Radio logicielle MSF utilisant un Raspberry Pi Pico

Décoder un signal temporel avec une radio logicielle (SDR) à base de RPi Pico

Antenne à très basse (VLF) fréquence à Anthorn (Dougsim, https://bit.ly/34HXeuG)

Martin Ossmann (Allemagne)

MSF est l'équivalent britannique de l'émetteur allemand de signaux horaires DCF77. Ce projet de radio logicielle montre comment un récepteur et un décodeur de ces signaux horaires (et d'autres) peuvent être mis en œuvre simplement et, surtout, à moindre coût. Comme matériel, vous n'aurez besoin que d'un Raspberry Pi Pico à prix modéré pour la réception, le décodage et l'affichage des informations du signal horaire MSF.

En Allemagne, l'émetteur DCF77 de Mainflingen transmet un signal horaire codé en ondes longues. Son équivalent basé au Royaume-Uni est le signal MSF, anciennement connu sous le nom de « The Rugby Clock » [1]. Il envoie des signaux horaires en utilisant un signal porteur à ondes longues de 60 kHz. Au début, il servait d'étalon de fréquence et émettait un paquet d'impulsions de cinq minutes deux fois par jour. Le protocole de transmission du signal a changé plusieurs fois au cours des dernières décennies, mais ce n'est qu'en 1977 que le codage a commencé à inclure des informations sur l'heure et la date pouvant être évaluées par le récepteur.

Projet

Au fil des ans, de nombreux circuits de réception/décodage qui utilisent le signal DCF77 ont été décrits dans divers articles d'Elektor, mais c'est pratiquement la première fois que la conception d'un récepteur MSF est présentée. Il y a eu, cependant, un circuit d'extension [2] pour le bon vieux *6502 Junior Computer* [3] décrit dans l'édition anglaise d'*Elektor*. L'eau a coulé sous les ponts depuis lors, et la technologie relative aux récepteurs/décodeurs a progressé à pas de géant. Dans cet article, nous allons utiliser les derniers principes « à jour » pour construire une radio définie par logiciel (SDR) à l'aide d'une petite carte à microcontrôleur. La carte Raspberry Pi Pico, qui utilise un CPU RP2040 cadencé à 125 MHz (équipé de deux cœurs à 32 bits ARM Cortex M0+), est un matériel approprié, voire prédestiné, pour cette application. Son convertisseur analogique numérique peut fonctionner à 500 Kéch/s. Toute cette puissance de traitement peut être achetée pour un montant presque négligeable de 5 € (voir l'encadré « **Produits** » ci-dessous). Nous présentons ici la mise en œuvre matérielle et logicielle d'un récepteur complet pour le signal temporel MSF de 60 kHz. Le récepteur complet, sans écran, mais avec une sortie RS232 et une connexion d'antenne, est représenté sur la **figure 1**.

Matériel

Tout d'abord, nous allons jeter un coup d'œil au matériel nécessaire pour construire la radio logicielle (SDR). Il y a juste quelques composants supplémentaires à connecter à notre carte Pico.

Entrée d'antenne : nous utilisons la broche d'entrée analogique ADC2 (GPIO28, sur la broche 34 de la carte Pico) pour la réception des signaux de l'antenne. Le CA/N utilise la tension interne de 3,3 V comme tension de référence. Cette broche doit donc être polarisée à la moitié de la tension de référence. Les deux résistances de la **figure 2** s'en chargent. Le condensateur C1 de 10 µF assure le couplage en courant alternatif du signal reçu.

Sortie RS232 : dans sa forme la plus simple (sans affichage à cristaux liquides), le récepteur utilise une interface série (115.200 bits/s) pour transmettre les données. L'interface est mise en œuvre par le circuit illustré à la **figure 3**. Il n'est pas possible d'utiliser le port USB pour produire les données série, car cela générerait des interruptions de manière imprévisible dans notre logiciel.

CN/A PWM : lorsqu'aucun écran LCD n'est connecté, il est possible d'utiliser les CN/A avec des signaux modulés en largeur d'impulsion (PWM) pour faciliter le débogage. Nous avons mis en place deux CN/A PWM avec les filtres passe-bas associés comme le montre la **figure 4**. En utilisant GPIO 2 et GPIO 3 comme sorties PWM, par exemple, le signal démodulé et les signaux Bit-Timer peuvent être affichés sur un oscilloscope (**figure 5**).

LCD : le module Arduino 8 bits de 3,5 pouces ILI9486 (version sans écran tactile SKU MAR3502 [4]) peut être utilisé comme écran LCD. Ce module Arduino de 3,5 pouces possède 480x320 pixels colorés et est commercialisé pour environ 10 €. La **figure 6** montre comment le connecter au Raspberry Pi Pico.

Le signal reçu est affiché sur l'écran LCD avec une forme d'onde montrant les informations de synchronisation des bits. Les informations sur l'heure de réception sont affichées en texte clair au-dessus de la forme d'onde (**figure 7**). Si vous ne souhaitez pas afficher ces informations, vous pouvez vous passer de l'écran LCD sans avoir à modifier le logiciel.

Antenne active : nous avons déjà abordé la connexion de l'antenne ; le circuit de l'antenne active est présenté à la **figure 8**. Il est essentiellement basé sur le double amplificateur opérationnel LM6132. Cet AOP est particulièrement adapté à cette application avec une tension de fonctionnement de 2,7 à 24 V, un produit de gain-bande passante de 10 MHz, une plage de signal d'entrée et de sortie comprise entre les deux polarités d'alimentation, et une faible consommation de courant de 360 µA par amplificateur. Bien sûr, d'autres AOP feraient aussi l'affaire

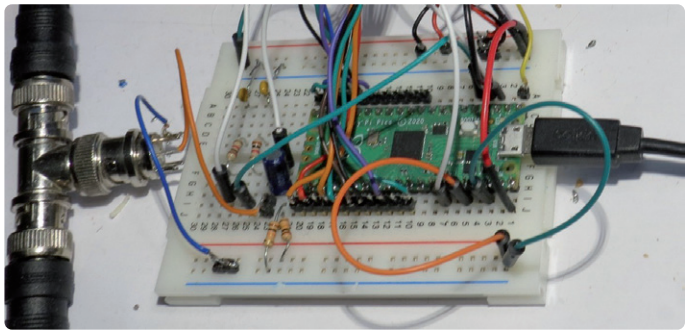


Figure 1. Carte Raspberry Pi Pico dans une radio définie par logiciel (SDR) pour la réception MSF.

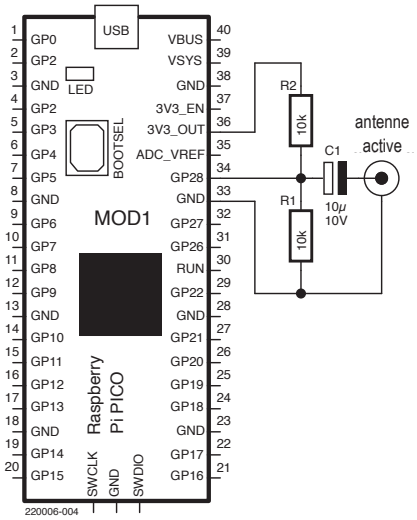


Figure 2. Composants requis à l'entrée du signal A/N.

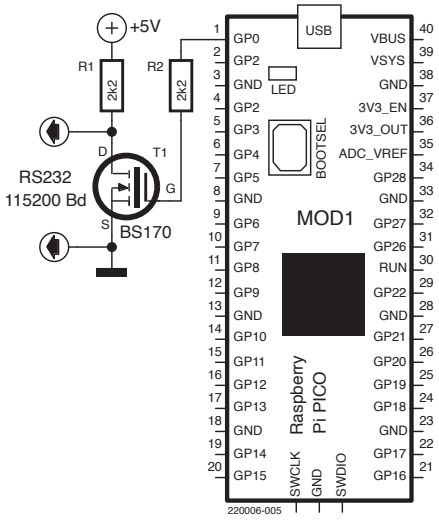


Figure 3. Sortie RS232 de la carte Pico.

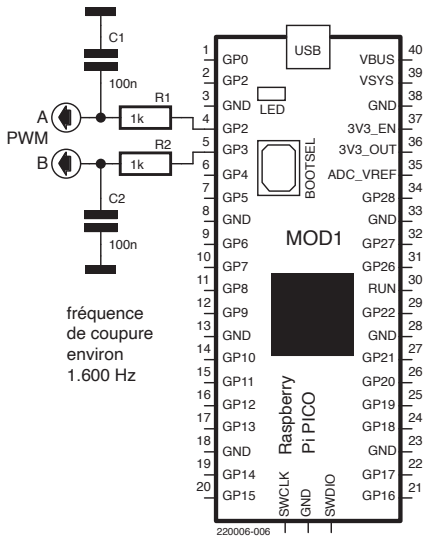


Figure 4. Deux filtres passe-bas pour les signaux de débogage CN/A-PWM.

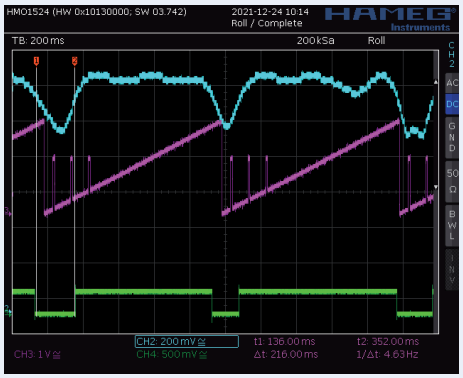


Figure 5. Les signaux du test PWM : La courbe du haut est la valeur de l'amplitude, celle du milieu est ***SecondTimer*** *montrant les impulsions de déclenchement de l'échantillonnage, et celle du bas est le signal numérique* ***sigValue.***

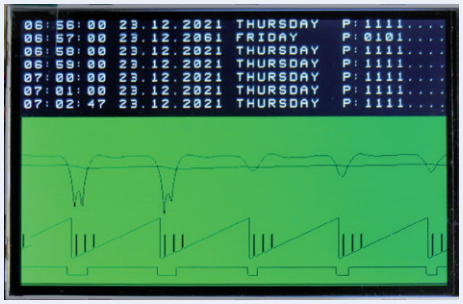


Figure 7. Informations du récepteur sur un écran LCD.

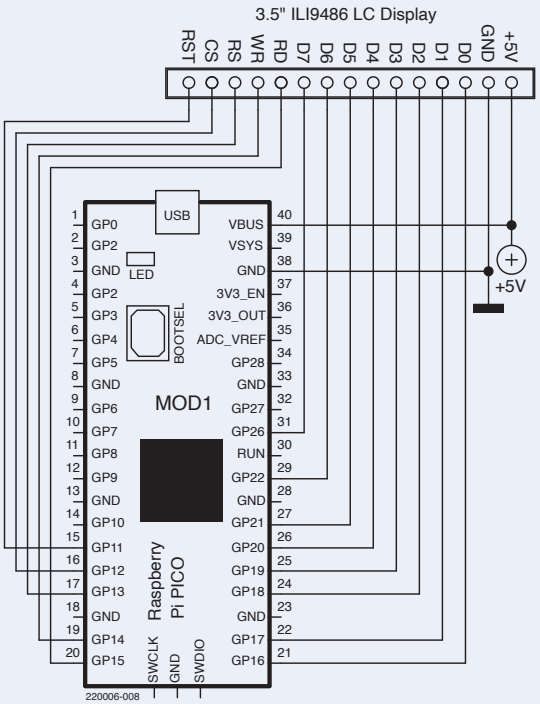
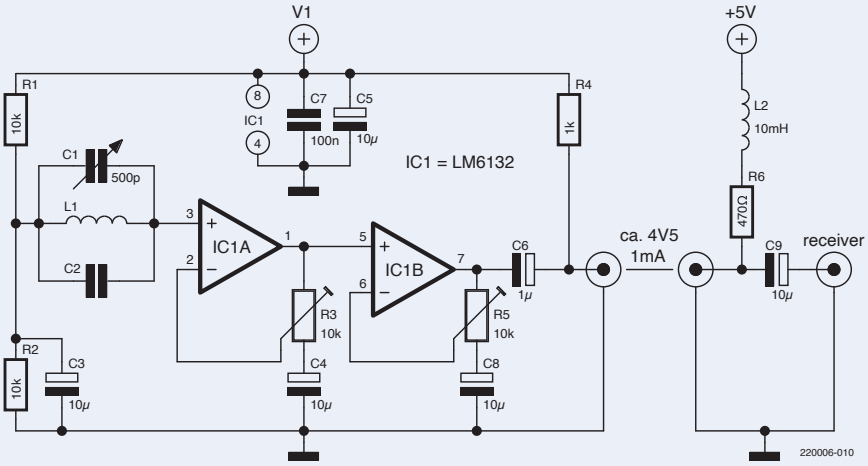


Figure 6. Connexions du LCD 3,5 pouces.



L1: 500 spires fil cuivre émaillé 0,2 mm sur barre de ferrite 10 mm x 180 mm

C2: comme requis

Figure 8. Antenne active pour le signal MSF de 60 kHz.

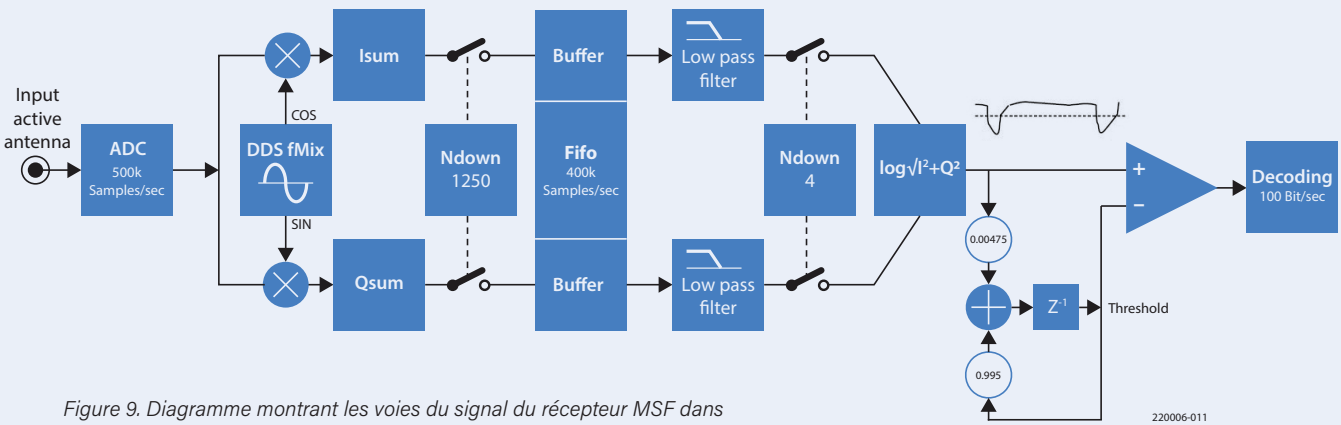


Figure 9. Diagramme montrant les voies du signal du récepteur MSF dans la carte Raspberry Pi Pico.

ici, mais si vous avez l'intention de remplacer le LM6132, vérifiez bien les spécifications.

Programmation du mélangeur d'entrée

Nous abordons maintenant la programmation. Les voies analogiques de la radio logicielle (SDR) sont structurées comme le montre la **figure 9**. Le Raspberry Pi Pico peut être programmé en utilisant différents langages de programmation. Pour cette application, nous avons choisi le langage C en utilisant l'environnement de développement Microsoft Visual Studio Code fonctionnant sur un PC sous Windows 10. Voyons comment fonctionnent les différentes parties.

La routine d'échantillonnage du CA/N est déclenchée par la modulation PWM. Elle est appelée 500.000 fois par seconde via une interruption. Le décalage `ADCoffset` = 2048 est soustrait de la valeur du CA/N et le résultat est ensuite multiplié par `ADCscale` = 10 (**listage 1**). La phase de l'oscillateur local (LO-DDS) est mise à jour et la valeur d'entrée est multipliée par le cosinus (signal en phase ou signal I) et le sinus (signal en quadrature de phase ou signal Q). Les produits sont additionnés sur 1250 échantillons (dans `Isum` et `Qsum`). Les valeurs sont ensuite (dans le **listage 2**) transmises à un dispositif PEPS (FIFO) pour un traitement ultérieur, qui se déroule alors à 500.000/s/1250 = 400 échantillons/s. Ce taux d'échantillonnage est si faible que tout le traitement ultérieur peut être effectué en utilisant des valeurs de variables doubles. Les valeurs sont lues à partir du FIFO et passent par un filtre passe-bas Butterworth d'ordre quatre, avec une fréquence de coupure de 3 Hz. Au cours du développement, il s'est avéré que cette basse fréquence de coupure était nécessaire, car l'antenne utilisée par l'auteur recevait de forts signaux parasites directement adjacents au signal utile. Cette opération est suivie d'un autre sous-échantillonnage, cette fois d'un facteur 4, de sorte que 100 échantillons/s sont ensuite traités.

La routine `msfSample()` du **listage 3** calcule ensuite l'amplitude `ampl` de la porteuse à partir des composantes I/Q. Le logarithme d'`ampl` est dérivé et stocké à son tour dans `ampl`, ce qui facilite le décodage des bits. Le seuil de niveau de commutation est dérivé d'`ampl` via un calcul de filtre récursif de premier ordre. Le signal `ampl` est ensuite comparé au seuil de niveau de commutation `threshold` pour déterminer sa valeur numérique de réception `sigValue`. Maintenant que le traitement du signal analogique a été traité, nous pouvons examiner comment les données sont récupérées à partir du signal reçu et la correspond avec les informations indiquant l'heure de la journée.

Lecture des bits

L'émetteur MSF envoie des impulsions de porteuse RF toutes les secondes, comme le montre la **figure 10**. À la seconde 0 de chaque minute, la porteuse s'éteint pendant 500 ms. La radio logicielle utilise cette impulsion pour la synchronisation. Les impulsions émises à chacune des 59 secondes suivantes contiennent deux bits d'information : A et B. Au début de chacune de ces secondes, la porteuse est éteinte pendant 100 ms (correspondant à 10 échantillons dans notre application). Si le bit A = 1, la porteuse reste éteinte pendant 100 ms supplémentaires, et si le bit B = 1, la porteuse est éteinte pendant 100 ms supplémentaires.

Dans `SecondTimer`, un timer, synchronisé à chaque seconde, tourne de 0 à 99. Le décodage logiciel fonctionne comme suit : dans `Duration`, la longueur d'impulsion de l'impulsion actuelle est mesurée. Si une absence de 0,5 s de la porteuse est détectée, `SecondTimer` est réglé sur la valeur 50 - 2 = 48 de sorte que le timer SecondTimer fonctionne maintenant de manière synchrone avec la seconde (**listage 4**). En

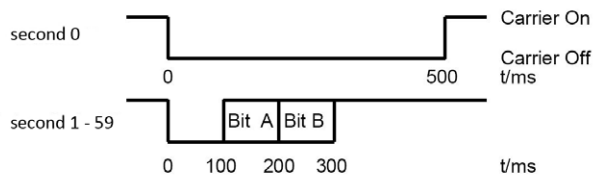


Figure 10. L'impulsion des secondes MSF montrant le codage Bit A et Bit B.

Year (BCD coded 0...99)								Meaning
80	40	20	10	8	4	2	1	BCD weighting
17A	18A	19A	20A	21A	22A	23A	24A	Bit

Month (BCD coded 1...12)					Meaning
10	8	4	2	1	BCD weighting
25A	26A	27A	28A	19A	Bit

Day of month (BCD coded 1...31)						Meaning
20	10	8	4	2	1	BCD weighting
30A	31A	32A	33A	34A	35A	Bit

Day of week (BCD coded 0...6)			Meaning
4	2	1	BCD weighting
36A	37A	38A	Bit

Hour (BCD coded 0...23)						Meaning
20	10	8	4	2	1	BCD weighting
39A	40A	41A	42A	43A	44A	Bit

Minute (BCD coded 0...59)							Meaning
40	20	10	8	4	2	1	BCD weighting
45A	46A	47A	48A	49A	50A	51A	Bit

Minute marker								Meaning
52A	53A	54A	55A	56A	57A	58A	59A	Bit
0	1	1	1	1	1	1	0	Value

Parity bits
Bit 54B with bit 17A to 24A results in odd number of bits
Bit 55B with bit 25A to 35A results in odd number of bits
Bit 56B with bit 36A to 38A results in odd number of bits
Bit 57B with bit 39A to 51A results in odd number of bits

Figure 11. Schéma de codage temporel MSF. (Source : [5]).

Listage 1. Échantillonnage A/N et fonctionnement de l'oscillateur local. Sommation et sous-échantillonnage avec un facteur de 1250 à 400 échantillons/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
```

Listage 2. Filtrage des valeurs I et Q et sous-échantillonnage par un facteur 4 à 100 échantillons/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 ;
            }
```

Listage 3. Calcul de l'amplitude ampl, du seuil de commutation « threshhold » et de la durée de l'impulsion de récupération des bits.

```
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
            }
```

Listage 4. Échantillonnage des bits A et B déclenché par 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
         } ;
      }
```

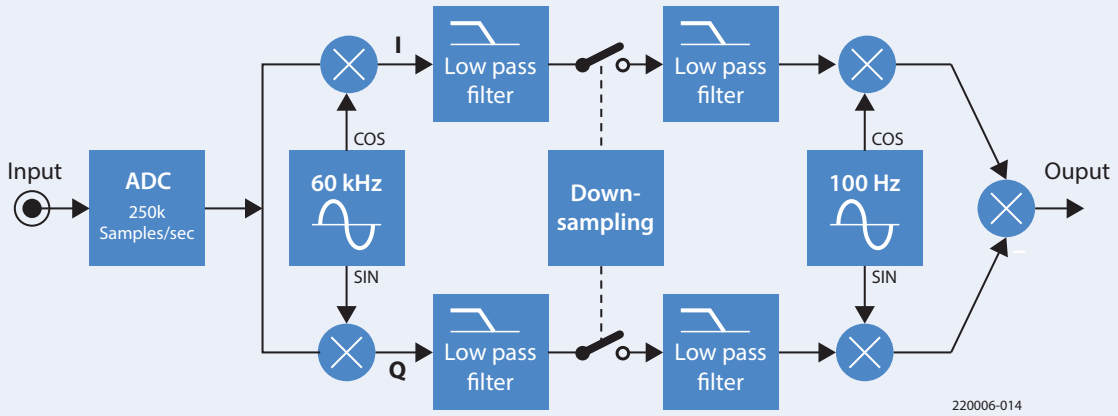


Figure 12. Conversion ascendante vers une FI de 100 Hz.

même temps, la minute est synchronisée en fixant la valeur de la seconde actuelle à 0 dans `doMinuteSync()`.
Grâce au timer `SecondTimer`, le signal reçu est échantillonné à la position mid-bit du Bit A et du Bit B (`SecondTimer==15` et `SecondTimer==25`) afin de déterminer les valeurs de ces bits transmis. Nous sortons simplement la valeur numérique reçue via la broche 4 du GPIO (broche 6 du Pico) :

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

La valeur de `SecondTimer` est également sortie ultérieurement via PWM pour être utilisé dans le débogage, tout comme les valeurs d'`ampl` et `DAC`. Ceci est réalisé avec les deux instructions suivantes :

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

Information sur le temps de décodage

Lorsque `SecondTimer` est synchronisé avec un décalage de 0,5 s, une minute s'est écoulée et nous pouvons évaluer la dernière information temporelle. Les bits de données reçus sont dans les valeurs `MSFbits[0 à 59]`. L'émetteur encode dans ces bits les informations indiquées dans la **figure 11.** Les informations relatives à l'heure et à la date sont ensuite simplement réorganisées comme dans le **listage 5** pour donner les heures et les minutes.
Nous affichons également les mêmes informations que nous envoyons via l'interface série sous forme de texte sur l'écran LCD. Ceci est fait en utilisant les instructions données dans le **listage 6**. On effectue un contrôle de parité sur les informations reçues comme dans le **listage 7**. Les bits contrôlés sont les bits A de l'information transmise. Les quatre bits de contrôle sont les bits B de chaque impulsion de seconde correspondante. Quatre contrôles de parité sont réalisés, l'intégrité de 12 bits au maximum est protégée par un bit de parité.

Signaux de débogage

Les récepteurs superhétérodynes (ou superhet) classiques mélangent le signal RF entrant avec un signal d'oscillateur local à fréquence variable pour produire une fréquence intermédiaire inférieure (ou FI). Le signal FI est ensuite filtré avec un filtre à bande relativement étroite. Le signal FI du récepteur MSF60 peut également être visualisé à l'aide d'un oscilloscope. Notre récepteur mélange le signal d'entrée jusqu'à FI = 0 Hz. Si vous voulez observer un signal FI CA, vous pouvez convertir le signal FI 0 en FI CA. Le schéma fonctionnel du circuit est illustré à la **figure 12**.
Le logiciel permettant d'effectuer la synthèse des canaux nécessaire est présenté dans le **listage 8**. Une sortie PWM est utilisée comme C/NA. Le signal FI modulé en amplitude à 100 Hz est illustré à la **figure 13**. Ceci conclut la conception et la construction du récepteur MSF. Nous avons utilisé un seul cœur du processeur dans cette application, ce qui laisse beaucoup de puissance de calcul disponible pour une extension. Le décodage des bits, par exemple, peut être amélioré en matière de tolérance aux erreurs. Il est possible de construire un récepteur DCF77 à peu près de la même manière, il suffirait d'adapter le processus de décodage des bits. La réception du signal MSF ici à Aix-la-Chapelle (Allemagne) est beaucoup plus faible que celle du signal DCF77 avec une radio logicielle (SDR) équivalente. Cela se traduit souvent par des bits de parité indiquant des erreurs dans les informations reçues, mais des messages sans erreur sont tout de même transmis assez fréquemment pour permettre l'affichage fiable d'informations précises sur l'heure du jour.

Travailler avec un RP2040

Nous avons démontré qu'avec très peu de matériel supplémentaire, la carte Raspberry Pi Pico peut être transformée en un MSF SDR complet. La tâche la plus complexe de cette réalisation est la construction de l'antenne active. Compte tenu de la puissance de traitement

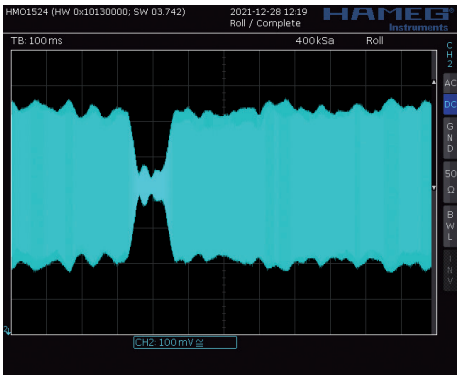


Figure 13. Signal FI de 100 Hz modulé en amplitude.

Listage 5. Décodage et sortie BCD en série des heures et des minutes.

```
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()
```

Listage 6. Affichage des informations sur les heures et les minutes sur l'écran LCD.

```
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
```

Listage 7. Contrôle de parité.

```
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) ;
```

Listage 8. Code de la synthèse multi-canal.

```
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
```

et du faible coût de la carte, cette application montre ce que même un amateur disposant de peu de ressources est capable de réaliser de nos jours. Vous lisez souvent à quel point la carte Pico est facile à programmer en Python, mais dans cette application, elle ne sera pas adaptée à la fréquence d'échantillonnage de 500 Kéch/s du signal d'entrée. Cependant, grâce au langage C, on peut manipuler plus directement le matériel du microcontrôleur et le programmer efficacement. Nous voyons ici que même sans l'avantage d'une unité à virgule flottante (FPU), le RP2040 est plus que capable d'implémenter un filtre numérique passe-bas.

220006-04

Des questions, des commentaires ?

Envoyez un courriel à l'auteur (ossmann@fh-aachen.de) ou contactez Elektor (redaction@elektor.fr).

PRODUITS

- **Elektor Raspberry Pi RTL-SDR Kit (livre et composants) (SKU 19518)**
  www.elektor.fr/19518
- **Elektor SDR Hands-on Kit (livre + shield SDR avec toroïde ferrite et câble)**
  www.elektor.fr/19041
- **Raspberry Pi Pico RP2040 (SKU 19562)**
  www.elektor.fr/19562

LIENS

[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, « Elektor Junior Computer », Elektor 1/2005: www.elektormagazine.fr/magazine/elektor-200501/10053
[4] Écran de 3,5 pouces : www.lcdwiki.com/3.5inch_Arduino_Display-UNO
[5] Services de temps et de fréquence NPL, « MSF 60 kHz Time and Date Code » : www.npl.co.uk/products-services/time-frequency/msf-radio-time-signal/msf_time_date_code