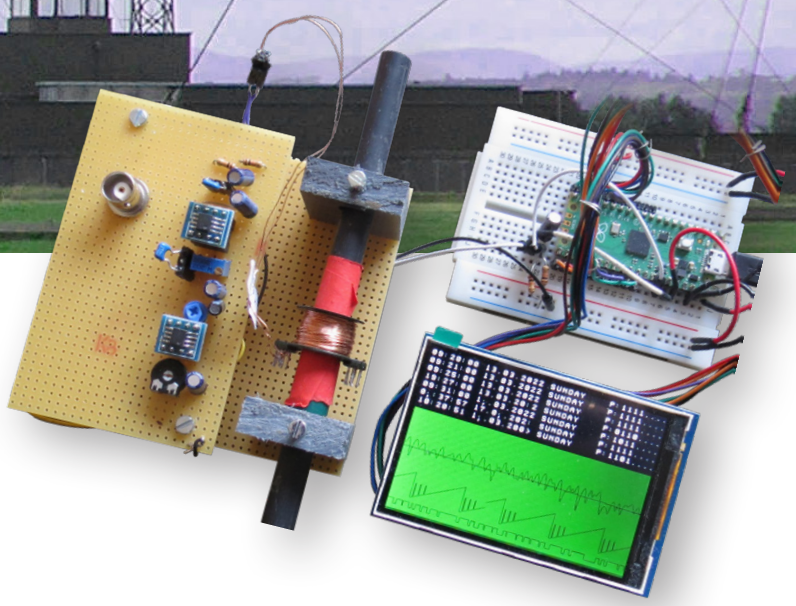
Een MSF-SDR met de Raspberry Pi Pico

decodeer het tijdsignaal met een Pi Pico-SDR

VLF-antenne in Anthorn (Dougsim, https://bit.ly/34HXeuG).



Martin Ossmann (Duitsland)

MSF is het Britse equivalent van de Duitse DCF77-tijdseinzender. Dit SDR-project laat zien hoe een ontvanger en decoder voor deze (en andere) tijdsignalen vrij eenvoudig en vooral goedkoop kunnen worden geïmplementeerd. Voor de hardware is niet veel meer nodig dan een goedkope Raspberry Pi Pico om de MSF-tijdsein-informatie te ontvangen, te decoderen en weer te geven.

In Duitsland zendt de DCF77-zender in Mainflingen op de lange golf een gecodeerd tijdsignaal uit. Het equivalent in het Verenigd Koninkrijk is het MSF-signaal dat vroeger bekend stond als "The Rugby Clock". Deze zendt tijdsignalen uit [1] onder gebruikmaking van een 60kHz-draaggolf. In de begindagen diende hij als frequentiestandaard en zond tweemaal per dag een pulstrein van vijf minuten uit. Het "transmissieprotocol" van het signaal is in de loop van de decennia herhaaldelijk gewijzigd, maar pas in 1977 omvatte het signaal ook informatie over de tijd en de datum, die door de ontvanger kon worden gedecodeerd en gebruikt.

Het project

In de loop der jaren zijn er in diverse Elektor-artikelen meerdere ontvanger/decoder-schakelingen beschreven die het DCF77-signaal gebruiken, maar dit is waarschijnlijk de eerste keer dat er een ontwerp voor een MSF-ontvanger verschijnt. Wel is er een uitbreiding [2] voor de good-old 6502 Junior Computer [3] beschreven in de Engelstalige editie van Elektor.

Sindsdien heeft de ontwikkeling niet stilgestaan en de technologie met betrekking tot ontvangers/decoders heeft enorme vooruitgang geboekt. In dit artikel zullen we de laatste *state of the art*-concepten gebruiken om een *software defined radio* (SDR) te bouwen met behulp van een klein microcontroller-board. Het Raspberry Pi Pico-board, dat gebruik maakt van een RP2040-CPU die geklokt wordt op 125 MHz (deze eigen controller van de Raspberry Pi Foundation is uitgerust met twee 32-bit ARM Cortex M0+ cores), is een geschikt (je zou haast zeggen 'voorbestemd') stukje hardware voor deze toepassing. Zijn A/D-omzetter haalt 500 ksps. En al deze 'power' kan worden gekocht voor een bijna belachelijke € 5 (zie **Gerelateerde producten**).

Hier laten we zien hoe een complete ontvanger in hard- en software kan worden geïmplementeerd voor het MSF-tijdsignaal op 60 kHz. De volledige ontvanger, zonder display maar met een RS232-uitgang en antenneaansluiting, is afgebeeld in **figuur 1**.

De hardware

Eerst zullen we de hardware bekijken die nodig is om de SDR te bouwen. Er zijn slechts een paar onderdelen nodig die op het Pico-board moeten worden aangesloten.

Antenne-ingang: we gebruiken de analoge ingang ADC2 (GPIO28, op het Pico-board pin 34) voor het antennesignaal. De ADC gebruikt de interne 3,3 V als referentiespanning. Deze pin moet daarom op de helft van de referentiespanning worden ingesteld. De twee weerstanden in **figuur 2** zorgen hiervoor. De 10µF-condensator C1 zorgt voor AC-koppeling van het binnenkomende signaal.

RS232 uitgang: in zijn eenvoudigste vorm (zonder LC-display) gebruikt de ontvanger een seriële interface (115.200 bit/s) om de gegevens uit te voeren. De interface wordt geïmplementeerd door de in **figuur 3** getekende schakeling. We kunnen de USB-poort niet gebruiken om de seriële gegevens uit te voeren, omdat dit op een onvoorspelbare manier interrupts bij de uitvoering van onze software zou veroorzaken.

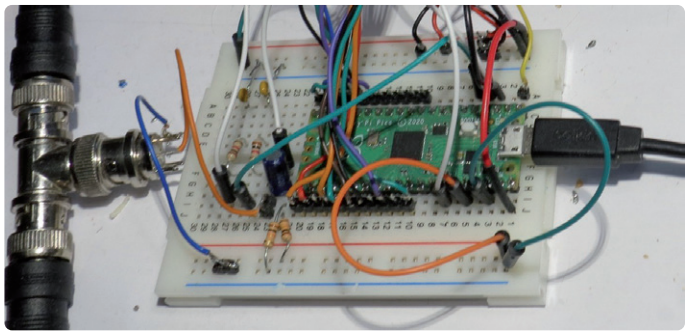
PWM DAC's: wanneer geen LC-display is aangesloten, is het mogelijk de DAC's te gebruiken met pulsbreedte-gemoduleerde (PWM) signalen om een gemakkelijk hulpmiddel bij het debuggen te creëren. We hebben twee PWM-DAC's opgezet met de bijbehorende laagdoorlaatfilters (**figuur 4**).

Door GPIO 2 en GPIO 3 als PWM-uitgangen te gebruiken, kunnen bijvoorbeeld het gedemoduleerde signaal en de Bit-Timer signalen op een 'scoop worden weergegeven (**figuur 5**).

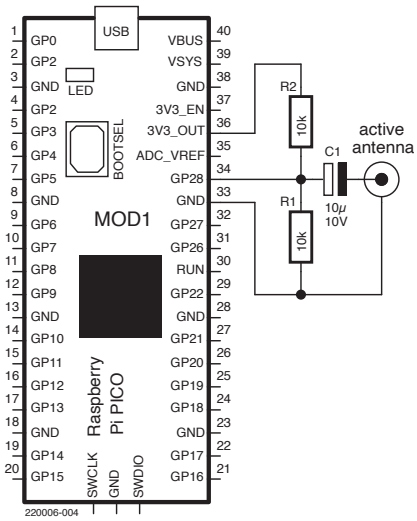
LCD: de *3,5-inch Arduino 8-bit module ILI9486* (versie SKU MAR3502 zonder touchscreen-functionaliteit [4]) kan worden gebruikt als display. Dit 3,5-inch Arduino-shield heeft 480x320 kleurenpixels en wordt verkocht voor ongeveer € 10. **Figuur 6** laat zien hoe het op de Raspberry Pi Pico wordt aangesloten.

Het ontvangen signaal wordt op het LC-display getoond samen met een golfvorm die informatie over de bit-timing levert. De ontvangen tijdinformatie wordt in leesbare vorm boven de golfvorm weergegeven (**figuur 7**). Als u geen behoefte hebt aan deze informatie, kunt u het LCD weglaten zonder dat u de software hoeft aan te passen.

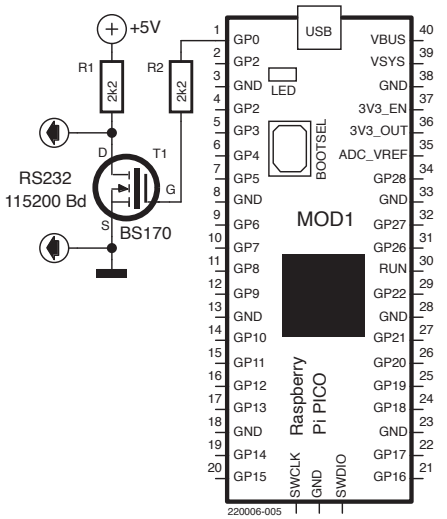
Actieve antenne: de aansluiting van de antenne is hierboven al besproken; het schema van de actieve antenne is te zien in **figuur 8**. Het hart ervan wordt gevormd door een dubbele opamp van het type LM6132. Deze opamp is bijzonder geschikt voor deze toepassing, met een voedingsspanning van 2,7...24 V, een versterking/bandbreedte-product van 10 MHz, rail-to-rail ingangs- en uitgangsspanningen



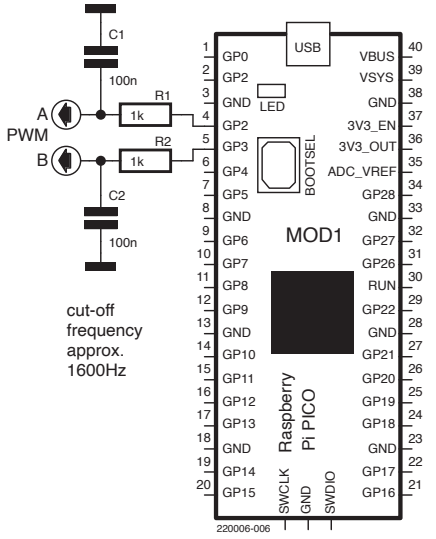
Figuur 1. Het Raspberry Pi Pico-board als software defined radio voor de ontvangst van het MSF-signaal.



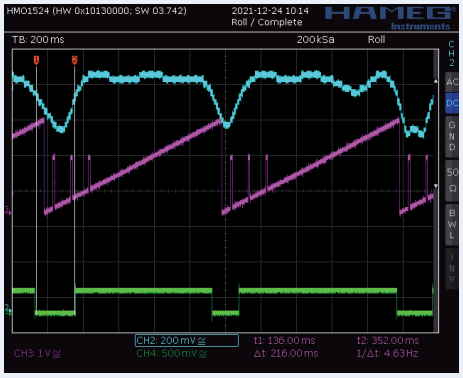
Figuur 2. Deze componenten moeten op de A/D-signaalingang worden aangesloten.



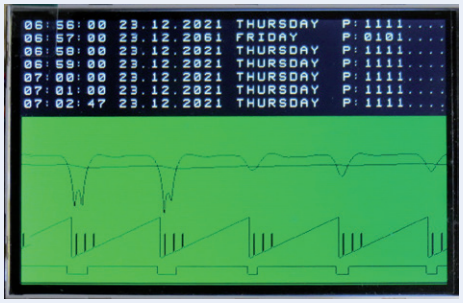
Figuur 3. De RS232-uitvoer van het Pico-board.



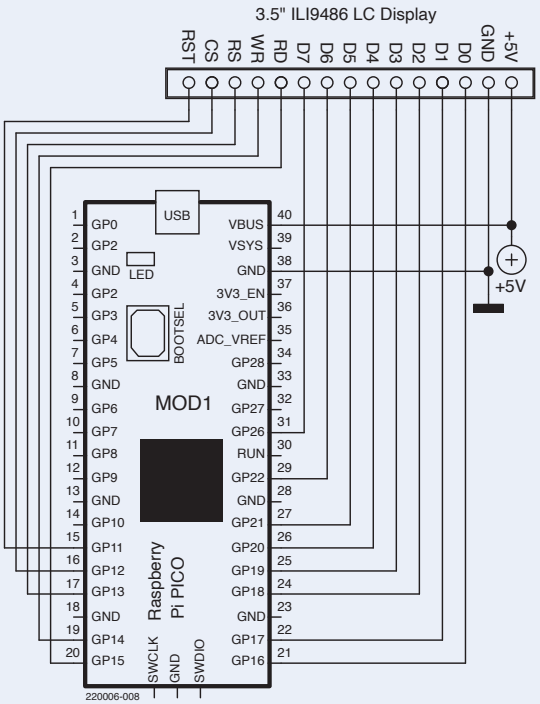
Figuur 4. Twee laagdoorlaatfilters voor de PWM-DAC debug-signalen.



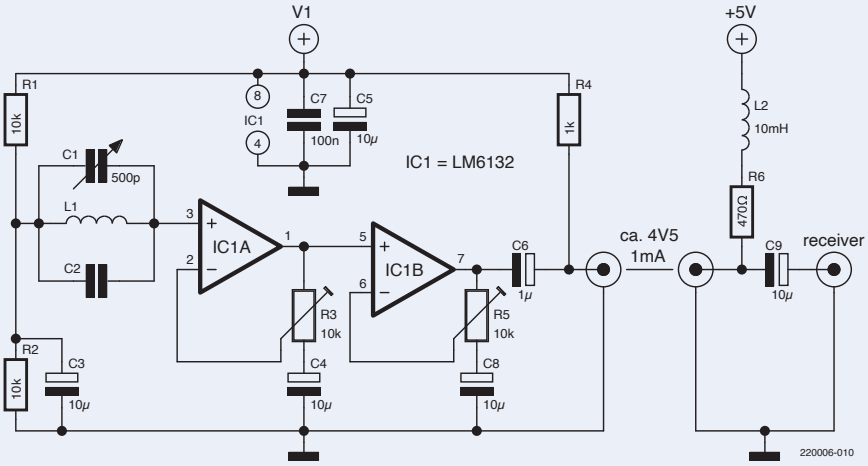
Figuur 5. De PWM-testsignalen: het bovenste spoor is de ampl-waarde, het middelste spoor is de SecondTimer die de Sampling-triggerpulsen toont en het onderste is het digitale sigValue-signaal.



Figuur 7. Ontvanger-informatie op een LC-display.

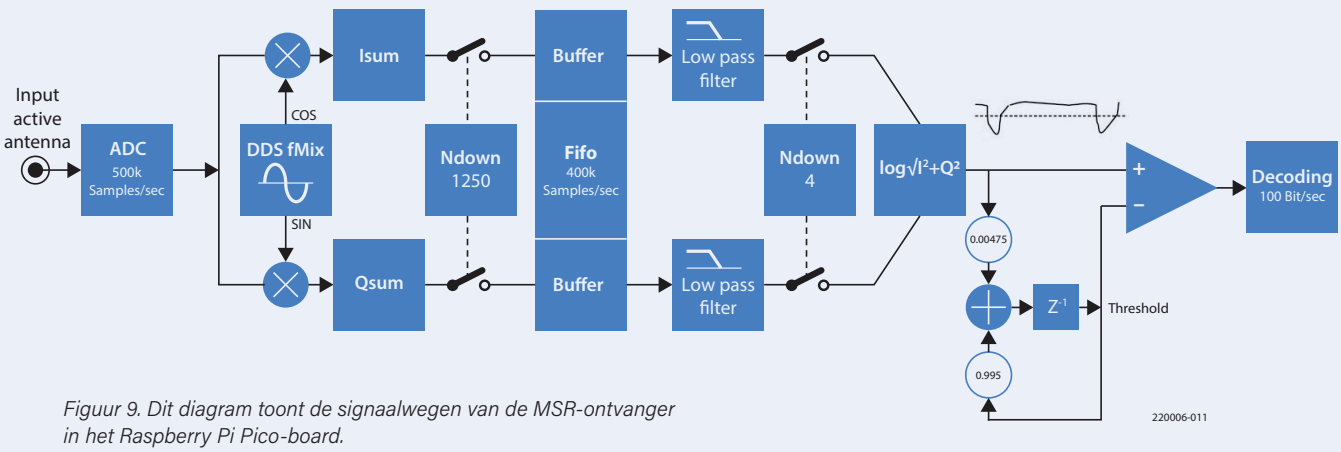


Figuur 6. Aansluiting van het 3,5" LC-display.



L1: 500 turns enamelled copper wire 0.2 mm on 10 mm ferrite rod (length 180 mm)
C2: as required

Figuur 8. Actieve antenne voor het 60-kHz MSF-signaal.



Figuur 9. Dit diagram toont de signaalwegen van de MSR-ontvanger in het Raspberry Pi Pico-board.

en een gering stroomverbruik van 360 µA per opamp. Ongetwijfeld zijn andere opamps hier ook bruikbaar, maar als u van plan bent een ander type te gebruiken, controleer dan eerst zorgvuldig of deze aan de specificaties voldoet.

Programmeren van de ingangsmixer

Na de hardware komen we bij de programmering. De analoge signaalwegen van de SDR hebben een structuur zoals geschetst in **figuur 9**. De Raspberry Pi Pico kan in verschillende talen worden geprogrammeerd. Voor deze toepassing hebben we gekozen voor C met behulp van de *Microsoft Visual Studio Code* ontwikkelomgeving, draaiend op een PC onder Windows 10. Laten we eens kijken hoe de verschillende onderdelen functioneren.

De ADC-bemonsteringsroutine wordt getriggerd door de PWM en 500.000 keer per seconde per interrupt aangeroepen. De offset `ADCoffset` = 2048 wordt afgetrokken van de ADC-waarde en het resultaat wordt vervolgens vermenigvuldigd met `ADCscale` = 10 (**listing 1**). De fase van de plaatselijke oscillator (LO-DDS) wordt opgehaald en de ingangswaarde wordt vermenigvuldigd met de cosinus (infase- of I-signaal) en de sinus (kwadratuur- of Q-signaal). De producten worden gesommeerd over 1250 samples (in `Isum` en `Qsum`). De waarden worden dan (in **listing 2**) doorgegeven aan een FIFO voor verdere verwerking, die dan plaatsvindt met 500.000/1250 = 400 samples/s. Deze bemonsteringsfrequentie is zo laag dat alle verdere operaties kunnen worden uitgevoerd met variabelen van het type double.

De waarden worden uit de FIFO uitgelezen en door een vierde-orde Butterworth-laagdoorlaatfilter met een afsnijfrequentie van 3 Hz geleid. Tijdens de ontwikkeling bleek dat deze lage afsnijfrequentie noodzakelijk was omdat de antenne van de auteur sterke stoorsignalen direct naast het gewenste signaal oppikte. Het filter wordt gevolgd door nog een downsampling-routine, dit keer met een factor 4, zodat uiteindelijk 100 samples/s worden verwerkt.

De `msfSample()`-routine in **listing 3** berekent dan de draaggolfamplitude `ampl` uit de I/Q-componenten. Dan wordt de logaritme van `ampl` berekend en op zijn beurt opgeslagen in `ampl`; dit maakt het gemakkelijker om de bits te decoderen.

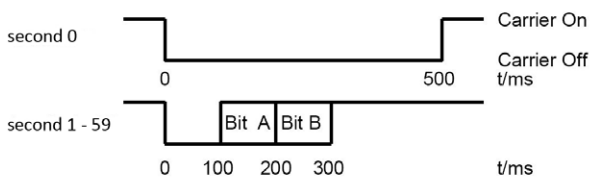
De schakeldrempel `threshold` wordt afgeleid van `ampl` via een eerste-orde recursieve filterberekening. Het signaal `ampl` wordt vervolgens vergeleken met `threshold` om de digitale ontvangstwaarde `sigValue` te bepalen.

Nu we de analoge signaalverwerking behandeld hebben, kunnen we bekijken hoe de gegevens worden teruggewonnen uit het ontvangen signaal en hoe dit correspondeert met de tijdinformatie.

De bits lezen

De MSF-zender zendt elke seconde HF-draaggolfpulsen uit zoals aangegeven in **figuur 10**. Op seconde 0 van elke minuut wordt de draaggolf gedurende 500 ms uitgeschakeld. De SDR gebruikt deze puls voor de synchronisatie. De pulsen die bij elk van de volgende 59 seconden worden uitgezonden, bevatten twee bits aan informatie: A en B. Aan het begin van elk van deze seconden is de draaggolf gedurende 100 ms uitgeschakeld (corresponderend met 10 samples in onze applicatie). Als bit A = 1, dan blijft de draaggolf 100 ms extra uitgeschakeld, en als bit B = 1, blijft de draaggolf nog eens 100 ms uit.

In `SecondTimer` loopt een timer, gesynchroniseerd met elke seconde, van 0 tot 99. De softwaredecodering werkt als volgt: in `Duration` wordt de lengte van de momentele puls gemeten. Als gedurende 0,5 s geen draaggolf wordt gedetecteerd, wordt SecondTimer ingesteld op de waarde 50 – 2 = 48, zodat deze timer nu synchroon met de seconde



Figuur 10. De MSF-secondenpulsen met bit A- en bit B-codering.

Jaar (BCD-gecodeerd 0...99)								betekenis
80	40	20	10	8	4	2	1	BCD-gewicht
17A	18A	19A	20A	21A	22A	23A	24A	bit

Maand (BCD-gecodeerd 1...12)					betekenis
10	8	4	2	1	BCD-gewicht
25A	26A	27A	28A	19A	bit

Dag van de maand (BCD-gecodeerd 1...31)						betekenis
20	10	8	4	2	1	BCD-gewicht
30A	31A	32A	33A	34A	35A	bit

Dag van de week (BCD-gecodeerd 0...6)			betekenis
4	2	1	BCD-gewicht
36A	37A	38A	bit

Uur (BCD-gecodeerd 0...23)						betekenis
20	10	8	4	2	1	BCD-gewicht
39A	40A	41A	42A	43A	44A	bit

Minuut (BCD-gecodeerd 0...59)							betekenis
40	20	10	8	4	2	1	BCD-gewicht
45A	46A	47A	48A	49A	50A	51A	bit

Minuut-marker								betekenis
52A	53A	54A	55A	56A	57A	58A	59A	bit
0	1	1	1	1	1	1	0	waarde

Parity bits
Bit 54B met bit 17A to 24A resulteert in oneven aantal bits
Bit 55B met bit 25A to 35A resulteert in oneven aantal bits
Bit 56B met bit 36A to 38A resulteert in oneven aantal bits
Bit 57B met bit 39A to 51A resulteert in oneven aantal bits

Figuur 11. Overzicht MSF-tijdcodering.

Listing 1. A/D-bemonstering en werking van de lokale oscillator. Sommeren en downsampling met een factor 1250 tot 400 samples/s.

```
int16_t adcv=(uint16_t) adc_hw→result;              //get ADC result
hw_set_bits(&adc_hw->cs, ADC_CS_START_ONCE_BITS);  //start ADC again
pwm_clear_irq(pwm_gpio_to_slice_num(PWM_PIN1));    //interrrupt flag
DDSp += DDSd ;                                      //increment LO-DDS phase
inputVal=ADCscale*(adcv-ADCoffset) ;               //offset and scaling
Isum += LOcosTab[DDSp>>24]*inputVal ;              //I-multiplication
Qsum += LOsinTab[DDSp>>24]*inputVal ;              //Q-multiplication
SampleTime++ ;                                     //refresh this step
if(sampleTime>=1250){                              //downsampling
      FIFO...}                                     //further steps
```

Listing 2. Filtering van de I- en Q-waarden en downsampling met een factor 4 tot 100 samples/s.

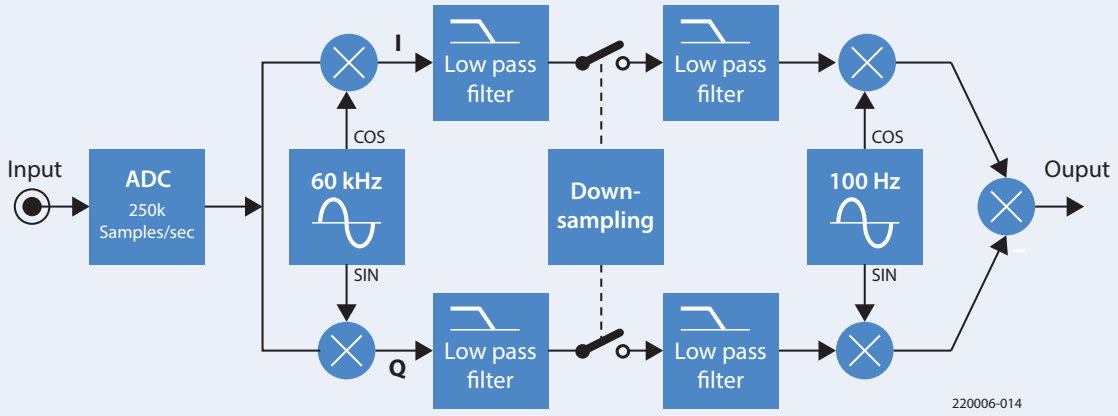
```
Isample=IntFifoI[IntFifoOutPtr] ;                  //get I-signal from FIFO
Qsample=IntFifoQ[IntFifoOutPtr] ;                  //get Q-signal from FIFO
IntFifoOutPtr=(IntFifoOutPtr+1) & IntFifoMask ;    //increment FIFO pointer
IfilOut = tprun(IIfil,Isample) ;                   //lowpass filter I-signal
QfilOut = tprun(QQfil,Qsample) ;                   //lowpass filter Q-signal
kdown++ ;
      if(kdown>=4){                                //downsampling factor 4
            msfSample(IfilOut,QfilOut) ;
            kdown=0 ;
            }
```

Listing 3. Berekening van de amplitude, de schakeldrempel en de pulsduur om de bits terug te winnen.

```
void msfSample(double ii, double qq){
      ampl=sqrt(ii*ii+qq*qq) ;                     //get carrier amplitude
      ampl=40*log(ampl+1) ;                        //log is better!
      threshold=0.995*threshold+0.005*ampl*0.95 ;  //recursive mean as threshold
      if(ampl>threshold){                          //comparator function
            sigValue=1 ;                           //digital value = 1
            }
      else {
            sigValue=0 ;                           //digital value = 0
            }
      doScope(ampl/2.0+20,threshold/2+20,sigValue*10+10, DAC/2.0+30) ;
      if(sigValue==lastSigValue){
            Duration++ ;                           //pulse goes on
            }
      else {                                       //pulse end reached
            tt=pulseForm(lastSigValue,Duration) ;  //get pulse character
            if(tt=='z'){
                  printf("sync on z") ;            //signalize sync
                  SecondTimer=50-2 ;               //sync SecondTimer
                  }
            printf("%c",tt) ;                      //display pulse character
            Duration=0 ;                           //new pulse length starts
            lastSigValue=sigValue ;                //update lastSigValue
            }
```

Listing 4. Sampling van bits A en B, getriggerd door SecondTimer.

```
IncSecondTimer() ;                        //SecondTimer runs from 0 to 99
DAC=SecondTimer ;                         //scope sawtooth signal
if (SecondTimer==5+0) { DAC=60 ; }        //scope signal pulse
if (SecondTimer==15+0) {                  //bit A sample time
   DAC=60 ;                               //scope signal pulse
   putMSFbit(Second,0) ;                  //clear bit store
   if (sigValue==0) {                     //if carrier switched off
      addMSFbit(Second,1) ;               //Bit A set true
      } ;
   }
   if (SecondTimer==25+0) {               //bit B sample time
      DAC=60 ;                            //scope signal pulse
      if (sigValue==0) {                  //if carrier switched off
         addMSFbit(Second,2) ;            //Bit B set true
         } ;
      }
```



Figuur 12. Omhoog converteren naar een IF van 100 Hz.

loopt (**listing 4**). Tegelijkertijd wordt de minuut gesynchroniseerd door de waarde van de huidige seconde op 0 te zetten in `doMinuteSync()`. Met behulp van `SecondTimer` wordt het ontvangen signaal bemonsterd in het midden van bit A en bit B (`SecondTimer==15` en `SecondTimer==25`) om te bepalen welke waarde deze bits hebben. De ontvangen digitale waarden voeren we eenvoudig uit via GPIO-Pin 4 (pin 6 van de Pico):

```
gpio_put(GPIO4, sigValue); // Output sigValue at
                           // pico GPIO4=pin 6
```

De waarde van `SecondTimer` wordt later ook voor debugging-doeleinden via PWM naar buiten gevoerd, net als de waarden van `ampl` en `DAC`. Dit wordt gedaan met de volgende twee statements:

```
pwm_set_gpio_level(PWM_PIN1, ampl/5.0 ); // Output
                                         // amplitude
pwm_set_gpio_level(PWM_PIN2, DAC );      // Output timing
```

Decoderen van de tijdinformatie

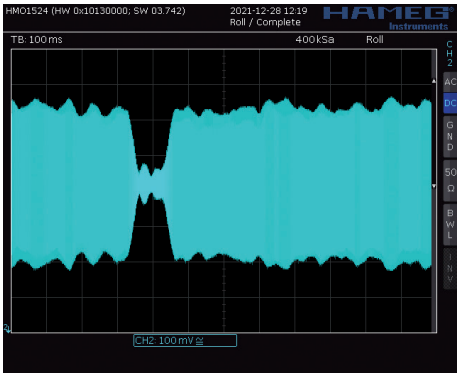
Telkens wanneer SecondTimer gesynchroniseerd is met de 0,5 s tijdens seconde nul, is er een minuut voorbij en kunnen we de laatst ontvangen tijdinformatie evalueren. De ontvangen databits staan in de waarden `MSFbits[0 ... 59]`. Bij de zender is de tijdinformatie conform **figuur 11** in deze bits gecodeerd.
De tijd- en datuminformatie wordt dan eenvoudigweg gereconstrueerd zoals in **listing 5** zodat we de uren en minuten krijgen.
We geven dezelfde informatie die we via de seriële interface versturen ook tekstueel op het LC-display weer. Dit doen we met behulp van de instructies in **listing 6**.
Een pariteitscontrole van de ontvangen informatie wordt uitgevoerd conform **listing 7**. De gecontroleerde bits zijn de A-bits van de verzonden informatie. De vier controlebits zijn de B-bits van de corresponderende secondenpuls. Er worden vier pariteitscontroles uitgevoerd, waarbij de integriteit van maximaal 12 bits wordt beschermd door één pariteitsbit.

Debug-signalen

Bij klassieke superhet-ontvangerontwerpen wordt het inkomende HF-signaal gemengd met een lokaal oscillatorsignaal met variabele frequentie om een lagere middenfrequentie (IF) te produceren. Het IF-signaal wordt vervolgens gefilterd met een relatief smalbandig filter.
Het IF-signaal van de MSF60 ontvanger kan ook worden bekeken met een oscilloscoop. Onze ontvanger mengt het ingangssignaal omlaag tot IF = 0 Hz. Als je een IF-AC-signaal wilt zien, kun je het 0-Hz IF signaal omhoog converteren naar een AC-IF. Het blokschema van de betreffende schakeling is getekend in **figuur 12**.
De software voor deze conversie is te zien in **listing 8**.
We gebruiken een PWM-uitgang als DAC. Het AM-gemoduleerde 100-Hz IF-signaal is te zien in **figuur 13**.
Hiermee zijn ontwerp en bouw van de MSF-ontvanger afgewerkt. Slechts één processor-core wordt in deze toepassing gebruikt, zodat er nog voldoende rekenkracht overblijft voor uitbreiding. De bit-decodering zou bijvoorbeeld foutresistenter kunnen worden gemaakt. Op vrijwel dezelfde wijze kan een DCF77-ontvanger worden gerealiseerd, alleen de decoderingsoftware moet dan worden aangepast. De ontvangst van het MSF-signaal is in Aken (Duitsland) veel lastiger dan van het DCF77-signaal (met een gelijkwaardige SDR). Dit resulteert vaak in pariteitsbits die fouten in de ontvangen informatie signaleren; er worden echter vaak genoeg foutloze berichten ontvangen om nauwkeurige informatie over de tijd van de dag betrouwbaar weer te geven.

Werken met een RP2040

We hebben gezien dat het Raspberry Pi Pico-board met heel weinig extra hardware kan worden omgetoverd tot een complete MSF-SDR. Het meest complexe element hierbij is de actieve antenne. Gezien de



Figuur 13. 100-Hz amplitudegemoduleerd IF-signaal.

Listing 5. Decodering en seriële BCD-uitvoer van uren en minuten.

```
void OutBCD2(int v){                          //issue 2 BCD digits
   uartPutc('0'+(v>>4)) ;                     //via serial interface
   uartPutc('0'+(v & 0xf)) ;
   }
int GetBCDbits(int StartPos , int Length){     //fetch length bits from MSFbits
   int v,k ;                                  //start at StartPos
   V=0 ;                                      //BCD coding
   for (k=0 ; k<Length ; k++ ){
      v=(v<<1) + ( MSFbits[StartPos++] & 1) ;
      }
   return v ;
   }
hours =GetBCDbits(39,6) ; OutBCD2(hours ) ;     //hours = bits 39 to 44
uartPutc(':') ;
minutes=GetBCDbits(45,7) ; OutBCD2(minutes) ;   //minutes = bits 45 to 51
UartBlank()
```

Listing 6. Uren en minuten op het LC-display.

```
LcdPutc(CRcode) ;                 //output carriage return
LcdPutc(LFcode) ;
LcdPutc('0'+((hours>>4)&0xF)) ;    //MS digit of hours
LcdPutc('0'+(( hours)&0xF)) ;      //LS digit of hours
LcdPutc(':') ;                    //separator
LcdPutc('0'+((minutes>>4)&0xF)) ;  //MS digit of minutes
LcdPutc('0'+(( minutes)&0xF)) ;    //LS digit of minutes
LcdPutc(':') ;                    //separator
```

Listing 7. Pariteitscontrole.

```
LcdPutc(CRcode) ;                          //output carriage return
int parity(int from , int to) {            //parity over A bits
int parity ;
int k ;
parity=0 ;
   for (k=from ; k<=to ; k++){
      parity ^= MSFbits[k] ;               //XOR with bits
      }
Parity &= 1 ;                              //select A bit
return parity ;
}
void parityCheck(int from , int to , int checkPosition) {
   int p ;
   p=parity(from,to) ;
   if ( (MSFbits[checkPosition] & 2)>0) {  //B bit is parity
      P ^= 1 ;                            //XOR parity bit
      }
   uartPutc(' ') ;
   uartPutc('P') ;
   uartPutc('=') ;
   uartPutc('0'+p) ;                       //output parity bit
   }
   ParityCheck(17,24,54) ;                 //four parity checks
   parityCheck(25,35,55) ;
   parityCheck(36,38,56) ;
   ParityCheck(39,51,57) ;
```

Listing 8. De code voor de softwarematige omhoog-conversie.

```
debugDDSp += debugDDSd ;              //100Hz phase update
v=IfilOut*cosTab[debugDDSp>>24]       //add I signal
   +QfilOut*sinTab[debugDDSp>>24] ;   //add Q signal
v=62+v/1024 ;                         //offset and scaling
pwm_set_gpio_level(PWM_PIN2,v );      //PWM output
```

rekenkracht en lage kosten van het board laat deze toepassing zien wat een hobbyist met weinig middelen tegenwoordig kan bereiken. Je leest vaak hoe gemakkelijk het Pico-board in Python geprogrammeerd kan worden, maar in deze toepassing is de de 500 k sample-rate van het ingangssignaal dan een struikelblok. In C kan de microcontroller-hardware echter directer worden aangesproken en efficiënter worden geprogrammeerd. We zien hier dat zelfs zonder een floating point-unit (FPU) de RP2040 nog steeds meer dan geschikt is voor het implementeren van een digitaal laagdoorlaatfilter.

220006-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via ossmann@fh-aachen.de of naar de redactie van Elektor via redactie@elektor.com.

GERELATEERDE PRODUCTEN

- Elektor Raspberry Pi RTL-SDR Kit (Book and Components) (SKU 19518)
  www.elektor.nl/19518
- Elektor SDR Hands-on Kit (Book and SDR-Shield with ferrite toroid and cable) (SKU 19041)
  www.elektor.nl/19041
- Raspberry Pi Pico RP2040 (SKU 19562)
  www.elektor.nl/19562

WEBLINKS

[1] "Time from NPL (MSF)," Wikipedia: https://en.wikipedia.org/wiki/Time_from_NPL_(MSF)
[2] "Time receiver for the Rugby MSF," Elektor 9/1982: www.elektormagazine.com/magazine/elektor-198209/44950
[3] J. Buiting, "Junior Computer", Elektor 1/2005: www.elektormagazine.nl/magazine/elektor-200501/14599
[4] 3.5"-display: www.lcdwiki.com/3.5inch_Arduino_Display-UNO
[5] NPL Time & Frequency Services, "MSF 60 kHz Time and Date Code": www.npl.co.uk/products-services/time-frequency/msf-radio-time-signal/msf_time_date_code