_t byte)**Deze functie wordt gebruikt om commando’s naar de driver te sturen om bijvoorbeeld het display te initialiseren of om te laten weten dat we data gaan sturen.
* **void ssd1306\_WriteData(uint8\_t\* buffer, size\_t buff\_size)**Deze functie gaat het framebuffer verzenden naar de driver.

****

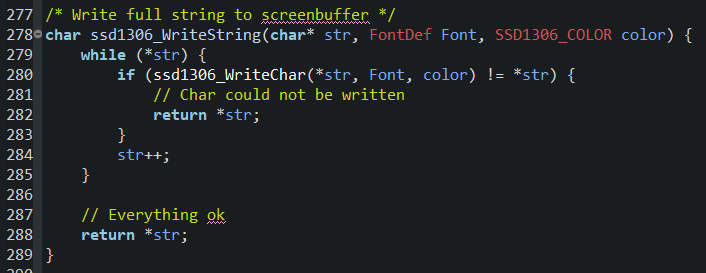
Figuur ssd1306.c hardware functions

Voor data uit de applicatie naar het scherm te sturen zijn er ook nog een aantal functies die we gaan gebruiken.

* **void ssd1306\_SetCursor(uint8\_t x, uint8\_t y)**

Deze functie wordt gebruikt om aan te geven op welke positie op het scherm we de data willen tonen, hier gaan we de x en y as meegeven.  
  
****

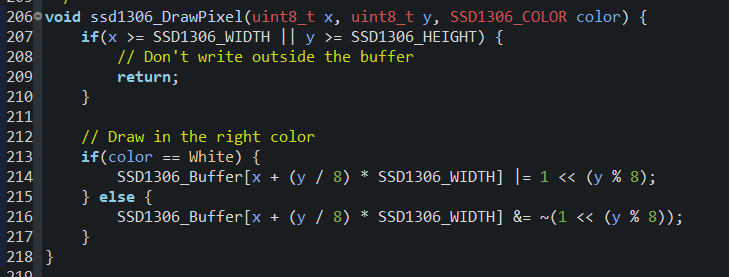
Figuur ssd1306\_SetCursor

* **char ssd1306\_WriteString(char\* str, FontDef Font, SSD1306\_COLOR color)**Wordt gebruikt voor het sturen van een string, deze functie maakt gebruik van de *WriteChar* functie.  
    
  ****

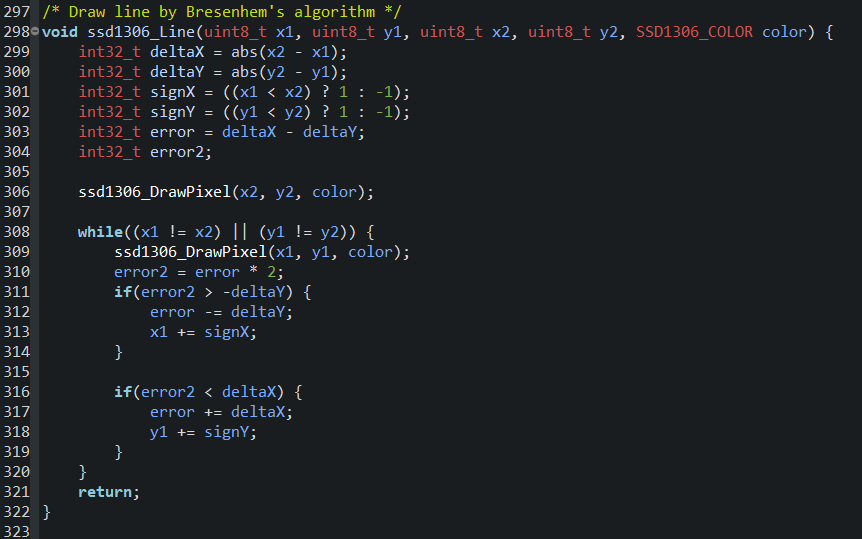
Figuur ssd1306\_WriteString

* **char ssd1306\_WriteChar(char ch, FontDef Font, SSD1306\_COLOR color)**Deze functie wordt gebruikt om een karakter te sturen daarvoor wordt gebruik gemaakt van de *DrawPixel* functie. De parameters die we hebben ingegeven bij de functie *SetCursor* worden hier gebruikt om met *DrawPixel* mee te geven.  
    
  

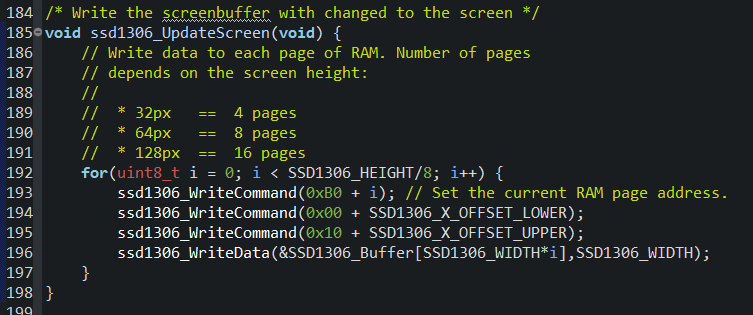
Figuur ssd1306\_WriteChar

* **void ssd1306\_DrawPixel(uint8\_t x, uint8\_t y, SSD1306\_COLOR color)**Deze functie gaat alle data in pixels in de framebuffer steken.  
    
  

Figuur ssd1306\_DrawPixel

* **void ssd1306\_Line(uint8\_t x1, uint8\_t y1, uint8\_t x2, uint8\_t y2, SSD1306\_COLOR color)**Met deze functie kunnen we een lijn tekenen. Hier wordt gebruik gemaakt van Bresenhem’s algorithme. Meer informatie hierover kan je vinden in de volgende link [Bresenham's line algorithm - Wikipedia](https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm)  
    
  

Figuur ssd1306\_Line

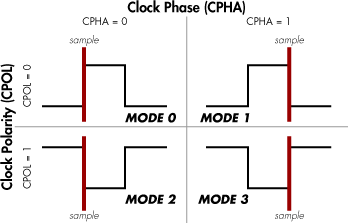
* **void ssd1306\_UpdateScreen(void)**Deze functie zal de framebuffer naar de driver sturen door gebruik te maken van de functie’s *WriteCommand* en *WriteData.*****

Figuur ssd1306\_UpdateScreen

### Onderzoek

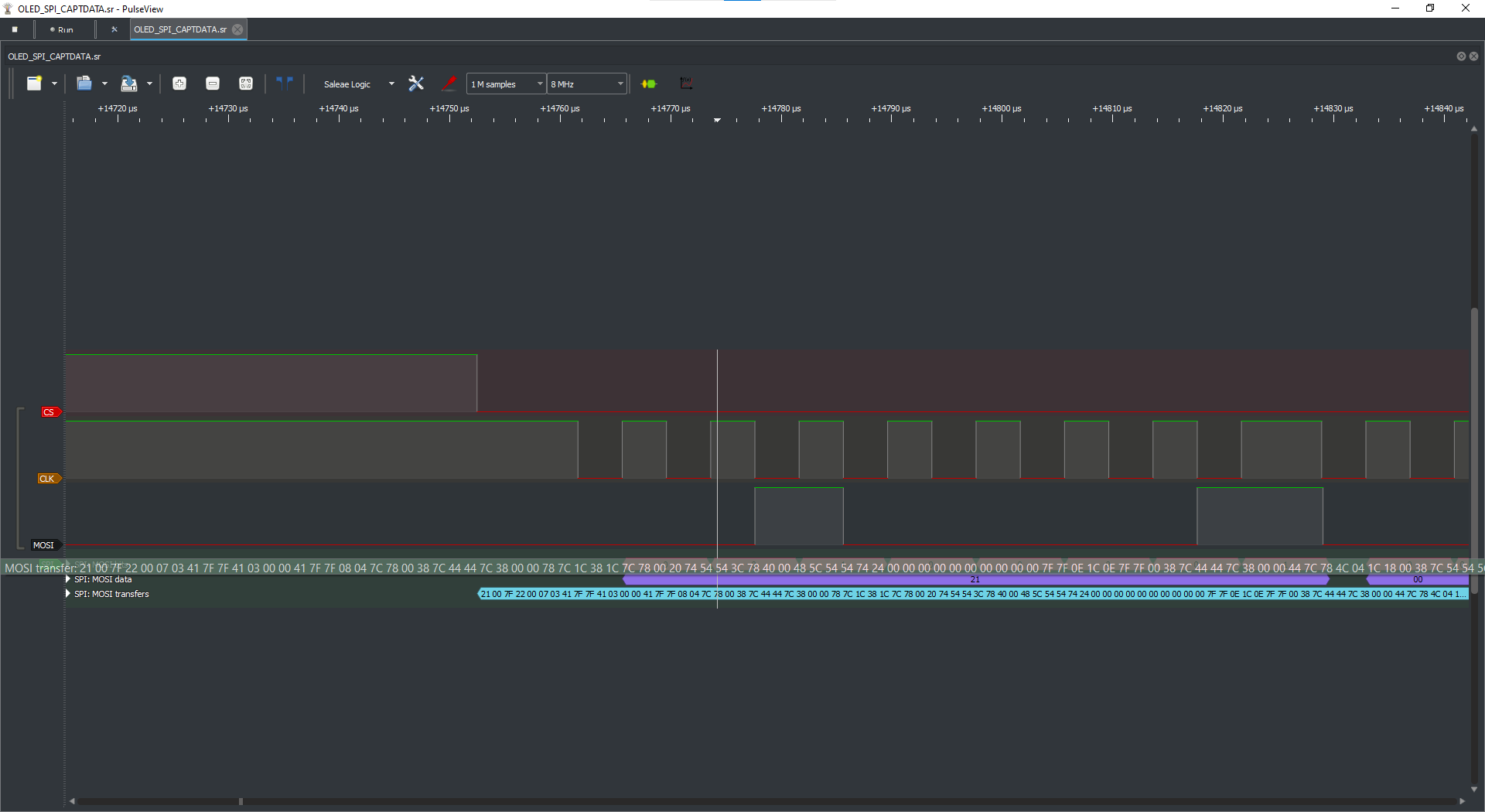
Voor dat we de configuratie in CubeMX kunnen doen hebben we nog een aantal gegevens nodig. SPI kan in verschillende modi worden gebruikt.

* Mode 0 -> Clock idle laag, sample 1ste flank
* Mode 1 -> Clock idle laag, sample 2de flank
* Mode 2 -> Clock idle hoog, sample 1ste flank
* Mode 3 -> Clock idle hoog, sample 2de flank



Figuur SPI modi

Welke we gaan gebruiken hangt af van de SSD1306 driver. Het scherm heb ik in een ander project al gebruikt, ik heb dan met een logic analyzer nagemeten hoe deze in het ander project was geconfigureerd. Het resultaat is te zien in de volgende afbeelding.



Figuur SPI modi analyzer

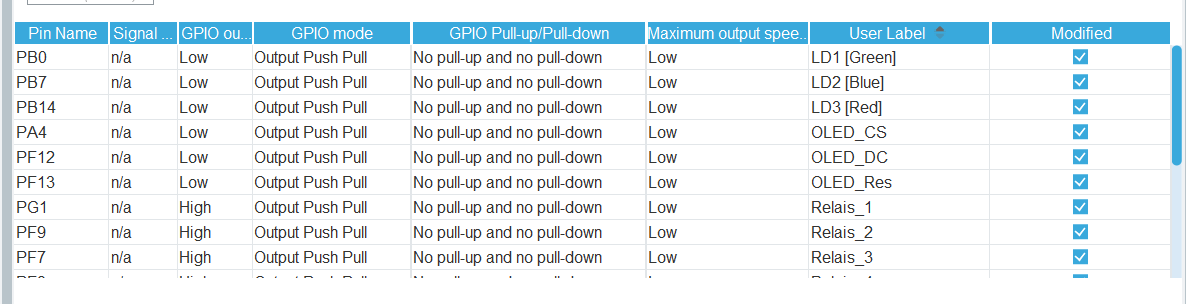
Zoals we kunnen zien is de clock idle hoog en wordt de data gesampeld op de 2de flank. Door dit zo te onderzoeken weet ik hoe SPI in CubeMx moet geconfigureerd worden.

### Configuratie

**GPIO**

Zoals ik eerder had aangehaald moeten we een paar benamingen van define’s toewijzen aan GPIO’s. De GPIO heb ik hier geconfigureerd, het betreft:

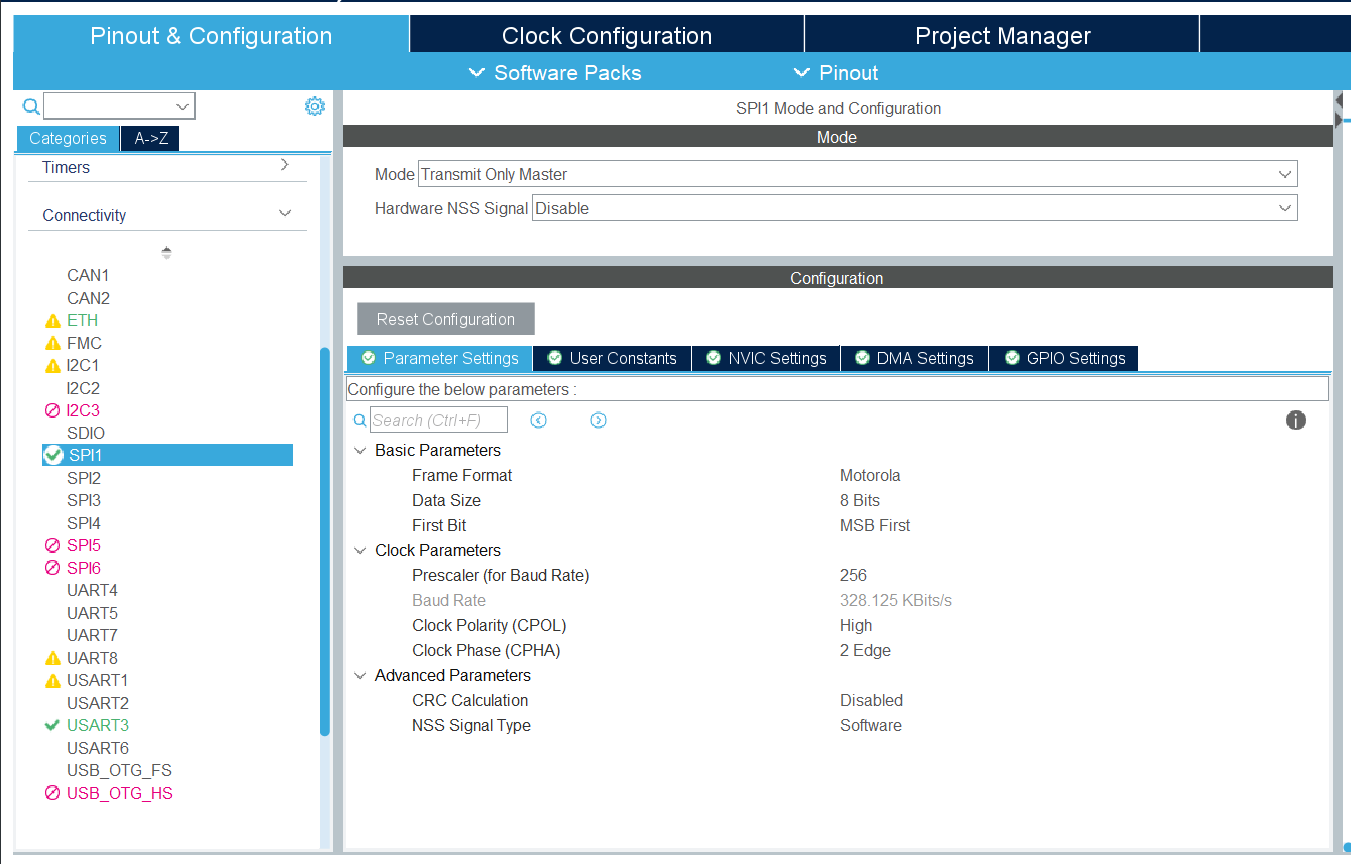
* PA4 -> OLED\_CS
* PF12 -> OLED\_DC
* PF13 -> OLED\_Res



Figuur 30 Config GPIO SPI

**SPI**

De SPI peripheral zelf wordt ook in CubeMX geconfigureerd. Ik heb voor de Transmit Only Master gekozen omdat we geen data terug verwachten en dus enkel MOSI gaan gebruiken. Verder hebben we uit het onderzoek geleerd dat we de mode van de SPI peripheral moeten configureren op Clock polariteit HIGH en Clock Phase 2de edge. Verder heb ik alles standaard gelaten. De configuratie kan terug gevonden worden in de onderstaande afbeelding.



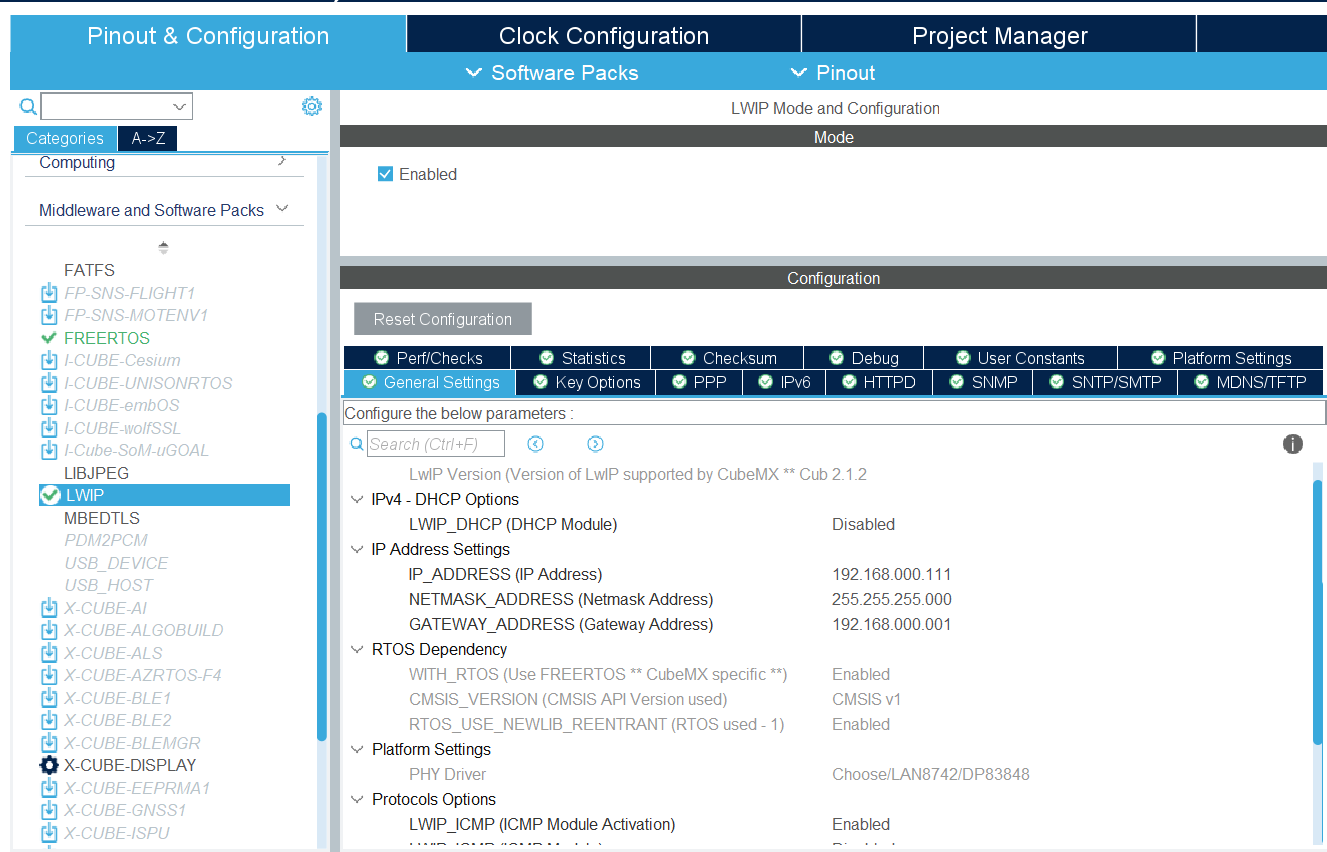
Figuur CubeMX SPI

## LwIP & MQTT

Voor het gebruik van LwIP en MQTT heb ik enkel LwIP moeten configureren in CubeMX. MQTT komt in een applicatie laag in de LwIP library zelf. LwIP zelf wordt enkel gebruikt door MQTT, in dit project heb ik geen applicatie geschreven dat rechtstreeks gebruik maakt (zonder MQTT) van LwIP.

### LwIP

Voor de configuratie heb ik enkel een static ip ingesteld en de PHY geselecteerd.



Figuur 32 CubeMX LwIP Static IP



Figuur CubeMX LwIP PHY

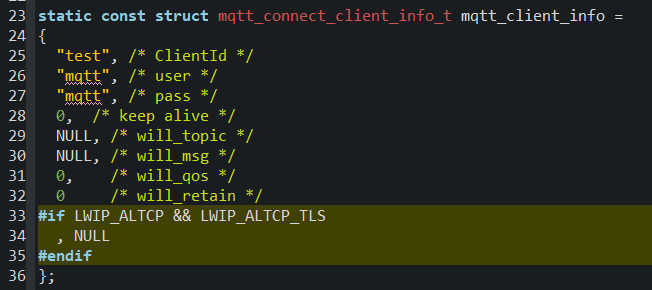
### MQTT

Voor het gebruik van MQTT zijn er een aantal structs en functies nodig. Dit is nodig voor het aanmaken van een client, de initialisatie, het ontvangen en het versturen van berichten. Alle functies kunnen terug gevonden worden in de mqtt\_global.c file.

* **mqtt\_client\_t\* mqtt\_client**Deze struct is nodig voor het aanmaken van een client, hier zitten alle benodigdheden in alsook het id van de client. De client wordt aangemaakt in de main file, vlak voor de initialisatie.  
  ****

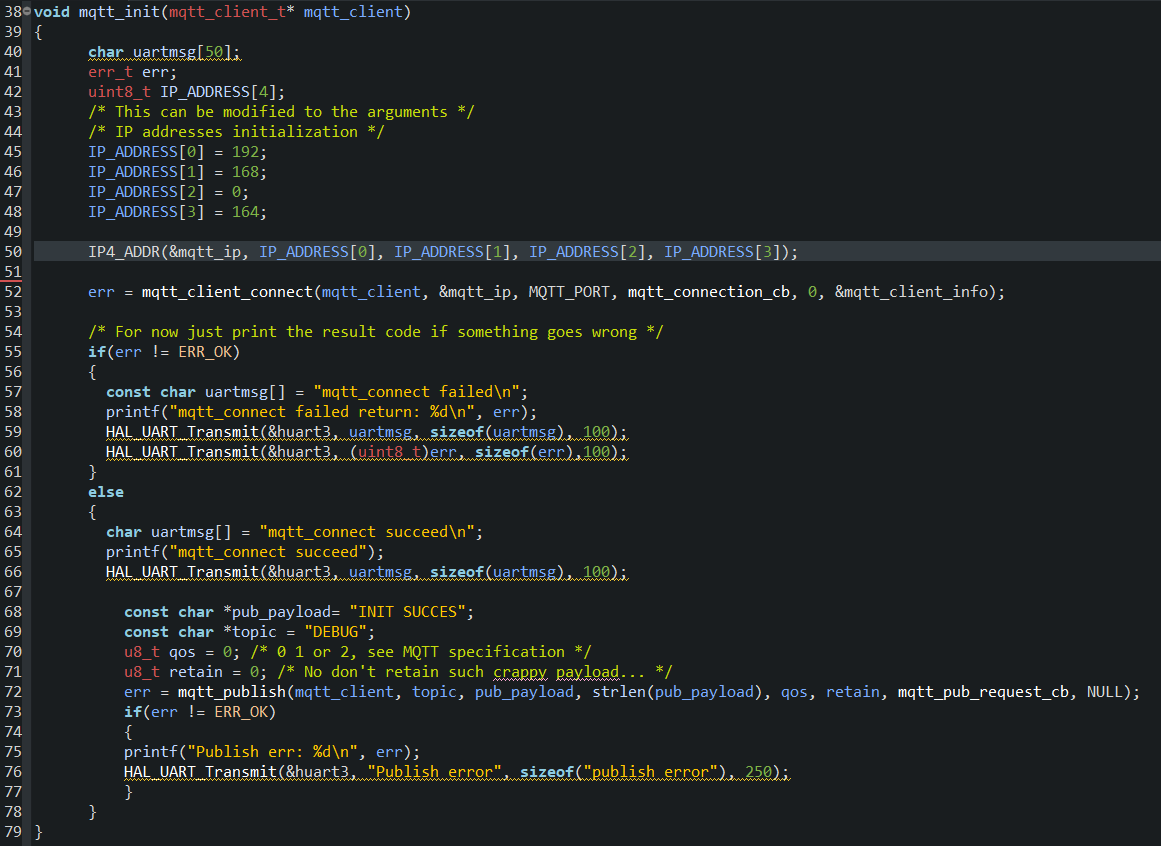
Figuur mqtt\_client\_new

* **static const struct mqtt\_connect\_client\_info\_t mqtt\_client\_info**Deze struct wordt gebruikt voor bijkomende client informatie zoals een username en password, etc.

****

Figuur mqtt\_client\_info

* **void mqtt\_init(mqtt\_client\_t\* mqtt\_client)**In deze functie wordt het IP toegewezen en wordt er verbinding gemaakt met de broker.

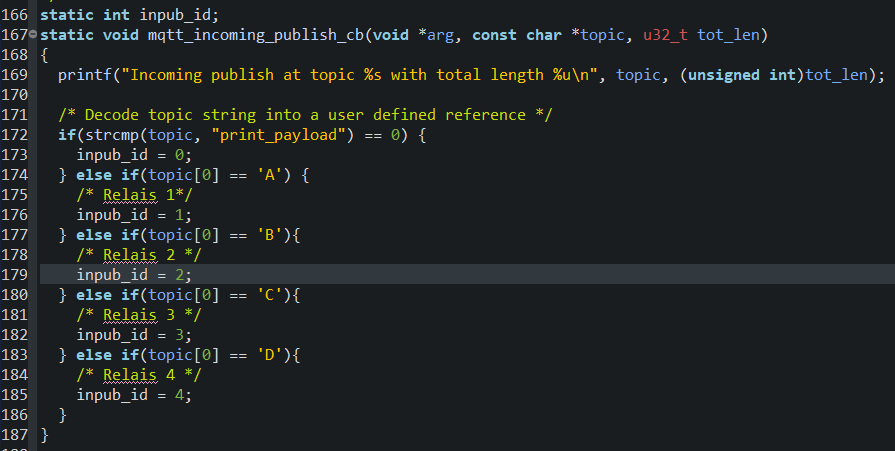
****

Figuur mqtt\_init

* **static void mqtt\_connection\_cb(mqtt\_client\_t \*client, void \*arg, mqtt\_connection\_status\_t status)**Deze functie is de callback functie die wordt aangeroepen als de connectie gelukt of mislukt is. Hier wordt er dan als de connectie gelukt is, geabonneerd op topics.  
    
  ****

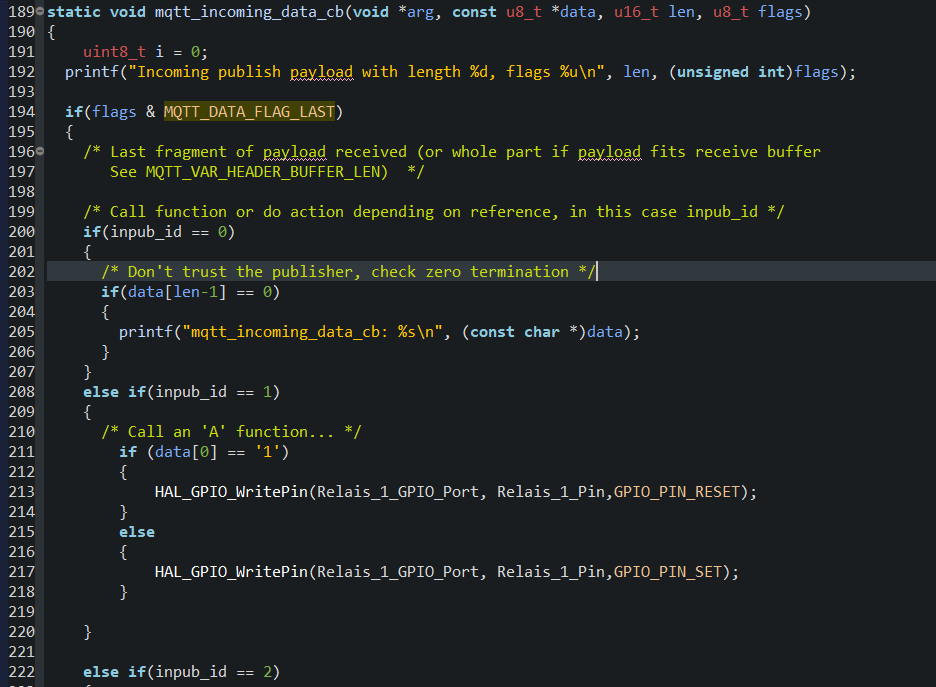
Figuur mqtt\_connection\_cb

* **static void mqtt\_incoming\_publish\_cb(void \*arg, const char \*topic, u32\_t tot\_len)**Deze callback wordt aangroepen wanneer er een bericht op een topic binnenkomt, hier wordt er gekeken naar het desbetreffende topic en er wordt een id gegeven.

****

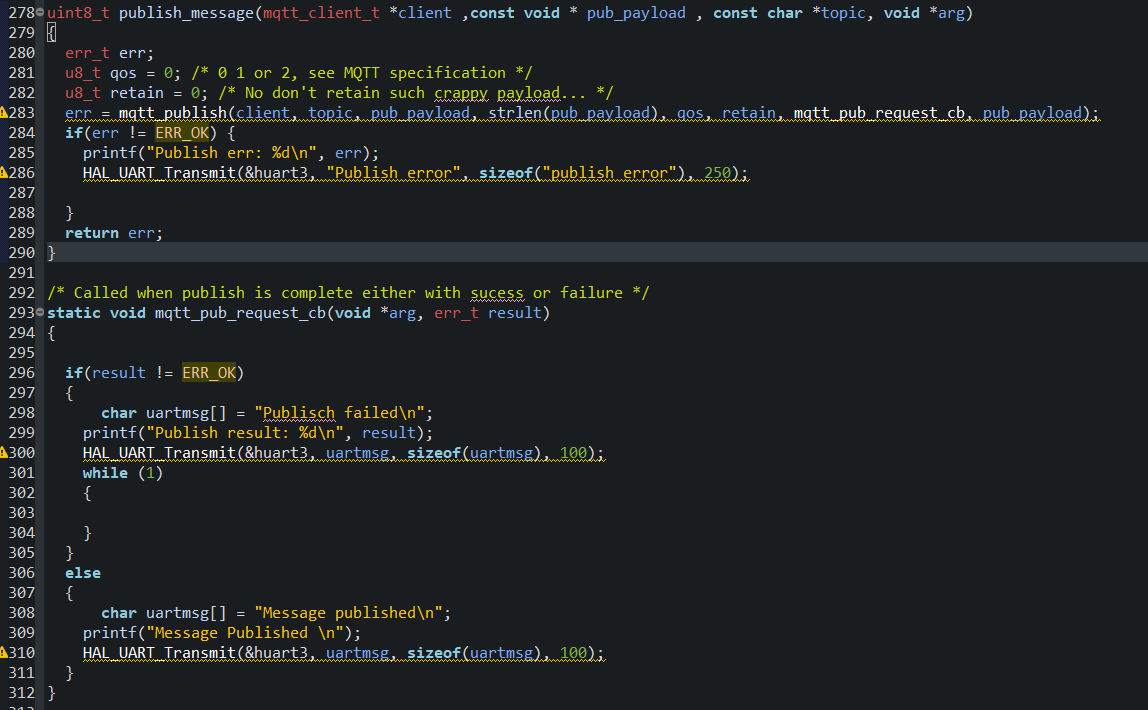
Figuur 38 mqtt\_incoming\_publisch\_cb

* **static void mqtt\_incoming\_data\_cb(void \*arg, const u8\_t \*data, u16\_t len, u8\_t flags)**Deze callback wordt ook aangeroepen als er een bericht is binnengekomen, hier gaan we vooral kijken op welk id er iets is binnengekomen en wat de data is. Hier gaan we er ook voor zorgen dat we relais gaan schakelen. Als we via Home Assistant een commando sturen voor het relais te schakelen gaat dat hier toekomen.

****

Figuur 39 mqtt\_incoming\_data\_cb

* **uint8\_t publish\_message(mqtt\_client\_t \*client ,const void \* pub\_payload , const char \*topic, void \*arg)**Met deze functie gaan we berichten versturen.

****

Figuur mqtt publisch\_message

## FreeRTOS

FreeRTOS CMSIS\_V1 is het operating system dat ik heb gebruikt in dit project.

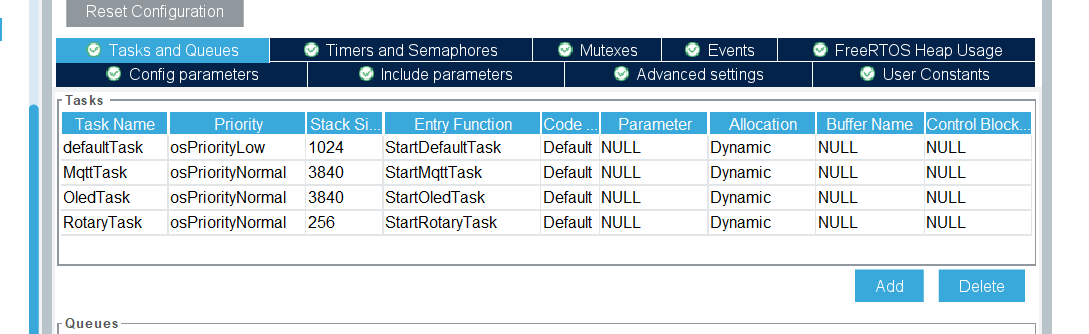
### Configuratie

In de configuratie heb ik gebruik gemaakt van HEAP size 75000 bytes en maak ik gebruik van heap\_3.



Figuur CubeMX RTOS memory

Het volgende dat is ingesteld zijn den tasks met hun priority’s en stacksize en de queue.

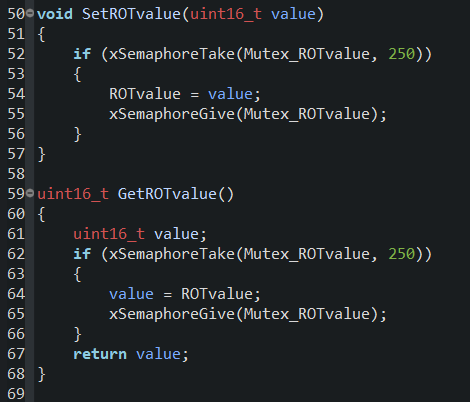


Figuur 42 CubeMX RTOS tasks en queue

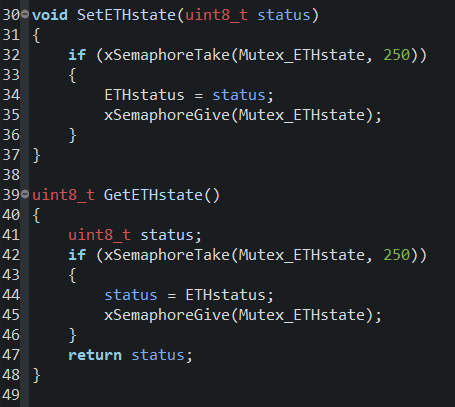
### Global setters & getters

Setters en Getters worden in dit geval gebruikt om data in of uit een shared resource te halen. We maken hier gebruik van een mutex, van het moment dat task A één van deze functies aanroept zal niemand anders toegang krijgen tot deze shared resource. Pas wanneer task A klaar is kan task B aan de shared resource. Ik heb dit gebruikt voor:

* **Rotary encode value**Omdat de value van de rotary encoder een bijna alle tasks wordt gebruikt is het een must deze te beveiligen.

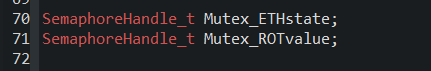
****

Figuur getter-setter rot value

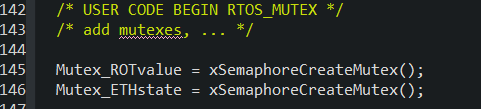
* **ETH state**Omdat ik ervoor moet zorgen dat de mqtt initialisatie pas wordt uitgevoerd wanneer de ethernet connectie up is ga ik er op deze manier voor zorgen. Deze setter wordt pas gebruik wanneer de ethernet initialisatie voltooid is. Pas dan worden alle functies van mqtt vrijgegeven.  
    
  ****

Figuur 44 getter-setter ETH state

De mutexen worden geinitialiseerd in de main file.



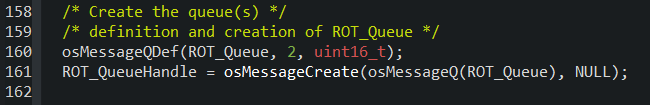
Figuur mutex handles



Figuur mutex create

### Queue

Omdat de rotary value uit een interrupt routine komt, wat trouwens geen deel uitmaakt van RTOS maar onafhankelijk ervan werkt kan dit niet zomaar als een globale variabele gedeeld worden. Mutexen kunnen ook niet gebruikt worden in een interrupt dus heb ik gekozen voor een queue. Het aanmaken gebeurd in de main file.



Figuur queue create

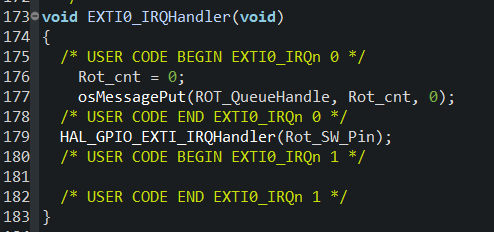
#### Queue interrupt routines

De queue ga ik als eerst gebruiken in de interrupt routine van de rotary encoder die ik al had toegelicht in [4.1.2 Test](#_Test). Hier da ik de value in de queue steken.



Figuur 48 queue put value rotary encoder

Verder ga ik in de routines ook gebruik maken van een reset knop waarmee de rotary value wordt gereset naar 0. Hier ga ik ook gebruik maken van de queue.



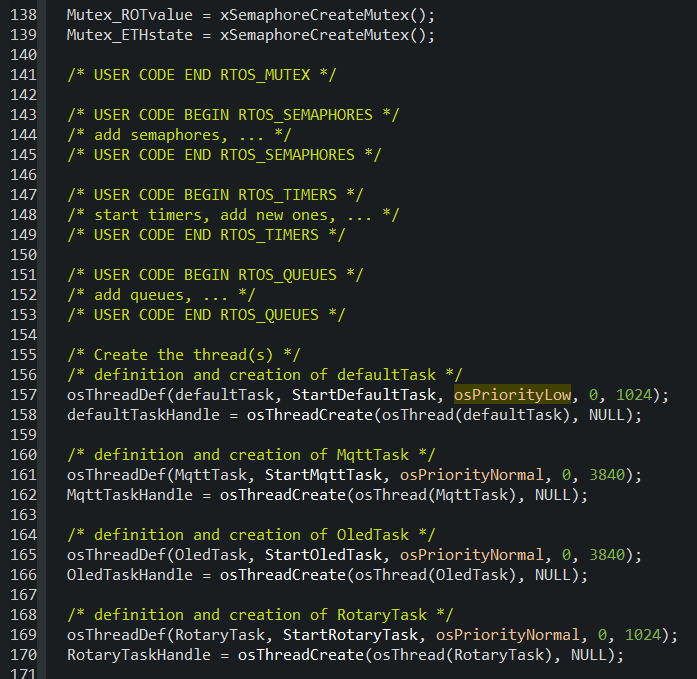
Figuur 49 queue put value reset rotary encoder

### Tasks

Zoals in de configuratie al is getoond heb ik gebruik gemaakt van 4 thread/tasks.

* DefaultTask
* MqttTask
* OledTask
* RotaryTask

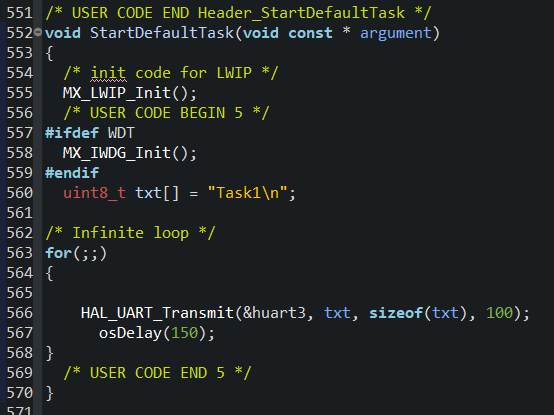
Het aanmaken van deze tasks gebeurt in de main file



Figuur create tasks

#### DefaultTask

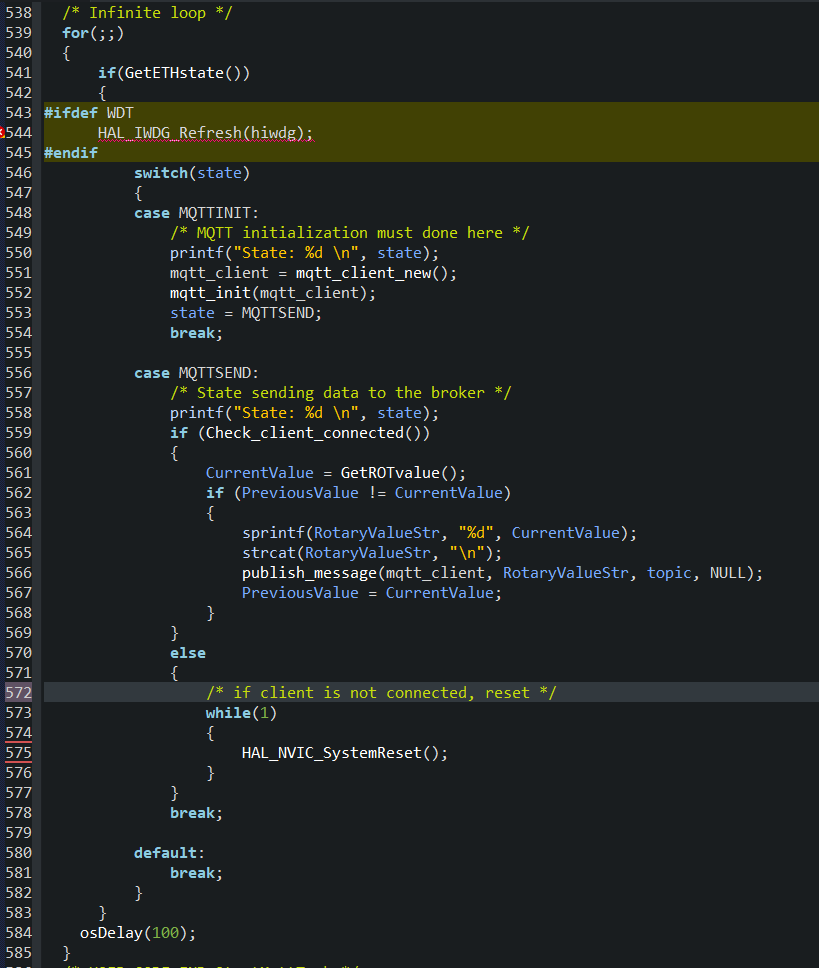
In de default task wordt er niets gedaan, enkel de initialisatie van LwIP en de watchdogtimer. Ik heb daarom ook gekozen om de priority op low te zetten aangezien we buiten de LwIP initialisatie en de Watchdogtimer niets hiermee doen.



Figuur DefaultTask

#### MqttTask

Deze task zorgt voor alles wat met MQTT te maken heeft, er zit een kleine state-machine in die er voor zorgt dat de initialisatie maar één keer gebeurd. Eens dat gedaan is wordt er enkel nog maar de value van de rotary encoder gestuurd via publish message. Voor dit gedaan wordt gaan we eerst de connectie met de broker controleren. Als dit niks oplevert gaan we naar een while(1) loop gaan en het systeem resetten, als er geen connectie is kunnen we niet veel meer doen, vandaar dat dit belangrijk is. Als we geconnecteerd zijn gaan we de rotary value halen met de rotary getter en kijken of deze veranderd is ten opzichte van de vorige value. Is dit het geval gaan we de value converteren naar een string en een linefeed eraan toevoegen. vervolgens wordt dit verzonden.



Figuur MqttTask

#### OledTask

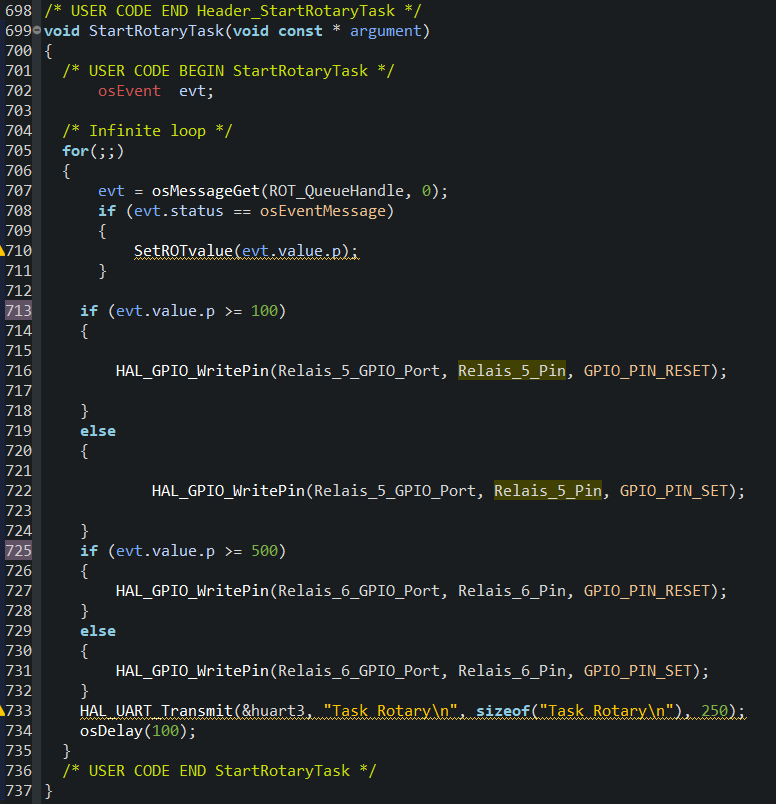
Deze task verzorgd alles wat met het display te maken heeft. Eerst gaat het de layout verzender die onveranderd blijft gedurende de applicatie. Daarna gaat het net zoals de MqttTask de rotary value ophalen met de rotary getter. Daarna gaan we kijken of deze veranderd is. Als deze veranderd is en de value is 0 dan gaan we de array helemaal leeg maken zodat de overige getallen worden verwijderd uit de array. Anders blijft dit op het scherm. Daarna gaan we de waarde als karakters in een aray steken. Dit wordt dan geprint op het scherm.



Figuur OledTask

#### RotaryTask

In de rotary task gaan we eerst controlleren of er een nieuwe waarde in de queue zit. Deze gaan we dan opslaan in de Rotary setter. Die we dan in al de andere tasks gaan gebruiken. Dit doe ik omdat wanneer ik dit uit de queue haal deze waarde ook effectief uit de queue wordt verwijderd. Daarom dat we deze gaan opslaan in de rotary setter. Die kunnen we dan blijven gebruiken totdat ze hier weer veranderd wordt. Verder ga ik controlleren als de opgehaalde waarde boven de 100 en boven de 500 zit zodat we hier ook weer relais kunnen schakelen.



Figuur RotaryTask