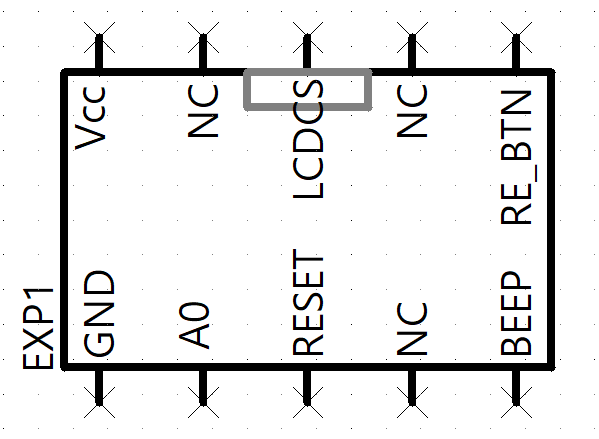
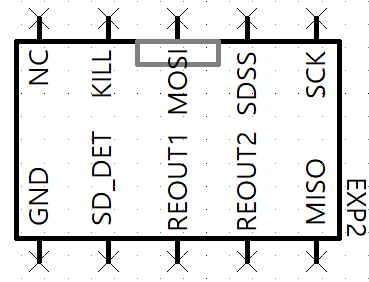
**Labo GLCD aansturen**

**Doelstelling**

Het doel van dit labo is : uitzoeken hoe we de grafische LCD kunnen aansturen. We gebruiken een bordje met de naam “MKS MINI12865 V2.0”. Het bestaat uit de LCD, zijn driver, een Rotary encoder, een buzzer, een SD-kaartlezer en een stopknop. Omdat ik de SD-kaartlezer en de knop niet nodig heb, worden ze niet besproken in dit labo.

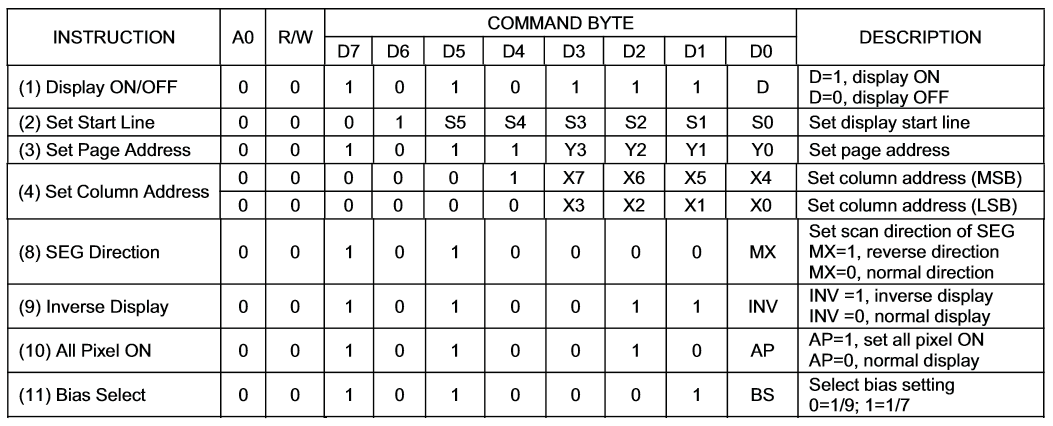
**Theorie**

Eerst zal ik de pin-out van de printplaat uitleggen en dan kan ik daar verder op opbouwen. Op de PCB zitten twee 10 pin connectoren genaamd EXP1 en EXP2. Ik bespreek alleen de pinnen die ik nodig heb. Dus alles met SD en KILL valt weg.

* Vcc, hier sluit je de voeding aan die tot max +12V mag zijn door een ingebouwde spanningsomvormer.
* GND, het is de ground van de hele PCB.
* A0, deze pin laat de driver weten of de gestuurde data een instructie is of pixeldata (instructies wanneer laag).
* LCDCS, bepaalt wanneer de slave inkomende data mag inlezen (mag inlezen wanneer laag).
* RESET, wordt gebruikt om een hardware reset te doen aan het begin van het programma om alle instellingen te resetten.
* BEEP, deze pin is voor de buzzer, die moet gewoon hoog gemaakt worden en de buzzer zal piepen zolang de pin hoog blijft.
* RE\_BTN, de output pin van het drukknop gedeelte van de Rotary encoder.
* REOUT1/2, de 2 output pinnen van de Rotary encoder.
* MOSI, de Master Out Slave In pin voor SPI. (Wordt nog meer uitgelegd)
* MISO, de Master In Slave Out pin voor SPI.
* SCK, de Serial Clock pin voor SPI0

De LCD heet 12864B11 en heeft een resolutie van 128x64. De driver van de LCD is de ST7567. Met instructies die nog zullen worden uitgelegd, kunnen we met die driver elke aparte pixel van de LCD aansturen.

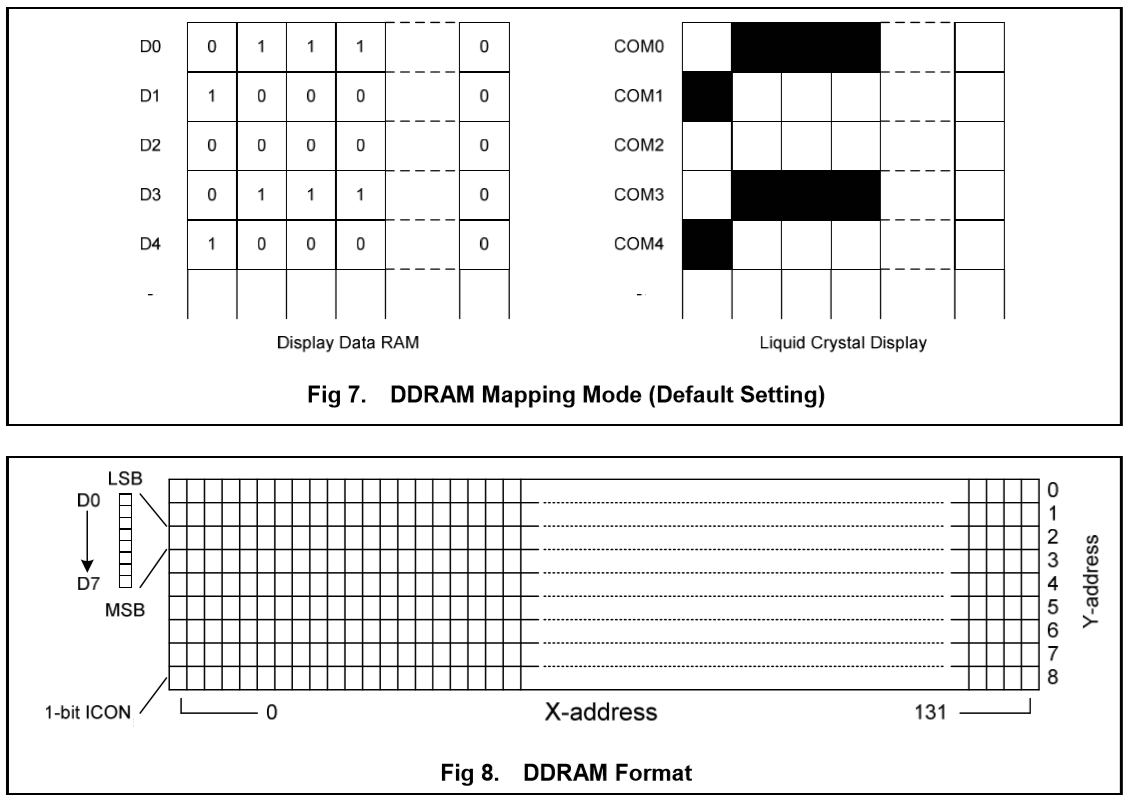
Maar voor we de pixels kunnen aansturen moeten er eerst een aantal regels vastgelegd worden. Eerst moet de initialisatie gebeuren. Hierbij gaan we de RESET pin gebruiken. Die moet heel kort laag en terug hoog gemaakt worden. Daardoor worden alle interne registers klaar gemaakt voor gebruik en gereset.

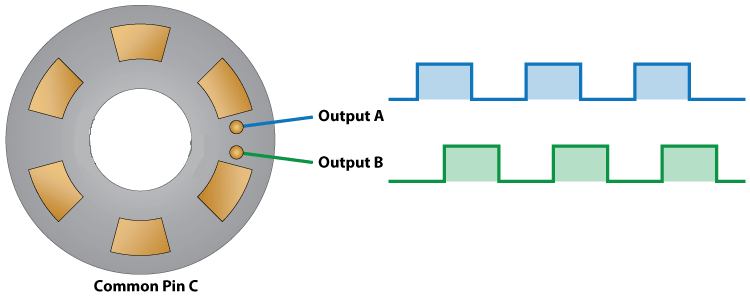
Na de reset moeten ook een aantal registers geconfigureerd worden. De registers zijn zichtbaar in de (verkorte) tabel hieronder vanuit de datasheet. Omdat het instructies zijn, moet je als je 1 van deze registers wil configureren, A0 laag maken. Instructie 2, 3 en 4 worden in het begin niet geconfigureerd want ze bepalen de coördinaat van de pixel die je wil aan of uit zetten.

Dit is hoe ik de drivers zou moeten configureren :

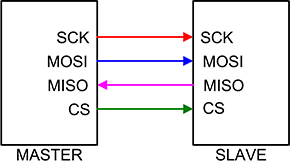
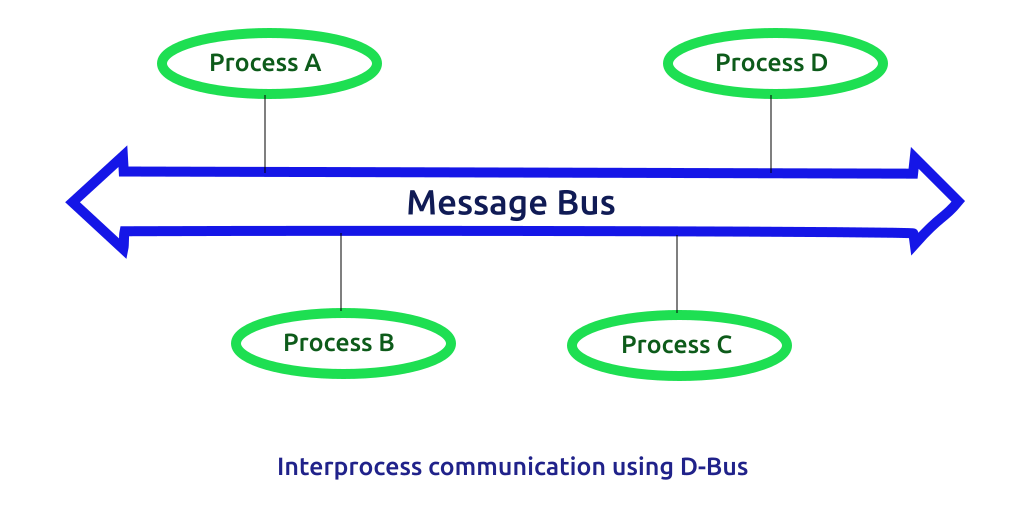
* Display OFF dus 0b1010 1110.
* SEG Direction normal dus 0b1010 0000.
* Inverse Display OFF dus 0b1010 0110.
* All Pixels normal dus 0b1010 0100.
* Bias is 1/9 dus 0b1010 0010

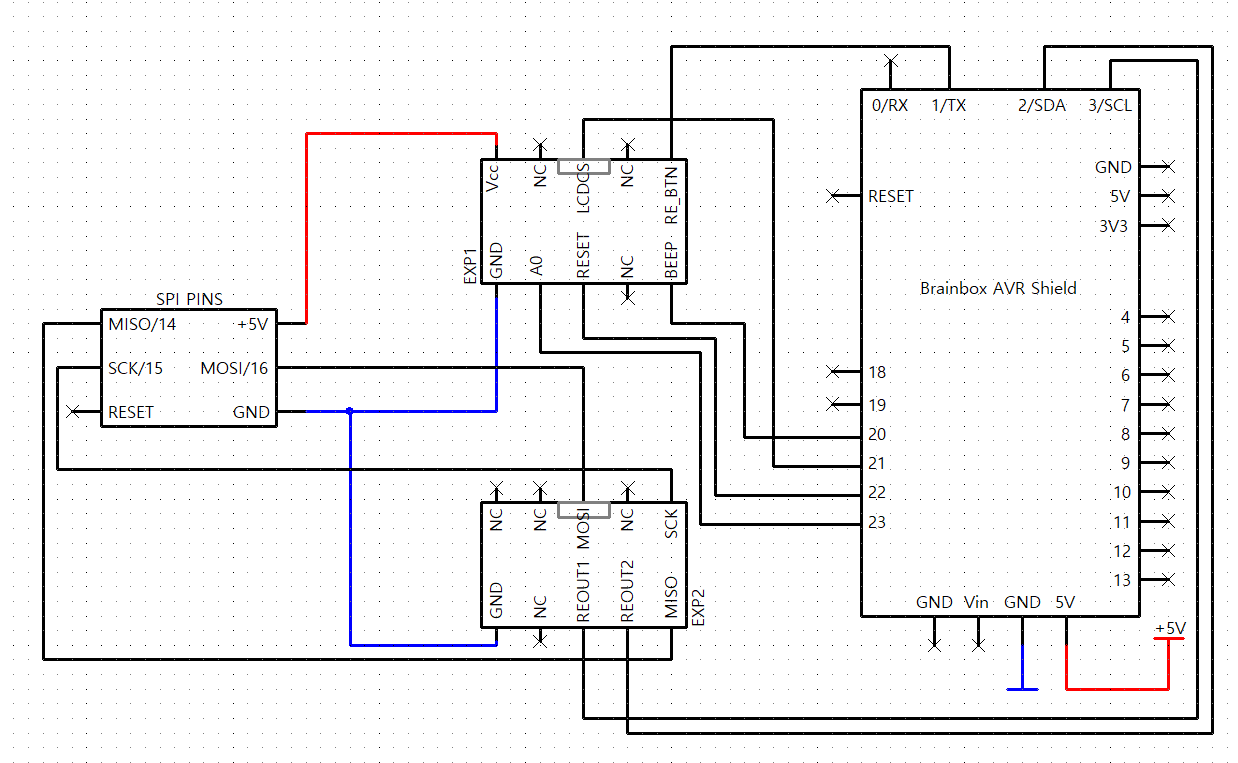
Het scherm bestaat uit 9 rijen / “pages” / Y coördinaten en 132 kolommen / columns / X coördinaten. Elke column bestaat uit 8 bits. Deze pixels worden allemaal individueel aangestuurd door het display data RAM / DDRAM van de driver. Dus als je een pixel zou willen aansturen zou je eerst de “page” en “column” moeten bepalen van de pixels die je wil aansturen en ook met een 8-bit getal bepalen welke pixels aan en welke uit zullen zijn (zichtbaar op de foto hieronder).

Waar ook rekening mee moet gehouden worden is het Display register. Dat register bepaalt wanneer de opgeslagen data in het DDRAM moet gedisplayd worden op de LCD, het zorgt voor de verbinding tussen de 2. Dus bij de initialisatie van het programma moet display OFF zijn (0b1010 1110). Daarna bepaal je de positie (de page en column) en het 8-bit getal dat bepaalt welke pixels aan en welke pixels uit zullen zijn. Tenslotte maak je dan terug een verbinding met de LCD en DDRAM door display ON te zetten (0b1010 1111) en zal de ingeschreven data gedisplayd worden.



Dit is hoe de binnenkant van een Rotary encoder er uit ziet. Maar eerst zal ik het nut van een Rotary encoder uitleggen. Het is net zoals een potentiometer, maar er is geen einde aan het draaien naar links/rechts. Hierdoor is het perfect voor projecten waarbij bijvoorbeeld door een menu moet gescrold worden. Een Rotary encoder heeft 2 uitgangen genaamd A en B maar ook een uitgang voor de knop en de GND. Wanneer er aan de Rotary encoder wordt gedraaid zal er een plaatje dat verbonden is aan de GND ook meedraaien (de vorm van het plaatje is een beetje zoals de toerentalteller plaat in het PID verslag). Dat plaatje zal contact maken met 2 aparte plaatjes die verbonden zijn aan A en B. Doordat het grote plaatje bijvoorbeeld eerst A aanraakt en daarna B kunnen we zien welke uitgang eerst hoog wordt en daaruit de draairichting bepalen.

SPI is een vorm van seriële communicatie via een bus, zoals I2C. Het bestaat uit 4 of 3 draden (de brainbox heeft CS niet). De SCK is de seriële klok die de datasnelheid bepaalt net zoals bij I2C. De MOSI is de draad waar de master data gaat sturen naar de slave. De MISO is de draad waar de slave data gaat terugsturen naar de master. Ten sloten de CS is al uitgelegd bij de pin-out van de GLCD.

**Schema**

**Beschrijving**

Het is een simpel programma. Het gebruikt de 3 componenten die in de doelstelling staan. Wanneer je aan de rotary encoder draait in wijzerzin dan zal er een waarde (zichtbaar op de seriële monitor) vergroten en als je tegenwijzerzin draait dan zal die verminderen. Als je dan op de knop (op de rotary encoder) zou drukken, dan zullen alle pixels op het scherm aan gaan en zal de buzzer ook even biepen. Als je voor de 2de keer drukt dan gaan de pixels terug uit en biept de buzzer ook natuurlijk.

**Code**

#include <SPI.h>

#define PD0 3

#define PD1 2

#define PD3 1

// A0 -> PF0

// RESET -> PF1

// CS -> PF4

// BEEP -> PF5

// RO\_1 -> PD0/SCL

// RO\_2 -> PD1/SDA

// RO\_Button -> PD3/TX

// Als 1 eerst laag is dan is het clockwise

// Functions voor wanneer er op de RO wordt gedrukt/gedraaid

void RO\_1**();**

void RO\_2**();**

void RO\_Button**();**

volatile boolean RO\_1\_Eerst**,** RO\_2\_Eerst**,** PixelsON **=** **false;** // Variable om te bepalen welke uitgang als eerste laag werd en of alle pixels al aan zijn of niet

boolean Tester1**,** Tester2**;** // Variable die bepalen wanneer er terug een interrupt mag zijn

long Teller **=** 0**;** // Teller om te zien hoeveel keer je al hebt gedraaid aan de RO

void setup**()**

**{**

attachInterrupt**(**digitalPinToInterrupt**(**PD0**),** RO\_1**,** FALLING**);** // Interrupt bij INT0 wanneer de pin van hoog naar laag gaat

attachInterrupt**(**digitalPinToInterrupt**(**PD1**),** RO\_2**,** FALLING**);** // Interrupt bij INT1 wanneer de pin van hoog naar laag gaat

attachInterrupt**(**digitalPinToInterrupt**(**PD3**),** RO\_Button**,** FALLING**);** // Interrupt bij INT3 wanneer de pin van hoog naar laag gaat

Serial**.**begin**(**9600**);**

SPI**.**begin**();**

PORTF **&=** 0b11111101**;** // Reset laag

delay**(**1**);**

PORTF **|=** 0b00000010**;** // Reset hoog

delay**(**1**);**

PORTF **&=** 0b11111110**;** // A0 laag

PORTF **|=** 0b00010000**;** // CS hoog

SPI**.**beginTransaction**(**SPISettings**(**4000000**,** MSBFIRST**,** SPI\_MODE0**));**

**}**

void loop**()**

**{**

**if(**RO\_1\_Eerst**)**

**{**

detachInterrupt**(**digitalPinToInterrupt**(**PD1**));** // Stop interrupts aan RO\_2

Teller**++;**

**while((**Tester1 **==** **false)** **|** **(**Tester2 **==** **false))**

**{**

Tester1 **=** digitalRead**(**PD0**);**

Tester2 **=** digitalRead**(**PD1**);**

**}**

RO\_1\_Eerst **=** **false;**

RO\_2\_Eerst **=** **false;**

attachInterrupt**(**digitalPinToInterrupt**(**PD1**),** RO\_2**,** FALLING**);** // Start interrupts terug aan RO\_2

**}**

**if(**RO\_2\_Eerst**)**

**{**

detachInterrupt**(**digitalPinToInterrupt**(**PD0**));** // Stop interrupts aan RO\_1

Teller**--;**

**while((**Tester1 **==** **false)** **|** **(**Tester2 **==** **false))**

**{**

Tester1 **=** digitalRead**(**PD0**);**

Tester2 **=** digitalRead**(**PD1**);**

**}**

RO\_1\_Eerst **=** **false;**

RO\_2\_Eerst **=** **false;**

attachInterrupt**(**digitalPinToInterrupt**(**PD0**),** RO\_1**,** FALLING**);** // Start interrupts terug aan RO\_1

**}**

Serial**.**println**(**Teller**);**

**}**

void RO\_1**()** // Interrupt functie wanneer 1 als eerste laag wordt

**{**

RO\_1\_Eerst **=** **true;**

RO\_2\_Eerst **=** **false;**

**}**

void RO\_2**()** // Interrupt functie wanneer 2 als eerste laag wordt

**{**

RO\_2\_Eerst **=** **true;**

RO\_1\_Eerst **=** **false;**

**}**

void RO\_Button**()** // Interrupt functie wanneer er op de knop wordt gedrukt

**{**

PORTF **|=** 0b00100000**;** // BEEP hoog

delay**(**1**);** // BEEP hoog houden voor 1ms

PORTF **&=** 0b11011111**;** // BEEP laag

**if(**PixelsON **==** **false)** // Zijn alle pixels uit

**{**

PORTF **&=** 0b11101111**;** // Chip select laag

SPI**.**transfer**(**0b10100101**);** // Zet alle pixels aan

PORTF **|=** 0b00010000**;** // Chip select hoog

PixelsON **=** **true;**

**}**

**else** // Zijn alle pixels aan

**{**

PORTF **&=** 0b11101111**;** // Chip select laag

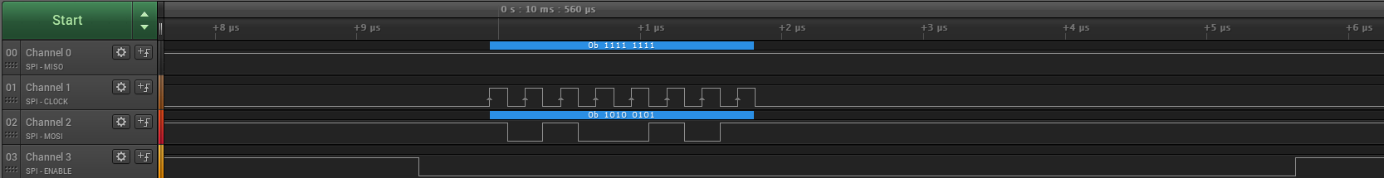
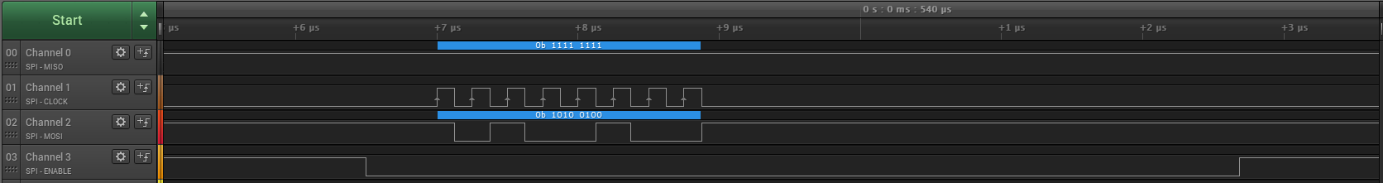
SPI**.**transfer**(**0b10100100**);** // Zet alle pixels uit

PORTF **|=** 0b00010000**;** // Chip select hoog

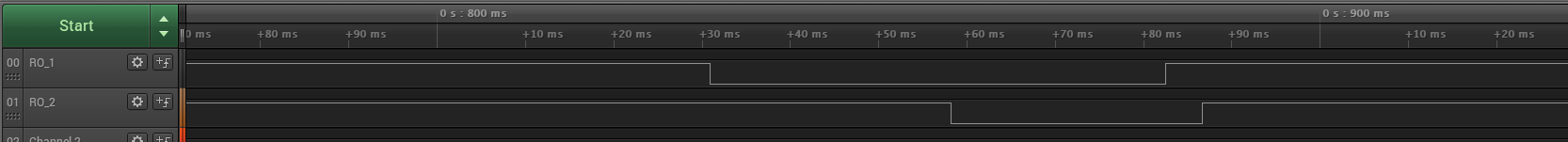
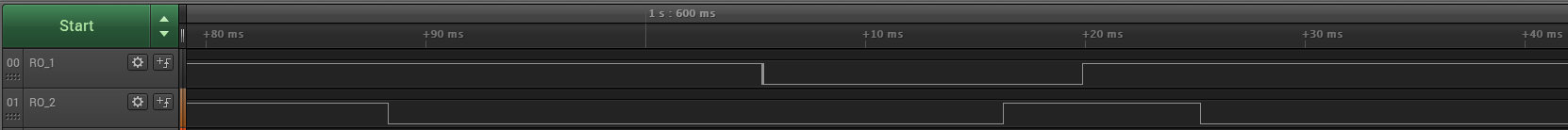
PixelsON **=** **false;**

**}**

**}**

**Metingen**

Hier zijn de 2 metingen van de data die naar de driver wordt gestuurd. We zien dat op de 1ste foto via de MOSI lijn 0b1010 0100 wordt gestuurd, dus alle pixels zijn uit dan. Op de 2de foto zien we dat via de MOSI lijn 0b1010 0101 wordt gestuurd, dus alle pixels zijn dan aan. Er wordt niets op de MISO lijn gestuurd want er moet niets terug naar de master worden gestuurd.



In deze 2 foto’s zien we duidelijk de 2 mogelijke uitkomsten uit de 2 uitgangen van de Rotary encoder. Bij de 1ste foto wordt RO\_1 of A eerst laag dus ik draaide in wijzerzin. In de 2de foto wordt RO\_2 of B eerst laag dus draaide ik in tegenwijzerzin.

**U8G2 Library**

De U8G2 library is een library voor meerdere GLCDs waarmee je op een simpele manier letters, figuren etc. op de LCD kunt displayen. De library kan een groot aantal LCDs aansturen maar dat maakt ons niet zo veel uit zolang het onze GLCD kan aansturen, wat de library ook kan.

Er zijn 3 modes waarmee de LCD kan worden aangestuurd. Single page, double page en full frame mode. Single page mode zorgt ervoor dat er 1 “page” wordt opgeslagen in het RAM van de brainbox. Double page mode doet hetzelfde als single page mode maar dan met 2 “pages” i.p.v. 1. Tenslotte doet full frame mode ook hetzelfde maar dan met een volledige kopie van de display. Dat is de snelste van de 3 modes, maar een nadeel ervan is dat die ook het meeste RAM van de brainbox inpakt.

**Besluit**

We hebben 2 manieren gevonden om de GLCD aan te sturen. Via SPI of via de library. Het is duidelijk dat we gaan met de library. De reden hiervoor zijn simpel : het is heel gemakkelijk te gebruiken, het maakt de code zeker een stuk simpeler en minder groot en het kan ons veel tijd besparen. Ik moet nog alleen later beslissen welke van de 3 modes ik ga gebruiken. De reden hiervoor is dat ik niet weet hoeveel procent van het RAM over gaat blijven na de “main” code voor de robot.