6 Seriële communicatie, het OLED display en de SD kaart

Deze week gaan we verder met programmeren. De basis is vorige week gelegd, deze wordt nu uitgebreid zodat alle componenten op de SMU uitgelezen of aangestuurd kunnen worden.

6.1 Seriële data uit de P1-poort

De SMU is gemaakt om gebruiksgegevens uit de slimme meter af te lezen. Zoals we eerder hadden gezien komt die data beschikbaar via de P1-poort. Tijd om die in wat meer detail te gaan bekijken.

THUISOPDRACHT 1:

Zoek nog eens op hoe de fysieke aansluiting van de P1-poort eruit ziet.

a) Hoeveel pinnen heeft de poort?
b) Wat is de functie van iedere pin?
c) Welke is/zijn van belang voor de data overdracht?

In de vorige weken heb je gezien dat de microcontroller pinnen kan uitlezen door te kijken of er een hoog of een laag signaal op staat. In principe kan je volgens deze methode 1 bit tegelijk versturen (een hoog of een laag signaal). Willen we verbruiksgegevens van de slimme meter versturen dan hebben we meer nodig dan 1 enkele bit. Er zijn twee manieren waarop we dat kunnen doen: parallelle en seriële dataoverdracht.

THUISOPDRACHT 2:

a) Zoek op wat parallelle dataoverdracht inhoudt.
b) Zoek op wat seriële dataoverdracht is.
c) Wat zijn de voor- en nadelen van de twee methodes?

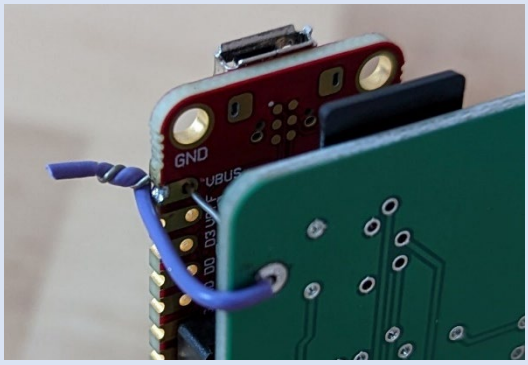
Via de P1-poort wordt data serieel verzonden. Welke data wordt verstuurd en hoe dat gaat is vastgelegd in een *protocol*. Een protocol is een standaard set aan regels waardoor elektronische apparaten met elkaar kunnen communiceren. In de Dutch Smart Meter Requirements (DSMR) staat precies beschreven hoe dit protocol er voor de slimme meter uitziet. Het hangt van de slimme meter die je thuis hebt af welk protocol jouw meter heeft: DSMR 2.2, DSMR 4.0, DSMR 4.2 of DSMR 5.0.

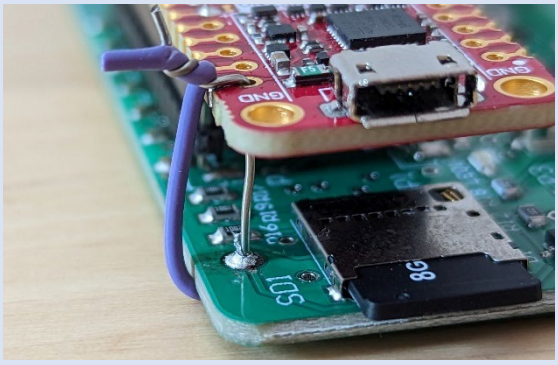
Let op: de DSMR2.2 meter stuurt elke 10 seconden een telegram, de andere meters elke seconde!

THUISOPDRACHT 3:

Geef het huidige verbruik van je slimme meter thuis weer op het OLED display:

Voor je data uit de P1-poort kan uitlezen is het belangrijk de VBUS-pin van het rode bordje door te verbinden met de onderliggende pad op de groene PCB. Dit kan met een los draadje, je hoeft niet te solderen:





Vervolgens kan je, afhankelijk van de DSMR versie van je meter, de hex-file op de microcontroller zetten:

- P1ShowCurrentUseOLED_dsmr_v2.hex voor DSMR2.2 meters
- P1ShowCurrentUseOLED_dsmr_v5.hex voor de andere DSMR meters

Verbind nu het SMU-bord met de P1-poort via de kabel die je te leen hebt. Deze past op de zes-pins header links op de PCB. Let op: het SMU-bord moet uiteraard ook nog steeds via de USB-kabel met je laptop verbonden zijn, anders heeft hij geen voeding!

Maak een foto van het OLED-display en laat zien dat het werkt.

THUISOPDRACHT 4:

Log en analyseer gedurende minimaal 24 uur het huidige verbruik van je slimme meter thuis:

1. Formatteer je SD kaart FAT32. (Zorg zelf voor een SD-kaart!)
2. Steek de SD kaart voorzichtig in de SMU print.
3. Gebruik, afhankelijk van de DSMR versie:
    - P1LogCurrentUseToSD_dsmr_v2.hex voor DSMR2.2 meters
    - P1LogCurrentUseToSD_dsmr_v5.hex voor de andere DSMR meters
4. Zet na 24 uur de SMU uit door de USB voeding los te halen.
5. Verwijder voorzichtig de SD kaart en analyseer de data (bijvoorbeeld met excel).

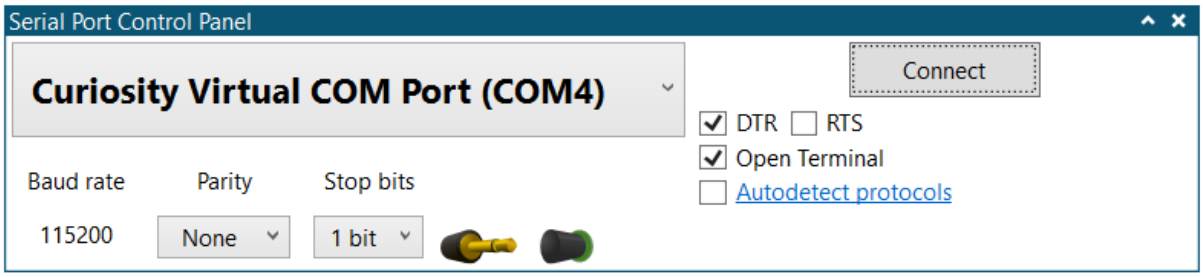
a) Wat is je sluipverbruik[7] en waardoor wordt dit veroorzaakt?
b) Zie je grote pieken in verbruik? Kan je die linken aan apparaten die toen aangingen?

[7] Sluipverbruik of lekkende elektriciteit is het gebruik van elektriciteit zonder dat men daar voordeel van heeft.

6.2 Seriële data uit de microcontroller

Je kunt de microcontroller ook serieel data laten versturen. Hiervoor gebruikt de microcontroller een stukje *periferie* (randapparatuur) genaamd de *UART* (Universal Asynchronous Receiver Transmitter).

De data die via de UART verstuurd wordt kan je zichtbaar maken op een Terminal. Deze kan je openen in Microchip Studio via: Tools → Data Visualizer (…→ Configuation → External Connection → Serial Port). Zorg dat je hierbij de Baud rate goed instelt (zie onderstaande opdracht), vink aan dat je een terminal wil zien en klik op Connect om contact te maken.



THUISOPDRACHT 5:

a) De snelheid waarmee data wordt verzonden wordt uitgedrukt in Baud. Zoek op wat dat betekent.
b) Zoek in de P1 companion standard van de DSMR op hoe snel achter elkaar er data wordt verzonden via deze poort.
c) Zoek in het DSMR protocol op hoe het telegram er uit ziet welke jouw meter verstuurt.

Gebruik P1ViewTelegram_dsmr_v2.hex of P1ViewTelegram_dsmr_v5.hex om het telegram zichtbaar te maken via de UART en de Terminal.

d) Vergelijk de informatie uit het protocol met een telegram van je slimme meter. Geef per regel aan welke informatie er wordt gegeven.
e) Beredeneer aan de hand van het telegram of je 1 of 3 fase aansluiting hebt.

Er zijn behoorlijk wat registers nodig om de UART juist te kunnen gebruiken. De code om de UART te kunnen gebruiken krijg je cadeau (zie ELO: UARTtemplate.c) maar het is wel belangrijk dat je weet hoe die code werkt, daarom staat deze hieronder stukje bij beetje uitgelegd.

De code begint met het inladen van een aantal header files en het definiëren van de kloksnelheid:

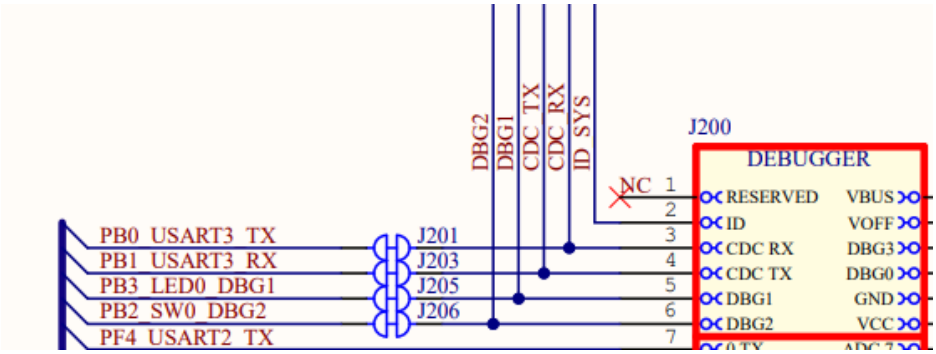
```
#define F_CPU (4000000UL)

#include <avr/io.h>
#include <avr/cpufunc.h>
#include <util/delay.h>
#include <string.h>
```

De klok wordt dus op 4MHz ingesteld. De header files worden gebruikt om een aantal functies te kunnen gebruiken die anders niet gedefinieerd zouden zijn. Avr/io.h bevat informatie over de IO-pinnen van onze microcontroller, avr/cpufunc.h bevat functies voor de CPU, util/delay.h zorgt ervoor dat we delay-functies kunnen gebruiken en string.h maakt het mogelijk om met arrays te werken.

In de regel die volgt wordt de waarde van het baudrate register uitgerekend, afhankelijk van de CPU klokfrequentie:

```
#define USART3_BAUD_RATE(BAUD_RATE)     ((float)(64 * F_CPU / (16 * (float)BAUD_RATE)) + 0.5)
```

We programmeren de microcontroller via de 'debugger'. In het schema van het microcontroller bordje kunnen we nagaan hoe de debugger met de microcontroller verbonden is:



De data die serieel van onze computer via de USB-kabel naar de microcontroller gaat moet via UART doorgegeven worden. In het bovenstaande schema zie je dat pin PB0 en pin PB1 van de microcontroller verbonden zijn met USART3. De microcontroller gebruikt 2 pinnen voor de seriële communicatie: eentje om te zenden (TX, verbonden met PB0), en eentje om te ontvangen (RX, verbonden met PB1). Deze pinnen moeten dus ook juist geconfigureerd worden: TX moet immers een uitgang worden en RX een ingang.

In de code worden deze pinnen als in- en output ingesteld in de *USART3_init* functie (zie hieronder). Behalve het instellen van de juiste pinnen wordt hierin ook de baudrate ingesteld, oftewel hoe snel de data verstuurd moet worden, en uit hoeveel data bits de informatie bestaat. Als laatste instructie wordt de transmitter aangezet.

```
void USART3_init(void)
{
    // AVR128DB48
    PORTB.DIRSET = PIN0_bm;                                 /* set pin 0 of PORT B (TXd) as output*/
    PORTB.DIRCLR = PIN1_bm;                                 /* set pin 1 of PORT B (RXd) as input*/

    USART3.BAUD = (uint16_t)(USART3_BAUD_RATE(9600));   /* set the baud rate*/

    USART3.CTRLC = USART_CHSIZE0_bm | USART_CHSIZE1_bm; /* set the data format to 8-bit*/

    USART3.CTRLB |= USART_TXEN_bm;                          /* enable transmitter*/
}
```

Dan volgt een functie om een karakter te versturen. Daarvoor wordt eerst gewacht tot de vorige transmissie klaar is, en vervolgens het te verzenden karakter klaargezet om te versturen:

```
void USART3_sendChar(char c)
{
    while(!(USART3.STATUS & USART_DREIF_bm))
    {
        ;
    }

    USART3.TXDATAL = c;
}
```

Tot slot volgt een functie om een string, oftewel een aantal karakters achterelkaar, te versturen:

```
void USART3_sendString(char *str)
{
    for(size_t i = 0; i < strlen(str); i++)
    {
        USART3_sendChar(str[i]);
    }
}
```

Het versturen gaat door met een for loop door de string te lopen, en dan karakter voor karakter te versturen met de *USART3_sendChar* functie.

Vanuit je eigen code (in de mail) kun je nu deze functie gebruiken om data te printen en op je laptop zichtbaar te maken. Om de UART te kunnen gebruiken typ je eerst *USART3_init();*

```
int main(void)
{
    USART3_init();

    while (1)
    {
        USART3_sendString("Hello World! \r\n");
    }
}
```

LABOPDRACHT 1:

a) Maak een programma voor de microcontroller waarbij je gebruik maakt van bovenstaande functies in het UARTtemplate.c, en die elke seconde je studentnummer naar de UART stuurt.
   Test je programma met het Terminal Window (te vinden onder het menu Tools → Data Visualizer).
b) Meet met de oscilloscoop hoe snel de microcontroller data verstuurt bij 9600 baud.
c) Pas de baudrate aan in de code naar 19200 baud en herhaal de meting.

Dit lijkt al een beetje op de printf functie die we eerder hebben gezien, maar wat als ik een waarde wil printen? Dat werkt niet met de *USART3_sendString* functie. Je zult dan eerst een stuk tekst moeten formateren en die vervolgens met *USART3_sendString* moeten versturen. Daarvoor gebruik je niet de *printf* maar de *sprintf* functie. Deze print de tekst in een buffer van karakters die je vervolgens kunt versturen: een tweetraps raket dus! Moeilijk? Nee hoor:

```
uint8_t counter = 10;
char buffer[20];

sprintf(buffer, "%d", counter);
USART3_sendString(buffer);
```

De P1-poort verstuurt dus een grote hoeveelheid informatie achter elkaar. Het zou leuk zijn als we (een deel van) die informatie direct kunnen aflezen. Het 7-segementsdisplay is hiervoor niet geschikt: hierop kan maar één karakter tegelijkertijd worden afgebeeld. Tijd dus om een ander display te leren kennen: het OLED display.

6.3 Het OLED display

Om meer dan één karakter tegelijk af te beelden gaan we gebruik maken van het OLED (Organic Light Emitting Diode) display op de SMU. Dit lijkt op de LED waarmee je al eerder kennis hebt gemaakt, maar dit display bevat een heleboel LED's, zogenaamde pixels, welke oplichten.

De pixels van het OLED display zijn uiteraard niet allemaal afzonderlijk verbonden met de microcontroller. De microcontroller op de SMU "praat" met een microcontroller die op het OLED display zit. Deze communicatie gaat via een tweedraads protocol genaamd I^2C. Hoe dit protocol precies werkt leer je in het tweede jaar van deze studie, voor nu is het voldoende te weten dat de microcontroller berichten stuurt om karakters te printen of een lijn te tekenen naar de microcontroller van het OLED display, en deze gaat vervolgens de individuele pixels aansturen.

Zoals gezegd is het OLED display een complex stukje elektronica welke "praat" met onze microcontroller. Je stuurt dus commando's naar het display. Deze commandostructuur kun je vinden in de datasheet en het vergt nogal wat leeswerk om deze handmatig om te zetten in code. Daarom krijg je in dit geval een aantal functies waarmee je het OLED display kunt gebruiken. Deze functies komen uit de GLCD library van github (https://github.com/efthymios-ks/AVR-SSD1306). We gebruiken deze library in onze template code (zie OLEDtemplate.c op ELO):

```
#define F_CPU  (4000000UL)

#include <avr/io.h>
#include <avr/cpufunc.h>
#include <util/delay.h>
#include <string.h>
#include "OLED/SSD1306.h"
#include "OLED/Font5x8.h"

int main(void)
{
	GLCD_Setup();
	GLCD_SetFont(Font5x8, 5, 8, GLCD_Overwrite);

	int i = 0;
	char buffer[10];

	while(1)
	{
		GLCD_GotoXY(10,25);
		sprintf(buffer, "%3d", i);
		GLCD_PrintString(buffer);
		GLCD_Render();
		_delay_ms(1000);
		i++;
	}
}
```

Functie	Toelichting
`GLCD_Setup();`	Voordat je het display kunt gebruiken moeten we het initialiseren. Ook worden in deze functie de I2C communicatiepinnen geïnitialiseerd.
`GLCD_SetFont(Font5x8, 5, 8, GLCD_Overwrite);`	Met deze functie zet je het lettertype van het display goed. Op dit moment is er alleen ondersteuning voor een 5x8 font.
`GLCD_GotoXY(20, 10);`	Met deze functie kun je de positie bepalen waar de tekst wordt geplaatst.
`sprintf(buffer, "%3d", i);`	Met deze functie kan je een variabele omzetten naar tekst.
`GLCD_PrintString(buffer);`	Met deze functie kun je een tekst weergeven. De tekst wordt weergegeven op de positie die je hebt ingesteld met de GLCD_GotoXY functie. Als je een variabele wilt weergeven moet je deze omzetten naar een tekst.
`GLCD_Render();`	Wellicht de meest belangrijke functie: de GLCD_Render functie zorgt er voor dat de data die klaargezet is naar het display wordt verstuurd.

LABOPDRACHT 2:

a) Ga na of je bovenstaande voorbeeldcode begrijpt en probeer te voorspellen wat er op het OLED display zal verschijnen.
b) Zet de voorbeeldcode op je microcontroller en kijk of er gebeurt wat je verwachtte.
c) Zorg nu dat er rechtsonder op het scherm geprint wordt ipv de oorspronkelijke plek.
d) Voeg bovenaan het scherm een zin naar keuze toe.

LABOPDRACHT 3:

a) Schrijf een programma welke optelt van 0 -> 9 en dan weer naar 0. Wacht steeds een seconde voor je de waarde verhoogt. Geef het cijfer weer op het OLED display.
b) Vergelijkbaar met vraag a), maar nu gebruik je de drukknop om de variabele te verhogen. Wat gebeurt er als je de knop ingedrukt houdt?
c) Maak je programma nu zo dat alleen bij een opgaande flank (dus als je de knop *in*drukt) de teller wordt verhoogd.

EXTRA OPDRACHT VOOR CREATIEVE STUDENTEN:

a) Behalve de functies die genoemd zijn in het stukje voorbeeldcode bevat de GLCD library nog meer functies. Zie de eerder genoemde github site. Maak een overzicht van een aantal van deze functies.
b) Maak met behulp van functies naar keuze een zo creatief mogelijk ontwerp voor op je OLED display. Maak hier een foto van en lever deze in via ELO.

6.4 SD kaart

Als derde mogelijkheid om data uit te sturen (naast UART en OLED-display) bevat de SMU een SD kaartslot. Hierin kun je een micro SD kaart stoppen welke je kunt lezen en schrijven. De SMU gebruikt deze SD kaart om het verbruik te loggen. De SD kaart wordt in de SPI mode gebruikt, wat wil zeggen dat er via een paar aansluitingen serieel data wordt uitgewisseld.

De SD kaart op de SMU is al op de juiste manier met de microcontroller verbonden. In het schema van de SMU kan je vinden via welke pinnen dat gebeurt. Het enige wat je nu nog hoeft te doen om het te gaan gebruiken is de juiste code uploaden. Om de kaart te kunnen gebruiken is de FatFS library geïntegreerd in de code van de microcontroller. Een volledig overzicht van de functies van de FatFS library is te vinden op http://elm-chan.org/fsw/ff/00index_e.html. Dit is best een complexe library met veel functies waarbij je rekening moet houden met de soort microcontroller die je gebruikt. Met behulp van deze functies is een stukje voorbeeldcode geschreven (SDtemplate op ELO), de gebruikte functies staan onder het voorbeeld toegelicht:

```
#define F_CPU   (4000000UL)

#include <avr/io.h>
#include <avr/cpufunc.h>
#include <util/delay.h>
#include <string.h>
#include "FatFS/ff.h"
#include <avr/interrupt.h>


ISR(TCA0_OVF_vect)
{
        disk_timerproc();       // Drive timer procedure of low level disk I/O module
        TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm;
}


int main(void)
{
        TCA0.SINGLE.INTCTRL = TCA_SINGLE_OVF_bm;
        TCA0.SINGLE.PER = 156; // tick every 10 msec = 100 hz: clock = 4000000 / 256 = 15625 hz
        TCA0.SINGLE.CTRLA = (TCA_SINGLE_CLKSEL1_bm | TCA_SINGLE_CLKSEL2_bm);
        TCA0.SINGLE.CTRLA |= TCA_SINGLE_ENABLE_bm;

        sei();

        FATFS FatFs;

        // init sdcard
        UINT bw;
        f_mount(&FatFs, "", 1);         // Give a work area to the FatFs module

        // open file
        FIL *fp = (FIL *)malloc(sizeof (FIL));
        if (f_open(fp, "file.txt", FA_WRITE | FA_CREATE_ALWAYS) == FR_OK) // Create a file
        {
                char *text = "Hello World! SDCard up and running!\r\n";
                f_write(fp, text, strlen(text), &bw);   // Write data to the file
                f_close(fp);// Close the file
        }

        while(1)
        {
        }
}
```

Functie	Toelichting
f_mount()	Zorgt er voor dat de SD kaart geschikt wordt gemaakt voor gebruik. Het systeem gaat uit van een geformatteerde SD kaart. Zorg er voor dat je deze voor gebruik FAT32 formatteert. Deze functie moet je dus altijd als eerste aanroepen. Hier worden ook de SPI pennen juist geïnitialiseerd.
f_open()	Opent het bestand op de SD kaart. Geef de naam van het bestand mee. De mode bepaalt wat er moet gebeuren als het bestand al op de SD kaart staat. Zie voor de verschillende modi de instructie bij de functie op http://elm-chan.org/fsw/ff/doc/open.html.
f_write()	Schrijft data in het bestand op de SD kaart. Je geeft de data mee, maar ook het aantal karakters dat geschreven moeten worden.
f_close()	Als je klaar bent met het schrijven van het bestand sluit je deze met f_close.

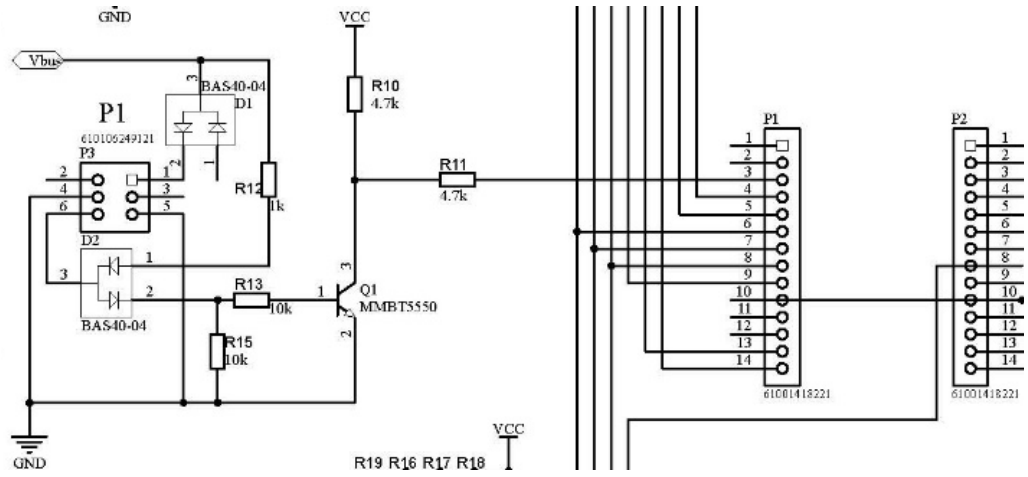
LABOPDRACHT 4:

Gebruik SDtemplate als basisprogramma en pas deze aan bij de volgende opdrachten:

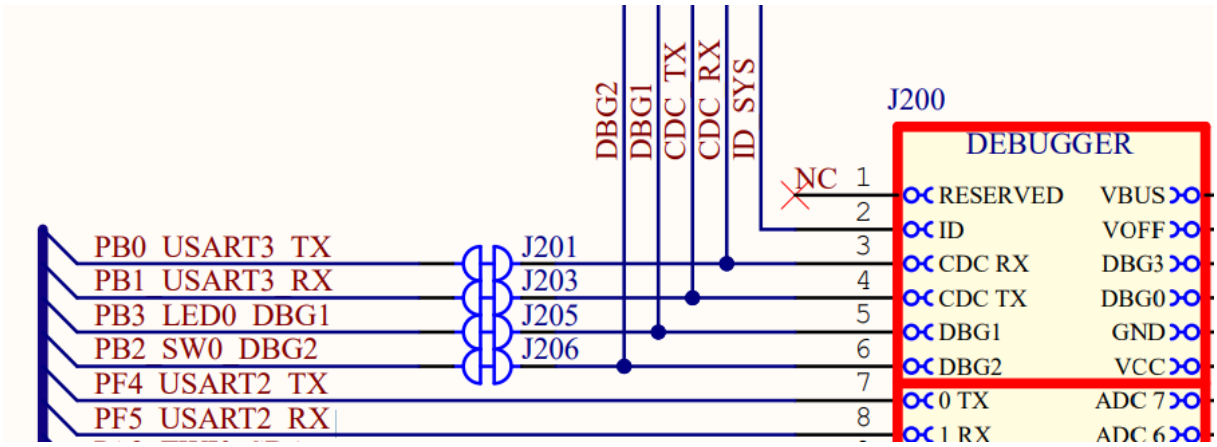
a) Schrijf een stuk code welke je naam en studentnummer print op de SD kaart.
b) Schrijf een stuk code welke elke seconde de waarde van de drie schakelaars gaat loggen op de SD kaart. Geef dit weer als binair patroon (bijvoorbeeld “1 1 0”).

6.5 P1-poort uitlezen

Je weet nu hoe je data zichtbaar kan maken via UART, OLED-display of op een SD-kaart. Om nu een echte P1-meter te maken hoef je alleen nog te weten hoe je de P1 poort uitleest. Zoals je in het schema van de SMU kan zien is de P1-aansluiting via twee pinnen verbonden met het microcontrollerbord: V_{bus} en PF15.



Pin PF15 is pin nummer 8 van het microcontroller bord. Als we deze in het schema van het Curiosity Nano bordje opzoeken vinden we dat deze pin wordt gebruikt om signalen te ontvangen (Rx) via UART2:



Willen we data uitlezen via P1, dan gebruiken we dus UART2. In de voorbeeldcode “P1template.c” die op ELO staat lezen we de data via P1 uit en maken die zichtbaar via de Terminal in Microchip Studio. Een groot deel van deze code stond ook in “UARTtemplate.c”, maar er zijn een paar verschillen.

Ten eerste de klok. In plaats van de standaardinstelling van 4 MHz, wordt in dit geval 16 MHz gebruikt:

```
#define F_CPU       (16000000UL)
```

Vervolgens wordt nu naast USART3 (om data naar de Terminal te sturen) ook USART2 gebruikt (om P1 data uit te lezen). Hiervoor wordt `void USART2_init(void)` aangemaakt:

```
void USART2_init(void)
{
    // AVR128DB48
    PORTF.DIRCLR = PIN5_bm;                                    /* set pin 5 of PORT B (RXd) as input*/

    PORTMUX_USARTROUTEA |= 0x10;

    if (dsmr == V2)
    {
        USART2.BAUD = (uint16_t)(USART2_BAUD_RATE(9600));      /* set the baud rate*/
        USART2.CTRLC = USART_PMODE1_bm | USART_CHSIZE1_bm;     /* set the data format to 7-bit, even parity*/
    }
    else
    {
        USART2.BAUD = (uint16_t)(USART2_BAUD_RATE(115200));    /* set the baud rate*/
        USART2.CTRLC = USART_CHSIZE1_bm | USART_CHSIZE0_bm;    /* set the data format to 8-bit, no parity*/
    }

    USART2.CTRLB |= USART_RXEN_bm;                             /* enable receiver*/
}
```

Zoals je ziet is er een klein verschil tussen DSMR2 en DSMR5, namelijk de baud rate die moet worden ingesteld.

In het stukje daarna wordt een functie aangemaakt waarmee we USART2 kunnen uitlezen:

```
uint8_t USART2_Read()
{
    while (!(USART2.STATUS & USART_RXCIF_bm));

    return USART2.RXDATAL;
}
```

De laatste functie voordat *main* begint is `void CLOCK_XOSCHF_crystal_init(void)`. Deze is er om ervoor te zorgen dat de 16MHz die we hebben ingesteld ook daadwerkelijk overal wordt doorgevoerd.

Binnen *main* beginnen we met het initializeren van de klok, USART2 en USART3. Vervolgens wordt binnen de while-loop de P1-poort uitgelezen en doorgestuurd naar de Terminal:

```
int main(void)
{
    char c;

    CLOCK_XOSCHF_crystal_init();

    USART2_init();
    USART3_init();

    /* Replace with your application code */
    while (1)
    {
        // echo received P1 data on USB port
        c = USART2_Read();
        USART3_sendChar(c);
    }
}
```

Nu is het voor de meeste mensen niet zo heel interessant om een volledig P1-telegram te zien. Gebruikers willen weten wat hun verbruik is. Kortom, het is nuttig om bepaalde informatie uit dit telegram te filteren en deze op bijvoorbeeld het OLED-display of de SD-kaart te laten zien.

Hoe bewerk je de seriële data die binnenkomt? Daarvoor heb je de *scanf* functie nodig. Ook hier moeten we de data uit de seriële poort halen, dus is er een speciale variant nodig net als bij de *printf* functie: *printf -> sprintf, scanf -> sscanf.* Deze leest dus de data uit een buffer. Je kunt in de telegram-gegevens kijken welke data je wilt bekijken. In het geval van het huidige verbruik heb je een stuk voor en een stuk na de komma.

```
if (sscanf(command, "1-0:1.7.0(%d.%d*kW)", &kw, &deckw) == 2)
        {
        sprintf(buffer, "%5d: %5d W", counter, kw * 1000 + deckw);
        }
```

Als het lukt om beide waarden toe te kennen (==2) dan kun je de kw en deckw waarden gebruiken om te loggen, te printen etc.

LABOPDRACHT 5:

a) Geef het huidige verbruik weer op het OLED display.
b) Wat moet je aanpassen om ook het gasverbruik uit de meter te halen?
c) Print het gasverbruik onder het huidige stroomverbruik.