Software des Projektes FreqGenNEU

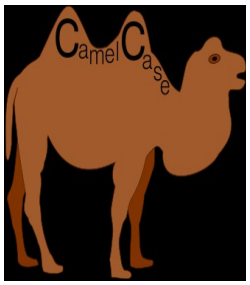
Axel Schultze, DK4AQ, V10a

Die Software stellt eine Experimental-Variante für Microcontroller-Einsteiger dar. Es wurde als Mikrorechner wurde die Arduino-Reihe und deren sehr einfache Entwicklungsumgebung gewählt. Die Idee ist, den Code an eigene Projekte anzupassen. Die Programmierung geschieht in C. Die Standardlibraries sind wie bei allen Microcontrollerprojekten der Welt an die Hardware angepasst.

Das Projekt zeigt die wesentlichen Teile und kann nach Belieben ergänzt und umgeschrieben werden. Daher hier auch eine komplette Beschreibung.

Die Software ist für eine kleine Mikrorechner unter den Bedingungen Echtzeit-Steuerungs-Software geschrieben. Es gibt wenig RAM für Variablen und die Rechengeschwindigkeit ist beschränkt.

Naming Conventions
Konstanten: Großschrift z.B. *KONSTANTE*
Variablen: Kleinschrift z.B. *frequenz*
Zuordnung Konstanten/Variablen: Vorsilbe mit Underline gertrennt z.B. *dre_val* => Drehgeber Wert
Zuordnung Funktionen: Beginn mit Kleinbuchstaben, Teile durch abwechselnde Groß/ Kleinschreibung ohne Underline (CamelCasing) z.B *initFreConfig()*

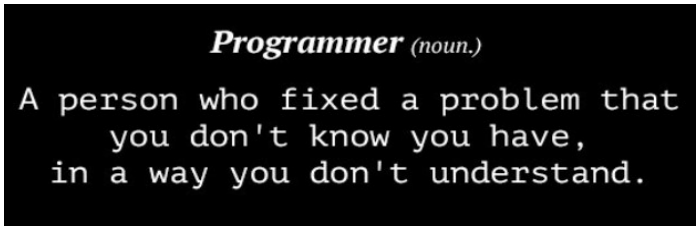


Quelle: Wikipedia

Funktionen und Variablentypen
Normalerweise wird in Coding Guidelines die großzügige Verwendung globaler Variablen als negativ angesehen. Bei der aus Echtzeitgründen notwendigen sequentielle Abarbeitung der Funktionen, müssen im Hintergrund die Eingangsdaten eines Moduls vom letzten Programmdurchlauf zur Bearbeitung verfügbar sein. Die Ausgangsdaten des Moduls müssen zwischengespeichert werden ,bis ein anderes Modul in einem nächsten Durchlauf darauf zugreifen kann. Für diese Programmstruktur sind globale Variable (für alle Module sichtbar) eine sinnvolle Lösung. Man muss darauf achten, dass nur ein Modul die Werte erzeugt und alle andern nur lesen.

Modularisierung wird hier nur als Ordnungsmittel eingesetzt. Die Module (bei Arduino Tabs, kleiner Pfeil oben rechts in der IDE) laufen meist nacheinander ab. In einigen Fällen rufen Module auch Untermodule auf. Professionelle Coding Guidelines gehen von einer sehr häufigen Verwendung von Modulebenen aus. Auch die Übergabe von Werten zwischen den Modulen werden dort in einer definierten Weise vorgegeben. In diesem Programm werden diese Regeln aus Struktur- und Laufzeitgründen nicht konsequent befolgt.

Ansonsten Gilt: Es ist Hobby, nicht alles so verbissen sehen



Quelle: http://www.hongkiat.com

Ablauf
Alle Funktionen auf der Ebene *FreqGenNeu* werden nacheinander aufgerufen (Round-Robbin-Scheduling). Ihre Gesamtablaufzeit resultiert auf der Addition der Laufzeiten der einzelnen Funktionen. Die Laufzeit der einzelnen Funktionen ist unterschiedlich je nach den Pfaden, die durch die anliegenden Bedingungen durchlaufen werden.

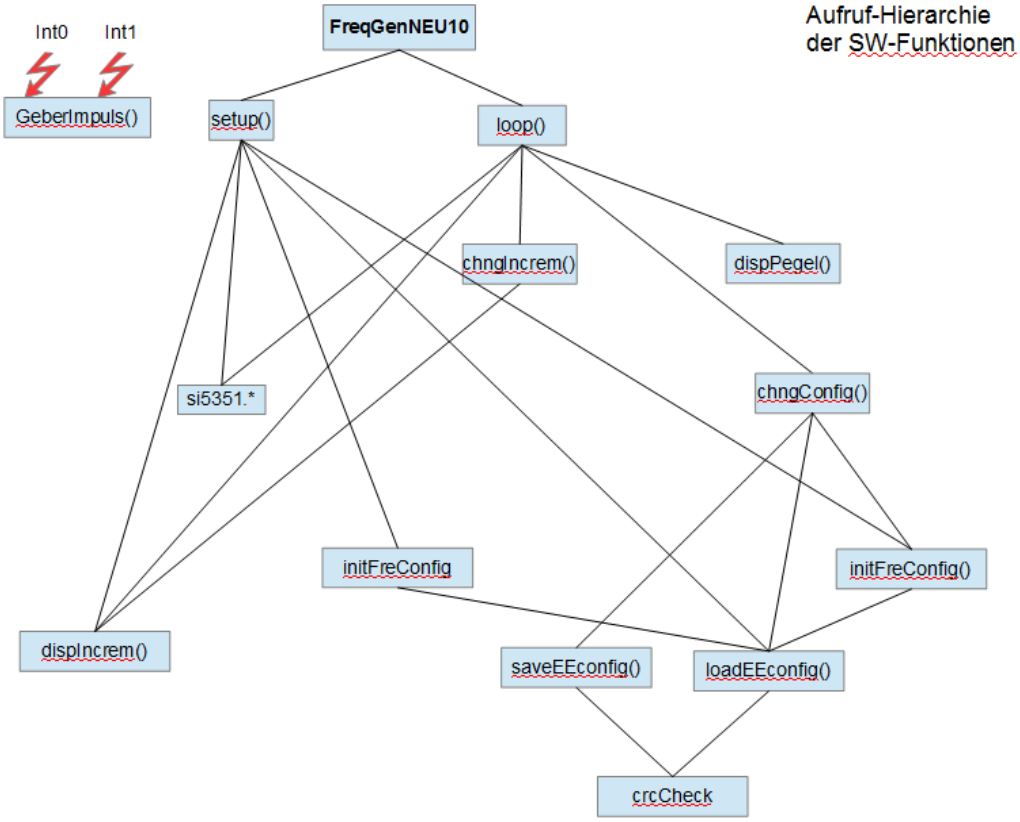
Interrupts
Es wurden zwei Interrupts für die Ausgänge des Opto-Drehgebers verwendet. Sie reagieren auf beide Flanken jedes Ausgangs. Diese Methode ist nur bei Opto-Encoder sinnvoll. Die Interrupts unterbrechen kurz die laufende Hauptschleife, lesen beide Pins ein und setzen ein Flag.

Laufzeit
Die Laufzeit des Gesamtprogramms ist im Prinzip unkrititsch. Sie wird nach oben nur durch den Bedieneindruck begrenzt:

- Die Reaktionszeit des Displays. Das Datenblatt des Display-Controllers verlangt zwischen den Kommandos vom µC einige ms Wartezeit. Die meisten moderen kompatibelen Controller sind glücklicherweise schneller. Falls es Anzeigefehler gibt, ist es sinnvoll *delay()*-Funktionen mit ca. 10-20ms zwischen zwei aufeinanderfolgende LCD-Kommandos einzufügen.
- Die Reaktionszeit auf Impulse vom Drehgeber hat höchste Priorität. Die Impulse selber werden per Interrupt erkannt unnd durch 2 Zeilen Software abgespeichert. Die Auswertung (Frequenzveränderung und Anzeige) geschieht in der Hauptschleife. Der Eindruck der sofortigen Reaktion auf den Drehgeber sollte auch bei schnellem Drehen nicht gestört werden. Falls tatsächlich mehrere Impuls zwischen zwei Auswertungen auftreten, dann werden diese „verschluckt“ und der letzte Impuls zum Zeitpunkt der Auswertung wird gewertet.
- Die Entprellung des Tasters „Frequenz-Inkrement-Stufe“ erfolgt als Non-Blocking-Entprellung. D.h. Die Entprellung blockiert den Programmablauf nicht durch eine Zeit-Warteschleife, sondern schaut in jedem Programmdurchlauf ob der Taster noch betätigt ist und die Entprellzeit (gemessen über einen durch durchlaufenden Hardware-Timer) schon erreicht ist
- Die Entprellung des Taster „EE“ zur Abspeicherung der aktuellen Frequenz und Inkrementwerte arbeitet bewusst mit einer Blocking-Entprellung. Um eine versehentliche Betätigung des Tasters zu vermeiden muss hier 3s der Taster betätigt werden bis die Bestätigung über das EE_Symbol kommt. Während der Zeit wird die Hauptschleife blockiert und sind alle anderen Aktivitäten unterdrückt.

FreqGenNEU
Diese Modul ist das Hauptmodul, die oberste Programmebene. In diesem Modul finden sich zu Beginn Definitionen von Variablen und Konstanten. Sie sind nach Teilfunktionen getrennt.

Die *#include*-Zeilen beschreiben, welche Libraries eingebunden werden. *LiquideChrystal* (LCD), *EEPROM* (Lesen/Schreiben EEPROM) und *Wire* (I2C) sind Standard-Libraries mit Funktionspaketen, die man unter Arduino/Hilfe/Referenz/Libraries finden kann. Dort sind ihre Eigenschaften beschrieben. *stdint* ermöglicht den Zugriff auf alle Registernamen des Prozessors, das ist für die Si5351-Library notwendig.
Eigene (oder aus dem Internet beschaffte) Libraries werden in einem speziellen Ordner gesucht, den die Arduino-IDE bei der Installation anlegt. Der Pfad wäre z.B. *C:\Users\Axel\Documents\Arduino\Libraries\Si5361*. Dort müssen jeweils ein *.cpp und *.h-File eingetragen sein. Arduino verwendet die Libraries in Source-Format. Das ist eigentlich in C-Systemen unüblich, erleichtert aber das Nachschauen im Notfall und hat für unser Projekt keine negativen Auswirkungen.



Alle Konstanten (mit Ausnahme der Zeiten...) sind symbolisch unter Namen abgelegt.
Das erleichert später die Änderung von z.B. Grenzen oder Zeichen zuverlässig über alle Module.
Die Konstanten wurden in den ersten 3 Buchstaben mit ihrer Funktionsgruppe versehen.

FRE, fre: Größe hat mit der Frequenzverstellung zu tun.
DRE, dre: Größe hat mit dem Drehgeber zu tun
EE: Grüße hat mit der EEPROM-Speicherung zu tun
PEG, peg: hat mit Pegel-Messung und -Anzeige zu tun

Alle globalen Variablen sind mit ihrem Typ (Größe/Format) definiert. u*nsigned int* heisst hier dass die Variable ein Wort breit ist (16 Bit*) und positive Zahlen von 0 – 65535 beinhalten kann.
(* Es wurden hier absichtlich ohne Hardware-unabhängige Formatierungen gearbeitet)
Die Typen haben die Eigenschaften wie in den meisten Lehrbüchern. Ausnahme sind l*ong, Float* und *Double*, die bei den 8-Bit-Arduinos nur 4 Byte lang sind. Eine bei Arduino nicht dokumentierte Ausnahme ist der Typ *long long*, der 8 Byte lang ist. Alle Variablentypen und ihre Eigenschaften können jederzeit in IDE unter Hilfe/Referenz betrachtet werden.
Im Bereich der Variablendefinitonen können Variablen auch gleich mit einem Wert bei Start des Programms vorbelegt werden. Dies wird insbesondere bei Strukturen und Feldern benutzt, weil es später viel umständlicher geht. Z.B. wird ein 1-dimensionales Feld (Vektor) *fre_increm[]* zur Ablage der Frequenzinkremente genutzt:

```
unsigned long fre_increm[6] =
{  1, FRE_INCREM_A, FRE_INCREM_B, FRE_INCREM_C, FRE_INCREM_D,
FRE_INCREM_E};
```

Neben der Konfiguration der Ein-/Ausgangs-Pins wird auch die LCD-Library *LiquidChristal* konfiguriert:

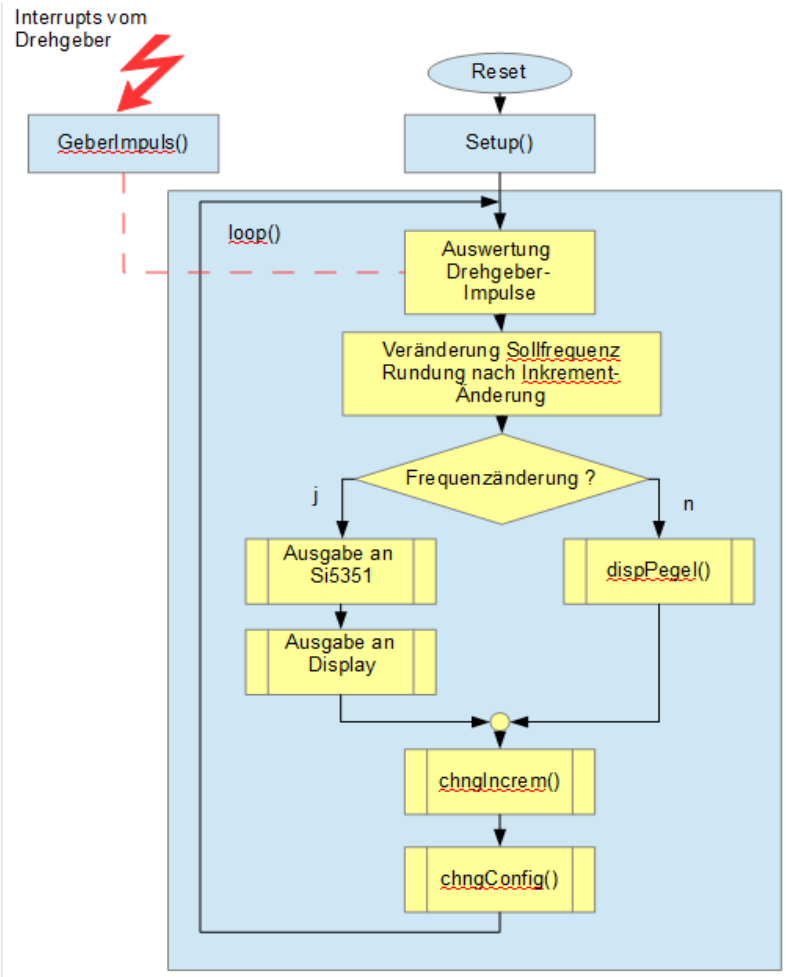
```
LiquidCrystal lcd(12, 11, 10, 9, 8, 7, 6);
```

Hier wird definiert wie das LCD-Displays mit den Prozessorpins verbunden sind (!Arduino-Nummerierung!). Einzelheiten im Kommentar.

Bei der Pegelanzeige werden kundenspezifische Zeichen zur Anzeige auf dem LCD-Display benötigt. Hier wurden pixelweise Zeichen definiert um die feinen Linien auf der Pegelanzeige zu bekommen. Dieses Feld wird später mit einem Ausruf aus der LCD-Library in den Grafik-Controller geladen, der max. 8 kundenspezifische Zeichen auf die Anzeige bringen kann.
Hier fällt auch noch das 2-dimensionale *Feld n[][] der Größe 8x7* auf.

```
byte n1[8] = {
   0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x15}; // Punkte 0-Limie
byte n2[8] = {
  0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x15}; // 1 Linie
byte n3[8] = {
  0x14,0x14,0x14,0x14,0x14,0x14,0x14,0x15}; // 2 Linien
byte n4[8] = {
  0x15,0x15,0x15,0x15,0x15,0x15,0x15,0x15}; //3 Linien
….....
```

Programmstruktur



Ablaufdiagramm FreqGenNEU

setup()

In der Funktion *setup()* befinden sich alle Programmzeilen, die nur einmal nach Reset durchlaufen werden. Sie dienen zur Initialiserung von Hardware und globaler Variablen.

Hier werden die Funktion von Port-Pins (In/Out), Interrupt-Pins zugiesen.
Es wird die Initialisierung des LCD-Controllers sowie der Download der kundenspezifischen Anzeige-Character vorgenommen. Es wird ein Initialisierungstext auf die Anzeige gebracht.

Es werden die Startwerte für Frequenz und Inkrement aus dem EEPROM geladen und die entsprechenden Variablen initalisiert und der Si5351 initalisiert und das erstemal die Startfrequenz auf den Oszillatorbaustein gegeben. Der Übergabewert *SI5351_CLK0* bedeutet, dass der Osillator Nummer 0 mit seinem Ausgang CLK0 aktiviert wird, man kann die beiden anderen Ausgänge CLK1, CLK2 getrennt aktivieren.

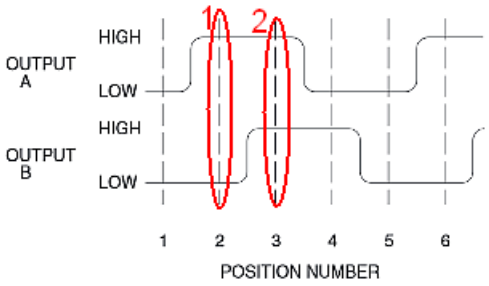
Danach erfolgt die Beschriftung des Displays für den Betrieb.

loop()

Die Funktion *loop()* ist eine Arduino-Spezialität, sie ist die eigentliche Hauptablaufschleife. Das aus dem Lehrbuch bekannte main() hat in Systemen ohne übergeordnetes Betriebssystem wenig Sinn,

da es auch keine Beendung des Programms und die Rückgabe der Rechnerkontrolle an das Betriebssystem gibt. *loop()* wird immer nach Ende von *setup()* gestartet und läuft solange bis entweder entweder Hardware-Reset oder die Abschaltung der Betriebsspannung auftritt. *loop()* ruft nacheinander die Funktionsteile auf.

Bearbeitung Drehgeber Pulse und Veräenderung der Sollfrequenz
Die Interruptroutine *GeberImpuls()* leistet die Vorarbeit und liefert ein Interruptflag *dre_p*, in dem sich die Anzahl von Interrupts nach dem letzten Auftreten sehen lässt. Aussederm werden die beiden Bitwerte für die Ausgangspegel des Drehgebers *dre_geber* geliefert. Vergleicht man den Wert des letzten Durchgangs mit dem aktuell gelesenen Wertes, so kann man bei jeder Interruptflanke sehen in welche Richtung der Grehgeber gedreht wurde.



CLOCKWISE ROTATION		
POSITION	OUTPUT A	OUTPUT B
1		
2	•	
3	•	•
4		•

• INDICATES LOGIC HIGH; BLANK INDICATES LOGIC LOW. CODE REPEATED EVERY 4 POSITIONS

Quelle: Datenblatt des Optischen Encoders ENC 62P22-H0S der Fa.

Es handelt sich um einen einschrittigen Code, der bei jedem Übergang eine eindeutige Identifikation der Änderungsrichtung zulässt.

Es wird eine Musterdifferenz zwischen dem aktuellen und dem vorigen Bitmuster erzegt und dann in einer *case*-Struktur entschieden, in welche Richtung die Frequenz geändert wird. Es kann bei jeder Flanke eine Frequenzänderung durchgeführt werden.

Ein unschöner Effekt bei der Umschaltung des Frequenz-Inkrements war, dass die unteren Stellen des Frequenzwertes stehen blieben, wenn man das Inkrement erhöhte. Deswegen wurde bei der ersten Betätigung des Drehgebers nach Umschaltung des Frequenzinkrementes eine richtungsabhängige Rundung eingeführt:

	Ohne Rundung	**Mit Rundung**
Altes Frequenzinkr.:	1Hz	1Hz
Alte Frequenz:	3700123Hz	3700123Hz
Umschaltung Frequenzinkr.:	100Hz	100Hz
Neue Freq. nach Freq.änderung:	3700223Hz usw.	3700200Hz

Die Rundung passiert richtungsabhängig und ist im *case*-Konstrukt der Mustererkennung untergebracht.

Frequenzausgabe
Die Programmteile zur Frequenzausgabe über Anzeige und den Si5351 werden aus Zeitgründen nur ausgeführt, wenn eine Frequenzänderung erfolgte. Die aktualisierte Sollfrequenz wird auf eine Ober- und Untergrenze begrenzt (Gründe: Grenzen des Oszillators / der Anwendung). Danach wird der Frequenzwert an die richtige Stelle auf dem Display ausgegeben. Die nicht beschriebenen Stellen der Frequenzanzeige werden mit Leerzeichen überschrieben, damit bei kleiner werdenden Zahlenwerten keine Restanzeigen stehenbleiben.

Die folgende Ausgabe der Frequenz an den Oszillatorbausteín ist das Kernstück des Programms:

```
// Ausgabe Frequenz an Si5351
si5351.set_pll(SI5351_PLL_FIXED, SI5351_PLLA);
//Casting der 4-Byte-Long Variablen auf eine 8-Byte Long-Variable
// Uebergabe der Sollfrequenz
si5351.set_freq((long long unsigned int)fre_frequenz * 100ULL, SI5351_PLL_FIXED, SI5351_CLK0);
fre_frequenz_alt = fre_frequenz;
fre_new = LOW;
```

Die Beschreibung der Library-Funktionen und deren Parameter finden sich den Beschreibungen der Library-Sourcefiles *Si5351.cpp*:

```
* set_pll(uint64_t pll_freq, enum si5351_pll target_pll)
*
* Set the specified PLL to a specific oscillation frequency
*
* pll_freq - Desired PLL frequency
* target_pll - Which PLL to set
*     (use the si5351_pll enum)
```

und

- ```
  * set_freq(uint64_t freq, uint64_t pll_freq, enum si5351_clock output)
  *
  * Sets the clock frequency of the specified CLK output
  *
  * freq - Output frequency in Hz
  * pll_freq - Frequency of the PLL driving the Multisynth
  *   Use a 0 to have the function choose a PLL frequency
  * clk - Clock output
  *   (use the si5351_clock enum)
  ```
-

Anschliessend wird das Flag *freq_new* gelöscht und für den nächsten Durchlauf die aktuelle Frequenz als alte Frequenz abgespeichert.

Anregung zur Erweiterung: Der Si5351 enthält eine Register zur Frequenzkorrektur von Abweichungen der Quarzfrequenz. Die Library enthält einen Funktionsaufruf dazu. Anstelle eine Frequenzkorrektur auf Eingabeebene, wirkt die interne Korrektur gleichmäßig auf einen breiten Frequenzbereich.

Wurde keine neue Frequenzänderung festgestellt wird anstelle der Frequenzausgabe über *dispPegel()* die Pegelanzeige eingelesen und aktualisiert. Da beide Programm große Laufzeiten haben, werden sie nie gleichzeitig ausgeführt. Die Frequenzausgabe hat Vorrang, die Pegelanzeige wird im nächsten Durchlauf aktualisiert.

Es folgen anschliessend noch die Aufrufe *chngIncrem()* zur Änderung der Inkrementstufe und *chngConfig()* zur Neuabspeicherung der aktuelle Konfiguration Frequenz / Frequenz-Inkrementstufe .

chngConfig( )

Die Funktion dient zur Abspeicherung der aktuellen Werte als Startwert als Einschalt-Konfiguration. Sie ermöglicht die Abspeicherung des Startwertes von Frequenz und Frequenz-Inkrement.
  ```

ins EEPROM durch die EE-Bedientaste. Die Funktion wird zyklisch bei jedem Programmzyklus aufgerufen. Es wird bei Aufruf der Funktion geprüft, ob die EE-Taste betätigt wurd. Der Pegel LOW ergibt sich aus der Schaltung, am Eingangspin liegt ein Pull-Up-Wiederstand. Die Taste zieht bei Betätigung den Eingang gegen Massepotential.

Ist die Taste nicht betätigt, dann wird der Rest der Programmzeilen übersprungen und die Funktion verlassen.
Ist der Taster betätigt, dann wird die Entprellung des Taster durch wiederholtes Abfragen nach einer Entprellzeit durchgeführt. Wenn der Taster damit ausreichend lange betätigt wurde, dann wird das „Aktiv"-Symbol „*" angezeigt. Nach einer Anzeigezeit von 1s werden die aktuellen Frequenz- und Inkrement-Werte in die Ablagestruktur im RAM geschrieben dann werden die aktuelle Frequenz und die aktuelle Inkremet-Stufe in die *EE_Ablage*-Struktur kopiert.

Mit Aufruf der Funktion *saveEEconfig()* wird die Ablage mit eine Checksumme gesichert und in das EEPROM gespeichert. Danach werden mit der Funktion *loadEEconfig()* die Werte wieder zurückgelesen Dabei wird der Checksumme überprüft. Ist die EEPROM-Speicherung gelungen, dann erscheint für 3s das Symbol „EEPROM Abspeicherung gelungen" 0Dh (Katana/Kanji mi ?):

Trat beim Schreiben eine Fehler auf, dann wird auf der Anzeige wird dann das Zeichen ('/') als Fehlermeldung für ausgegeben. Die aktuellen Werte on Frequenz und Frequenz-Inkrement werden nicht verändert.

Die Funktion wird erst verlassen, wenn die EE-Taste als nichtbetätigt erkannt wurde.

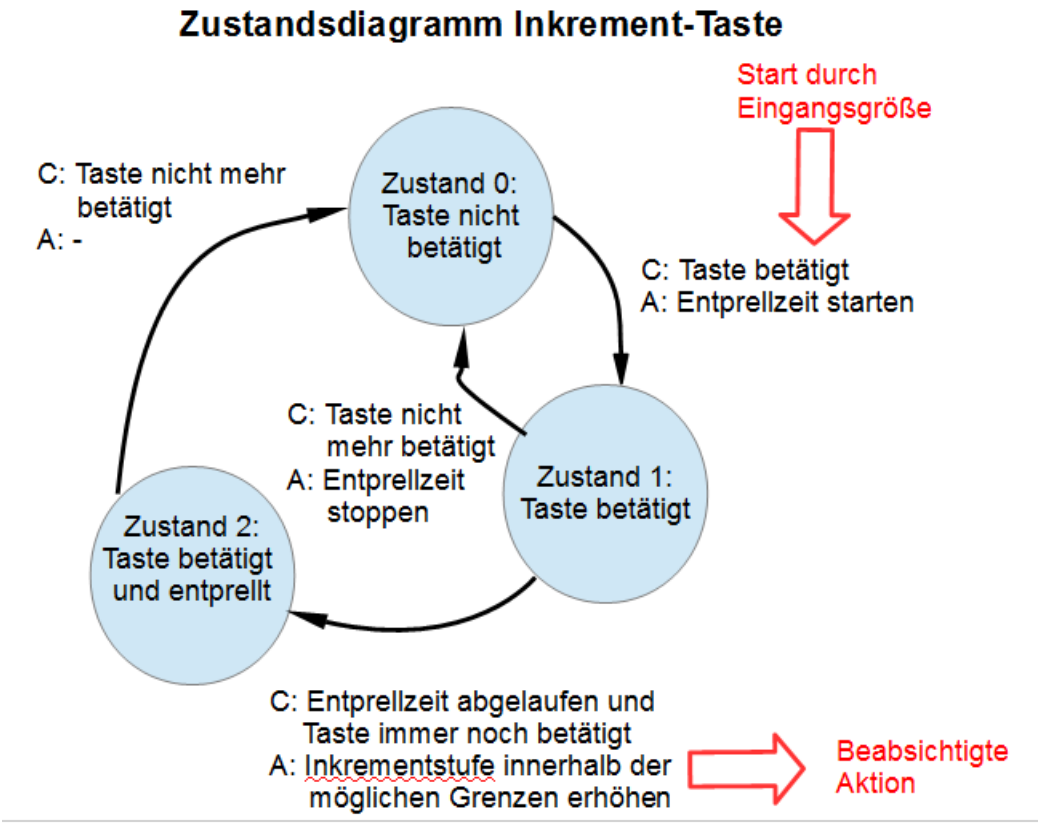
Die Notwendigkeit der Speicherabsicherung liegt in der endliche Anzahl der Schreibzyklen des EEPROM-Speichers. (Meist ist es heute eine Emulation durch FLASH-Speicher, die sich in mehreren Speicherbänken umschalten um eine dem EEPROM-ähnliche Anzahl Schreibzyklen zur ermöglichen). Beim Atmega328 werden ca. 100.000 Schreibzyklen angegeben. Eine weitere Fehlerquelle sind Spannungseinbrüche auf der Betriebsspannung währen des Schreibens.

Anregungen:

- Es können mehrere Bänder mit Bandgrenzen angelegt werden.
- Bei jedem Abschalten könnten die gerade eingestellten Werte in das EEPROM geschrieben werden. Die Betriebsspannung muss dann jedoch durch zusätzliche Maßnahme noch solange anliegen, bis das EEPROM geschrieben wurde (z.B. Relais durch SW gesteuert.)

chngIncrem()

Dies Funktion bewirkt Umschaltung des Frequenz-Inkrements durch den Drehgeber-Taster. Diese Funktion wird evtl. während des Abstimmens mehrfach ausgeführt und sollte damit den Programmdurchlauf nicht wesentlich stören. Daher wird hier die Entprellung des Tasters als „Non-Blocking-" Funktion benötigt. Dazu eignet sich am besten die Struktur eines Zustandsautomaten (Finite State Machine FSM). Er kann als Statechart dargestellt werden:



Das Konstrukt liest sich nicht als Ablaufplan, es zeigt vielmehr 3 Betriebsmodi (genannt Zustände) die solange bei jedem Durchlauf abgearbeitet werden, solange sich nicht durch eine Entscheidung eine Umschaltung auf einen anderen Betriebmodus ereignet. Diese Umschaltung (Zustandsübergänge) geschieht durch eine logische Bedingung (C: Condition) und löst beim Durchlaufen (während der Umschaltung) eine Aktion (A: Action) aus. Diese Aktionen können interne Variable, aber auch externe Größen, z.B. Ausgänge betreffen. Wenn das Programm also in einem Zustand ist, dann ist damit nicht automatisch ein Ausgangswert betroffen.

Solche Konstrukte erlauben eine gute Planung der Effekte und lassen sich leicht mit case-Konstrukten erstellen. Der Vorteil dieser Realisierung ist die Tatsache, dass der Programmablauf ungestört über den gerade aktuell eingestellten Betriebsmodus (Zustand) weiterläuft bis die Bedingung zum Übergang in den nächsten Betriebsmodus eintritt (z.B. Erreichen der Entprellzeit). Die Zeitmessung geschieht hierbei durch einen Hardwaretimer im Hintergrund, der quarzgetrieben ständig Zahlen generiert, die im Abstand von einer Millisekunde weitergezählt werden: die Funktion *millis()* aus der Arduino-Standardbibliothek. Der aktuelle Zustand wird in einer Zustandsvariablen *dreTastZust* gespeichert. Die Weiterschaltung des Zustands geschieht durch Änderung dieses Wertes.

crcCheck()

Diese Funktion soll eine CRC-Checksumme (Cyclic Redundncy Check) in Form eines Bytes über die im EEPROM abgelegten Bytes bilden.
Diese Checksumme wird vor dem Schreiben in das EEPROM als zusätzlicher Datenwert gebildet und abgespeichert. Nach dem Lesen der Bytes aus dem EEPROM kann durch erneutes Bilden der CRC-Checksumme ermittelt werden ob die Datenablage korrekt erfolgte.

dispIncrem()

Diese Funktion zeigt den Wert des Frequenz-Inkrements auf dem Display an.

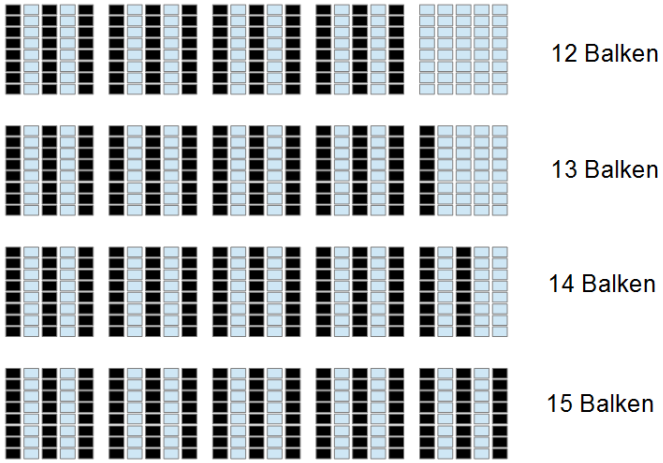
Es wird der Cursor des Display mit der Funktion *lcd.setCursor()* aus der Arduino-LiquidCristal-Bibliothek auf den definierten Platz auf dem Display gesetzt.
Dann wird die alte Darstellung durch Leerzeichen überschrieben, die neue Darstellung könnte ja weniger Stellen haben und die nichtbenutzten Stellen sollen ja dunkel bleiben.
Wenn der aktuelle Wert des Frequenz-Inkrements < 1000Hz ist, dann wird die Zahl ohne Zusätze ausgegeben. Wenn der Wert des Frequenz-Inkrements >= 1000Hz ist dann wird der Wert durch 1000 geteilt, der Wert ausgegeben und mitdem Zecihen 'k' wie kilo ausgegeben.

Freq.inkrem. Anzeige
100Hz → **100**
1000Hz → **1k**

dispPegel()

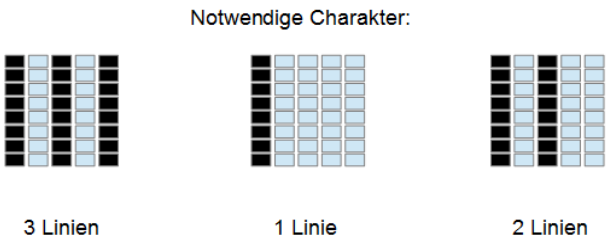
Mit dieser Funktion erfolgt das Schreiben der Pegel-Symbole auf die LCD- Anzeige. Die Funktion soll experimentell zeigen, welche Qualität und Nutzbarkeit der Anzeigen von Bargrafen auf einer Character-orientierten LCD-Anzeige möglich ist. Die Nutzung könnte z.B. für Batteriespannung, S-Meter oder SWR-Anzeige genutzt werden. Ein Bargraf kann die Tendenzen eines Signals besser erfassbar machen als ein Zahlenwert. Wie kann man unter Nutzung weniger Character soetwas realisieren und wie schnell ist diese Anzeige ?

Die Idee ist die Verwendung von kundenspezifischen Zeichen etwa der Form:



Durch Aneinanderreihung von kundenspezifischen Linien-Charactern kann man mit wenigen Charactern einen Balken mit hoher Auflösung darstellen.

Man benötigt dazu nur 3 kundenspezifische Character:



dispPegel() verwendet 6 Character (*PEG_BREIT*). Das ergibt 3 x 6 = 18 mögliche Feinstriche.

Das Analogsignal wird über einen Analogpin des Prozessors eingelesen. Es wird durch die Anzahl der Anzahl Character geteilt. Da im Zahlensystem Integer gerechnet wird, fallen Reste < 1 weg und man bekommt die Anzahl der 3-Strich-Blöcke, die dargestellt werden müssen. Der Rest wird mit dem Modulo-Operator % ermittelt. Er ergibt den Rest der Teileroperation: Mit diesem Rest wird entschieden, ob der letzte dargestellte Character entweder 0,1,2 oder 3 Striche haben sollte. Die Werte 0 und drei ergeben theoretisch entweder „einen leeren Character“ oder „einen 3-Linien-Character“. Beides müsste durch die vorherige ganzzahlige Teilung und Verarbeitung bereits erledigt worden sein.

Anzeigemodule mit HD44780-kompatiblen Controllern verwenden häufig komplette Kopien des Datenblatts des ursprünglichen Controllers. Dessen Reaktionszeiten auf Kommandos ist nicht unerheblich daher finden sich im Code delay()-Funktionen, die Wartezeiten im ms-Bereich vor dem Übergeben des nächsten Kommandos erreichen sollen. Es besteht jedoch der Eindruck, dass viele neue Controller auch schneller Beschrieben werden können. Gerade bei häufiger Schreibtätigkeit, wie bei dieser Funktion ist das Ausprobierewn geringerer Wartezeiten sinnvoll.

Anregungen:

- Die Löschung aller Zeichen vor Neubeschreibung könnte ersetzt werden durch ein Beschreieben der noch verbleibenden Charactern NACH der Balkendarstellung
- Eine Überprüfung ab welchem Character der alte Balken im aktuellen Programmdurchlauf überschrieben werden kann spart Schreibaktionen. Die ersten Character bleibe dann einfach stehen.

initFreConfig()

Diese Funktion initialisiert die Konfiguration der Frequenzeinstellung (Startfrequenz, Frequenz-Inkrement).
Nach Aufruf wird das EEPROM-Anzeigefeld gelöscht und der Wert „EE“ eingetragen.

Mit der Funktion *loadEEconfig()* wird der Inhalt des EEPROMS in die Frequenz-Ablage im RAM kopiert. Dann wird die Prüfinformation *.check* auf Gültigkeit überprüft. Ist die Prüfinformatio OK, dann wird das Zeichen „gültig“ 0Dh ausgegeben.

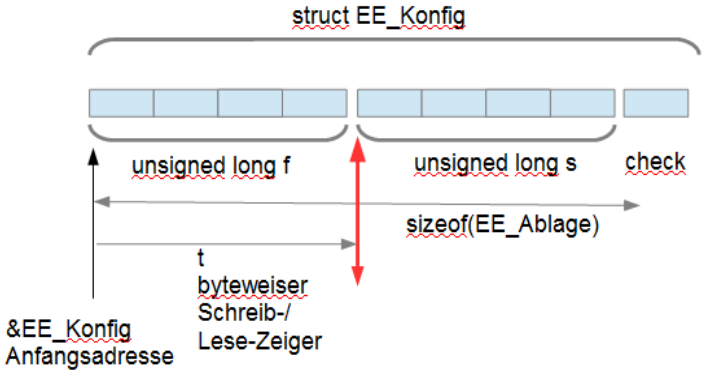
Ist dies nicht der Fall so wird die Frequenzablage mit den Intialiasierungkonstanten *FRE_START* und *FRE_INCREM_C* gefüllt. Es wird das Zeichen „ ungültig“ B0h '-' ausgegeben.

loadEEconfig()

Dieses Modul erledigt den lesenden Zugriff auf das EEPROM. Dabei wird das Ablagefeld in einzelnen Bytes mittels der Arduino-Funktion *EEPROM.read()* aus dem EEPROM geholt. Die EEPROM-Ablagestruktur wird als eine Reihe von Bytes betrachte, die bei der RAM-Adresse *&EE_Ablage* startet und die Länge *sizeof(EE_Ablage)* hat. Mit der Zählvariablen *t* werden die Bytes aus der Reihe zum Lesen adressiert.

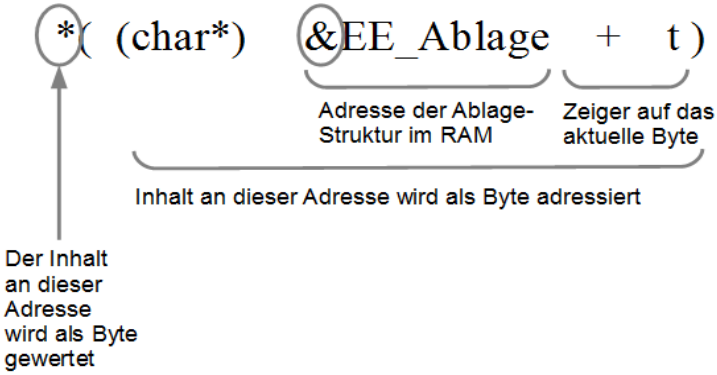
Die Adressierung der EE_Ablage als Bytes und das Ablegen der unstrukturierten Bytes aus dem EEPROM wird über eine pfiffige Zeigerkonstruktion durchgeführt. C konnte als erste Hochsprache mit physikalisch Adressen und typisierten Zeigern rechnen. Wird ein Zeiger vom Typ Byte definiert dann ändern sich die pysikalischen Adressen bei Zeigererhöhung um 1. Wird ein Zeiger vom Typ Long definiert, so ändert sich die Physikalische Adresse um 4. Diese Eigenschaft zusammen mit den Operatoren & (Adresse von...) und * (Inhalt von...) sowie der Möglichkeit eine Variable durch

Type-Casting als veränderten Datentyp anzusprechen sind hier sehr hilfreich.

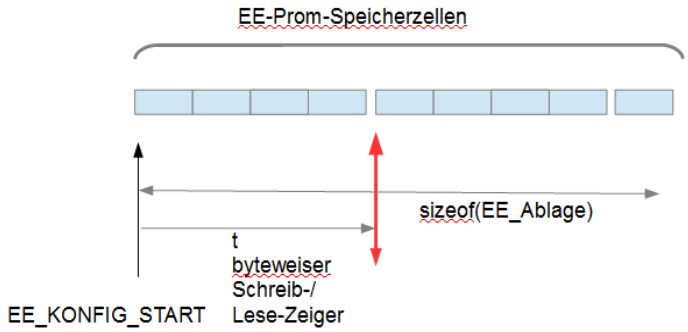



```
struct EE_Konfig
{
  unsigned long f; //Ablage Startfrequenz
  unsigned long s; // Ablage Stufe Frequenz-Inkrement
  byte check; // Ablage Kennzeichnung Gueltigkeit
}
```

Struktur *EE_Konfig* im RAM und byteweiser Zugriff



Code für die byteweise Adressierung des RAM-Ablagefeldes



Zugriff der Dateninhalte im EEPROM

```
EEPROM.read(EE_KONFIG_START + t)
```

Code für byteweisen Zugriff im EEPROM

saveEEconfig()

Diese Funktion schreibt die Inhalte der Struktur *EE_Ablage* in das EEPROM. Es bedient sich dabei der Adressierung im RAM und im EEPROM wie in der vorigen Funktion gezeigten Mechanismen.