Laboration 3 - Spara data på SD-kort

Hur man sparar och läser data från minneskort

Data genereras av Arduinon och skickas till datorn via en USB-sladd. Men hur gör vi om man inte kan vara uppkopplad till datorn direkt via en USB-sladd? Om en Arduino används för att samla in data och den inte är kopplad till en dator behöver den ett sätt att antingen skicka eller lagra datan tills den kan hämtas. I denna labb ska vi lära oss hur man lagrar data på ett SD-kort med en Arduino. SD-kort är enkla minneskort som finns i många olika apparater t.ex. telefoner och kameror.

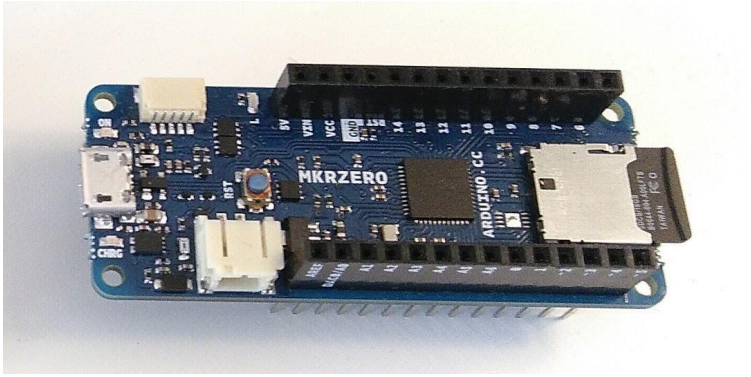
Material

- Arduino MKR ZERO
- USB-sladd för att koppla Arduino till datorn
- Dator
- SD-kort
- Batteri 9V
- Step down voltage
- BMP280
- Kopplingsplatta
- Sladdar för koppling
- Diod & Resistor (Valfritt)

E-post: cansat@au.se
Telefon: 070-000 90 56
Senast uppdaterad: 5/8 2022

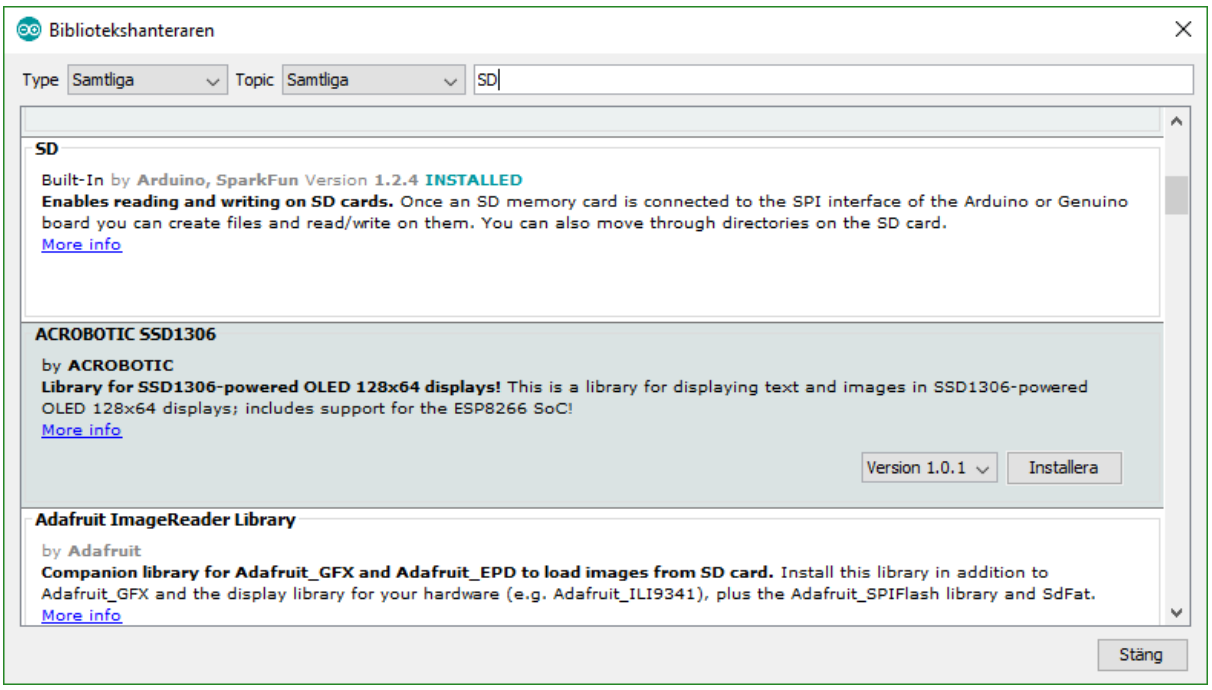
Koppling

Vissa typer av Arduino har en inbyggd minneskortsläsare och vissa har det inte. En Arduino MKR ZERO, som vi använder, har en inbyggd läsare för SD-kort.



Testa funktionen av SD-kortet

För att kunna använda hårdvaran behöver vi rätt mjukvara. Denna hittar man bland de inbyggda biblioteken i Arduinons program. Gå till Verktyg -> Hantera bibliotek och sök efter SD och installera det. Biblioteket som ska installeras heter “SD”.



Det första vi ska göra är att använda en exempelskiss för att kontrollera att allt fungerar. Hitta den under Fil -> Exempel -> SD -> CardInfo. Eftersom vi har en inbyggd SD-kortläsare behöver vi ändra värdet på en variabel i koden. Ändra

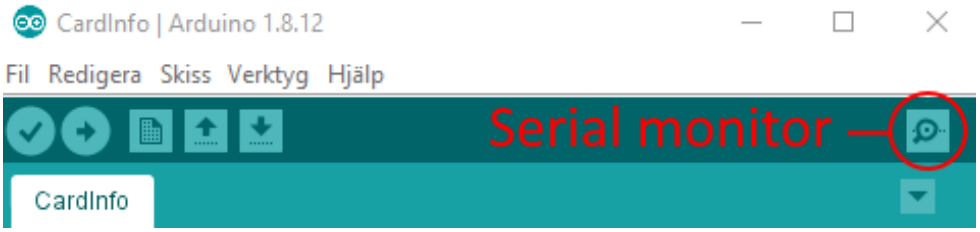
- `const int chipSelect = 4;`

till

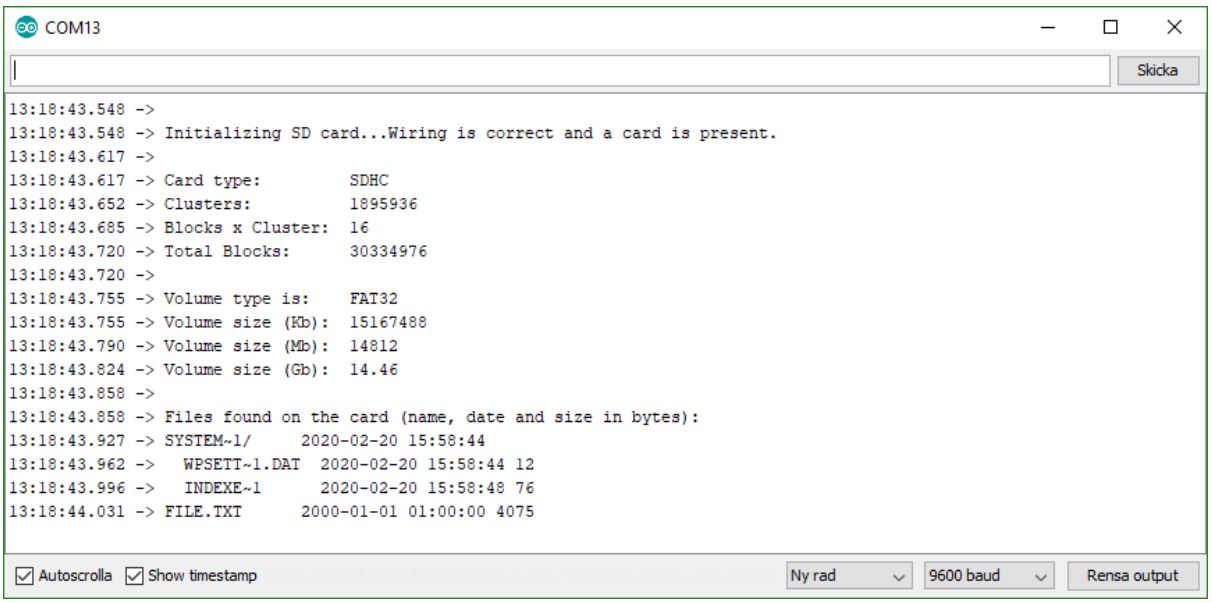
- `const int chipSelect = SDCARD_SS_PIN;`

Utöver det behöver inget ändras. Öppna Serial monitor och ladda upp skissen.

Inforuta, kom ihåg:



Programmet kommer testa kortet i olika steg och skriva ut resultaten i *serial monitorn.* Om något är fel kommer det skrivas där. Här är ett exempel på hur det kan se ut:



Koda ett program som skriver och läser data från SD-kortet

Om allting ser ut att fungera som det ska, öppna en ny skiss. I denna labb ska vi skriva siffror i en textfil på SD-kortet och sedan läsa dem. I förra labben använde vi `Serial.print()` for att skriva text i serial monitor. I denna labb kommer vi använda `file.print()` för att skriva text i filen.

Notera att även `println` funkar med `file`.

Om man vill repetera en bit kod utan att lägga den i `loop()` eller bara repetera den ett begränsat antal gånger eller tills ett visst kriterium uppfylls använder man en for- eller while-loop. Syntaxen för dessa är:

```
for( start; kriterium; steg )
{
     kod
}
```

Notera: semikolon mellan parametrarna i definitionen.

`start` genomförs först, nästan alltid en variabeldeklaration.
`kriterium` testas innan varje varv i loopen, om det är sant utförs koden i steg och sedan koden i loopen. Ofta är detta en kontroll av hur många varv loopen har gjort.

steg är en bit kod som kör varje gång kriterium är sant. Oftast en ökning av variabeln som definierades i start. Vanligtvis används ++ efter variabeln vilket ökar den med 1.

> **Inforuta, kom ihåg:** Ett kriterium måste vara ett booleskt uttryck, dvs en jämförelse av två objekt med < mindre än, <= mindre eller lika med, > större än, >= större eller lika med, == lika med eller != inte lika med.

Exempel:

```
for(int k = 0; k <= 10; k++) {
      Serial.println(k);
}
```

kommer skriva ut siffrorna 0 t.o.m. 10

Den andra typen av loop är en while-loop. Den används när något ska repeteras tills ett kriterium uppfylls snarare än ett bestämt antal gånger. Syntaxen är

```
while( kriterium )
{
     kod
}
```

där *kriterium* utvärderas innan varje loop och om det är sant så kör loopen, är det falskt så bryts den. Notera att det är lätt att skapa en while-loop som aldrig bryts.

Exempel:

```
int k = 0;
while(k <= 10) {
      Serial.println(k);
      k++;
}
```

kommer också skriva ut siffrorna 0 till 10, men här används ett externt kriterium istället för ett internt för loopen.

En sak som är lite lurigt är att det går att bara skriva en vanlig siffra (t.ex. en int variabel) som villkor i en `while`-loop (och andra ställen som tar villkor). Det som händer då är att det tolkas så att alla tal utom 0 är sant och 0 är falskt.

Vi kan återanvända en del kod från exemplet vi kollade på nyss. Till att börja med inkluderar vi biblioteket vi installerade med kommandot

- `#include <SD.h>`

(Kom ihåg! Inget semikolon efter `include`). Vi behöver ange vilken pin CS är kopplad till med en normal variabeldefinition,

- `int chipSelect = SDCARD_SS_PIN;.`

Inga andra pins behöver definieras eftersom läsaren är inbyggd. Sedan skriver vi

- `File file;`

för att skapa file-objektet vi ska använda senare.

I setup() startar vi vår Serial monitor på samma sätt som innan,

- `Serial.begin(9600);`

Om man använder en Arduino kopplad till en dator med en USB-sladd måste man inkludera en paus efter man startar Serial monitor för att ge mjukvaran tid att starta innan man börjar skicka meddelanden. Detta görs med en kompakt while-loop,

- `while(!Serial);`

> **Inforuta, kom ihåg:** Ett utropstecken betyder "inte", så != betyder "inte lika med" och `!Serial` betyder "inte `Serial`" dvs att Serial monitor inte har startat än.

Detta gör att programmet stannar i loopen tills Serial har startat. Vi kan strunta i klammerparenteserna eftersom loopen inte ska köra någon kod. Nästa steg är att starta mjukvaran som driver SD-kortet. Det gör vi genom att skriva

- `SD.begin(chipSelect);`

För säkerhets skull skriver vi vår data i en ny fil och inte en gammal som redan har data i sig. För att göra det kollar vi först om en fil finns. Det gör man med `SD.exists("data.txt");` som tar värdet sant eller falskt beroende på om en fil med det namnet existerar eller inte. Om filen finns tar vi bort den med `SD.remove("data.txt");` dvs

- `if (SD.exists("data.txt")) {`

```
        SD.remove("data.txt");
    }
```

I `loop()` börjar vi med att öppna filen vi ska skriva i. Det görs med

- `file = SD.open("data.txt", FILE_WRITE);`

Detta öppnar filen med namnet `data.txt` för att skriva data i den. Om filen inte finns så skapas en ny tom fil automatiskt. För att skriva ett slumpat tal till `data.txt` börjar vi med att skapa den siffran med

- `int randNum = random(10);`

`random()` är en inbyggd funktion som ger en slumpad siffra mellan 0 och värdet man anger. Sedan skriver vi in siffran i filen med

- `file.println(randNum);`

Allt man skriver in i filen hamnar automatiskt sist. När siffran är skriven stänger vi filen igen med

- `file.close();`

För att läsa filen behöver vi öppna den i läsläge. Det görs med
`file = SD.open("data.txt", FILE_READ);`

När programmet läser en fil kan man tänka sig att den gör det med ett läshuvud som går längs alla bokstäver i dokumentet. När man anropar den normala läsfunktionen svarar den med den bokstav som läshuvudet står över och flyttar sedan till nästa. Om man anropar läsfunktionen många gånger efter varandra kan man läsa hela filen.

För att läsa filen och skriva ut resultatet till Serial monitor borde vi börja med någon form av avskiljare för att göra utskriften mer lättläst, något i stil med

- `Serial.println("--- Data start here ---");`

Sedan behöver vi göra den faktiska läsningen. Vi kommer läsa en bokstav i taget. Vi vet att det kommer vara en siffra men det är ändå av datatypen text. Vi skapar en variabel för att hålla denna bokstav med

- `char character;`

Vi använder `file.read()` för att läsa filen. Vi vill läsa filen tills vi har nått slutet, alltså vill vi använda en loop. För att enkelt kolla om det finns någon siffra att läsa anropar man `file.available()`. Den berättar hur mycket data som finns kvar att läsa. Genom att sätta `file.available()` som villkor i vår `while`-loop kommer den att gå tills antalet siffror som är kvar att läsa är 0. Vi skriver det så här:

- ```
  while (file.available()) {
      character = file.read();
      Serial.write(character);
  }
  ```

Notera att vi använder `Serial.write()` här istället för `Serial.print(). Detta` då det är för att `write` används för att skriva data som är bytes och `char` är en typ av bytes som presenteras som bokstäver. Sedan stänger vi filen igen med

- `file.close();`

Avsluta `loop()` med en

- `delay(5000);`

så det är möjligt att hinna läsa det som skrivs ut.
  ```

Ladda upp programmet till Arduinon. Utskriften borde se ut som nedan. Om det inte fungerar gör det inget. Gå bara vidare till nästa uppgift.

Utökad kontroll

När programmet körs nu så borde det fungera, men det kanske inte gör det eftersom att läsa och skriva till SD-kort är ganska komplicerat. För att göra det mycket lättare att hitta eventuella fel som inte dyker upp i valideringen borde vi lägga till kontroller i koden som talar om för oss om något går fel.

Detta kan vi göra med hjälp av *if-satser* som skriver ett meddelande i Serial monitor om något är fel. Villkoren i sådana if-satser kan vara något kryptiska men vi ska förklara dem. Alla funktioner vi kommer använda nu finns listade i referensen för SD-biblioteket[1].

Den första kontrollen man kan göra är att SD-kortet startade som det skulle. Funktionen `SD.begin(pin)` tar värdet sant om starten lyckades på den angivna pinen och falskt om den misslyckades. Om vi ändrar i `setup()` från

- `SD.begin(chipSelect);`

till

- `if (!SD.begin(chipSelect)) {`

[1] https://www.arduino.cc/en/Reference/SD

```
        Serial.println("Could not initialize SD card.");
    }
```

kommer koden göra samma saker som innan men vi kommer då få ett meddelande om något gick fel i det här steget. Vi kan utöka vår kontroll för att också se om filen data.txt finns, vilket vi kan göra i `setup()`. Genom att lägga till ett `Serial.println("File exists.";)` och en if-sats. Sedan i if-satsen kan vi få veta om programmet går in i den eller inte. `SD.remove()` tar, på samma sätt som `SD.begin()`, värdet sant eller falskt beroende på om den lyckades eller inte. Om vi byter koden

```
if (SD.exists("data.txt")) {
    SD.remove("data.txt");
}
```

till

```
if (SD.exists("data.txt")) {
    Serial.println("File exists.");
    if (SD.remove("data.txt")) {
        Serial.println("File removed successfully.");
    }
}
```

kommer programmer undersöka om dessa handlingar lyckades och meddela oss om de gjorde det. Vidare så är `SD.open()` gjort så att om den inte lyckas öppna en fil tar den värdet falskt. Vi kan använda det genom att lägga all kod som beror på att vi lyckas öppna en fil i en if-sats som tar file som villkor. Så vi ändrar

```
int randNum = random(10);
file.println(randNum);
file.close();
```

till

```
if (file) {
    int randNum = random(10);
    file.println(randNum);
    file.close();
    Serial.print("Wrote number: ");
    Serial.println(randNum);
}
```

På samma sätt ändrar vi koden när vi öppnar filen för att läsa den. Från

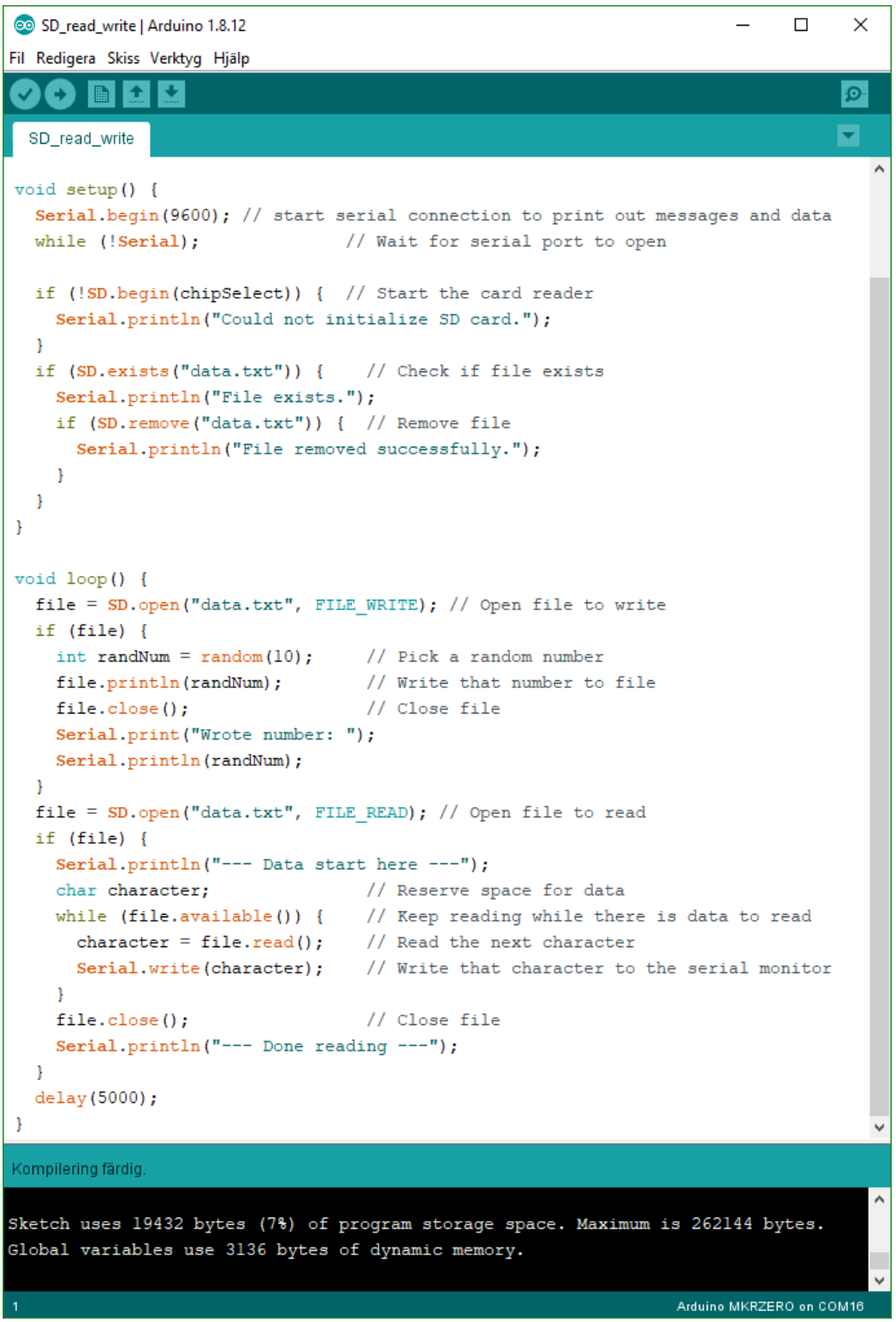
```
Serial.println("--- Data start here ---");
char character;
while (file.available()) {
    character = file.read();
    Serial.write(character);
```

```
    }
```

- `file.close();`

till

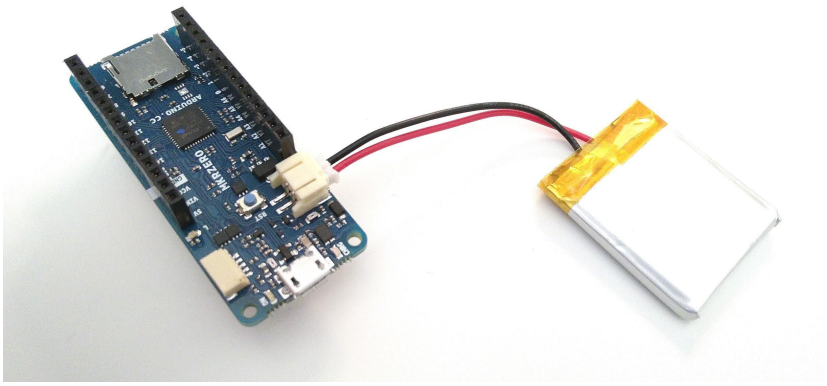
- ```
  if (file) {
      Serial.println("--- Data start here ---");
      char character;
      while (file.available()) {
          character = file.read();
          Serial.write(character);
      }
      file.close();
      Serial.println("--- Done reading ---");
  }
  ```
  ```


SD_read_write | Arduino 1.8.12
Fil Redigera Skiss Verktyg Hjälp
SD_read_write
void setup() {
Serial.begin(9600); // start serial connection to print out messages and data
while (!Serial); // Wait for serial port to open
if (!SD.begin(chipSelect)) { // Start the card reader
Serial.println("Could not initialize SD card.");
}
if (SD.exists("data.txt")) { // Check if file exists
Serial.println("File exists.");
if (SD.remove("data.txt")) { // Remove file
Serial.println("File removed successfully.");
}
}
}
void loop() {
file = SD.open("data.txt", FILE_WRITE); // Open file to write
if (file) {
int randNum = random(10); // Pick a random number
file.println(randNum); // Write that number to file
file.close(); // Close file
Serial.print("Wrote number: ");
Serial.println(randNum);
}
file = SD.open("data.txt", FILE_READ); // Open file to read
if (file) {
Serial.println("--- Data start here ---");
char character; // Reserve space for data
while (file.available()) { // Keep reading while there is data to read
character = file.read(); // Read the next character
Serial.write(character); // Write that character to the serial monitor
}
file.close(); // Close file
Serial.println("--- Done reading ---");
}
delay(5000);
}
Kompilering färdig.
Sketch uses 19432 bytes (7%) of program storage space. Maximum is 262144 bytes.
Global variables use 3136 bytes of dynamic memory.
1
Arduino MKRZERO on COM16

Validera och provkör skissen. Får du någon utskrift nu? Fungerar programmet? Extra: ta loss SD-kortet från Arduinon och kör programmet för att se om du får felmeddelande.

Arduino med batteridrift

Nu är du redo att skicka ut Arduinon att samla data på egen hand, förutom att den fortfarande sitter fast i datorn. Som tur är så är det enkelt att driva en arduino med ett batteri. Pinarna Vin (Voltage in) och GND (jord) används som strömingångar. En Arduino MKR Zero kan ta emot max 5V, så den kan inte drivas av ett 9V-batteri utan att en spänningsregulator också används. En MKR Zero har en speciell strömkontakt som kan används istället för VIN. Denna kontakt sitter på sidan och passar med en JST-PH-kontakt. Den är gjord speciellt för att passa till 3,7V Lithium Polymer (LiPo) batterier. Om Arduinon drivs med mindre än 5V kommer den inte att ge 5V ut själv. När Arduinon drivs med ett batteri kommer den ladda och köra det program den hade när den var uppkopplad mot en dator sist. Notera att man inte kan ha `while(!Serial)` i kod som körs på en Arduino som inte är kopplad direkt till en dator eftersom det kommandot gör att koden väntar tills Serial monitor är öppnad och igång vilket aldrig händer om Arduinon inte är kopplad till en dator.



Mobil datainsamling

Nu ska du använda alla kunskaper du samlat på dig hittills! Kombinera denna labb med förra labben och spela in en dataserie när Arduinon genomgår något experiment, t.ex. Ta med Arduinon till olika höjder eller ändra temperaturen kring sensorn. Mätdatan kan lagras på ett SD-kort och läsas på en dator med SD-kortläsare.

- Koppla den sista ordinarie kretsen som gjordes i labb 2 och ladda programmet den använde.

- Modifiera programmet så att det inte bara skriver mätdatan till Serial monitor utan också skriver in det på SD-kortet, dvs kombinera programmet från labb 2 med den som just skrevs fast istället för en slumpad siffra så skrivs det aktuella mätvärdet ner i filen och det behövs ingen del som läser data.
- Koppla in ett batteri eller en powerbank till Arduinon efter att du har skickat koden som följer till den. Använd en "step down voltage" om du använder ett 9V batteri. Här kan det behöva lödas.

Om man vill skriva koden så att det går att lagra mätdata från flera olika sensorer samtidigt behöver man lägga in någon form av avskiljare, t.ex.

- ```
  file.print(t);
  file.print(" - ");
  file.println(p);
  ```

För att synliggöra om Arduinon arbetar eller inte kan man lägga in en snabb blinkning av den inbyggda dioden eller en extern varje gång ett värde skrivs till SD-kortet.

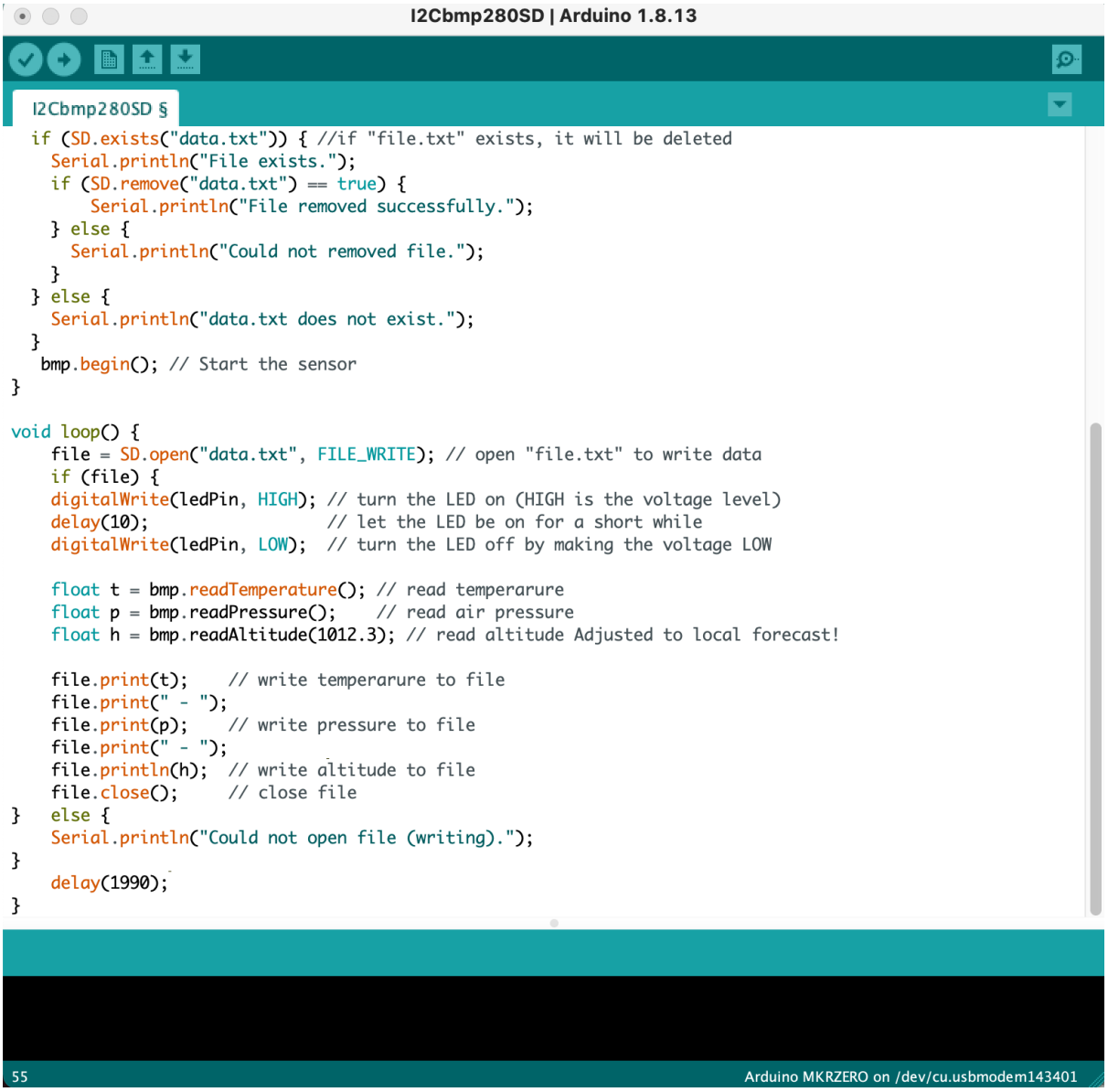
- ```
  digitalWrite(ledPin, HIGH);
  delay(10);
  digitalWrite(ledPin, LOW);
  ```

Om du gör detta, glöm inte att öppna `ledPin` i `setup()` med

- `pinMode(ledPin, OUTPUT);`
-

Om man väljer att koppla in en extern diod med en resistor, koppla den som i förra labben när vi hade en pulserande diod.


I2Cbmp280SD | Arduino 1.8.13
I2Cbmp280SD §
#include "Adafruit_BMP280.h" // Include the library with code for the sensor
#include <SD.h> // Include the library with code for the SD card
Adafruit_BMP280 bmp; // I2C
int ledPin = LED_BUILTIN; // When using built in pin
//int ledPin = 10; // When using external led with resistor
int chipSelect = SDCARD_SS_PIN; //chip select pin for the MicroSD Card Adapter
File file; // file object that is used to read and write data
void setup() {
pinMode(ledPin, OUTPUT); // Open the LED pin
Serial.begin(9600); // start serial connection to print out debug messages and data
pinMode(chipSelect, OUTPUT); // chip select pin must be set to OUTPUT mode
if (!SD.begin(chipSelect)) { // Initialize SD card
Serial.println("Could not initialize SD card."); // if return value is false, something went wrong.
} else {
Serial.println("SD card initialized");
}
if (SD.exists("data.txt")) { //if "file.txt" exists, it will be deleted
Serial.println("File exists.");
if (SD.remove("data.txt") == true) {
Serial.println("File removed successfully.");
} else {
Serial.println("Could not removed file.");
}
} else {
Serial.println("data.txt does not exist.");
}
bmp.begin(); // Start the sensor
}
void loop() {
Uppladdning färdig.
done in 0.033 seconds
CPU reset.
14
Arduino MKRZERO on /dev/cu.usbmodem143401



För en lite svårare utmaning: hur kan man skriva koden så att det går att lagra mätdata från flera olika sensorer?

Vill man läsa av vad som skrivits på SD-kortet genom att koppla Arduinon till datorn kan man använda samma program som användes att skriva och läsa av data från ett SD-kort. Kom ihåg att ta bort koden som tar bort filen om den finns och koden som skriver till ett SD-kort. Vi vill bara läsa av kortet. Ta bort:

- ```
  if (SD.remove("data.txt")) {
      Serial.println("File removed successfully.");
  ```
  ```

```
}
```

och

- ```
  file = SD.open("data.txt", FILE_WRITE);
  ```
- ```
  if (file) {
      int randNum = random(10);
      file.println(randNum);
      file.close();
      Serial.print("Wrote number: ");
      Serial.println(randNum); }
  ```

Eftersom vi inte behöver läsa av kortet mer än en gång kan koden som läser flyttas till `setup` och `loop` kan lämnas tom.

Extra övning

Skriv ett program som skickar text på en fil på ett SD-kort som morsesignaler från en lysdiod.