EEPROM Strukturen speichern über SPI-Bus

Experiment EEPROMstruct 7

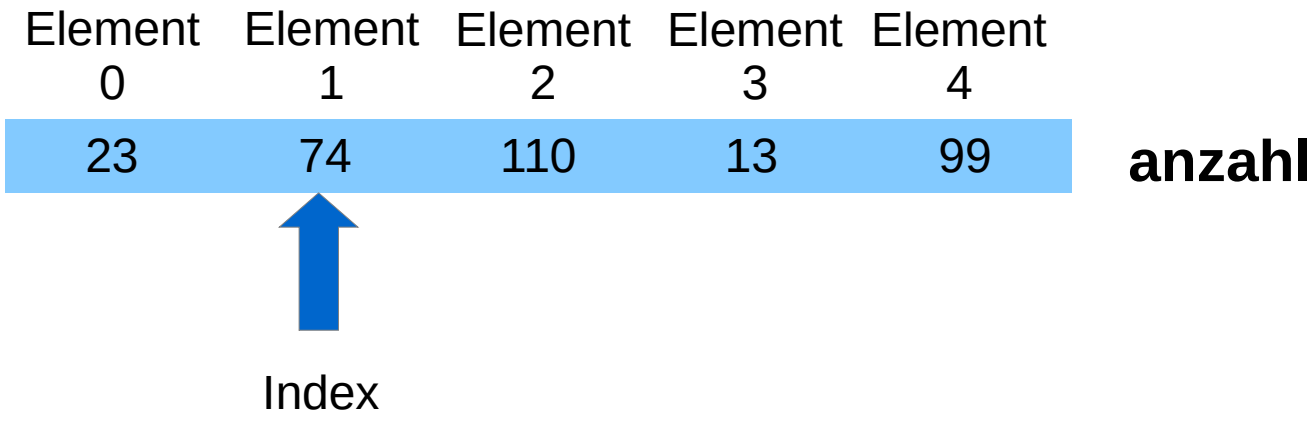
A.Schultze / DK4AQ 18.06.2013

Es soll eine Tabelle aus verschiedenartigen Informationen im EEPROM abgelegt und wieder gelesen werden. Solche Tabellen dienen z.B. zur Abspeicherung von Startwerten und Korrekturwerten für Geräte. Auch die letzte Einstellung vor Abschaltung des gerätes kann man so speichern..

Ein typisches Beispiel ist die Stationseinstellung an Radios, Fernsehern oder Funkgeräten:

Kanal	Repeater	Frequenz/kHz
54	Steinberg	145675,0
723	Wolfsburg	439050,0
58	Bocksberg	145725,0
56	Bergen	145700,0
63	Deister	145787,5
742	Celle	439275,0
746	Brocken	439325,0
684	Wurmberg	438550,0

Bisher haben wir mit „normalen Variablen oder mit Arrays (Feldern) gearbeitet. Arrays enthalten eine Anzahl Elemente, die gleiche Bedeutungen haben.




```
unsigned char anzahl[5];    // Deklaration eines Arrays mit 5 unsigned char
                            //und dem Namen anzahl
```

anzahl[1] greift auf das Element 1 des Arrays zu.(nicht Element 0 !!!)

```
anzahl[3] = 13;
x = anzahl[1];
anzahl[ ] = { 23, 74,110,13,99};
anzahl[ ] = { “Hello“ };
```

Wenn Informationen abgespeichert werden sollen, dann gehören häufig unterschiedliche Kategorien von Informationen zusammen. Dazu bietet C die Möglichkeit an, eigene Variablentypen zu definieren, in denen mehrere Kategorien zusammengeheftet werden.

```
typedef struct
    {
    int kanalnummer;
    char kanalname[10];
    unsigned long frequenz;
    } kanal_struct;
```

Für ein Fuinkgerät sollen folgende Informationen zur Auswahl stehen:

Kanalnummer	0...999
Kanalname	9 ASCII-Zeichen
Frequenz	140-500 Mhz mit einer Auflösung von 0,1kHz

Mit der obigen Definition einer neuen Struktur lassen sich weitere strukturierte Variablen deklarieren :

```
kanal_struct kanal_daten;
kanal_struct eeprom_daten

kanal_daten.frequenz = 4390500;
Kanal = kanal_daten.kanalnummer;
```

Man kann auf diese Strukturvariablen Einzelelementweise zugreifen.
ACHTUNG ! Textstrings lassen sich zwar als String bei der Intialisierung der Struktur einfach eingeben, jedoch können sie später nur über spezielle String-Funktionen zugegriffen werden (z.B. strcopy).

Wenn man nun eine Fernbedienung oder ein Senderauswahlmenü als Ziel hat, kann man die neuen Strukturen auch als Tabelle anlegen (Array of Arrays):

```
kanal_struct kanal_daten[8] =
 {
   {54,"Steinberg",1456750},
   {723,"Wolfsburg",4390500},
   {58, "Bocksberg",1457250},
   {56, "Bergen   ",1457000},
   {63, "Deister  ",1457875},
   {742,"Celle    ",4392750},
   {746,"Brocken  ",4393250},
   {684,"Wurmberg ",4385500}
 };
```

Arrays:
Zuweisung Strings in dieser Form nur bei Initialisierung möglich !

Damit wurd eine Array *kanaldaten* mit 8 Einträgen definiert. Alle Einträge sind vom Typ *kanal_struct*. Die Einträge werden zur Initialisierung gleich mit Werten gefüllt.

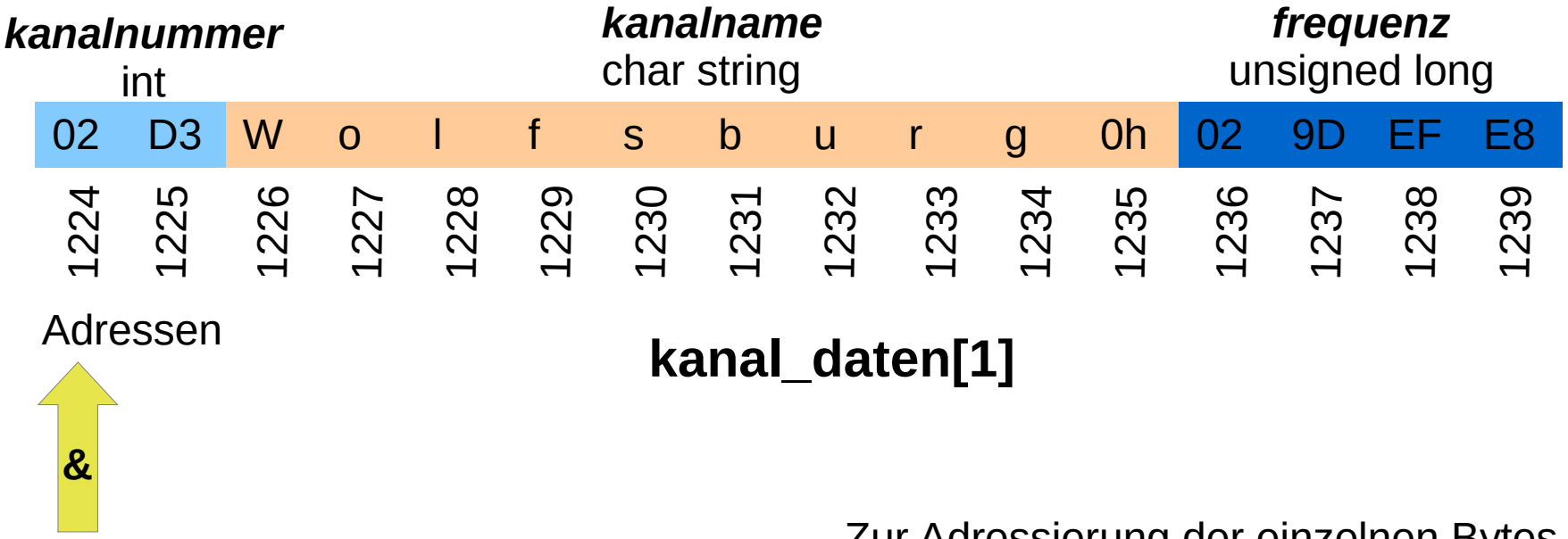
Wird nun z.B. kanal_daten[2] adressiert, so stehen über den Punktoperator die zugehörigen Werte zur Verfügung.

```
x = kanal_daten[2].kanalnummer
Serial.print( kanaldaten[2].kanalname
y = frequenz
```

Das Problem beim Transport dieser Daten in das EEPROM ist, dass es nur Bytes akzeptiert. Wie gewinnt man aus der beschriebenen Struktur nun Bytes und trasportiert sie ins EEPROM und kann sie beim Lesen auch wieder zusammenbauen ?

& **:** Adressoperator, bedeutet “Speicheradresse von“

***** **:** Inhaltsoperator, bedeutet „Inhalt von“



&kanal_daten[0] -> 1224

Zur Adressierung der einzelnen Bytes werden Pointer verwendet. Pointer haben typabhängige Eigenschaften.

```
byte *lesezeiger;
int *wert_zeig;
long *freq_zeig;
```

- Byte-Zeiger ändern ihre Speicheradresse bei jedem Inkrement um den Wert 1
- Int-Zeiger ändern ihre speciehradresse bei jedem Inkrement um den Wert 2
- Long-Zeiger ändern ihre speciehradresse bei jedem Inkrement um den Wert 4

Wenn wir also byteweise lesen wollen, dann benötigen wir einen Bytezeiger.

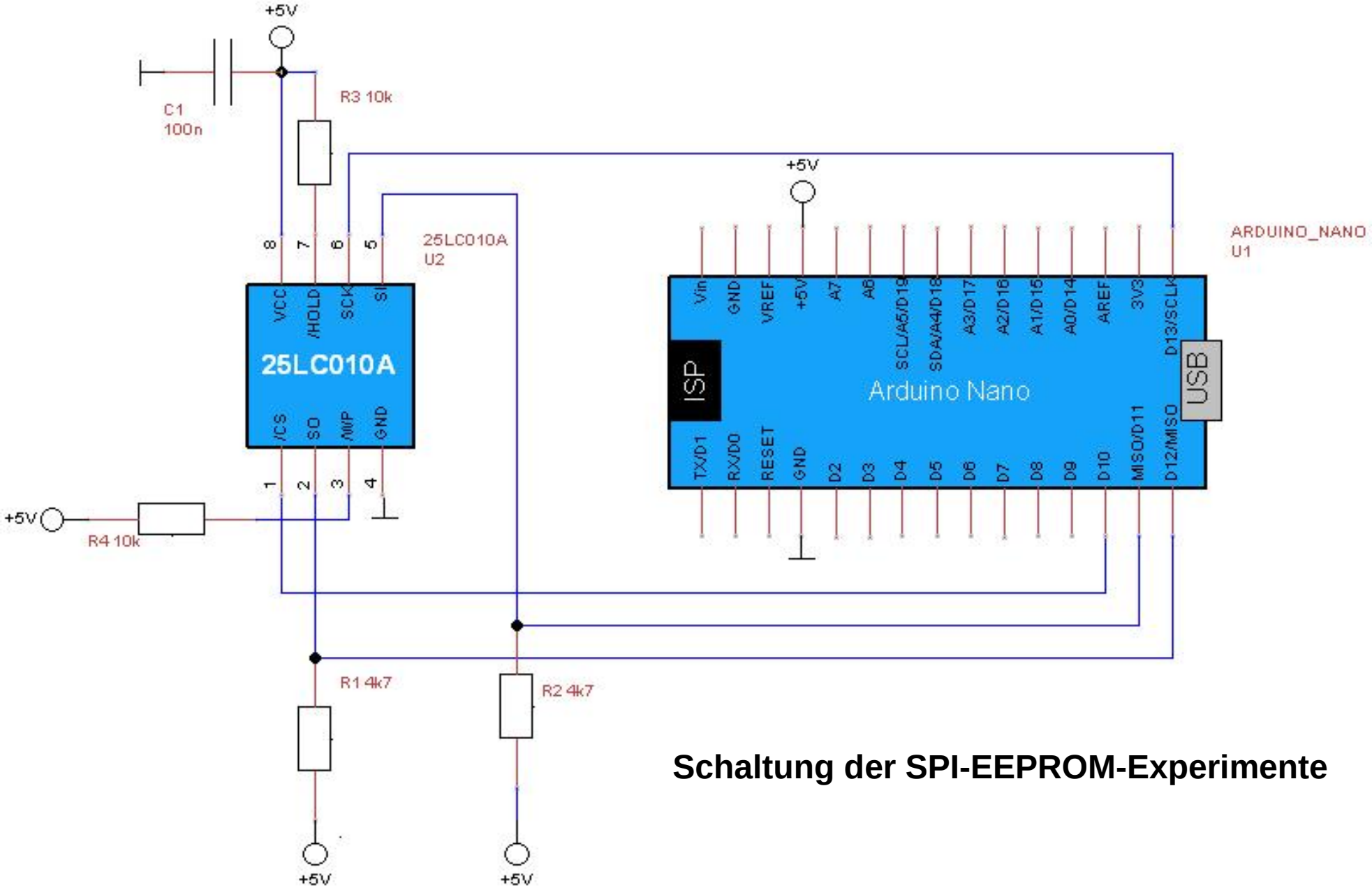
```
byte *b_ptr;
```

```
b_ptr = (byte *)&akt_datensatz + i;
```

Das Konstrukt ***&akt_datensatz*** wird immer die erste Speicheradresse des gesamten Konstruktes angegeben. Mit ***&akt_datensatz + i*** kann man über i die Adresse erhöhen. ***(byte *)*** bedeutet, dass der Ausdruck als Byte-Adresse gelesen wird. Der *b_ptr* zeigt also gesteuert über *i* auf ein Byte in der Struktur.

```
*b_ptr = SPI.transfer(i);
```

Damit kann man Daten in den Inhalt der Speicherstelle, ablegen, auf *b_ptr* zeigt.



Schaltung der SPI-EEPROM-Experimente

SPI_EEPROstruct (1)

```
// SPI_EEPROMstruct
// Serielles EEPROM 25LC010A ueber SPI
// A.Schultze/DK4AQ, 16.06.2013


#include <SPI.h>
#include "EEP_cmd.h"

// Definition des Portpins Chip Select auf D10
#define CS 10

//Adresse fuer Schreib-Lese-Versuche
#define BASE_ADR 0
#define PAGE_LIMIT 16
#define SATZ_MAX 8

unsigned int base_adr = BASE_ADR; // Startadresse Blockoperation im EEPROM
byte block_max = PAGE_LIMIT; // Blockgroesse Schreiben/Lesen
byte wr_byte = 0; //Zwischenspeicher Schreib-Byte
byte rd_byte = 0; //Zwischenspeicher Lese-Byte
byte ee_adr = 0; // interne Adresse EE-Speicher
```

SPI_EEPROstruct (2)

```
// Definiton eines komplexen Datentyps kanal_struct aus
//unterschiedlichen Einzeltypen
typedef struct
  {
  int kanalnummer;
  char kanalname[10];
  unsigned long frequenz;
  }kanal_struct;
// Array von 8 Elementen des Typs kanal_struct
// Direkte Zuweisung nur bei der Initialisierung !
//Alle Klammern geschweift !
 kanal_struct kanal_daten[8] =
 {{54,"Steinberg",1456750},
 {723,"Wolfsburg",4390500},
 {58, "Bocksberg",1457250},
 {56, "Bergen   ",1457000},
 {63, "Deister  ",1457875},
 {742,"Celle    ",4392750},
 {746,"Brocken  ",4393250},
 {684,"Wurmberg ",4385500}};
```

Deklaration und Initialisierung des Arrays (2D)

SPI_EEPROstruct (3)

```
 kanal_struct lese_daten[8];

kanal_struct akt_datensatz = {0,".",0};

byte lesewert[PAGE_LIMIT];// Daten, die empfangen wurden
byte schreibwert[PAGE_LIMIT];// Daten, die geschrieben werden sollen
byte i = 0;//Zaehlvariable Bytes
byte d = 0;//Zaehlvariable Datensätze

void setup()
{
  //Deklaration CS als Ausgang
  pinMode(CS,OUTPUT);
  digitalWrite(CS,HIGH);

  //Init. der Libraries
  Serial.begin(9600);
  SPI.begin();

  //Parameter fuer SPI-Bus
  SPI.setBitOrder( MSBFIRST);
```

SPI_EEPROstruct (4)

```
SPI.setClockDivider(SPI_CLOCK_DIV8);
SPI.setDataMode(SPI_MODE0 );

Serial.println("EEPROMstruct");
Serial.println("===========");

//EEPROM Kommando Freigabe Schreiben
EEP_WREN();

//EEPROM Kommando Memory Protection Modus
wr_byte = WP_NO;
EEP_setMMP(wr_byte);

Serial.println("Kommandobyte");
Serial.println(wr_byte,BIN);

//EEPROM Kommando Freigabe Schreiben
EEP_WREN();
```

SPI_EEPROstruct (5)

```
SPI.setClockDivider(SPI_CLOCK_DIV8);
SPI.setDataMode(SPI_MODE0 );

Serial.println("EEPROMstruct");
Serial.println("===========");

//EEPROM Kommando Freigabe Schreiben
EEP_WREN();

//EEPROM Kommando Memory Protection Modus
wr_byte = WP_NO;
EEP_setMMP(wr_byte);

Serial.println("Kommandobyte");
Serial.println(wr_byte,BIN);

//EEPROM Kommando Freigabe Schreiben
EEP_WREN();

//EEPROM Kommando Status lesen
rd_byte = EEP_RdStatus();
```

Ende setup( )

```
  Serial.println("Statusbyte");
  Serial.println(rd_byte,BIN);

}
```

Ende setup( )

SPI_EEPROstruct (7)

```
void loop()
{
  //EEPROM alle Datensaetze schreiben
  for(d=0;d<SATZ_MAX;d++)
  {
    akt_datensatz = kanal_daten[d];
    base_adr = d * PAGE_LIMIT;
    //EEPROM Datensatz schreiben
    for(i=0;i<block_max;i++)
    {
      //EEPROM Byte schreiben
      EEP_WrData(base_adr,i);
    }
  }
```

Schleife:
vom Datensatz 0 bis zum Ende der Tabelle

akt_datensatz bekommt zu schreibenden Inhalt

Startadresse Schreiben im EEPROM übergeben

Schleife:
Vom Element 0 .kanalname bis Element 2 .frequenz ausgeben

```
void loop()
{
  //EEPROM alle Datensaetze schreiben
  for(d=0;d<SATZ_MAX;d++)
  {
    akt_datensatz = kanal_daten[d];
    base_adr = d * PAGE_LIMIT;
    //EEPROM Datensatz schreiben
    for(i=0;i<block_max;i++)
    {
      //EEPROM Byte schreiben
      EEP_WrData(base_adr,i);
    }
  }
```

Schleife:
vom Datensatz 0 bis zum Ende der Tabelle

Schleife:
Vom Element 0 .kanalname bis Element 2 .frequenz ausgeben

```
//EEPROM alle Datensaetze lesen
for(d=0;d<SATZ_MAX;d++)
{
  //EEPROM Datensatz lesen
  base_adr = d * PAGE_LIMIT;
  //EEPROM Byte lesen
  for(i=0;i<block_max;i++)
  {
    EEP_RdData(base_adr, i);
  }
  lese_daten[d] = akt_datensatz;
}
```

Schleife:
vom Datensatz 0 bis zum Ende der Tabelle

Startadresse Lesen im EEPROM übergeben

Schleife:
Vom Element 0 .kanalname bis Element 2 .frequenz einlesen

```
//Ausgabe Ergebnis
Serial.println("Schreiben aus RAM in EEPROM");
Serial.println("----------------------------");
Serial.println("Kanal   Repeater        Frequenz/kHz");
for(d=0;d<SATZ_MAX;d++)
{
  Serial.print(kanal_daten[d].kanalnummer);
  Serial.write(0x09);
  Serial.print(kanal_daten[d].kanalname);
  Serial.write(0x09);
  Serial.print((kanal_daten[d].frequenz)/10);
  Serial.print(',');
  Serial.print((kanal_daten[d].frequenz)%10);
  Serial.println(' ');
}
Serial.println(" ");
```

```
  Serial.println("Lesen aus EEPROM");
  Serial.println("-----------------");
  Serial.println("Kanal   Repeater         Frequenz/kHz");
  for(d=0;d<SATZ_MAX;d++)
  {
    Serial.print(lese_daten[d].kanalnummer);
    Serial.write(0x09);
    Serial.print(lese_daten[d].kanalname);
    Serial.write(0x09);

    Serial.print((lese_daten[d].frequenz)/10);
    Serial.print(',');
    Serial.print((lese_daten[d].frequenz)%10);
    Serial.println(' ');
  }
  Serial.println(" ");
  //Warten auf Eingabe ueber seriellen Monitor
  while (1);

}
```

EEPROM Daten byteweise lesen

```
//EEPROM Byte lesen
void EEP_RdData(int baseadr,byte i)
{
     //Variable vom Typ Byte-Pointer
     byte *b_ptr;

     digitalWrite(CS,LOW);
     SPI.transfer(READ);
     SPI.transfer(baseadr+i);
     //Bytepointer = als Byte-Adresswert(Adresse akt_datensatz + Offsetadr. 1)
     b_ptr = (byte *)&akt_datensatz + i;
     //Inhalt des durch Bytepointer adressierten Speicher mit empfangenen Byte füllen
     *b_ptr = SPI.transfer(i);

     digitalWrite(CS,HIGH);
     return;

     //ALternativlösung: *((byte *)(&akt_datensatz + i))=SPI.transfer(i);
}
```

EEPROM Status lesen

```
//EEPROM Status lesen
byte EEP_RdStatus()
{
  byte r_byte;
  //EEPROM Status lesen
  digitalWrite(CS,LOW);
  SPI.transfer(RDSR);
  r_byte = SPI.transfer(0);
  digitalWrite(CS,HIGH);
  return(r_byte);
}
```

EEPROM Freigabe Schreiben

```
void EEP_WREN()
{
 //EEPROM initalisieren, Write Enable
 digitalWrite(CS,LOW);
 SPI.transfer(WREN);
 digitalWrite(CS,HIGH);
 delay(5);
 return;
}
```

EEPROM Daten byteweise schreiben

```
// EEPROM Byte schreiben
void EEP_WrData(unsigned int baseadr,byte i)
{
  byte w_byte;
  byte *b_ptr;

  //Bytepointer =
  //als Byte-Adresswert(Adresse akt_datensatz + Offsetadr. 1)
  b_ptr = (byte *)&akt_datensatz + i;
  //Inhalt des durch Bytepointer adressierten
  //Speicher mit empfangenen Byte füllen
  w_byte = *b_ptr;
  digitalWrite(CS,LOW);
  SPI.transfer(WRITE);
  SPI.transfer(baseadr + i);
  SPI.transfer(w_byte);
  EEP_WREN();
  delay(5);
  return;
}
```

EEPROM Memory Management setzen

```
// EEPROM Set Memory Protection
void EEP_setMMP(byte pr_byte)
{
 //EEPROM Kommando memory Protection aufheben
  digitalWrite(CS,LOW);
  SPI.transfer(WRSR); //Komando-Byte
  SPI.transfer(pr_byte);// Daten-Byte
  digitalWrite(CS,HIGH);
  delay(5);
  return;
}
```

Symbolische Konstanten für 25L010A

```
// Instruction Set 25L010A
//Lesen
#define READ 0x03
//Schreiben
#define WRITE 0x02
//Schreiben sperren
#define WRI 0x04
//Schreiben freigeben
#define WREN 0x06
//Lesen Statusregister
#define RDSR 0x05
//Schreiben Statusregister
#define WRSR 0x01

// Bitmuster Schreibschutz BP1/BP0
//kein Schreibschutz
#define WP_NO 0b00000000
//Schreibschutz oberes Viertel (60h-7Fh)
#define WP_UQ 0b00000100
//Schreibschutz obere Haelfte (40h-7Fh
#define WP_UH 0b00001000
```

Ergebnis der EEPROM-Schreib- und Lese-Aktion auf dem seriellen Monitor

```
EEPROMstruct
===========
Kommandobyte
0
Statusbyte
10
Schreiben aus RAM in EEPROM
---------------------------
Kanal   Repeater        Frequenz/kHz
54      Steinberg       145675,0
723     Wolfsburg       439050,0
58      Bocksberg       145725,0
56      Bergen          145700,0
63      Deister         145787,5
742     Celle           439275,0
746     Brocken         439325,0
684     Wurmberg        438550,0

Lesen aus EEPROM
----------------
Kanal   Repeater        Frequenz/kHz
54      Steinberg       145675,0
723     Wolfsburg       439050,0
58      Bocksberg       145725,0
56      Bergen          145700,0
63      Deister         145787,5
742     Celle           439275,0
746     Brocken         439325,0
684     Wurmberg        438550,0
```