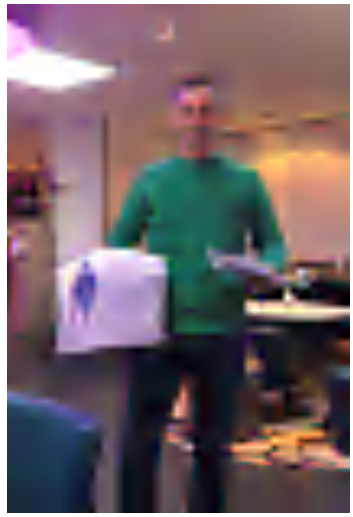
Radio zendamateur Examen

Om 15.15 begon het examen en bleef ik achter in de kantine en had een leuk gesprek met novice amateur Hans PD0HZS die zijn F-examen net had afgesloten. Hij kwam uit Alkmaar, en hij herkende zelfs mijn call (regio Tilburg) en de activiteiten rond scouting en onze QRZ pagina (websites van zendamateurs). Ik maakte nog een praatje met de examen-commissie, die er binnenkort mee stopt als het examen bij het CBR komt. Vanaf deze plek nogmaals hartelijk dank aan de Stichting Radio Examens voor de vele examens die jullie hebben verzorgd.

Geslaagd?
Gerard kwam na ongeveer 25 minuten naar buiten. Opgelucht maar toch nog even afwachten want je mag alleen de antwoorden meenemen. De vragen blijven in het lokaal. We bleven tot het einde hangen, en kregen toen het verlossende antwoord, want de antwoorden komen dan ook online. Geslaagd, met 8 foutjes, net iets meer als verwacht, maar dat zal een zorg zijn, want het papiertje is binnen!
82% van de Novice kandidaten was geslaagd. Het examen Full License dat eerder was gehouden, was blijkbaar moeilijk want daar slaagde slechts 50%

Hoe kijk je, nu je geslaagd bent terug op je cursus en examen?

Ik sprak laatst met mijn vrouw dat ik in het begin erg opkeek tegen de cursus, vooral de techniek. Hoe dichter ik echter bij het examen kwam, merkte ik dat ik ook een persoonlijke ontwikkeling doormaakte. Ik pakte makkelijker kleine klusjes op om iets te repareren en ik heb meer geduld en zoek de dingen uit als ik het niet weet.


Geslaagd! Dat lucht op

Nu zo snel mogelijk on air, want daar heb ik het voor gedaan. De meeste spullen heb ik al staan om op VHF en UHF uit te komen. Na een tijdje te hebben geluisterd kan ik straks ook mee doen.
73 en hopelijk inspireert het mensen om ook hun licentie te gaan halen en anderen om te zorgen voor een stukje coaching.

Het artikel is geschreven vanuit de Onafhankelijke Radioamateurs Brabant, de O-R-B www.o-r-b.nl
Wekelijks verzorgen we sinds 2004 vanuit Tilburg om 20.00 een gezellige radioronde op 145.400.

73 Frank PF1SCT

APRS Transceiver - Inleiding tot de software.

Robert de Kok, PA2RDK

Zoals aangekondigd in de vorige Razzies, ben ik de afgelopen weken bezig geweest met mijn nieuwe APRS VHF transceiver. Allereerst de software.

Omdat mijn kwaliteiten niet voorzien in de zelfstandige ontwikkeling van een userinterface, heb ik mij laten inspireren door de SI4735 radio van Gert PE0MGB en de layout van het scherm van mijn FT991A.

Hierbij kwam ik tot de layout zoals getoond op de volgende bladzijde.

Omdat ik de wensen en eisen nog niet allemaal wil en kan overzien, is een Agile werkwijze voor de hand liggend. Dit betekent dat de software flexibel moet worden ingericht, zodat het

eenvoudig is om er knopjes bij te bouwen en de layout en functionaliteit eenvoudig aan te passen.

Omdat de wens bestaat de radio ook middels internet te kunnen bedienen, is het noodzakelijk de functies zodanig aan te roepen dat dit niet alleen met de knopjes mogelijk is. Het programma moet zeker geen spaghetti worden, want dan loop ik binnen een paar weken hartstikke vast en wordt onderhoud en uitbreiding een drama en is de lol van programmeren snel over. Kortom het programma moet netjes modulair worden opgezet.

Nu wil ik zeker niet beweren dat ik een fantastische programmeur ben en onder druk van tijd en haast ben ook ik regelmatig in de verleiding een (te) korte route te kiezen maar ik ben wel zo arrogant te denken dat ik weet hoe het zou moeten.

Daarom wil ik dit eerste deel van de bouw van de transceiver wijden aan een aantal gebruiken en technieken en mogelijkheden met betrekking tot de ontwikkeling van de (Arduino) software in het algemeen, zodat deze onderhoudbaar, leesbaar en uit te breiden of aan te passen is voor derden. De tekst is niet bedoeld als beginnerscursus programmeren, ik ga er van uit dat je al de nodige ervaring hebt.

Voor velen zal het gesneden koek zijn, maar wellicht pik je toch er iets van op!

Voor diegenen die de transceiver willen bouwen en gewoon gebruik willen maken van het programma en helemaal geen aspiratie hebben zelf met software aan de gang te gaan, moeten nog even geduld hebben, volgende maand word ik concreter.

De bouw is nog volop in ontwikkeling maar mocht je erg nieuwgierig zijn of haast hebben, het programma en het schema in de huidige staat zijn beschikbaar op Github.

Het modulair bouwen van software vraagt eigenlijk om het object georiënteerd (=object oriented =OO) bouwen. Deze route heb ik om verschillende redenen niet gekozen, maar de principes van OO ondersteun ik wel maximaal.

De belangrijkste kenmerken zijn dat een module zelfstandig functioneert, de functie waarvoor de module is gebouwd altijd wordt uitgevoerd door deze module, er wordt dus geen code gekopieerd. Aanvullend geldt dat een module zich met één losstaande taak bezighoudt en niet afhankelijk is van andere delen van het programma.

Een goed voorbeeld is het tekenen van het scherm. Deze tekent alleen het scherm en doet geen berekeningen aan bijvoorbeeld frequenties. Een module kan wel uit sub modules bestaan, bijvoorbeeld het tekenen van het scherm bestaat onder andere uit modules om knoppen, de meter en de frequentie informatie te tekenen. Deze sub modules moeten ook zelfstandig te gebruiken zijn.

Waarom doen we dit zo: er zijn verschillende redenen om de frequentie opnieuw te tekenen, als deze handmatig wordt veranderd, als de PTT wordt ingedrukt, als er iets met APRS gebeurt etc. Als we dan steeds opnieuw de code voor het tekenen van de frequentie herhalen en we besluiten bijvoorbeeld een ander lettertype te gaan gebruiken dan dienen we dit op de verschillende plekken te doen. Het risico er een te vergeten is groot met als gevolg dat we bijvoorbeeld bij het indrukken van de PTT

opeens tegen een ander lettertype aan kijken. Dat ziet er niet uit natuurlijk!

Om deze reden kan een functie met maar één regel heel functioneel blijken: Stel dat het tekenen van de frequentie er als volgt uit ziet:

```
tft.drawString(rxFrequency, 220,90,1);
```

(Hierin is rxFrequency de te tonen frequentie, 220 de positie op x-as, 90 de positie op de y-as en 1 het fontnummer.)

Dan is het heel aantrekkelijk deze regel te kopiëren, maar als de frequentie een beetje verplaatst moet worden dan mogen we dit op verschillende plekken aanpassen.

Als we er deze functie van maken, hoeven we alleen die aan te roepen en kunnen deze op een plaats aanpassen als de frequentie een beetje verplaatst moet worden:

```
void function drawFrequency(){
  tft.drawString(rxFrequency, 220,90,1);
}
```

In de functie drawScreen kunnen we de functie drawFrequency aanroepen:

```
void function drawScreen(){
  drawFrequency();
}
```

Maar natuurlijk kunnen we drawFrequency op andere plekken ook aanroepen.

Laten we deze functie een beetje uitbreiden, stel we hebben een globale variabele (globaal wil zeggen dat de variabele overal in het programma beschikbaar is, hierover later meer.) ‘isTransmitting’. Dit is een boolean, true als er wordt gezonden en false als er wordt ontvangen. Als er wordt gezonden moet de frequentie in het rood worden getoond, bij ontvangst in het geel. Dit zou bijvoorbeeld zo kunnen:

```
void function drawFrequency(){
  If (isTransmitting) {
    tft.setTextColor(TFT_BLACK, TFT_RED);
  } else {
    tft.setTextColor(TFT_BLACK, TFT_YELLOW);
  }
  tft.drawString(rxFrequency, 220,90,1);
}
```

Dit zal gewoon werken, maar de functie (module) is nu afhankelijk van het bestaan van de variabele ‘isTransmitting’ en kan dus niet meer zelfstandig bestaan. Dit moet je daarom niet willen. Geloof mij, programmeer een poosje door en ik krijg vanzelf ergens een keer gelijk van je.

```
void function drawFrequency(bool setTXColor){
  If (setTXColor) {
    tft.setTextColor(TFT_BLACK, TFT_RED);
  } else {
    tft.setTextColor(TFT_BLACK, TFT_YELLOW);
  }
  tft.drawString(rxFrequency, 220,90,1);
}
```

Dit is een nettere oplossing, we gebruiken nu een lokale variabele ‘setTXColor’ (een lokale variabele is alleen bekend binnen de functie). De waarde van de variabele geven we mee met de aanroep van de functie, dus als volgt:

```
void function drawScreen(){
  drawFrequency(isTransmitting);
}
```

De aanroep drawFrequency(); werkt nu niet meer, we moeten nu een boolean parameter meegeven, dus drawFrequency(isTransmitting);

Technisch is dit een prima oplossing, maar het kan flexibeler. Als we de gewenste kleur meegeven in plaats van de variabele ‘isTransmitting’ kunnen we ook een derde of vierde kleur gebruiken. De functie ziet er dan als volgt uit:

```
void function drawFrequency(uint16_t btnColor){
  tft.setTextColor(TFT_BLACK, btnColor);
  tft.drawString(rxFrequency, 220,90,1);
}
```

En de aanroep van de functie kan dan de volgende zijn:

```
void function drawScreen(){
  if (isTransmitting){
    drawFrequency(TFT_RED);
  } else {
  drawFrequency(TFT_YELLOW);
  }
}
```

Even tussendoor: Nu ben ik niet per definitie een voorstander van het proberen zoveel mogelijk functionaliteit op een regel te proppen, ik weet het, het is een sport en er worden hele wedstrijden rondom georganiseerd, maar bovenstaand voorbeeld is mij toch te dol. De hele functie kan verkort worden naar:

```
void function drawScreen() {
  drawFrequency(isTransmitting?TFT_RED:TFT_YELLOW);
}
```

Veel talen ondersteunen deze schrijfwijze, het is een (dubbel) verkorte if/else.
Eigenlijk staat er if (isTransmitting==true) then TFT_RED else TFT_YELLOW.

IsTransmitting is een boolean, dus isTransmitting is al isTransmitting en hoeft dus niet te worden uitgeschreven met ==true. Andersom kan je ook op deze manier op false checken:

```
 drawFrequency(!isTransmitting?TFT_YELLOW:TFT_RED);
```

Dit doet hetzelfde en betekent:
if (isTransmitting==false) then TFT_YELLOW else TFT_RED. Het uitroepteken voor isTransmitting betekent 'not', dus staat er eigenlijk 'if not isTransmitting then TFT_YELLOW else TFT_RED'. Het vraagteken vervangt de 'if', en de : vervangt de 'else'.

Deze wijze van een if/else opschrijven kan je nesten, dus in elkaar vlechten. Stel we hebben nog een globale variabele 'openSquelch', deze is true als de squelch open staat en er dus geluid uit de speaker komt. We willen dat de frequentie groen is als de squelch open is, rood als er wordt gezonden en anders geel. Dit lukt op deze wijze:

```
drawFrequency(isTransmitting?TFT_RED:openSquelch?
TFT_GREEN:TFT_YELLOW);
```

De eerste else is nu vervangen door een tweede if/else paartje.

Verder met onze functie: Welke en het aantal parameters dat je wilt meegeven aan een functie is oneindig. Stel dat je ook het fontnummer extern wilt kunnen bepalen, dan gaat de functie er zo uitzien:

```
void function drawFrequency(uint16_t btnColor, int fontNummer){
  tft.setTextColor(TFT_BLACK, btnColor);
  tft.drawString(rxFrequency, 220,90,fontNummer);
}
```

Bij de aanroep van de functie is het nu wel verplicht het fontnummer mee te geven, laatstgenoemde voorbeeld gaat er dan als volgt uitzien:

```
drawFrequency(isTransmitting?TFT_RED:openSquelch?
TFT_GREEN:TFT_YELLOW, 1);
```

Duidelijk? Mooi.

Wel vervelend dat we nu altijd de twee parameters moeten meegeven, het gebruikte fontnummer is vrijwel altijd 1 en maar op 1 plek in het programma afwijkend. Daarvoor zijn er overloads bedacht, een functie wordt overloaded door dezelfde functie maar met minder of afwijkende parameters:

```
void function drawFrequency(uint16_t btnColor){
  drawFrequency(btnColor,1)
}
```

Deze functie kan bestaan naast onze eerste functie. De functie heeft maar 1 parameter en roept dezelfde functie met 2 parameters aan, waarbij de tweede parameter (het fontnummer altijd een 1 is. Je kunt hierbij nog verder gaan:

```
void function drawFrequency(){
  drawFrequency(TFT_YELLOW,1)
}
```

Dit roept ook onze eerste functie aan, altijd met de kleur TFT_YELLOW en font 1.
Maar ook dit kan:

```
void function drawFrequency(bool isTransmitting){
  drawFrequency(isTransmitting?TFT_RED:TFT_YELLOW,1)
}
```

We geven nu een boolean mee in plaats van een kleur en fontnummer, is die true dan wordt de frequentie rood, anders geel. Het fontnummer is altijd 1.

De compiler weet welke versie (welke overload) van een functie gebruikt moet worden. Ik maak veelvuldig gebruik van deze mogelijkheid, kijk maar eens naar de functie ‘drawButton’ in het transceiver programma.

Nog even tussendoor: over TFT_YELLOW en de andere kleuren, dit is een handig grapje in c++ (Arduino is eigenlijk gewoon C++ met een schilletje eromheen om het eenvoudiger te maken). Dit is een globale constante. Een constante wordt in Arduino gedefinieerd met #define TFT_YELLOW 0xFFE0. Hierbij is 0xFFE0 de HEX-waarde van de kleur geel. Als we nu iets geel willen kleuren kunnen we TFT_YELLOW gebruiken in plaats van 0xFFE0. Een stuk beter te lezen en onthouden toch?

Dit soort defines of constantes worden veelvuldig gebruikt en maken het lezen van het programma een stuk makkelijker. Het is een goede gewoonte om alle defines bovenin het programma te zetten.
#define kan worden gebruikt voor hex waardes, maar ook voor getallen of strings.

In plaats van #define wordt ook regelmatig const gebruikt, #DEFINE aprsFreq 144800 gedraagt zich hetzelfde als const uint32_t aprsFreq = 144800;

Technisch gaat de compiler anders om met de twee verschillende methodes, maar het resultaat is hetzelfde. In beide gevallen hebben we in het programma een globale variabele aprsFreq beschikbaar met de waarde 144800. De inhoud van de variabele kan niet worden gewijzigd, is constant.

We hebben nu variabelen aan een functie meegegeven, andersom kan een functie ook gegevens teruggeven. In bovenstaande voorbeelden beginnen alle functies met ‘void’, dit betekent leeg, er komt geen waarde terug van de functie. Een functie een waarde laten teruggeven kan op deze wijze:

```
int myIntFunction(){
  return 3;
}

String myStringFunction(){
  return “Text”;
}

bool myBoolFunction(){
  return true;
}
```

In plaats van void begint de functiedefinitie met het type variabele dat wordt teruggegeven. Dus in bovenstaande voorbeelden een integer, een string en een boolean. De functie MOET een waarde teruggeven, anders wil het programma niet compileren.
Dit gaat dus fout:

```
bool myBoolFunction(){
  Bool test = true;
  If (test) return true;
}
```

Wist je trouwens dat if (test) hetzelfde betekent als if (test==true)?

If (!test) betekent if (test==false)!
Er wordt alleen een waarde teruggegeven als test true is. De compiler gaat hierop onderuit.
Ondanks dat test hard op true is gezet. Op deze manier werkt het wel:

```
bool myBoolFunction(){
  Bool returnValue = false;
  Bool test = true;
  If (test) returnValue = true;
  return returnValue;
}
```

Spreekt voor zich denk ik. Het woord ‘return’ wordt altijd gevolgd door de returnwaarde en breekt de functie af. Het mag dus ook zo:

```
bool myBoolFunction(){
  Bool test = true;
  If (test) return true;
  return false;
}
```

Doet precies hetzelfde maar is (vind ik) slechter leesbaar. Ik ben er een voorstander van de functie altijd op de laatste regel te laten eindigen met een return waarvan de waarde ergens in de functie wordt gezet. Maar dit is persoonlijk.

Een voorbeeld: stel we willen het bepalen van de kleur van de frequentie in een functie stoppen:

```
uint16_t getFrequencyColor(bool isTransmitting, bool openSquelch) {
  uint16_t returnColor = TFT_WHITE;
      returnColor = isTransmitting?TFT_RED:openSquelch?TFT_GREEN:TFT_YELLOW;
  return returnColor;
}
```

De aanroep wordt dan:

```
drawFrequency(getFrequencyColor(isTransmitting, openSquelch));
```

Vaak wordt een returnvalue gebruikt om aan te geven of een functie gelukt is:

```
bool setFrequency(){
  returnValue = false;
  set de frequentie;
  if (frequentie klopt) returnValue = true;
  return returnValue;
}
```

Waarom doen we dit allemaal zo vraag je je wellicht af. Er zijn verschillende redenen voor, allereerst het geheugen in een ESP32 of willekeurig welke microprocessor is niet oneindig. We dienen hier met zorg mee om te gaan. Door een variabele alleen in een functie te definiëren en gebruiken, bestaat deze variabele alleen in de functie. Idealiter zitten er in het geheugen alleen de variabelen die we nodig hebben. We kunnen het geheugen dus voor verschillende variabelen gebruiken. Ten tweede, de code wordt een stuk beter leesbaar en zoals aan het begin van dit verhaal al aangehaald, de functies kunnen zelfstandig en autonoom functioneren. Dit op zich hoeft geen doel te zijn, maar het maakt een functie wel veel beter herbruikbaar.

We hebben het continu over variabelen en natuurlijk weten we allemaal dat er verschillende soorten variabelen bestaan. Grofweg zijn er nummers, booleans en strings. Maar er valt een hoop meer over te vertellen.

Allereerst de numerieke variabelen, Arduino kent onder andere int, int8_t, uint8_t, int16_t, int32_t en float. Er zijn er nog meer. Maar wat betekent het:

int8_t is een integer (een geheel getal) van maximaal 8 bits. Met 8 bits kan je 255 waardes definiëren, dus kan een int8_t een getal tussen -127 en +127 bevatten. Je snapt het al, een int16_t bestaat uit 16 bits en kan dus een getal tussen -32767 en +32767 bevatten. Als je geen negatieve getallen nodig hebt kun je overwegen de uint8_t of uint16_t types gebruiken. De u staat voor unsigned. Een 8bits uint kan daarom een getal tussen 0 en 255 bevatten en een 16 bits tussen 0 en 65535. Arduino is heel eenvoudig in de afhandeling van integers

hetgeen best leuke bugs kan opleveren. Als je stelt uint8_t x = 256, dan vindt de compiler dat x=0, na 255 krijg je weer 0, 257 is dus 1, 258 is 2 en zo verder, tot 511, dat is 255 en 512 is weer 0!

Een int in de Arduino omgeving is per definitie een int16_t, dus 2 bytes groot. Als het nummer in de variabele nooit groter wordt dan 127, gebruikt de variabele dus 2 keer zoveel geheugen dan noodzakelijk, zonde, gebruik dan bij voorkeur een int8_t!

Als je geen negatieve waardes nodig hebt, gebruik dan een uint. Technisch maakt het meestal niet uit, maar het komt de leesbaarheid ten goede.

Een float bestaat uit 4 bytes en kan een waarde tussen -3.4028235E+38 en 3.4028235E+38 bevatten.

Een boolean kan alleen ja/nee of true/false of 1/0 bevatten en is dus maar 1 bit groot. Als je aan een boolean genoeg hebt, gebruik dan ook een boolean want dat is het meest economisch.

Elke integer kan ook als boolean worden gebruikt in de Arduino omgeving, maar realiseer je dat je tenminste 7 bits geheugen weggooit.

Een string bevat een tekst met een willekeurige lengte. Hier kom ik nog op terug.

We komen ook de byte en char tegen. Technisch zijn het, net als de int8_t en uint8_t beiden variabelen

We kunnen ook een eigen soort variabele maken. Dit is een ‘struct’. Een struct is een vorm van een variabele waarin we meerdere eigenschappen van die variabele vastleggen. In de transceiver maak ik hier veelvuldig gebruik van.
Een voorbeeld hiervan is de struct ‘Button’:

```
typedef struct // Buttons
{
   const char *name;     // Buttonname
   const char *caption;  // Buttoncaption
   char waarde[12];      // Buttontext
   uint16_t pageNo;
   uint16_t xPos;
   uint16_t yPos;
   uint16_t width;
   uint16_t height;
   uint16_t btnColor;
   uint16_t bckColor;
} Button;
```

Een variabele van het type ‘Button’ bestaat dus uit 10 variabelen (name, caption, waarde, pageNo etc..). Een ander voorbeeld is de struct ‘Sfreq’:

```
typedef struct  // Frequency parts
{
  int fMHZ;
  int fKHz;
} SFreq;
```

Zoals we hierboven hebben gezien, een functie kan maar een variabele teruggeven. Maar die variabele kan prima een struct zijn. Op deze manier kunnen we dus meerdere variabelen teruggeven vanuit een functie.

In het transceiver programma maak ik voor het rekenwerk aan de frequentie gebruik van kanalen (channels) in plaats van de frequentie. Channel 0 komt overeen met 144.000MHz, channel 160 is 146.000MHz, channel 22880 is 430.000MHz, 23680 is 440.000MHz. Inderdaad, een 12.5 KHz raster, tellend vanaf 144.000MHz.

In het programma heb ik een functie om het kanaal om te rekenen naar een frequentie, maar voor het tekenen van de frequentie is het makkelijk om de MHz’en en KHz’en als losse variabelen terug te krijgen. Dat doe ik met de volgende functie:

```
SFreq getFreq(int channel){
  int fMHz = floor(channel/80) + 144;
  int fKHz = (channel - (floor(channel/80)*80))*125;
  SFreq sFreq = {fMHz,fKHz};
  return sFreq;
}
```

Weer even tussendoor: Wat gebeurt er hier: er zitten 80 stappen van 12.5 KHz in een megaherz. Dus in fMHz stoppen we channel/80 zonder het deel achter de komma en tellen hier 144 bij op.

*In fKHZ stoppen we channel – (fMHz*80) en vermenigvuldigen dit met 125.*
Bijvoorbeeld channel 100: 100/80 = 1 en rest 20. fMHz wordt dus 1+144 = 145. fKHz wordt 100 – (1 80) = 20, 20 *125 = 2500, dus fKHz = 2500.*

fKHZ is eigenlijk in stappen van 0,1 kHz, dus 2500 betekent 250,0

Als we de transceiver willen ombouwen naar een 5KHz raster, hoeven we dus alleen hier een aanpassing te maken, Dat is nog eens efficiënt programmeren!

In deze functie komt alles samen, de ingaande parameter is het kanaal dat we willen omrekenen, we retourneren een zelf gedefinieerde variabele van het type SFreq. SFreq bestaat weer uit 2 variabelen fMHz en fKHz.

Een voorbeeld van de werking:

```
SFreq sFreq = getFreq(140);
Serial.printf("%01d.%04d",sFreq.fMHz,sFreq.fKHz);
```

Er wordt een variabele sFreq van het type (struct) SFreq gemaakt en gevuld met de waardes van channel ‘rxChannel’. In sFreq zijn 2 variabelen beschikbaar, sFreq.fMHz en sFreq.fKHz.
Dit resulteert in de tekst 145.7500.

Weer een stukje tussendoor: De functie printf is een heel handige om variabelen in een tekst te kunnen zetten. In bijvoorbeeld Serial.printf(“De frequentie is %01dMHz en %04dKHz, dit is in de %s band",sFreq.fMHz,sFreq.fKHz,”twee meter”);

Op de plaats ‘%01d’ komt de integer fMHz te staan, de 01 wil zeggen dat er minimaal 1 positie wordt gevuld en de d wil zeggen een getal. ‘%04d’ is een getal met minimaal 4 posities. 25 wordt zodoende 0025. Hier komt fKHz te staan. ‘%s’ wordt vervangen door de string ‘twee meter’.

Deze werkwijze kan ook worden gebruikt om een buffer te vullen:

Sprintf((buf, “De frequentie is %01dMHz en %04dKHz, dit is in de %s band", sFreq.fMHz, sFreq.fKHz,”twee meter”);
Serial.print(buf);

De buffer ‘buf’ wordt gevuld en op de tweede regel geprint.

Ik wil het hebben over array’s. Een array is een lijst van variabelen. Een array kan worden gemaakt van alle variabele types, dus ook van structs.

Voorbeelden van arrays:

```
int myInts[6];
int myInts[] = {2, 4, 8, 3, 6};
int myInts[5] = {2, 4, -8, 3, 2};
char message[6] = "hello";
SFreq sFreqs[4];
```

De eerste is een array van 6 int’s waarvan de waardes niet zijn vastgelegd.
De tweede en derde zijn allebei array’s met 5 vastgelegde int’s. De schrijfwijze van de derde is overdreven, technisch mag het wel maar het is verwarrend.
De vierde is een array van 6 chars (bytes die worden geïnterpreteerd als ASCII karakters).
De vijfde is een array van 4 SFreq’s, deze zijn alle vier nog leeg.
Een element van een array kan worden opgevraagd of gezet met zijn index: van de 2e en 3e array is myInts[0]=2, myInts[1]=4 etc. Elementen

in een array tellen dus vanaf 0, zijn zerobased.

```
int myInts[4] = {2, 4, -8, 3, 2};
int myInts[6] = {2, 4, -8, 3, 2};
```

De eerste regel gaat fout met de melding:
error: too many initializers for 'int [4]', een array met 5 waardes en er zijn maar 4 plekken gereserveerd
De tweede regel is spannend, de compiler vindt het goed, maar zijn er nu 5 of 6 plekken?

char message[6] = "hello"; heeft maar 5 letters maar toch is de array vol. Een array met characters (bytes die een teken vertegenwoordigen) wordt altijd afgesloten met een character null. Dit is geen 0, wat dat is ASCII 48. Kijk maar eens hier. Ik kom hier later nog op terug.

In het transceiver programma maak ik veel gebruik van array's. Een belangrijke is de array met buttons, dit is een array van structs van het type Button. Maar er is ook een array met repeaters en een array met CTCSS codes in gebruik.

In arrays wil je over het algemeen zoeken naar een element of door de verschillende elementen lopen. Een goed voorbeeld hiervan is de functie drawButtons(), deze tekent (onder voorwaarden) alle knoppen uit de array met buttons. Met een for loop kun je door alle items in de array met buttons lopen:

```
for (int i=0; i<aantal buttons; i++) {//etc..}
```

Het venijn zit in 'aantal buttons', dit moet je wel kunnen achterhalen. Nu is er een functie sizeof() beschikbaar. Het had leuk geweest als sizeof() het aantal elementen in het array had verteld, maar dat doet het niet. sizeof() vertelt de grootte van de variabele in bytes. Met sizeof(buttons) blijkt dus de grootte van de hele array buttons. Maar er is een leuke oplossing: sizeof(button[0]) geeft de grootte aan van de variabele button[0] in de array buttons. Dit mag natuurlijk ook button[1] of button[12] zijn, tenminste als die bestaan in de array.

Sizeof(buttons)/sizeof(button[0]) geeft dus het aantal elementen aan in de array buttons.
De for loop ziet er dus zo uit:

```
for (int i=0; i<sizeof(buttons)/sizeof(buttons[0]); i++) {//etc…}
```

Het zoeken in een array gebeurt niet anders, loop door de array totdat je het juiste item vindt:

```
Button findButtonByName(String name){
 for (int i=0; i<sizeof(buttons)/sizeof(buttons[0]); i++) {
                    if (String(buttons[i].name)==name) return
findButtonInfo(buttons[i]);
 }
 return buttons[0];
}
```

Spreekt voor zich hoop ik. Hierbij blijkt meteen hoe fijn het is dat een functie een struct als returnwaarde kan teruggeven. Met de aanroep:

```
Button mijnKnop = findButtonByName("Scan");
```

hebben we alle gegevens van de button "Scan" te pakken in de variabele mijnKnop gestopt.. Een echte functie om te zoeken in een array is er niet, maar kan je natuurlijk wel zelf maken.

Ik had het er net al over: char message[6] = "hello"; Dit is een array van characters (lettertekens). String message = "hello"; lijkt en is ook vrijwel hetzelfde. Maar niet helemaal. In de Arduino omgeving kan gebruik gemaakt worden van strings, maar C++ kan dit standaard niet. In C++ is een string altijd een array van characters afgesloten met een char(null). Het werken met char arrays is een beetje lastiger dan het werken met strings, maar ik wil er toch voor pleiten dat je probeert eraan te wennen. Daarvoor heb ik een goede reden, De compiler gaat jou proberen te helpen met het gebruik van strings, maar deze zijn voor de compiler niet ideaal. Vooral niet omdat de lengte niet is gedefinieerd en de char(null) ontbreekt waarmee heel mooi het einde van een string bepaald kan worden. Het voorbeeld is technisch niet helemaal correct maar geeft een beeld van wat er gebeurt in het geheugen van

een processor bij gebruik van een string. Stel drie variabelen:

```
Int i=5;
String s=”Oliebol”;
Char c=’X’.
```

In het geheugens staat nu 5OliebolX.
We veranderen nu de string in “Appelflap”. Dit past in het geheugen niet op de plek van “Oliebol”, hiervoor zijn 2 extra posities noodzakelijk. Het geheugen gaat er dan zo uitzien:5_____XAppelflap. Voor string s is een nieuw plekje gevonden en in het geheugen is een gat ontstaan. Het opvullen van dit gat met nieuwe variabelen is voor de processor lastig.
Hadden we dit gedaan:

```
Int i=5;
Char s[20]=”Oliebol”;
Char c=’X’.
```

Dan had het geheugen er zo uitgezien: 5Oliebol0_____X. En na het veranderen 5Appelflap0_____X. Het geheugen hoeft dan niet te groeien en er hoeven geen variabelen te worden verplaatst. Neem dit serieus als je wilt dat jouw programma 24/7 blijft doordraaien. Het manipuleren van strings leidt vrijwel altijd tot regelmatig (1 keer per week is ook regelmatig) crashen, omdat het geheugen vol zit door losse stukjes string.

Als er belangstelling voor blijkt wil ik nog wel eens verder uitweiden over deze materie.

Als laatste deze maand wil ik het hebben over de naamgeving van variabelen. Je doet jezelf een lol als je hiermee probeert consequent te zijn, dit komt de leesbaarheid van de code ten goede. Bezuinig niet op de lengte, dat kost echt geen geheugen.
De variabelen rf en tf zeggen niemand iets, rxFrequency en txFrequency zijn meteen duidelijk. De tijd die het extra typewerk kost, verdien je ruimschoots terug!

Om dezelfde reden, herkenbaarheid van de code, is de schrijfwijze ook belangrijk: we zien vaak camelCase, PascalCase en Snake_Case voorbij komen.

camelCase, dus beginnen met een kleine letter, alles aan elkaar en elk nieuw woord met een hoofdletter, gebruik je voor variabelen. Een variabele begint dus altijd met een kleine letter.

PascalCase, dus beginnen met een hoofdletter, alles aan elkaar en elk nieuw woord met een hoofdletter, gebruik je voor functies en structs. Tenminste zou het moet en zijn vind ik. In Arduino gaat dit al mis omdat de functies loop() en setup() beiden met een kleine letter beginnen.

CAPITALS, dus alles hoofdletters, gebruik je voor defines.

Snake_Case is uit de mode, teveel onzinnige underscores.

Soms kom je nog wel eens de Hungarian notation tegen, daarbij geeft de eerste letter of letters aan wat voor soort variabele het betreft, dus strName voor string, iName voor integer, bName voor boolean, fName voor functie. Een beetje achterhaald, maar soms wel lekker duidelijk.

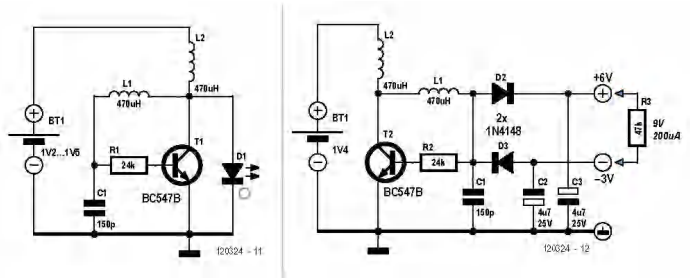
Ik weet het, ik ben absoluut te betrappen op fouten wat betreft naamgeving, maar ik doe mijn best.

Volgende maand gaan we wat dieper in op het programma zelf. Dan wordt het vast interessanter.

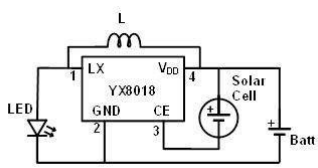
PA3CNO's Blog

Afgelopen maand hoorde ik tijdens het vaste avondrondje Henny PA3HK bezig met de reparatie van een oude versterker waarvan de transistoren moeilijk te verkrijgen waren. Ik heb dat probleem zelf ook wel eens gehad: een apparaat dat onderdelen bevat die defect zijn gegaan maar die bij de reguliere onderdelenleveranciers op internet niet meer te krijgen zijn. Iedereen kent natuurlijk Conrad, maar die is zijn assortiment aan het beperken, richt zich steeds meer op SMD componenten en aan HF doen ze sowieso niet veel. In Duitsland hebben we Reichelt, die vaak wat goedkoper is en ook meer spullen heeft voor de experimenterende HF amateur. Ga je meer de professionele kant op, dan is daar Mouser die heel veel heeft, maar een stuk duurder is en ook nogal wat verzendkosten rekent. Wat minder bekend is Farnell, die nog meer op de professionele markt gericht is. Maar een bedrijf wat bijna niemand kent, is Donberg.ie. Deze site levert de meest onmogelijke onderdelen. Als ze het niet hebben, is het nooit gemaakt. Ze zijn stervensduur, maar als je geen keus hebt is het wel de enige mogelijkheid om aan historische onderdelen te komen. Misschien een tip. Zet 'm in je bookmarks voor het geval je een keer tegen kapotte onderdelen aanloopt die je nergens meer kunt vinden.

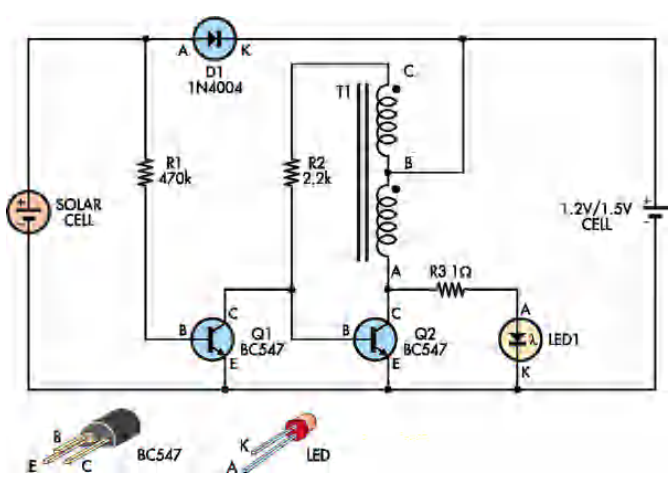
In de Elektuur nieuwsbrief stond een stukje over een één-transistor spanningsomvormer zoals die toegepast wordt in van die tuinverlichting met zonnepaneeltje. Daar zit vaak maar één AA oplaadbare batterij in en die zijn nominaal 1,2V: een spanning die te laag is om zelfs maar een rode LED (die de laagste doorlaat spanning heeft) op te laten branden. Zo'n omvormertje zorgt er dan voor dat de LED blijft branden tot de batterijspanning tot zo'n 0,8V gedaald is. De auteur van het artikel beschrijft hoe hij de schakeling ombouwde zodat er 9V bij 200uA uit te halen is. Geen idee wat je daar mee moet, maar misschien heb je er een toepassing voor.



Wat me intrigeerde, is hoe die zonnecel dan zodanig aangesloten wordt dat de batterij geladen wordt en het geheel automatisch inschakelt als het donker wordt. Enig speurwerk op internet liet zien dat in de meeste gevallen gewoon een chip toegepast wordt. Dat heb ik ook gezien bij de reparatie van mijn tuinlampjes, want hoe goed ze die dingen ook zeggen te maken, uiteindelijk lopen ze vol water, rotten de LED's van de print en geven de historische NiCd batterijen het op. Uiteindelijk vond ik een schema met transistoren, maar niet met één transistor, zie de schakeling hieronder.



Als de zonnecel spanning levert, is Q1 geopend en die neemt de basisspanning van Q2 weg. De accu wordt dan geladen via D1. Maar dan brandt de LED toch via T1? Nee, want daar is de accuspanning te laag voor zoals ik al schreef. Wordt het donker, dan gaat Q1 dicht, Q2 kan daardoor oscilleren en er wordt voldoende spanning opgewekt om de LED te laten branden. Ingenieus niet?



Heb je recent nog op aprs.fi gekeken? Is je wat opgevallen? De site aprs.fi is afgestapt van het gebruik van Google Maps. Al in 2018 sloeg de sitebeheerder alarm omdat met het gehanteerde billing beleid de rekening voor het gebruik op zou kunnen lopen tot €4000-5000 per maand. En nu is de bom kennelijk gebarsten: aprs.fi is overgestapt op het gebruik van OpenStreetMap. Sommige dingen zullen dus anders zijn, maar het belangrijkste werkt nog: kunnen zien waar stations zich bevinden. Mooi toch?

Afdelingsnieuws

Verslagen zijn we door het bericht dat op 24 januari j.l. Anneke van Strien, PD4EJP, XYL van Paul PA3DFR, overleden is. We wisten dat Anneke al een tijd ziek was, maar haar overlijden kwam toch nog onverwacht snel. We wensen Paul en zijn familie veel sterkte toe met het verwerken van dit verlies.

Op woensdag 11 januari hadden we dan onze eerste bijeenkomst van het nieuwe jaar en ook de eerste die in ons nieuwe onderkomen gehouden werd: buurthuis 't Span aan de Sullivanlijn 31 in Zoetermeer. De ruimte is een stuk luxer dan ons oude onderkomen van de Minigolfclub: er is een bar waar we niet zelf meer achter hoeven te staan en die voldoende aanbod heeft voor onze wensen. We zijn er dus blij mee, zie de foto rechtsonder.

Voor de maand februari zijn de clubavonden op de woensdagen 8 en 22, waarbij op de 8e de QSL-manager weer aanwezig zal zijn voor het uitwisselen van de kaarten. Vanaf 20:00 kan iedereen dan weer bij ons terecht op de nieuwe locatie: Sullivanlijn 31, 2728BR Zoetermeer. Het buurthuis ligt enigszins van de weg, aan het Aldo van Eyckpark. Voor nieuwkomers is het soms even zoeken, maar als je langs de (momenteel in verbouwing zijnde) snackbar loopt, dan ligt het aan je linkerhand.

Nog even over het weerstation project: dat is nog in ontwikkeling. Het blijkt dat de problemen waar nabouwers tegenaan lopen dusdanig groot zijn, dat we het over een andere boeg gaan gooien. Hardwarematig is het allemaal heel eenvoudig, maar de software er op de juiste manier inkrijgen is een hersenkraker. Daarom is onze softwaredokter nu bezig om het weerstation te voorzien van een webpagina waarop je de configuratie kunt aanpassen. Als dat werkt, kunnen we de ESP32's geprogrammeerd uitleveren en de problemen beperken. Ook de functionaliteit is uitgebreid: je ziet nu ook de bandcondities en de MUF zoals deze door de ionosonde in Dourbes gerapporteerd wordt.