Arduino -8- geheugen

Handleiding van Helpmij.nl

Auteur: drejansen

juli 2022

Het geheugen van Arduino

Vorige keer ging het om extra pootjes (pinnen), nu wat extra geheugen. Eerst maar eens vertellen over het interne geheugen. In elke Arduino zijn drie typen geheugen aanwezig:

FLASH geheugen (niet vluchtig geheugen):

De Nano is er in twee uitvoeringen: ATMega 168 met 16K en de ATMega 328p heeft 32K. De Uno (ATMega 328) heeft 32K en de Mega (ATMega 1280) heeft 128KByte.
Een klein deel, ca 1 KB, is voor de bootloader gereserveerd, de rest is voor je eigen programma (sketch).
Dit geheugen kan je 10.000 keer herschrijven. Strings kan je het beste in dit (FLASH) geheugen plaatsen.

SRAM (statisch geheugen)

Ook hier twee typen Nano's: ATMega 168 heeft 1K en de ATMega 328p heeft 2KByte. De Uno (ATMega 328) heeft 2K en de Mega (ATMega 1280) heeft 8KByte. In SRAM staan de variabelen en manipulatie van de sketch. Dit geheugen gaat verloren wanneer de Arduino wordt uitgeschakeld. Strings en variabelen komen standaard in SRAM. Heb je veel strings dan kan je in de geheugenproblemen komen. Vooral vaste strings kan je dan beter elders plaatsen. Snel veranderende variabelen kan je het best in dit (SRAM) geheugen blijven plaatsen.
Standaard komt alles in het SRAM geheugen. Met de **F(..)** functie plaats je data in het FLASH geheugen:

```
Serial.println("Deze string staat in SRAM");      Serial.println(F("Deze string staat in FLASH"));
```

E-EPROM (elektrisch wisbaar geheugen):

De Nano, weer in twee varianten: ATMega 168 heeft 512 bytes en ATMega 328p 1K. De Uno (ATMega 328) 1K en de Mega (ATMega 1280) heeft 4K. Dit is 'vast' geheugen voor opslag van gegevens die voor lange tijd bewaard moet blijven. Dit geheugen kan je 100.000 keer herschrijven. Als de Arduino uit staat blijft de data staan.

EEPROM schrijven.

Hier wordt de waarde 123 in EEPROM, op locatie -0- geschreven: Let op, maximaal te schrijven waarde is 255.

De INO bestanden (sketchen) kan je hier downloaden:
http://magazine.helpmij.nl/uploads/2022/04/Arduino-geheugen-I2C.doc

Er wordt één getal in het interne EEPROM geheugen op locatie -0- geschreven. Schakel de Arduino uit en weer aan, dan kan je zien dat de waarde is blijven staan.

EEPROM uitlezen.

Bestand om een blok data, of de hele EEPROM te lezen, zie apart te downloaden bestand

Data loggen.

Een reeks variabelen, bv het temperatuur verloop gedurende een hele dag registreren. Hier wordt de gemeten waarde van de analoge ingang -0- gemeten. Omdat deze waarden 10 bits breed zijn, worden de gemeten waarden door 4 gedeeld, waardoor ze 8 bits breed worden.

EEPROM afmeting.

Hoe groot was ook al weer de EEPROM van deze Arduino? Na het runnen van dit programma weet je het.

EEPROM Clear

Wil je met een schone lei beginnen, dan voer je dit programma uit. Het zoekt zelf uit hoe groot het geheugen is, en maakt alles schoon. **digitalWrite(13, millis() & 128);** // zodra de EEPROM is gewist gaat de led knipperen.
Deze manier van knipperen maakt gebruik van de altijd lopende millis teller. De werking: de LED status wordt steeds vergeleken met het 8e bitje van de millis waarde. Na elke 128 ms. veranderd het 8e bitje van de millis teller, dit is dan meteen de knipperfrequentie. De AND functie werkt slechts op één byte, trager gaat dus niet, 256 is het 9e bitje. Bij 255 blijft de LED continu branden, omdat het allemaal enen zijn. Bij 64 is de frequentie dubbel. Het 7e bit is dan -1-.

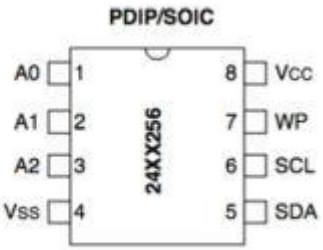
Tips voor zuiniger geheugengebruik

1) F() zet alle strings in het flash geheugen, dit zal SRAM schelen.
2) Verwijder alle onnodig variabelen, functies en #includes.
3) Gebruik de correcte array grootte, als je bijvoorbeeld a[55] reserveert en je gebruikt maar a[5], dan blijft onnodig geheugen gereserveerd.
4) Gebruik zo veel mogelijk functies, alle parameters en variabelen die naar een functie wijzen, worden automatisch geleegd.
5) Gebruik de juiste datatypen zoals int, Double, Float, voor je variabelen.
6) Gebruik zo veel mogelijk lokale variabelen dit is sneller en gebruikt minder geheugen.

Geheugensteuntje aanbrengen.

Extern geheugen toevoegen I2C EEPROM (AT24C32 = 4 KB) Wanneer je alle geheugen hebt gebruikt en je komt toch te kort, dan prik je er toch wat bij! Deze komt samen met een klokchip op één printje: DS3231 AT24C32 IIC
Te koop voor € 2,50 bij Alie Express (Tot zo ver deze sterspot.):
https://nl.aliexpress.com/item/1005004046676483.html?spm=a2g0o.productlist.0.0.62a47de22GH6X8&algo_pvid=16b13599-5740-4caa-88e2-a4daa4d31797&algo_exp_id=16b13599-5740-4caa-88e2-a4daa4d31797-1&pdp_ext_f=%7B%22sku_id%22%3A%2212000027849475256%22%7D&pdp_pi=-1%3B2.33%3B-1%3B-1%40salePrice%3BEUR%3Bsearch-mainSearch

Wanneer je dit printje hebt aangesloten en scant op I2C adressen, dan zie je twee slave adressen: 0x57 en 0x68. Nummer 57 is de geheugenchip, in het oranje ovaal. nummer 68 is de klok.
Nu behandelen we het geheugen adres: 0x57. De -0x- geeft aan dat het een HEX adres is.



Waarschuwing:

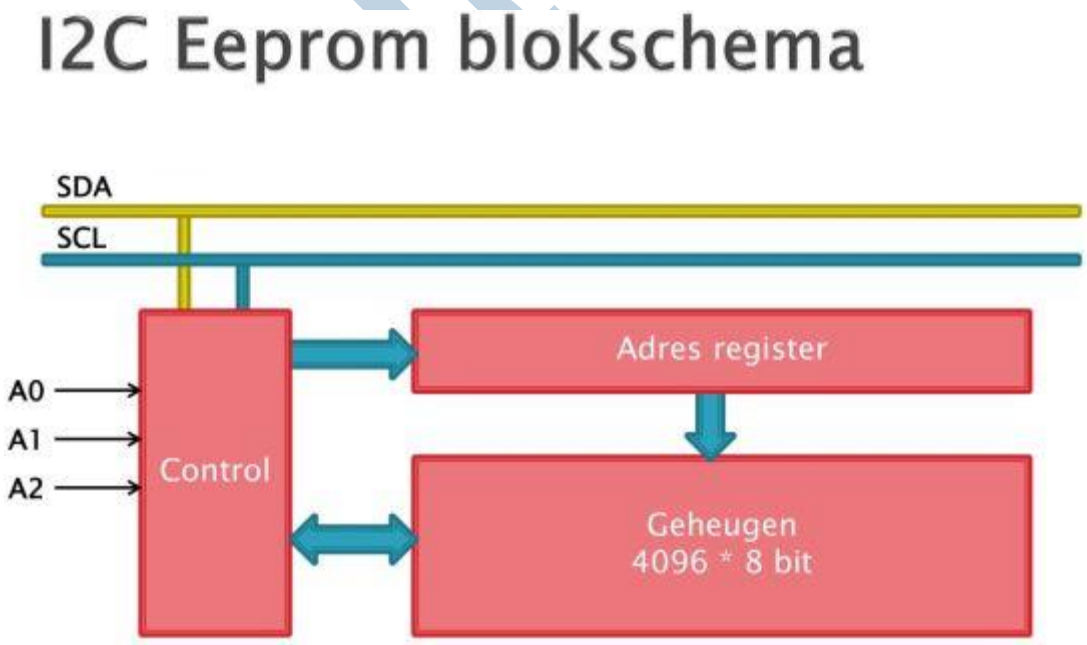
Op de print zit een batterij, dat via een weerstandje aangesloten. Haal dat weerstandje weg, want een lithium batterij laat zich niet via een weerstandje laden. De batterij zal kapot gaan. Het bedoelde weerstandje staat in het rode vierkantje. De batterij is voor de klok, maar daarover de volgende keer meer.

Het geheugen is opgebouwd uit 128 blokken van 32 cellen.

Per 'run' kan je maximaal 30 geheugencellen lezen of schrijven, daarna moet je een volgend blok van 30 cellen aanroepen. In de datasheet staat 32 cellen, maar ik mis de laatste 2 bytes van elk blok. Wil je vanaf adres 10 in het betreffende blok schrijven, dan heb je nog maar 20 geheugenplaatsen in dat blok, daarna moet je naar het volgende blok omschakelen. Ik mis telkens 2 bytes, Waar die blijven, weet ik niet, mogelijk voor het pointeradres.

Het benaderen van extern geheugen gaat in drie stappen:

-1- schrijf het MSB naar het geheugenadres van de chip (8 bits)
-2- schrijf het LSB naar het geheugenadres van de chip (8 bits)
-3- stuur de data die je op dat adres wilt opbergen, of lees de aldaar opgeborgen data.



Het lezen en schrijven van één byte maar de externe EEPROM

Er wordt op veel plaatsen een korte pauze ingelast, dat is om het externe geheugen de gelegenheid te geven om de data op te slaan of op te halen. Nadat ik deze pauzes had uitgeschakeld werkte het programma (bij mij) nog steeds. Dat wil niet zeggen dat die pauzes overbodig zijn, kwestie van uitproberen.

Verschil tussen 'write' en 'print'.

In de sketch staat **Serial.write** en **Serial.print**. wat is het verschil?

Serial.**write** stuurt de ruwe data, zoals je het hebt opgegeven.
Serial.**print** vertaalt de data naar de bijhorende ASCII waarde.

Net als bij de interne EEPROM, kan je ook extern een reeks data schrijven of lezen. Hier gaat het per 'pagina' of 'blok'. Een blok is 32 bytes. (ik mis er twee, zodat ik er slechts 30 heb. In het volgende programma wordt het alfabet weggeschreven en uitgelezen. Experimenteer eens met .write en .print, dan zie je meteen het verschil.

De pointer

De pointer is de plek vanwaar het geheugen wordt uitgelezen. Vervolgens kan je achter elkaar een hele pagina (30 bytes) uitlezen, daarna moet je weer opnieuw de pointer naar de volgende pagina verplaatsen, anders begint hij weer aan het begin van de betreffende pagina. Zo ook bij het schrijven, na 30 bytes opnieuw de pointer (32 bytes) verplaatsen. Let op!! Een pagina is 32 bytes, maar twee bytes raken telkens zoek. (Foutje in chip?)

Het pointer adres is een word. (16 bits)

Welke MSB en LSB moet ik invoeren voor het juiste pointer adres? De pointer omvat 2 bytes, een word. Het programma dat dit voor mij uitzoekt staat als laatst in de rij.

Volgende keer de klok op dit printje, dit is een tijdklok, maar ook schakelklok, alarmklok, etc.

De INO bestanden (sketchen) kan je hier downloaden:
http://magazine.helpmij.nl/uploads/2022/04/Arduino-geheugen-I2C.doc