To urządzenie generuje częstotliwości od 0,1 Hz do ... 4400 MHz. Wyświetlacz TFT 480x360

Celem tego badania nie jest stworzenie nadajnika radiowego, ale generatora laboratoryjnego o bardzo małej mocy (<1 mW, tj. <0 dBm), pozwalającego na eksperymentowanie i regulację obwodów BF, HF, VHF, UHF, SHF jak na przykład filtry, aby skonfrontować teorię z doświadczeniem bez promieniowania. Urządzenie musi być zamknięte w metalowej obudowie podłączonej do uziemienia. Sygnał wyjściowy będzie w gnieździe SMA. Filtr HF zostanie włożony do zasilacza. Zobacz linki na dole strony dotyczące przydzielania częstotliwości radiowych. Jakakolwiek transmisja w pasmach VHF i UHF jest zabroniona (amatorska amatorska transmisja radiowa) i zostanie szybko zidentyfikowana, co może doprowadzić Cię bezpośrednio do więzienia. Amatorzy radiowi będą mogli dostosować tę realizację do własnych potrzeb.To urządzenie generuje częstotliwości od 0,1 Hz do ... 4400 MHz. Wyświetlacz TFT 480x360

peut vous amener directement en prison. Les radio-amateurs pourront adapter cette réalisation pour leurs besoins propres.

Cliquez sur une image ci-dessous pour visualiser l'album. Certaines sont commentées

Table des matières :

34 - Version (v10_1) Ajout d'un DDS AD9850

35 - Version (v10_2) Wobulation permise également pour l'AD9850

36 - Tests de wobulation en mode AD9850

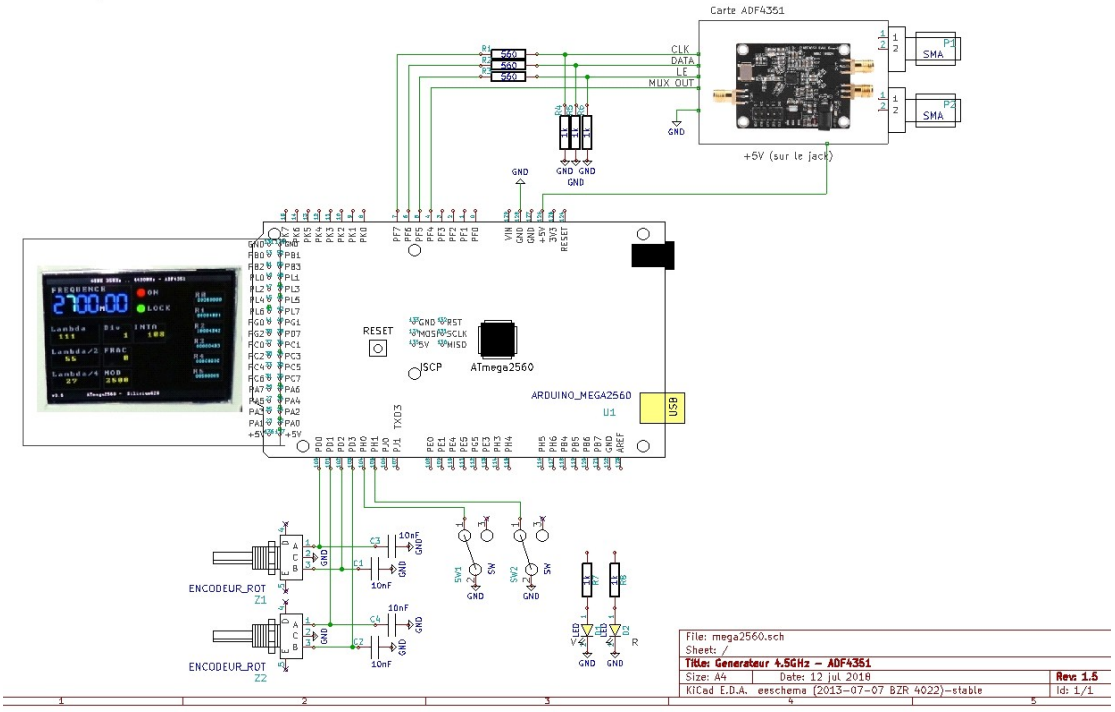
37 - La bobine d'excitation

38 - -

39 - ---

40 - Une tit' face avant...

41 - La réalisation d'un OM (F1CHM)



Trzyprzewodowe łącze szeregowe jest wystarczające do sterowania ADF4351:

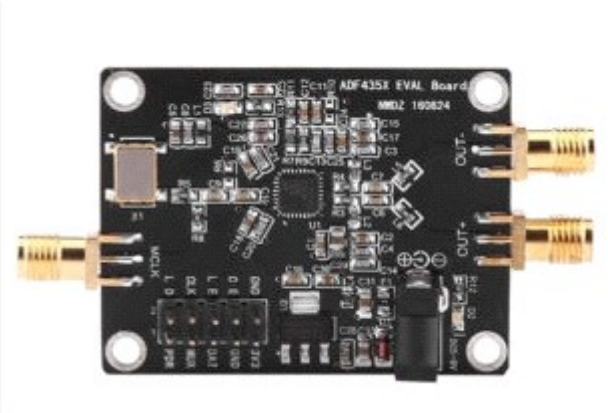
- dane
- zegar

Te trzy sygnały są generowane przez ATmega na jego porcie F z poziomem logicznym 0..5V

Ale ADF4351 działa pod 3V3, stąd obecność trzech dzielników napięcia rezystora (560 ohm - 1k).

Czwarty sygnał (MUX OUT) to logiczne wyjście ADF4351 wskazujące na dobrą synchronizację.

3 Karta ADF4351



Ta karta znaleziona w sieci za podwójną cenę samego komponentu jest mile widziana, wiedząc, że ADF4351 jest prezentowany w mikroskopijnym pakiecie LFCSP o rozstawie 0,5 mm.

4- moduł wyświetlacza TFT 480 x 320 pikseli

Ten moduł jest przeznaczony do bezpośredniego interfejsu Arduino Mega2560. Rozdzielczość 480 x 360 pikseli dla rozmiaru 3 cali zapewnia doskonałą ostrość obrazu (kolor). Biblioteka „UTFT” dostępna dla Arduino bardzo mu odpowiada.

5 Płyta Arduino Mega2560

Ta karta nie jest już dostępna. Świetny klasyk, którego sercem jest ATmega 2560, jeden z najpotężniejszych przed przejściem na ARM (te ostatnie są używane w szczególności w Raspberry Pi).

ATmega 2560 to:

256 kB pamięci flash

8 KB pamięci RAM

4 KB EEprom

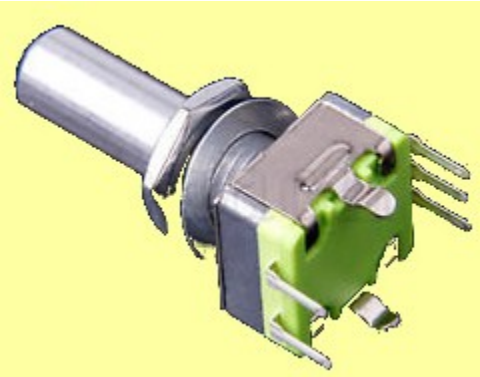
Cała gama urządzeń peryferyjnych (USART, SPI, liczniki, timery ...)

86 linii portów I / O

etc ...

Wybrałem go, aby łatwo połączyć wyświetlacz. Jeśli chodzi o ADF4351, niektóre latają za pomocą prostego i mini Arduino Mini.

6 Kodery obrotowe kod szary



Poświęciłem artykuł temu koderowi: TUTAJ.

Umożliwiają zwiększenie / zmniejszenie częstotliwości wyjściowej oraz wybór cyfry do modyfikacji.

7 Arkusz danych ADF4351: szczegóły

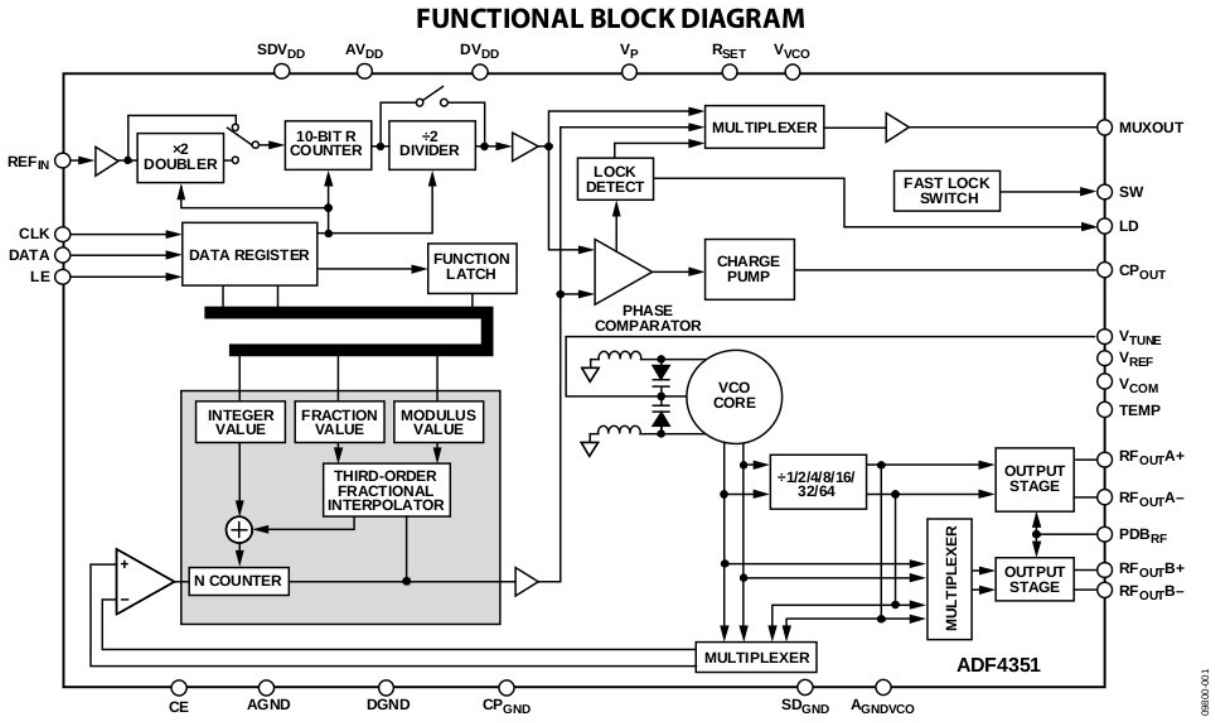


Figure 1.

Ten schemat funkcjonalny zdziwił mnie. Tylko czytając i czytając tekst pdf zrozumiałem głęboko transcendentny charakter szarego prostokąta na dole po lewej! Odtąd magiczna formuła, która następuje, staje się świetlistą ... uff!

Z drugiej strony odczyt pdf sugeruje, że obwód ten został zaprojektowany do wyposażenia urządzeń telefonii komórkowej GSM i innych smartfonów działających w UHF (1800 MHz) i pasmach.

Ogólna zasada ADF4351 jest następująca:

Zestaw trzech VFO (przełączalnych) obejmuje pasmo częstotliwości od 2 GHz do 4 GHz. Niższe częstotliwości (do 35 MHz) są uzyskiwane przez podział przez / 2, / 4, / 8, / 16, / 32, / 64, jak widać w moim pliku niższej klasy ADF4351 na tej stronie ,

9 Program C ++ dla Arduino Mega2560

Kod

```
/*
Firmware pour carte générateur de signaux ADF4351
et carte Uno Mega2560 + afficheur 480x320 non tactile.
par Silicium628.

Ce fichier source "GeneUHF_ADF4351.cpp" est libre, vous pouvez le redistribuer et/ou le modifier selon
les termes de la Licence Publique Générale GNU.

En ce qui concerne certaines bibliothèques incluses, issues du domaine "Arduino" (en particulier UTFT), il faut voir au cas par cas.
*/

/**
REMARQUES :
-les ports utilisés doivent être définis dans les fichiers "AD9850.h" et "AD9951.h"
-il faut copier le dossier "UTFT" dans le dossier des librairies /usr/share/arduino/libraries ET définir les permissions qui vont bien

**/


//================================
#define version "v4.0"
```

```
//================================

#include "GeneUHF_ADF4351.h"


#include <avr/io.h>
#include <stdint.h>
#include <stdlib.h>
#include <util/delay.h>
#include <math.h>

#include "UTFT.cpp"
#include "uart2560_628.c"
#include "ADF4351-628v2.h";

#define portPIN_switch PINH

#define pin_switch1 0b00000001
#define pin_switch2 0b00000010


// Declare which fonts we will be using
extern uint8_t SmallFont[];
extern uint8_t BigFont[];
extern uint8_t SevenSegNumFont[];

/
*************************************************************************************************************
//choisir le bon driver suivant, en fonction du type d'afficheur (tous deux des 480x320 qui se ressemblent, achetés au même fournisseur) ******
//la library correcte doit être installée dans le bon dossier ARDUINO, avec les permissions d'accès correctes !

UTFT TFT480(CTE32HR,38,39,40,41);
UTFT TFT480(HX8357C,38,39,40,41);
*************************************************************************************************************/

       UTFT TFT480(HX8357C,38,39,40,41);

       ADF4351 CarteADF4351;

       AffiNombre pannel_1;
```

```
62.       AffiNombre pannel_2;
63.       AffiNombre pannel_3;
64.       AffiNombre pannel_4;
65.       AffiNombre pannel_5;
66.       AffiNombre pannel_6;
67.       AffiNombre pannel_7;
68.       AffiNombre pannel_8;
69.       AffiNombre pannel_9;
70.
71.       AffiNombre pannel_R0;
72.       AffiNombre pannel_R1;
73.       AffiNombre pannel_R2;
74.       AffiNombre pannel_R3;
75.       AffiNombre pannel_R4;
76.       AffiNombre pannel_R5;
77.
78.       LED LED1;
79.       LED LED2;
80.
81.       uint8_t mode = 0;
82.
83.       uint16_t x0, y0;
84.       uint16_t x_7seg, y_7seg;
85.
86.   uint32_t frequence , memo_frequence; // fréquence du signal de sortie ; en
   multiples de 10 kHz ; ex: 240000 représente 2400.00MHz
87.  uint32_t frq_min;
88.  uint32_t frq_max;
89.  uint32_t   delta_F;
90.  uint32_t lambda; // longueur d'onde en mm
91.
92.       uint8_t pos; // position du multiplicateur
93.       uint8_t pos_curseur; // position du curseur (pour affichage)
94.       uint8_t  switches,  memo_switches;  //   8  bits  permettant  de  mémoriser
   l'état de 8 inverseurs
95.
96.  uint32_t pas;
97.  uint8_t etat;
98.  uint16_t compteur1;
99.
100.        uint8_t envoyer_data;
101.        float position;
102.
103.            String txt_hexa[33];
104.
```

```
105.
106.      /**
107.      RAPPEL variables avr-gcc (vérifiable avec le .map)
108.
109.      char                              1   -128 .. 127 ou caractères
110.      unsigned char 1      0 .. 255
111.      uint8_t                           1     (c'est   la   même   chose   que
   l'affreux 'unsigned char')
112.      char toto[n]  n
113.      int                                     2     -32768 .. 32767
114.      int16_t                           2     idem 'int'
115.      short int                  2      pareil que int (?)
116.      unsigned int  2      0 .. 65535
117.      uint16_t                   2      idem 'unsigned int'
118.      long int                   4 octets  -2 147 483 648 à 2 147 483 647
119.      int32_t                           4     octets -> 32 bits   ;      idem
   long int
120.      long long int 8      octets -> 64 bits
121.      unsigned long int   4      octets -> 32 bits   ;      0 .. 4 294 967 295
   (4,2 x 10^9)
122.      uint32_t                   4      32 bits      ;      idem  'unsigned  long
   int'
123.      float                             4
124.      double                     ATTENTION ! 4 octets (oui, 32 bits ! et pas 64
   bits (8 octets) comme en C standard)
125.
126.      La déclaration  char JOUR[7][9];
127.      réserve l'espace en mémoire pour 7 mots contenant 9 caractères (dont 8
   caractères significatifs).
128.      **/
129.
130.      void init_ports (void)  // ports perso
131.      {
132.      // 0 = entree, 1=sortie   ;   les 1 sur les pins en entrees activent les
   R de Pull Up (tirage à VCC)
133.
134.        DDRD  = 0b11110000;
135.        PORTD = 0b00001111;
136.
137.        DDRF  = 0b11101111;
138.        PORTF = 0b00000000;
139.
140.        DDRH  = 0b11111100;
141.        PORTH = 0b00000011;
142.
```

```
143.      //  DDRJ  = 0b11111110; // PJ0 = RXD3  - PJ1 = TXD3 (attention c'est PJ1 qui est au bord du connecteur sur la carte Arduino mega2560)
144.      //  PORTJ = 0b00000001;
145.
146.
147.       // REMARQUE : les ports utilisés doivent être définis dans les fichiers "AD4351.h" et  "AD4351.h"
148.      }
149.
150.
151.
152.      void int_EXT_setup()
153.      {
154.            // voir 15. External Interrupts (ATmega2560.pdf p:109)
155.
156.            EICRA |= 0b00001010;    // bits[1,0] = 10 -> int0 sur front descendant;  bits[3,2] = 10 -> int1 sur front descendant;   - p:110
157.            EIMSK |= 0b00000011;  // bit[0]=1 ->  INT0 enable  ; bit[1]=1 -> INT1 enable  -   parag 15.2.3  EIMSK - p:111
158.      }
159.
160.
161.
162.      void init_variables(void)
163.      {
164.            envoyer_data = 0;
165.            frequence = 4000;   // 40.00 MHz
166.            frq_min = 3200;           //     32.00 MHz - toutefois la PLL décroche au dessous de 32.4MHz (voir led LOCK)
167.            frq_max = 440000;   // 4400.00 MHz = 4.4 GHz
168.            pos_curseur = 3;
169.
170.            pas = 1000;
171.            switches =0;
172.            memo_switches =0;
173.
174.            compteur1 =0;
175.
176.            uint8_t x0=10;
177.            uint8_t y0=20;
178.
179.            delta_F = 100;  // 1.00 MHz
180.      }
181.
182.
```

```
183.
184.
185.      ISR (INT0_vect)
186.      {
187.      //interruption sur front descendant sur l'entree Int0
188.      // declenchee par la rotation du codeur_ROT (1)  ->  +/-frequence
189.
190.            etat = PIND & 0b00000100;
191.
192.            {
193.                  if (etat == 0b00000100)
194.                  {
195.                        if (frequence >= (frq_min + pas)) {frequence -= pas;}
196.                  }
197.                  else
198.                  {
199.                        if ( (frequence+pas)  <= frq_max) {frequence += pas;}
200.                        if  (  (frequence+pas)    >    frq_max)  {frequence  =
   frq_max;}
201.                  }
202.                  envoyer_data = 1;
203.            }
204.      }
205.
206.
207.
208.      ISR (INT1_vect)
209.      {
210.      //interruption sur front descendant sur l'entree Int1
211.      // declenchee par la rotation du codeur_ROT (2) -> pas
212.            uint8_t n;
213.            etat = PIND & 0b00001000;
214.
215.            if (etat == 0)
216.            {
217.                  if (pos_curseur > 1) {pos_curseur--;}
218.            }
219.            else
220.            {
221.                  if (pos_curseur < 6) {pos_curseur++ ;}
222.            }
223.            pas=1;
224.            for (n=1; n<pos_curseur; n++)    {      pas *=10;     }
225.            envoyer_data = 1;
226.      }
```

```
227.
228.
229.      ISR(TIMER3_COMPA_vect)
230.      { }
231.
232.
233.      void trace_ecran()
234.      {
235.            TFT480.setColor(10, 10, 5);
236.            TFT480.fillRect(0, 14, 479, 309);
237.            trace_entete();
238.
239.            TFT480.setBackColor(0, 0, 0);
240.        TFT480.setColor(180,180,180); // couleur de l'étiquette uniquement
241.        TFT480.setFont(BigFont);
242.
243.            if(mode==0)
244.            {
245.                  pannel_1.init(10, 20, 210, 80,"FREQUENCE");
246.
247.                  pannel_2.init(5, 120, 130, 50,"Lambda  ");
248.                  pannel_3.init(5, 180, 130, 50,"Lambda/2");
249.                  pannel_4.init(5, 240, 130, 50,"Lambda/4");
250.
251.
252.                  uint16_t xp1 = 145;
253.                  uint16_t yp1 = 120;
254.                  uint8_t dy1 = 60;
255.                  pannel_5.init(xp1, yp1, 75, 50,"Div");
256.                  pannel_6.init(xp1, yp1+dy1, 75, 50,"FRAC");
257.                  pannel_7.init(xp1, yp1+dy1*2, 75, 50,"MOD");
258.
259.                  pannel_8.init(xp1+80, yp1, 85, 50,"INTA");
260.
261.                  uint16_t xp2 = 385;
262.                  uint16_t yp2 = 40;
263.                  uint8_t dy2 = 40;
264.                  pannel_R0.init(xp2, yp2,     80, 35,"R0");
265.                  pannel_R1.init(xp2, yp2+dy2,  80, 35,"R1");
266.                  pannel_R2.init(xp2, yp2+dy2*2,  80, 35,"R2");
267.                  pannel_R3.init(xp2, yp2+dy2*3, 80, 35,"R3");
268.                  pannel_R4.init(xp2, yp2+dy2*4, 80, 35,"R4");
269.                  pannel_R5.init(xp2, yp2+dy2*5, 80, 35,"R5");
270.            }
271.            else
```

```
272.             {
273.                   pannel_1.init(10, 40, 210, 80,"F centrale");
274.                   pannel_9.init(10, 130, 210, 35,"delta F");
275.             }
276.       }
277.
278.
279.
280.
281.       void trace_entete()
282.       {
283.             TFT480.setColor(64, 64, 64);
284.         TFT480.fillRect(0, 0, 479, 13); // bandeau en haut
285.
286.         TFT480.setColor(30, 30, 30);
287.         TFT480.fillRect(0, 300, 479, 319);  // bandeau en bas
288.
289.
290.       //  TFT480.setBackColor(64, 64, 64);
291.         TFT480.setColor(255,255,255);
292.         TFT480.print("GENE 35MHz .. 4400MHz - ADF4351", CENTER, 1);
293.
294.       //  TFT480.setBackColor(64, 64, 64);
295.         TFT480.setColor(255,255,255);
296.         TFT480.print(version, 5, 300);
297.         TFT480.setColor(255,255,255);
298.         TFT480.print("ATmega2560 - ", 100, 300);
299.         TFT480.setColor(255,255,255);
300.         TFT480.print("Silicium628", 210, 300);
301.
302.       }
303.
304.
305.       void TFT_aff_ICI()
306.       {
307.             TFT480.setFont(BigFont);
308.         TFT480.setColor(255,255,255);
309.         TFT480.print("ICI", 0, 220);
310.       }
311.
312.
313.
314.
315.       void TFT_affiche_chiffre_7seg(uint8_t num)
316.       {
```

```
317.            TFT480.setFont(SevenSegNumFont);
318.            TFT480.printNumI(num, x_7seg, y_7seg);
319.      }
320.
321.
322.
323.      void  TFT_aff_nb_form3  (uint32_t  valeur,  uint8_t  nb_chiffres,  uint8_t
   curseur, uint8_t R, uint8_t G, uint8_t B)
324.      {
325.      //affiche un nombre en representation decimale en separant les groupes de
   3 chiffres
326.            unsigned char r ;
327.            char tbl[7];
328.            uint8_t i;
329.
330.            curseur=nb_chiffres +1 -curseur;
331.
332.            for (i=1; i<=nb_chiffres; i++)
333.            {
334.                  r=valeur % 10; // modulo (reste de la division)
335.                  valeur /= 10; // quotient
336.                  tbl[i]=r;
337.            }
338.            for (i=1; i<= nb_chiffres; i++)
339.            {
340.                  TFT480.setColor(R,G,B);
341.
342.                  if (i==curseur)
343.                  {
344.                        TFT480.setColor(0,250,250);
345.                  }
346.                  TFT_affiche_chiffre_7seg(tbl[nb_chiffres +1 -i]);
347.                  x_7seg+=30;
348.
349.                  uint8_t m, k, u;
350.                  m =nb_chiffres-6;
351.                  k =nb_chiffres-3;
352.                  u =nb_chiffres;
353.
354.                  //if (i== 3)
355.                  if (i== m)
356.                  {
357.                        TFT480.setFont(BigFont);
358.                        TFT480.setColor(100,100,100);
359.                        TFT480.print("M",x_7seg, y_7seg+30);
```

```
360.                         x_7seg+=15;
361.                   }
362.                   //if (i== 6)
363.                   if (i== k)
364.                   {
365.                         TFT480.setFont(BigFont);
366.                         TFT480.setColor(100,100,100);
367.                         TFT480.print("k",x_7seg, y_7seg+30);
368.                         x_7seg+=15;
369.                   }
370.                   //if (i== 9)
371.                   if (i== u)
372.                   {
373.                         TFT480.setFont(BigFont);
374.                         TFT480.setColor(100,100,100);
375.                         TFT480.print("Hz",x_7seg, y_7seg+30);
376.                   }
377.             }
378.       }
379.
380.
381.       void TFT_aff_nb_form3b (uint32_t valeur, uint8_t nb_chiffres, uint8_t
    curseur, uint8_t R, uint8_t G, uint8_t B)
382.       {
383.       //affiche un nombre en representation decimale avec separateur 'M' à
    gauche des 2 chiffres de droite
384.
385.             unsigned char r ;
386.             char tbl[7];
387.             uint8_t i;
388.
389.             curseur=nb_chiffres +1 -curseur;
390.
391.             for (i=1; i<=nb_chiffres; i++)
392.             {
393.                   r=valeur % 10; // modulo (reste de la division)
394.                   valeur /= 10; // quotient
395.                   tbl[i]=r;
396.             }
397.             for (i=1; i<= nb_chiffres; i++)
398.             {
399.                   TFT480.setColor(R,G,B);
400.
401.                   if (i==curseur)
402.                   {
```

```
                        TFT480.setColor(0,250,250);
                }
                TFT_affiche_chiffre_7seg(tbl[nb_chiffres +1 -i]);
                x_7seg+=30;

                uint8_t m = nb_chiffres-2;

                if (i==m)
                {
                        TFT480.setFont(BigFont);
                        TFT480.setColor(200,200,200);
                        TFT480.print("M",x_7seg,  y_7seg+30);  // affichage du separateur 'M' (M comme Megahertz)
                        x_7seg+=15;
                }
            }
      }




      //  ================================================================================================================


      void setup()
      {
        init_ports();
        init_variables();

      //     USART_Init();

        TFT480.InitLCD();
       // TFT480.rotate_screen_180();
        TFT480.setFont(SmallFont);
        TFT480.clrScr();

        int_EXT_setup();

            CarteADF4351.init();

        trace_ecran();
        affiche_leds();

```

```
445.              sei(); // enable interruptions
446.              envoyer_data = 1;
447.       }
448.
449.
450.
451.       void affiche_leds()
452.       {
453.              LED1.init(245, 80, 0,255,0, "LOCK" );
454.              LED2.init(245, 40, 255,0,0, "ON" );
455.       }
456.
457.
458.       void calcul_lambda()
459.       {
460.              lambda = 3e7/frequence; // en mm
461.       }
462.
463.
464.
465.        void loop()
466.       {
467.              uint8_t r1;
468.
469.              while(1)
470.              {
471.       //             memo_mode = mode;
472.       //             memo_switches = switches;
473.
474.              //     mode = portPIN_switch   & pin_switch1; // mode = 0 ou 1
   suivant la position du switch1
475.              //     if ((portPIN_switch  & pin_switch2) == 0) mode = 3;  // =2
476.
477.                     if   ((portPIN_switch   &   pin_switch1)   ==   pin_switch1)
   { switches |= 0b00000001; } else { switches &= 0b11111110; }
478.                     if   ((portPIN_switch   &   pin_switch2)   ==   pin_switch2)
   { switches |= 0b00000010; } else { switches &= 0b11111101; }
479.                     if (memo_switches != switches)   {      }
480.
481.                     if   ((port_PIN_ADF4351   &   pin_MUXOUT)   ==   pin_MUXOUT)
   { LED1.setEtat(1); } else { LED1.setEtat(0);}
482.                     LED2.setEtat(1); // pour faire joli !
483.
484.                     if(envoyer_data > 0)
485.                     {
```

```
                        envoyer_data = 0;

                        calcul_lambda();
                        _delay_ms(1);
                        if ( (switches & 0b00000010) == 0)      {
        affiche_leds();     }

                        double F2;
                        F2 = frequence / 100.0; // F2 en MHz (avec 2 décimales)

                        CarteADF4351.setFreq(F2);

                        pannel_1.affiche_frequence(frequence, 0, 100, 255); // (avec de grands chiffres 7 segments)

                        if (mode == 0)
                        {
                                uint8_t DIV = CarteADF4351.DIV;
                                pannel_5.affiche_valeur(DIV, 4, "  ");

                                uint16_t FRAC = CarteADF4351.FRAC;
                                pannel_6.affiche_valeur(FRAC, 4, "  ");

                                uint32_t MOD = CarteADF4351.MOD;
                                pannel_7.affiche_valeur(MOD, 4, "  ");

                                uint32_t INTA = CarteADF4351.INTA;
                                pannel_8.affiche_valeur(INTA, 5, "  ");

                                _delay_ms(1);


        //unite="mm";
                                pannel_2.affiche_valeur(lambda, 4, "mm");
                                pannel_3.affiche_valeur(lambda/2, 4, "mm");
                                pannel_4.affiche_valeur(lambda/4, 4, "mm");


        pannel_R0.affiche_HEXA(CarteADF4351.registers[0]);

        pannel_R1.affiche_HEXA(CarteADF4351.registers[1]);

        pannel_R2.affiche_HEXA(CarteADF4351.registers[2]);
```

```
524.
          pannel_R3.affiche_HEXA(CarteADF4351.registers[3]);
525.
          pannel_R4.affiche_HEXA(CarteADF4351.registers[4]);
526.
          pannel_R5.affiche_HEXA(CarteADF4351.registers[5]);
527.                         }
528.                         else
529.                         {
530.                               pannel_9.affiche_valeur(delta_F, 4, "M ");
531.                         }
532.                         _delay_ms(1);
533.                   }
534.
535.      /**
536.                   if((portPIN_RAZ & pin_RAZ) == 0) // RAZ de la fréquence sur
   appui de l'encodeurs rotatif du haut (qui comprend un switch)
537.                   {
538.                         if ( (switches & 0b00000001) == 0) {frequence =0;
   envoyer_data += 1; }
539.
540.
541.                         if ((switches & 0b00000010) == 0)
542.                         { // soft reset
543.                               switches == 0;
544.                               TFT480.clrScr();
545.                               TFT480.setColor(30, 30, 30);
546.                               TFT480.fillRect(0, 150, 479, 200);  // bandeau
547.                               TFT480.setColor(255,0,0);
548.                               TFT480.print("SOFT RESET", CENTER, 160);
549.                               _delay_ms(1000);
550.                               TFT480.clrScr();
551.                               setup();
552.                         }
553.
554.                   }
555.            **/
556.
557.                   _delay_ms(1);
558.
559.            }
560.
561.      }
562.
```

```
563.      /**
   ************************************************************************
   ***
564.                                                CLASS AffiNombre

565.      ************************************************************************
   *************/
566.
567.      // Constructeur
568.      AffiNombre::AffiNombre()
569.      {
570.
571.      }
572.
573.
574.      void AffiNombre::init(uint16_t x, uint16_t y, uint16_t dx, uint16_t dy,
   char txt_etiquette_i[10])
575.      {
576.            x0 = x;
577.            y0 = y;
578.
579.            for (int i=0; i<10; i++) {txt_etiquette[i]=txt_etiquette_i[i];}
580.            txt_etiquette[10]='\0';  // zero terminal
581.
582.            TFT480.setColor(10, 10, 5);
583.            TFT480.fillRect(x0, y0, x0+dx, y0+dy);
584.            TFT480.setColor(100, 100, 100);
585.            TFT480.drawRect(x0, y0, x0+dx, y0+dy);
586.            TFT480.setBackColor(0, 0, 0);
587.        TFT480.setColor(180,180,180); // couleur de l'étiquette
588.        TFT480.setFont(BigFont);
589.            TFT480.print(txt_etiquette, x0, y0); // Etiquette
590.      }
591.
592.
593.      void AffiNombre::affiche_frequence(uint32_t F_i, uint8_t R, uint8_t G,
   uint8_t B)
594.      {
595.            x_7seg = x0+10;
596.            y_7seg = y0+20;
597.            uint8_t p2;
598.            p2 = pos_curseur;
599.            TFT_aff_nb_form3b  (F_i,  6,  p2,  R,  G,  B);  //  6  chiffres
   significatifs
600.      }
```

```
601.
602.
603.
604.      void  AffiNombre::affiche_valeur(uint16_t  valeur,  uint8_t  nb_chiffres,
   char txt_unite_i[3])
605.      {
606.            for (int i=0; i<2; i++) {txt_unite[i]=txt_unite_i[i];}
607.            txt_unite[2]='\0';  // zero terminal
608.
609.            TFT480.setColor(220,220,0);
610.            TFT480.setFont(BigFont);
611.            TFT480.printNumI(valeur, x0+5,y0+20, nb_chiffres, ' ');
612.            TFT480.setFont(BigFont);
613.            TFT480.print(txt_unite, x0+80,y0+20); // ex : mm, kHz, etc...
614.      //     TFT480.print("ab", x0+80,y0+20); // ex : mm, kHz, etc...
615.      }
616.
617.
618.      void AffiNombre::affiche_HEXA(uint32_t valeur)
619.      {
620.      // affiche un nombre en representation hexadécimale
621.      // 16 nb_signes hexa max
622.
623.            uint8_t r;
624.            uint8_t i;
625.
626.            char tbl[9];
627.            char signes[17] = "0123456789ABCDEF";
628.
629.            for (i=0; i<8; i++)
630.            {
631.                  r= valeur % 16; // modulo (reste de la division)
632.                  valeur /= 16; // quotient
633.                  tbl[7-i]=signes[r];
634.            };
635.            tbl[8]='\0';
636.
637.            TFT480.setColor(0,255,255);
638.            TFT480.setFont(SmallFont);
639.            TFT480.print(tbl, x0+10,y0+15);
640.      }
641.
642.
643.
```

```
644.      /**
   ********************************************************************************
   ***
645.                                                CLASS LED

646.      ********************************************************************************
   **************/
647.      // Constructeur
648.      LED::LED()
649.      {
650.
651.      }
652.
653.
654.      void LED::init(uint16_t x_i, uint16_t y_i, uint8_t R_i, uint8_t G_i,
   uint8_t B_i, char txt_etiquette[5])
655.      {
656.            x=x_i;
657.            y=y_i;
658.
659.            R=R_i;
660.            G=G_i;
661.            B=B_i;
662.
663.            TFT480.setColor(40,40,40); // gris
664.            TFT480.fillCircle(x, y, 10); // dessine la led éteinte (cercle
   plein grisé)
665.
666.            TFT480.setColor(180,180,180);
667.            TFT480.setFont(BigFont);
668.            TFT480.print(txt_etiquette, x+15, y-8); // Etiquette
669.      }
670.
671.
672.
673.      void LED::setEtat(uint8_t etat_i)
674.      {
675.            if(etat_i == 0)
676.            {
677.                  TFT480.setColor(40,40,40); // gris
678.                  TFT480.fillCircle(x, y, 10);
679.            }
680.            if(etat_i == 1)
681.            {
682.                  TFT480.setColor(R,G,B); // couleur choisie
```

```
683.                    TFT480.fillCircle(x, y, 10);
684.             }
685.      }
```

Plik GeneUHF_ADF4351.h

```
1.
2.  #ifndef GENEUHF_ADF4351_H
3.  #define GENEUHF_ADF4351_H
4.
5.  #include <stdint.h>
6.
7.
8.  class AffiNombre
9.  {
10. public:
11.
12.       uint16_t x0;
13.       uint16_t y0;
14.
15.       AffiNombre();
16.
17.       void  init(uint16_t  x,  uint16_t  y,  uint16_t  dx,  uint16_t  dy,char
    txt_etiquette[10]);
18. //     void setTexte(char[3]);
19.       void affiche_frequence(uint32_t F_i, uint8_t R, uint8_t G, uint8_t B);
20.       void   affiche_valeur(uint16_t   valeur,   uint8_t   nb_chiffres,   char
    txt_unite_i[3]);
21.       void affiche_HEXA(uint32_t valeur);
22.
23.
24. private:
25.       char txt_etiquette[10];
26.       char txt_unite[3];
27.
28. };
29.
30.
31. class LED
32. {
33. public:
34.
35.       uint16_t x;
36.       uint16_t y;
37.
```

```
38.       uint8_t R;
39.       uint8_t G;
40.       uint8_t B;
41.
42.  LED();
43.
44.       void init(uint16_t x_i, uint16_t y_i, uint8_t R_i, uint8_t G_i, uint8_t
   B_i, char txt_etiquette[5]);
45.       void setEtat(uint8_t etat_i);  // = 0 ou 1
46.
47. private:
48.       char txt_etiquette[5];
49.
50. };
51.
52.
53. /***** autres declarations ******/
54.
55. void init_ports (void);
56. void init_variables(void);
57. void trace_ecran();
58. void trace_entete();
59. void TFT_affiche_chiffre_7seg(uint8_t num);
60. void TFT_aff_nb_form3 (uint32_t valeur, uint8_t nb_chiffres, uint8_t curseur,
   uint8_t R, uint8_t G, uint8_t B);
61. void TFT_aff_nb_form3b (uint32_t valeur, uint8_t nb_chiffres, uint8_t curseur,
   uint8_t R, uint8_t G, uint8_t B);
62. void TFT_aff_nb_form3c (uint32_t valeur, uint8_t nb_chiffres, uint8_t curseur,
   uint8_t R, uint8_t G, uint8_t B);
63.
64.
65. void calcul_lambda();
66. void affiche_leds();
67.
68.
69. #endif
70.
```

10 La klasa ADF4351 wersja 1

Kod źródłowy ec C

```
1. /**
```

```
2.        ADF4351-628v2.cpp
3.        Firmware pour piloter un circuit DDS : ADF4351
4.        avec un AVR atmega2560
5.  **/
6.
7.  /*
8.        Ce fichier source "GeneUHF_ADF4351.cpp" est libre, vous pouvez le
    redistribuer et/ou le modifier selon
9.        les termes de la Licence Publique Générale GNU.
10.
11.       Un grand merci à Alain Fort F1CJN feb 2,2016
12.       pour son travail :
13.
14.       "ADF4251 and Arduino
15.       update march 7, 2016 (ROBOT V1.1 and V1.0)"
16.       -http://f6kbf.free.fr/html/ADF4351%20and%20Arduino_Fr_Gb.htm
17.       -http://f6kbf.free.fr/html/ADF4351_LCD_07032016.zip
18.
19.       dont je me suis servi pour créer cette objet (class) en C++
20.       J'ai aussi utilisé mon propre travail effectué pour les DDS AD9850 et
    AD9951
21.
22.       Silicium628 - Juillet 2018
23.
24. */
25.
26. #include <avr/io.h>
27. #include <util/delay.h>
28. #include "ADF4351-628v2.h"
29.
30. #define bitSet(value, bit) ((value) |= (1UL << (bit)))   // 1UL signifie la
    valeur 1 au format Unsigned Long
31. #define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
32. #define bitWrite(value, bit, bitvalue) (bitvalue ? bitSet(value, bit) :
    bitClear(value, bit))
33.
34. // Constructeur
35. ADF4351::ADF4351() { }
36.
37.
38. void ADF4351::init()
39. {
40.       set_LE();
41.       Write_All_Register();
42.       PFDRFout=25;
```

```
43.       RFint=7000;
44.       RFout = RFint/100 ; // fréquence de sortie
45.       OutputChannelSpacing = 0.01; // Pas de fréquence = 10kHz
46. }
47.
48.
49. void ADF4351::mute(uint8_t valeur)
50. {
51.       if (valeur == 1)   {port_ADF4351 &= ~pin_CE;} else {port_ADF4351 |=
    pin_CE;}
52. }
53.
54.
55.
56. void ADF4351::impulse_clk()
57. {
58.  //_delay_us(1);  // temps necessaire à la stabilisation du niveau envoyé
59.   port_ADF4351 |=  pin_CLK;
60.  _delay_us(1);
61.   port_ADF4351 &= ~pin_CLK; // l'operateur "~" donne le complement (inverse des
    bits). ne pas confondre avec l'opérateur "!"
62.  _delay_us(1);
63. }
64.
65.
66. void ADF4351::impulse_LE()
67. {
68. // _delay_us(10);
69.   port_ADF4351 |=  pin_LE;
70.  _delay_us(1);
71.   port_ADF4351 &= ~pin_LE;
72.  _delay_us(1);
73. }
74.
75.
76. void ADF4351::set_LE()
77. {
78.   port_ADF4351 |=  pin_LE;
79. }
80.
81.
82. void ADF4351::reset_LE()
83. {
84.   port_ADF4351 &= ~pin_LE;
85. }
```

```
86.
87.
88. void ADF4351::set_DATA()
89. {
90.   port_ADF4351 |=  pin_DATA;
91. }
92.
93.
94. void ADF4351::reset_DATA()
95. {
96.   port_ADF4351 &= ~pin_DATA;
97. }
98.
99.
100.
101.      void ADF4351::out1octet(uint8_t octet_i)
102.      {
103.      // MSB first (= mode par defaut)
104.            uint8_t i;
105.        uint8_t masque;
106.
107.            for (i=0;  i < 8; i++)
108.            {
109.                  masque = 0b10000000 >> i; // le '1' se deplace de gauche a
    droite
110.                  if ( (octet_i & masque) != 0) { port_ADF4351 |=  pin_DATA; }
           else { port_ADF4351 &= ~pin_DATA; }
111.                  impulse_clk();
112.            }
113.      }
114.
115.
116.
117.      void  ADF4351::Write_Register32(const  uint32_t  data)     //Programme  un
    registre 32bits
118.      {
119.            uint8_t n; // numéro du registre (0..5) codé par les trois bits de
    poids faible
120.            uint8_t octet_i;
121.
122.            n = data && 0b00000111;
123.            if (data != memo_registers[n]) // on ne programme le registre que
    si son contenu doit changer. (cela accelère le programme)
124.            {
125.                  memo_registers[n] = data;
```

```
126.                    reset_LE();
127.                    for (int i = 3; i >= 0; i--) // boucle sur 4 x 8bits
128.                    {
129.                           octet_i=(data >> 8 * i) & 0xFF; // décalage et
    masquage de l'octet
130.                           out1octet(octet_i);
131.                    }
132.                    impulse_LE();
133.      // LE = Load Enable. When LE goes high, the data stored in the 32-bit
    shift register is loaded
134.      // into the register that is selected by the three control bits (ce sont
    les trois bits de poids faible)
135.             }
136.      }
137.
138.
139.      void ADF4351::Write_All_Register()   // Programme tous les registres de
    l'ADF4351
140.      {
141.             for (int i = 5; i >= 0; i--)   // programmation ADF4351 en
    commencant par R5
142.             {
143.                    Write_Register32(registers[i]);
144.             }
145.      }
146.
147.
148.      void ADF4351::setFreq(double frequence)
149.      {
150.        RFout = frequence;
151.
152.             registers[4] = 0x8C803C; // cette valeur n'est pas arbitraire. En
    particulier les bits 20,21,22 doivent être =0.
153.      // ce qui est le cas : 0x8C803C = 0b 1000 1100 1000 0000 0011 1100
154.             if (RFout >= 2200)               { DIV = 1; }
155.             else if (RFout >= 1100)  { DIV = 2;           registers[4]      +=
    (1UL<<20); } // 1UL signifie la valeur 1 au format Unsigned Long en memoire
    (uint32_t -> 4octets = 32 bits)
156.             else if (RFout >= 550)   { DIV = 4;           registers[4]      +=
    (2UL<<20); }
157.             else if (RFout >= 275)   { DIV = 8;           registers[4]      +=
    (3UL<<20); }
158.             else if (RFout >= 137.5) { DIV = 16;          registers[4]      +=
    (4UL<<20); }
```

```
159.            else if (RFout >= 68.75) { DIV = 32;          registers[4]      +=
   (5UL<<20); }
160.
         else  { DIV = 64;          registers[4] += (6UL<<20); }
161.
162.            INTA = (RFout * DIV) / PFDRFout;
163.            MOD = (PFDRFout / OutputChannelSpacing);
164.            FRACF = (((RFout * DIV) / PFDRFout) - INTA) * MOD;
165.            FRAC = round(FRACF); // On arrondit le résultat
166.
167.            registers[0] = 0;
168.            registers[0]  =  INTA  <<  15;  //  décalage  de  15  rangs  pour  ranger
   cette valeur (16 bits) à sa place dans R0 ; voir p:15 du pdf
169.            registers[0] = registers[0] + (FRAC << 3); // decalage de 3 rangs
   vers la gauche pour le rentrer dans les bits [DB3..DB14]
170.
171.            registers[1] = 0;
172.            registers[1] = MOD << 3; // decalage de 3 pour le rentrer dans les
   bits [DB3..DB14]
173.            registers[1] = registers[1] + 1 ; // ajout de l'adresse "001"
174.            bitSet (registers[1], 27); // PR1-> Prescaler = 8/9 ; voir p:16 du
   pdf
175.
176.            bitSet (registers[2], 28); // Digital lock == "110" sur b28 b27 b26
177.            bitSet (registers[2], 27); // digital lock
178.            bitClear (registers[2], 26); // digital lock
179.
180.      // bitSet (registers[4], 10); //MTLD: Supply current to the RF output
   stage to be shut down until the part achieves lock
181.
182.            Write_All_Register();  // Programme tous les registres de l'ADF4351
183.      }
184.
185.
186.
187.
```

Plik ADF4351-628v2.h

Kod źródłowy en C

```
1. /**
2.        ADF4351-628v2.h
3.
4.        Firmware pour piloter un circuit DDS : ADF4351
5.        avec un AVR atmega2560
```

```
6.
7.  **/
8.
9.
10. #ifndef      ADF4351_628V2_H
11. #define      ADF4351_628V2_H
12.
13. #pragma      once
14.
15. #define      port_ADF4351 PORTF  // sur portF
16. #define      port_PIN_ADF4351 PINF
17.
18. #define      pin_CLK                   0b10000000          //    (sortie)    sur
    portF
19. #define      pin_DATA            0b01000000          // (sortie) sur portF
20. #define      pin_LE              0b00100000          // (sortie) sur portF
21. #define      pin_MUXOUT    0b00010000          // (*ENTREE*) sur PINF
22. #define      pin_CE              0b00001000          // (sortie) sur portF
23.
24. class ADF4351
25. {
26.
27. public:
28.
29.       uint32_t  registers[6]  =  {0x4580A8,  0x80080C9,  0x4E42,  0x4B3,  0xBC803C,
    0x580005};
30.       uint32_t memo_registers[6] = {0, 0, 0, 0, 0, 0};
31.
32.       uint32_t RFint, RFintold, INTA, RFcalc, PDRFout, MOD, FRAC;
33.
34.       double RFout, REFin, PFDRFout, OutputChannelSpacing, FRACF;
35.
36.       double RFoutMin = 35; // 35 MHz
37.       double RFoutMax = 4400; // 4400MHz = 4.4 GHz
38.       double REFinMax = 250;
39.       double PDFMax = 32;
40.
41.       uint8_t DIV;
42.       uint8_t lock=2;
43.
44.       ADF4351();
45.
46.       void init();
47.
48.       void reset_LE();
```

```
49.       void set_LE();
50.       void reset_DATA();
51.       void set_DATA();
52.       void mute(uint8_t valeur);
53.       void impulse_clk();
54.       void impulse_LE();
55.       void out1octet(uint8_t octet_i);
56.       void  Write_Register32(const  uint32_t  data);     //Programme  un  registre
   32bits
57.       void Write_All_Register(); // Programme tous les registres de l'ADF4351
58.       void setFreq(double frequence);  // en MHz (avec decimales)
59.
60. private:
61.
62.       bool _powerdown, _auxEnabled, _rfEnabled, _feedbackType;
63.
64. };
65.
66. #endif
67.
```

Wielkie podziękowania dla Alain Fort F1CJN luty 2,2016

za jego pracę:

„ADF4251 i Arduino

aktualizacja 7 marca 2016 r. (ROBOT V1.1 i V1.0) ”

-http: //f6kbf.free.fr/html/ADF4351%20and%20Arduino_Fr_Gb.htm

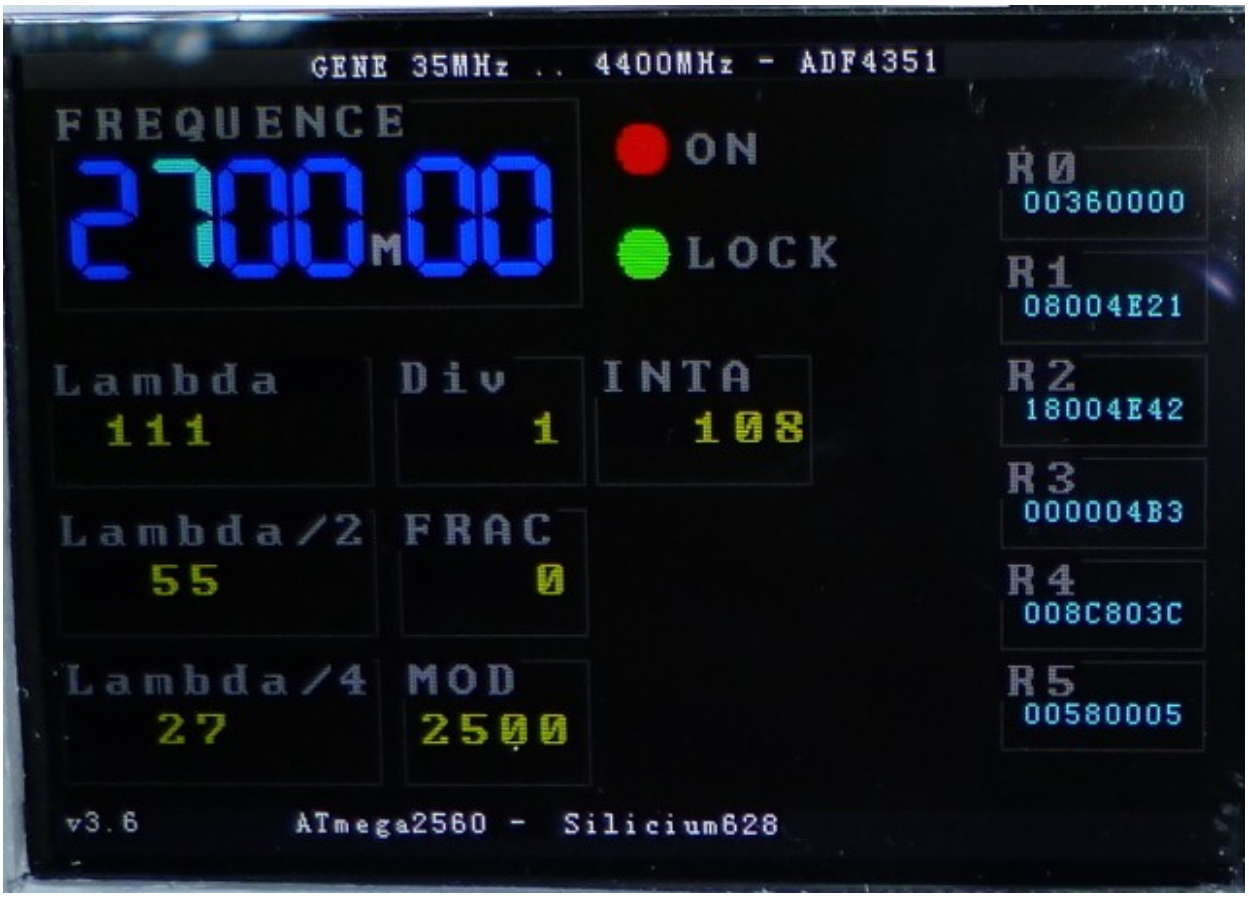
-http: //f6kbf.free.fr/html/ADF4351_LCD_07032016.zip

którego użyłem do utworzenia tego obiektu (klasy) w C ++

Korzystałem również z własnej pracy wykonanej dla DDS AD9850 i AD9951

Krzem 628 - lipiec 2018

11 Szczegół wyświetlacza



Są wyświetlane:

Częstotliwość

Odpowiednia długość fali lambda

Lambda / 2 i lambda / 4

Główne parametry przesłane do ADF4351 (DIV, FRAC, MOD, INTA)

Zawartość sześciu rejestrów ADF4351 jest wyświetlana w systemie szesnastkowym

Wszystko aktualizowane oczywiście w czasie rzeczywistym.

Podświetlona cyfra (dla częstotliwości) jest tą, która będzie zwiększana / zmniejszana przez górny koder przyrostowy.

dolny koder służy do wyboru cyfry, która ma zostać zmodyfikowana.

12 Co planuję zrobić dalej?

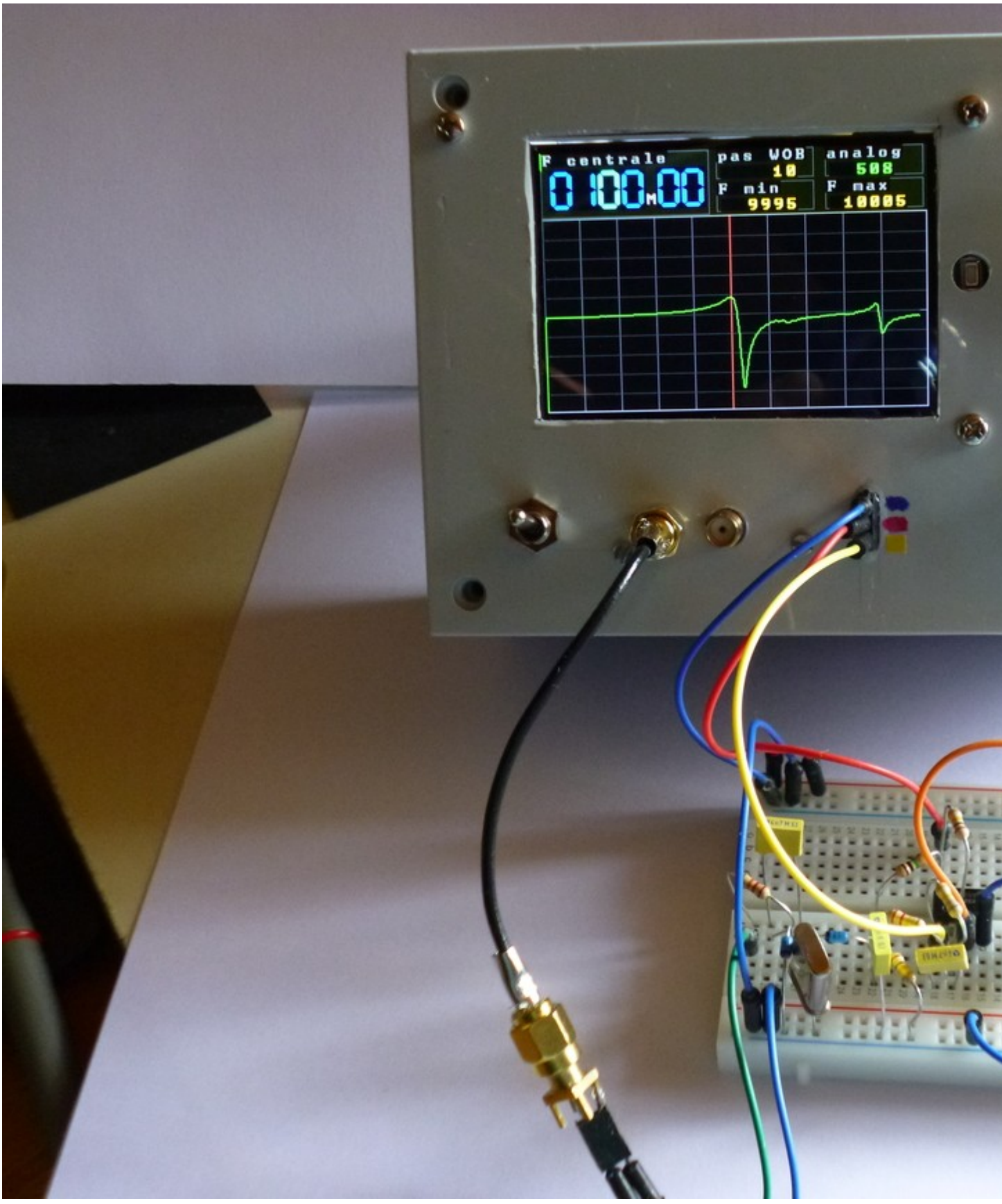
19 lipca 2018: Mamy tutaj generator częstotliwości o bardzo szerokim zakresie i

posiadający własny wyświetlacz w wysokiej rozdzielczości i kolorze. Możliwe jest zatem stworzenie autonomicznego wobuloskopu, tj. Wobulatora z własnym wyświetlaczem do wykreślania krzywej charakterystyki częstotliwościowej filtra, ponadto UHF.

Należy jednak wziąć pod uwagę, że obwód ADF4351 zapewnia sinusoidalny sygnał trybu podstawowego tylko dla częstotliwości w zakresie od 2,2 GHz do 4,4 GHz. Niższe częstotliwości są uzyskiwane przez dzielniki logiczne, a zatem nie są czystym przebiegiem sinusoidalnym, ale „kwadratowym”.

W rezultacie osiągalne będzie po prostu wobuloskop dla zakresu 2 GHz -> 4 GHz, który nie jest tak zły i bardzo wygodny w eksperymentowaniu z częstotliwością 2,4 GHz, na przykład dla wielu aplikacji (WiFi, Bluetooth, zdalne sterowanie RC, radioamatorzy ...)

13 Co zrobiłem dalej: WOBULATOR

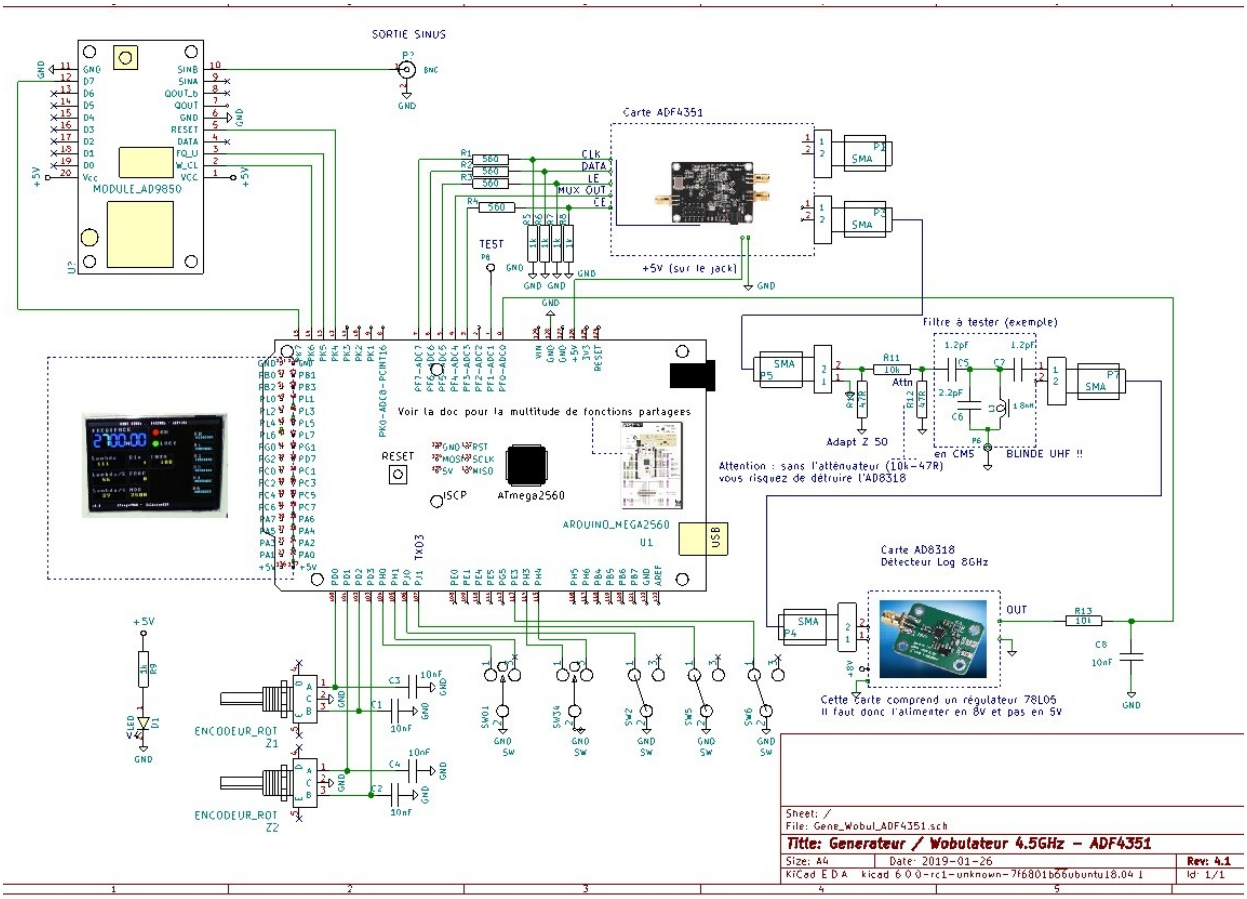

F centrale
0 100.00
pas WOB
10
analog
508
F min
9995
F max
10005

20 lipca 2018: To jest to samo urządzenie: prosty falownik (ten po lewej) umożliwia przejście z prostego trybu generatora UHF do trybu wobulatora (tj. Generator z częstotliwością przesuwną (przez małe skoki) i wyświetlanie krzywa odpowiedzi uzyskana w czasie rzeczywistym). Wszystkie autonomiczne (nie ma potrzeby używania komputera lub laptopa lub smartfona ...)

Część analogowa (prostownik diodowy + wzmacniacz) łączy się z generatorem za pomocą prostego mikro-gniazda z trzema stykami o podziałce 1/10 cala (2,54 mm) (GND, + 5 V i sygnał o bardzo niskiej częstotliwości). Sygnał UHF pożycza złącze SMA tak, jak powinno.

Jak widać na zdjęciu, część analogowa nie jest jeszcze zaimplementowana na płytce drukowanej, nie będzie długa ... Ale programowanie aplikacji jest zakończone, jest w 100% funkcjonalne.

14 Nowy schemat, gen UHF + wobulator

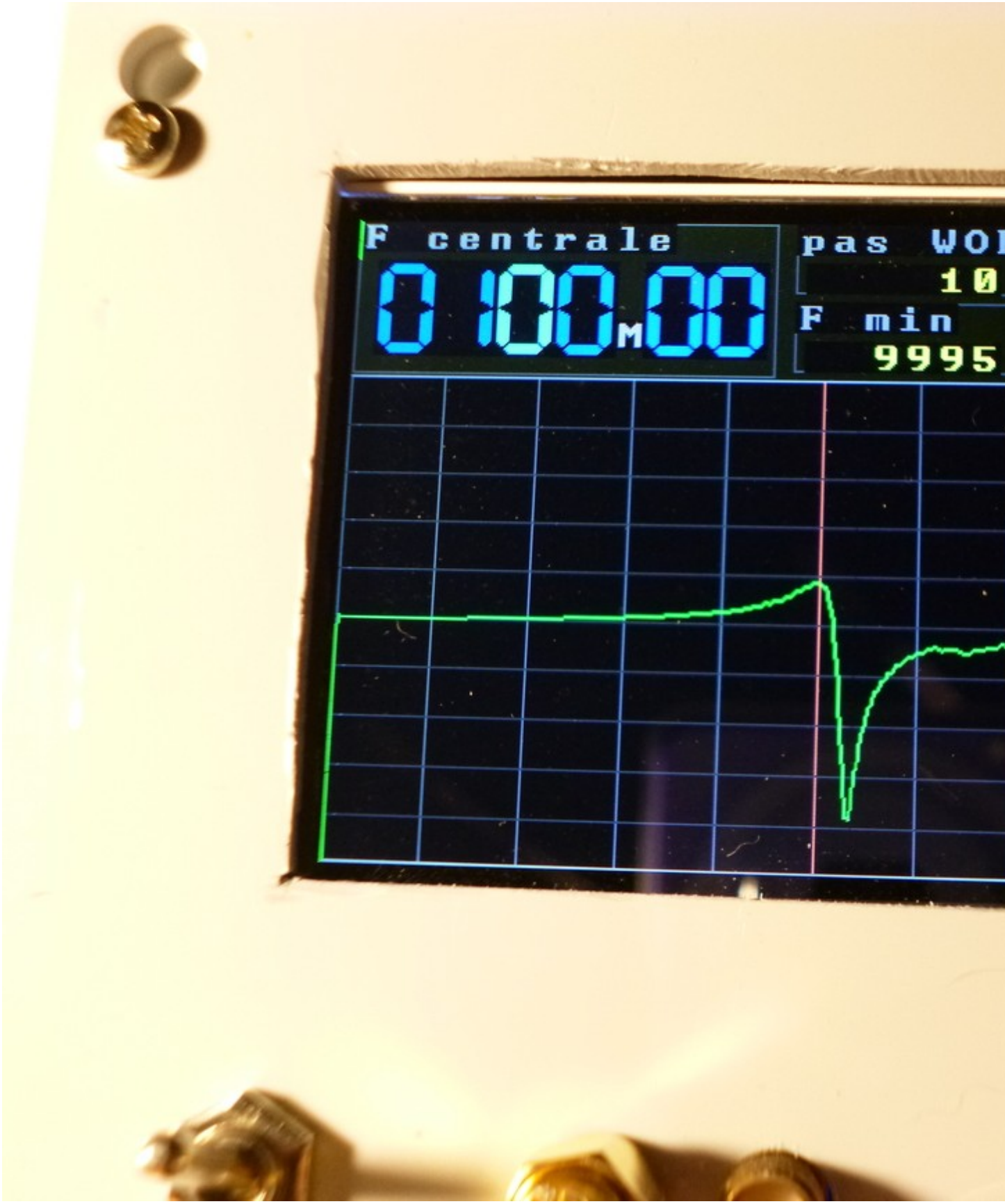


Część analogowa, która zawiera dwie diody UHF do wykrywania amplitudy sygnału, jest teraz wyposażona w logarytmiczny detektor obwiedni, który może pracować do 8 GHz. Komórka RC trochę wygładza sygnał, zanim zaatakuje ADC ATmega2560.

Detektor AD8318 składa się z kilku kaskadowych wzmacniaczy SHF. Może więc oferować dynamikę 70 dB z rozdzielczością 1 dB (patrz arkusz danych). Ale uwaga, nie jest to kwestia zastosowania sygnału 70dBm lub spawania łukowego !!!

Jego zasięg działania wynosi -60dBm (tak, z wyprzedzeniem) do 0dBm. Wystarczy powiedzieć, że musimy starannie chronić wszystkie części UHF, aby nie uzyskać odbiornika TNT, GSM, 3G, 4G, WIFI, BlueTooth, nie pasożytniczych zasilaczy, zasilania niektórych lamp LED 230V itp. ...

15 Odpowiedź częstotliwościowa kwarcu 100 MHz


F centrale
0 100.00
M
pas
10
F min
9995

Czerwona linia w środku wizjera wskazuje częstotliwość 100,00 MHz

„Skok WOB = 10” oznacza 10 kHz / kwadrat. (tak, kiloherc, nie megaherc!)

Całkowita szerokość wykresu przedstawia 100kHz.

Wszystkie te wartości można modyfikować za pomocą enkoderów obrotowych (częstotliwość środkowa i krok).

Uwaga: Drugi rezonans po prawej jest prawdopodobnie obrazem widma spowodowanym przez niepożądaną harmoniczną w pobliżu nośnika, która podczas przemiatania przechodzi z częstotliwością 100,00 MHz. Zidentyfikowałem te harmoniczne od samego początku i doszedłem do wniosku, że wobulator będzie niezawodny tylko w zakresie od 2 GHz do 4 GHz, dla którego sygnał wyjściowy jest sinusoidalny, bezpośrednio z VCO ADF4351. W każdym razie będziemy musieli poeksperymentować trochę więcej, zanim cokolwiek skończymy.

21 lipca 2018 r .: Po zbadaniu sygnału 100 MHz do analizatora widma nie wykryłem najmniejszego fałszywego sygnału o częstotliwości 40 kHz od nośnika. Czy powinniśmy wyciągnąć wniosek, że obserwujemy rzeczywisty rezonans wtórny kwarcu przy 100,040 MHz? Należy jednak zauważyć, że jesteśmy blisko granicy rozdzielczości spektralnej analogowego analizatora HM5010 ... To trochę jak widzenie planety pozasłonecznej w pobliżu jej gwiazdy ... Jak to, co robi nasz wobulator znacznie lepiej niż „prawdziwy” analizator widma! (nieco z punktu widzenia funkcji HM5010, to prawda).

23 lipca 2018 r .: Jeśli ktoś popycha badania wzdłuż większego zakresu częstotliwości, zdaje sobie sprawę, że kwarc 100 MHz przedstawia rezonanse dla następujących częstotliwości:

- 60 MHz
- 100 MHz
- 140 MHz
- 180 MHz

malejących amplitud wraz ze wzrostem częstotliwości. Jest to więc kwarcowy „wydźwięk” dotyczący częstotliwości zaznaczonej powyżej. Częstotliwości rezonansowe są oddalone od siebie o 20 MHz i są po prostu nieparzystymi (wielokrotnymi) harmonicznymi częstotliwości podstawowej = 20 MHz.

- 60 MHz = 20 x 3
- 100 MHz = 20 x 5
- 140 MHz = 20 x 7
- 180 MHz = 20 x 9

Wszystko to należało się spodziewać, częstotliwości kwarcowe> 30 MHz są w zasadzie podtekstami, w przeciwieństwie do rezonatora fali powierzchniowej (SAW dla Surface Acoustic Wave).

Uwaga:

Dlaczego kwarc rezonuje elektrycznie na nieparzystych wielokrotnościach jego częstotliwości podstawowej, a nie (także) na wielu parach?

Gdy kwarc jest elektrycznie nakłaniany, ustalany jest stacjonarny mechaniczny układ fal, który wibruje. Pojawiają się brzuchy i węzły wibracji. Dzięki swojej budowie cząsteczki tworzące kwarc są piezoelektryczne, to znaczy wykazują przestrzenną dysymetrię swoich wewnętrznych ładunków elektrycznych, gdy ulegają mechanicznemu naprężeniu (ciśnieniu). Wszystkie te ładunki sumują się wzdłuż osi oscylacji, tak że wypadkowa między potencjałem elektrycznym odpowiadającym brzuchowi i odpowiadająca węzłowi jest zasadniczo asymetryczna. Napięcie na zaciskach, to znaczy między powierzchniami, jest maksymalne, gdy odległość między tymi powierzchniami jest wielokrotnością tego, co oddziela węzeł brzucha. Dwa węzły lub dwa brzuchy oddalone są o połowę długości fali. A ta oddzielająca brzuch od węzła jest warta lambda / 4. (Zobacz moją prezentację TUTAJ.)

Rezonans elektryczny (maksymalne napięcie AC na zaciskach) występuje zatem, gdy grubość kwarcu wynosi lambda / 4 lub wielokrotność lambda / 4. Co jest napisane:

E = (2n + 1) * lambda / 4

(2n + 1) jest nieparzystą liczbą całkowitą (1, 3, 5, 7 ...)

skąd:

lambda = 4E / (2n + 1)

F = v / lambda = (2n + 1) v / 4E

z v: prędkość propagacji fali „akustycznej” w kwarcu

F = v / 4E ... 3V / 4E ... 5V / 4E ...

wywołując F0 = v / 4E najniższą z tych częstotliwości, nazywaną częstotliwością podstawową, uzyskujemy sekwencję harmonicznych:

F = Fo ... 3Fo ... 5Fo ... 7Fo ... itd ...

Harmoniczne parzystej rangi po prostu nie pokazują różnicy potencjałów między powierzchniami (rozkład ładunków jest wtedy symetryczny), więc nie ma wykrywalnego efektu piezoelektrycznego.

Tak, ale nadal nie wyjaśnia małej drugorzędnej (-ych) rezonansu (-ów) przy 40 kHz głównego.

16 Wersja programu géné + wobulateur

Kod źródłowy en C

```
1. /*
2. Firmware pour piloter:
3. - une carte générateur de signaux ADF4351
4. - une carte géné AD9850
5. - une carte Uno Mega2560
6. - afficheur 480x320 non tactile.
7.
8. par Silicium628.
9.
10. CE fichier source "Gene_Wobul_double_ADF4351.cpp" est libre, vous pouvez le redistribuer et/ou le modifier selon
11. les termes de la Licence Publique Générale GNU.
12.
13. En ce qui concerne certaines bibliothèques "incluses" (par la directive "#include"), issues du domaine "Arduino", il faut voir au cas par cas.
14. En particulier la library "UTFT" n'est pas libre, il y a un copyright dans l'entête, que je cite:
15. "UTFT.h - Arduino/chipKit library support for Color TFT LCD Boards  Copyright (C)2010-2014 Henning Karlsen. All right reserved"
16. Fin de citation.
17. */
18.
19.
20. //================================
21. #define version "v10.2"
22. //================================
23.
24. #include "Gene_Wobul_ADF4351_v10_2.h"
25.
26.
27. #include <avr/io.h>
28. #include <stdint.h>
```

```
#include <stdlib.h>
#include <util/delay.h>
#include <math.h>

#include "UTFT.cpp"
#include "uart2560_628.c"
#include "ADF4351-628v5.h";
#include "AD9850-628v1.h"
#include <EEPROM.h>

#define portPIN_switch0 PINH
#define portPIN_switch1 PINH
#define portPIN_switch2 PINJ
#define portPIN_switch3 PINH
#define portPIN_switch4 PINH
#define portPIN_switch5 PINJ
#define portPIN_switch6 PINE

#define pin_rot0 0b00000100
#define pin_rot1 0b00001000

#define pin_switch0 0b00000001
#define pin_switch1 0b00000010
#define pin_switch2 0b00000001
#define pin_switch3 0b00001000
#define pin_switch4 0b00010000
#define pin_switch5 0b00000010
#define pin_switch6 0b00001000


#define Crt_AD9850  1
#define Crt_ADF4351 2

#define mode_GENE          1
#define mode_WOBU_dB       2
#define mode_WOBU_mV       3

#define md_saisie_frq 1
#define md_saisie_Hz_div 2
#define md_saisie_dB_div 3


extern uint8_t SmallFont[];
extern uint8_t BigFont[];
extern uint8_t SevenSegNumFont[];
```

```
74.
75.
76. UTFT TFT480(HX8357C,38,39,40,41);
77.
78. AD9850 CarteAD9850;
79. ADF4351 CarteADF4351;
80.
81. AffiRect Affi_1a; // fréquence AD9850
82. AffiRect Affi_1b; // fréquence ADF4351
83. AffiRect Affi_1c; // afficheur unique pour la fréquence en mode wobulation
84.
85. AffiRect Affi_2a;   // lambda AD9850
86. AffiRect Affi_2b;   // lambda ADF4351
87.
88. AffiRect Affi_3a;   // lambda/2 AD9850
89. AffiRect Affi_3b;   // lambda/2 ADF4351
90.
91. AffiRect Affi_4a;   // lambda/4 AD9850
92. AffiRect Affi_4b;   // lambda/4 ADF4351
93.
94. AffiRect Affi_5;
95. AffiRect Affi_6;
96. AffiRect Affi_7;
97. AffiRect Affi_8;
98. AffiRect Affi_9;
99. AffiRect Affi_10;
100.      AffiRect Affi_11;
101.      AffiRect Affi_12;
102.      AffiRect Affi_13;
103.      AffiRect Affi_14;
104.
105.      AffiRect Affi_R0;
106.      AffiRect Affi_R1;
107.      AffiRect Affi_R2;
108.      AffiRect Affi_R3;
109.      AffiRect Affi_R4;
110.      AffiRect Affi_R5;
111.
112.      AffiV AffiV_01;
113.      AffiV AffiV_02;
114.
115.      LED LED_a;   // "LOCK" pour l'ADF4351
116.      LED LED_b1; // "ON" pour l'AD9850
117.      LED LED_b2; // "ON" pour l'ADF4351
118.
```

```
119.      LED LED0;
120.      LED LED1;
121.      LED LED2;
122.      LED LED3;
123.      LED LED4;
124.
125.      uint8_t couleur[3];
126.      uint8_t rouge[3]={255, 0, 0};
127.      uint8_t orange[3]={245, 121, 0};
128.      uint8_t jaune[3]={255, 255, 0};
129.      uint8_t jaune_sombre[3]={220, 220, 0};
130.      uint8_t jaune_tres_sombre[3]={100, 100, 0};
131.      uint8_t vert[3]={0, 255, 0};
132.      uint8_t cyan[3]={0, 255, 255};
133.      uint8_t cyan_sombre[3]={0, 200, 200};
134.      uint8_t bleu_pale[3]={160, 240, 255};
135.      uint8_t bleu_clair[3]={0, 100, 255};
136.      uint8_t bleu[3]={0, 0, 255};
137.      uint8_t violet[3]={200, 0, 255};
138.      uint8_t noir[3]={0, 0, 0};
139.      uint8_t gris[3]={150, 150, 150};
140.      uint8_t blanc[3]={255, 255, 255};
141.
142.      uint8_t couleur_gain[3];
143.
144.      uint8_t switches;
145.      uint8_t memo_switches;
146.
147.
148.      uint8_t SW_mode;    //  =  1..2..3    (par  lecture  d'un  inverseur  3
   positions)   1=[gene freq] ; 2=[Wobu dB] ; 3=[Wobu mV]
149.      uint8_t SW_saisie; // = 1..2..3  (par lecture d'un inverseur 3 positions)
          1=[Freq] ; 2=[MHz/div] ; 3=[dB & offset]
150.      uint8_t SW_rapide;  // = 0..1    (par lecture d'un inverseur 2 positions)
   0=[balayage normal]  ; 1=[balayage rapide]
151.      uint8_t SW_polarite;      //  =  0..1     (par  lecture  d'un  inverseur  2
   positions)   sens de variation de la tension du signal acquisition
152.      uint8_t SW_carte;   // = 1..2    (par lecture d'un inverseur 2 positions)
          1=[AD9850] ; 2=[ADF4351]
153.
154.      parametres params1;
155.      int addr_params1=0; // en EEprom
156.
157.      uint16_t x0, y0;
158.      uint16_t x_7seg, y_7seg;
```

```
159.
160.      uint32_t memo_frequence1; // fréquence de sortie de l'AD9850;
161.      uint32_t  memo_frequence2;  //  fréquence  de  sortie  de  l'ADF4351;    en
   multiples de 10 kHz
162.
163.      uint32_t frq_centrale;
164.      uint32_t frq_min_AD98, frq_max_AD98;
165.      uint32_t frq_min_ADF43, frq_max_ADF43;
166.
167.      uint32_t pas_frq_gene_AD98;
168.      uint32_t pas_frq_gene_ADF43;
169.      uint32_t pas_frq_wobu_AD98;
170.      uint32_t pas_frq_wobu_ADF43;
171.
172.      float F_mark;
173.      float lambda_a; // longueur d'onde en cm AD9850
174.      float lambda_b; // longueur d'onde en cm ADF4351
175.      float F_affi_min, F_affi_max;
176.
177.      uint8_t DIV;
178.      uint16_t FRAC;
179.      uint32_t MOD;
180.      uint32_t INTA;
181.
182.
183.
184.      uint32_t kHz_div;  // pas de wobulation
185.      uint32_t facteurs_mult[13] = {1, 2 , 5, 10, 20, 50, 100, 200, 500, 1000,
   2000, 5000, 10000};
186.
187.      uint8_t dB_div;
188.      uint32_t mV_div;
189.      int polarite = 1;
190.
191.      uint8_t etat;
192.      uint16_t compteur1;
193.      uint8_t raffraichir;
194.      uint8_t afficher_freq;
195.
196.      uint8_t envoyer_data;
197.      uint8_t stop =0;
198.      float position;
199.
200.      uint16_t valeur_lue;  // par le convertisseur ADC
201.      uint8_t depassement;
```

```
202.
203.      uint32_t EE_adrs_freq0;
204.
205.      uint16_t tableau_acq[470];
206.
207.
208.      /**
209.      RAPPEL variables avr-gcc
210.
211.      TYPE                        nb octets
212.      char                              1   -128 .. 127 (ou caractères)
213.      unsigned char 1      0 .. 255
214.      uint8_t                           1     (c'est   la   même   chose   que
   l'affreux 'unsigned char')
215.      char toto[n]  n
216.      int                               2      -32768 .. 32767
217.      int16_t                           2      idem 'int'
218.      short int                 2      pareil que int (?)
219.      unsigned int  2      0 .. 65535
220.      uint16_t                  2      idem 'unsigned int'
221.      long int                  4 octets  -2 147 483 648 à 2 147 483 647
222.      int32_t                           4      octets -> 32 bits   ;      idem
   long int
223.      long long int 8      octets -> 64 bits
224.      unsigned long int   4      octets -> 32 bits   ;      0  ..  4  294  967  295
   (4,2 x 10^9)
225.      uint32_t                  4      32 bits      ;      idem  'unsigned  long
   int'
226.      float                             4
227.      double                    ATTENTION ! 4 octets (oui, 32 bits ! et pas 64
   bits (8 octets) comme en C standard)
228.
229.      La déclaration  char JOUR[7][9];
230.      réserve l'espace en mémoire pour 7 mots contenant 9 caractères (dont 8
   caractères significatifs).
231.      **/
232.
233.      void init_ports (void)
234.      {
235.      // 0 = entree, 1=sortie   ;   les 1 sur les pins en entrees activent les
   R de Pull Up (tirage à VCC)
236.
237.        DDRD  = 0b11110000;
238.        PORTD = 0b00001111;
239.
```

```
240.        DDRE  = 0b11110111;
241.        PORTE = 0b00001000;
242.
243.        DDRF  = 0b11101110;
244.        PORTF = 0b00000000;
245.
246.        DDRH  = 0b11100100;
247.        PORTH = 0b00011011;
248.
249.        DDRJ  = 0b11111100;
250.        PORTJ = 0b00000011;
251.
252.        DDRK  = 0b11111111;
253.        PORTK = 0b00000000;
254.
255.       // REMARQUE : les ports et pins utilisés par le circuit ADF4351 sont
    définis dans les fichiers "ADF4351-628v{x}.h"
256.      }
257.
258.
259.      /**
260.      The ADC module contains a prescaler, which generates an acceptable ADC
    clock fre-
261.      quency from any CPU frequency above 100 kHz. The prescaling is set by the
    ADPS bits
262.      in ADCSRA
263.      **/
264.
265.      void InitADC()
266.      {
267.      //SFIOR   = 0b10000000; // ADTS2,1,0 = 100 -> ADC declenché par Timer0
    Overflow - p:218
268.
269.      /** ========= REGITRE ADMUX ======= **/
270.
271.        ADMUX  = 0b11000000;
272.      //Bit 7:6 = ADC Reference Selection = 11 -> Internal 2.56V Voltage
    Reference with external capacitor at AREF pin - p:281 du datasheet ATmega2560
273.      //Bits 0:5 - Analog Channel Selection Bits et gain éventuel - p282
274.      // ici: Select pin ADC0 using MUX avec ref tension  2.56V interne - pas
    de gain.
275.
276.      /** ========= REGITRE ADCSRA ======= **/
277.      // voir p:285
```

```
278.         ADCSRA = 0b10000101;  // Activate ADC; Prescaler=f/32; bit5=0 -> Auto-
   Trigger desable
279.
280.      // bits [0..2] = ADPS -> Prescaler
281.      // ADPS = 010; Prescaler=f/4            gamme 8-90 kHz
282.      // ADPS = 011; Prescaler=f/8            gamme 4-56 kHz
283.      // ADPS = 100; Prescaler=f/16           gamme 2-31 kHz
284.      // ADPS = 101; Prescaler=f/32           gamme 1-15 kHz
285.      // ADPS = 110; Prescaler=f/64           gamme 300Hz- 8kHz
286.      // ADPS = 111; Prescaler=f/128    gamme 200Hz- 4300 Hz
287.
288.      //bit5 = ADATE (ADC Auto Trigger Enable)
289.      //Bit7 – ADEN: ADC Enable Writing this bit to one enables the ADC. By
   writing it to zero, the ADC is turned off.
290.
291.      }
292.
293.      template <class T> void EEPROM_write_Struct(int ee, const T& value)
294.      {
295.          const byte* p = (const byte*)(const void*)&value;
296.          for (int i = 0; i < sizeof(value); i++)    {    EEPROM.write(ee++,  *p+
   +); }
297.      }
298.
299.
300.      template <class T> void EEPROM_read_Struct(int ee, T& value)
301.      {
302.          byte* p = (byte*)(void*)&value;
303.           for (int i = 0; i < sizeof(value); i++) {   *p++ = EEPROM.read(ee+
   +); }
304.      }
305.
306.
307.
308.      void save_params_to_EEPROM()
309.      {
310.            EEPROM_write_Struct(addr_params1, params1);
311.
312.      }
313.
314.
315.      void load_params_from_EEPROM()
316.      {
317.            EEPROM_read_Struct(addr_params1, params1);
318.      }
```

```
319.
320.
321.      int freeRam()
322.      {
323.            extern int __heap_start, *__brkval;
324.            int v;
325.            return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int)
   __brkval);
326.      }
327.
328.
329.      void int_EXT_setup()
330.      {
331.            // voir 15. External Interrupts (ATmega2560.pdf p:109)
332.
333.            EICRA |= 0b00001010;    // bits[1,0] = 10 -> int0 sur front
   descendant;  bits[3,2] = 10 -> int1 sur front descendant;   - p:110
334.            EIMSK |= 0b00000011;  // bit[0]=1 ->  INT0 enable  ; bit[1]=1 ->
   INT1 enable  -   parag 15.2.3  EIMSK - p:111
335.      }
336.
337.
338.      void setCouleur(uint8_t *couleur_i, uint8_t R, uint8_t V,uint8_t  B)
339.      {
340.            couleur_i[0]=R;
341.            couleur_i[1]=V;
342.            couleur_i[2]=B;
343.      }
344.
345.
346.      void init_variables(void)
347.      {
348.            load_params_from_EEPROM();
349.      // penser à changer la valeur de la clé suivante après toute modif de la
   "struct parametres" pour forcer le reparamétrage
350.            if (params1.cle != 187)     // à la première programmation la clé
   ne sera pas correcte => reset bas niveau
351.            {
352.                  params1.cle = 187; // cle ok pour les fois suivantes (ainsi
   on ne repassera plus pas cette condition)
353.                  params1.offset_y =0;
354.                  params1.gain=1;
355.                  params1.pos_curFreq_gene_AD98 = 4;  // valeur de départ; on
   peut la modifier (détermine le digit en surbrillance pour la fréquence)
356.                  params1.pos_curFreq_gene_ADF43 = 4;
```

```
357.                    params1.pos_curFreq_wobu_AD98 = 4;
358.                    params1.pos_curFreq_wobu_ADF43 = 4;  // valeur de départ; on
    peut la modifier (détermine le facteur kHz/div au départ)
359.                    params1.pos_curseur_wob = 2;
360.                    params1.pos_mark = 300;
361.                    params1.frequence_AD9850 = 10000;// 1.0 kHz
362.                    params1.frequence_ADF4351 = 3500;// 35.00 MHz
363.            }
364.
365.            pas_frq_gene_AD98=1; for (int n=1; n<params1.pos_curFreq_gene_AD98;
    n++)   {      pas_frq_gene_AD98 *=10;    } // <- ne surtout pas toucher cette
    ligne de calcul auto
366.            pas_frq_gene_ADF43=1;           for           (int           n=1;
    n<params1.pos_curFreq_gene_ADF43; n++)  {      pas_frq_gene_ADF43 *=10;   } // <-
    ne surtout pas toucher cette ligne de calcul auto
367.            pas_frq_wobu_AD98=1; for (int n=1; n<params1.pos_curFreq_wobu_AD98;
    n++)   {      pas_frq_wobu_AD98 *=10;    } // <- ne surtout pas toucher cette
    ligne de calcul auto
368.            pas_frq_wobu_ADF43=1;           for           (int           n=1;
    n<params1.pos_curFreq_wobu_ADF43; n++)  {      pas_frq_wobu_ADF43 *=10;   } // <-
    ne surtout pas toucher cette ligne de calcul auto
369.
370.            if (SW_carte == Crt_AD9850) {kHz_div=1;}
371.            if (SW_carte == Crt_ADF4351) {kHz_div=10;}
372.            kHz_div *= facteurs_mult[params1.pos_curseur_wob];   //    <-    ne
    surtout pas toucher cette ligne de calcul auto
373.
374.            dB_div =1;    dB_div *= facteurs_mult[params1.gain];  //    <-    ne
    surtout pas toucher cette ligne de calcul auto
375.            mV_div =1;    mV_div *= facteurs_mult[params1.gain];  //    <-    ne
    surtout pas toucher cette ligne de calcul auto
376.            setCouleur(couleur_gain, 0, 255, 0);
377.            envoyer_data = 0;
378.
379.            frq_min_AD98 = 1; // 1 dixième de Hz -> 0.1Hz
380.            frq_max_AD98 = 50e6 * 10;  //  50  000  000  0  représentant   50  000
    000.0 Hz  soit (50e6 x 10) dixièmes de Hz -> 50.0MHz
381.            borne_frequence_AD9850();
382.
383.
384.            frq_min_ADF43 = 3300;           //    33.00   MHz   -   La   PLL   de
    l'ADF4351 décroche en dessous de 32MHz environ (voir led LOCK en mode GENE)
385.            frq_max_ADF43 = 440000;    //4400.00 MHz = 4.4 GHz
386.            borne_frequence_ADF4351();
387.
```

```
388.            compteur1 = 0;
389.            depassement = 0;
390.            raffraichir=1;
391.            afficher_freq=1;
392.      }
393.
394.
395.      void borne_frequence_AD9850()
396.      {
397.            if    (    (params1.frequence_AD9850)        <        frq_min_AD98)
   {params1.frequence_AD9850 = frq_min_AD98;}
398.            if    (    (params1.frequence_AD9850)        >        frq_max_AD98)
   {params1.frequence_AD9850 = frq_max_AD98;}
399.      }
400.
401.      void borne_frequence_ADF4351()
402.      {
403.            if    (    (params1.frequence_ADF4351)       <       frq_min_ADF43)
   {params1.frequence_ADF4351 = frq_min_ADF43;}
404.            if    (    (params1.frequence_ADF4351)       >       frq_max_ADF43)
   {params1.frequence_ADF4351 = frq_max_ADF43;}
405.      }
406.
407.
408.      void inc_FRQ_AD9850()
409.      {
410.            uint32_t pF3m = pas_frq_wobu_AD98 * 10000;
411.            if (SW_mode == mode_GENE)
412.            {
413.                  if  (  (params1.frequence_AD9850  +  pas_frq_gene_AD98)  <=
   frq_max_AD98) {params1.frequence_AD9850 += pas_frq_gene_AD98;}
414.                  borne_frequence_AD9850();
415.            }
416.            if ((SW_mode == mode_WOBU_dB) || (SW_mode == mode_WOBU_mV))
417.            {
418.                  if  (  (params1.frequence_AD9850  +  pF3m)  <=  frq_max_AD98)
   {params1.frequence_AD9850 += pF3m;}
419.                  borne_frequence_AD9850();
420.            }
421.      }
422.
423.
424.      void dec_FRQ_AD9850()
425.      {
426.            uint32_t pF3m = pas_frq_wobu_AD98 * 10000;
```

```
427.            if (SW_mode == mode_GENE)
428.            {
429.                  if    (params1.frequence_AD9850    >=    pas_frq_gene_AD98)
   {params1.frequence_AD9850 -= pas_frq_gene_AD98;}
430.                  borne_frequence_AD9850();
431.            }
432.            if ((SW_mode == mode_WOBU_dB) || (SW_mode == mode_WOBU_mV))
433.            {
434.                  if        (params1.frequence_AD9850        >=        pF3m)
   {params1.frequence_AD9850 -= pF3m;}
435.                  borne_frequence_AD9850();
436.            }
437.      }
438.
439.
440.      void inc_FRQ_ADF4351()
441.      {
442.            if (SW_mode == mode_GENE)
443.            {
444.                  if  (  (params1.frequence_ADF4351  +  pas_frq_gene_ADF43)  <=
   frq_max_ADF43) {params1.frequence_ADF4351 += pas_frq_gene_ADF43;}
445.                  borne_frequence_ADF4351();
446.            }
447.            if ((SW_mode == mode_WOBU_dB) || (SW_mode == mode_WOBU_mV))
448.            {
449.                  if  (  (params1.frequence_ADF4351  +  pas_frq_wobu_ADF43)  <=
   frq_max_ADF43) {params1.frequence_ADF4351 += pas_frq_wobu_ADF43;}
450.                  borne_frequence_ADF4351();
451.            }
452.      }
453.
454.      void dec_FRQ_ADF4351()
455.      {
456.            if (SW_mode == mode_GENE)
457.            {
458.                  if    (params1.frequence_ADF4351    >=    pas_frq_gene_ADF43)
   {params1.frequence_ADF4351 -= pas_frq_gene_ADF43;}
459.                  borne_frequence_ADF4351();
460.            }
461.            if ((SW_mode == mode_WOBU_dB) || (SW_mode == mode_WOBU_mV))
462.            {
463.                  if    (params1.frequence_ADF4351    >=    pas_frq_wobu_ADF43)
   {params1.frequence_ADF4351 -= pas_frq_wobu_ADF43;}
464.                  borne_frequence_ADF4351();
465.            }
```

```
466.      }
467.
468.
469.      void inc_offset()
470.      {
471.            params1.offset_y += 10;
472.            if ( params1.offset_y  >  2000) {params1.offset_y = 2000;}
473.            Affi_12.affiche_valeur(params1.offset_y, 5, "");
474.      }
475.
476.      void dec_offset()
477.      {
478.            params1.offset_y -= 10;
479.            if (params1.offset_y <  -2000) {params1.offset_y = -2000;}
480.            Affi_12.affiche_valeur(params1.offset_y, 5, "");
481.      }
482.
483.
484.
485.      void inc_pos_curseur_frq()
486.      {
487.            uint8_t cur_max;
488.            uint8_t n;
489.      //Rappel : SW_mode ->      1=[gene freq] ; 2=[Wobu dB] ; 3=[Wobu mV]
490.
491.            if (SW_carte == Crt_AD9850)
492.            {
493.                  if (SW_mode == mode_GENE)
494.                  {
495.                        cur_max = 9;
496.                        if    (params1.pos_curFreq_gene_AD98    <    cur_max)
   {params1.pos_curFreq_gene_AD98++ ;}
497.                        pas_frq_gene_AD98=1;
498.                        for   (n=1;   n<params1.pos_curFreq_gene_AD98;   n++)
         {     pas_frq_gene_AD98 *=10;    }
499.                  }
500.                  else   //   ((SW_mode   ==   mode_WOBU_dB)   ||   (SW_mode   ==
   mode_WOBU_mV))  // WOBULATION
501.                  {
502.                        cur_max = 5;
503.                        if    (params1.pos_curFreq_wobu_AD98    <    cur_max)
   {params1.pos_curFreq_wobu_AD98++ ;}
504.                        pas_frq_wobu_AD98=1;
505.                        for   (n=1;   n<params1.pos_curFreq_wobu_AD98;   n++)
         {     pas_frq_wobu_AD98 *=10;    }
```

```
506.                    }
507.             }
508.
509.             if (SW_carte == Crt_ADF4351)
510.             {
511.                    if (SW_mode == mode_GENE)
512.                    {
513.                           cur_max = 6;
514.                           if    (params1.pos_curFreq_gene_ADF43    <    cur_max)
    {params1.pos_curFreq_gene_ADF43++ ;}
515.                           pas_frq_gene_ADF43=1;
516.                           for   (n=1;   n<params1.pos_curFreq_gene_ADF43;   n++)
          {      pas_frq_gene_ADF43 *=10;   }
517.                    }
518.
519.                    else   //   ((SW_mode   ==   mode_WOBU_dB)   ||   (SW_mode   ==
    mode_WOBU_mV))  // WOBULATION
520.                    {
521.                           cur_max = 6;
522.                           if    (params1.pos_curFreq_wobu_ADF43    <    cur_max)
    {params1.pos_curFreq_wobu_ADF43++ ;}
523.                           pas_frq_wobu_ADF43=1;
524.                           for   (n=1;   n<params1.pos_curFreq_wobu_ADF43;   n++)
          {      pas_frq_wobu_ADF43 *=10;   }
525.                    }
526.
527.             }
528.      }
529.
530.
531.      void dec_pos_curseur_frq()
532.      {
533.             uint8_t n;
534.             if (SW_carte == Crt_AD9850)
535.             {
536.                    if (SW_mode == mode_GENE)
537.                    {
538.                           if      (params1.pos_curFreq_gene_AD98      >      1)
    {params1.pos_curFreq_gene_AD98--;}
539.                           pas_frq_gene_AD98=1;
540.                           for   (n=1;   n<params1.pos_curFreq_gene_AD98;   n++)
          {      pas_frq_gene_AD98 *=10;    }
541.                    }
542.                    if ((SW_mode == mode_WOBU_dB) || (SW_mode == mode_WOBU_mV))
    // WOBULATION
```

```
543.                    {
544.                           if      (params1.pos_curFreq_wobu_AD98      >      1)
   {params1.pos_curFreq_wobu_AD98--;}
545.                           pas_frq_wobu_AD98=1;
546.                           for    (n=1;    n<params1.pos_curFreq_wobu_AD98;    n++)
          {      pas_frq_wobu_AD98 *=10;    }
547.                    }
548.             }
549.
550.             if (SW_carte == Crt_ADF4351)
551.             {
552.                    if (SW_mode == mode_GENE) // GENE
553.                    {
554.                           if      (params1.pos_curFreq_gene_ADF43      >      1)
   {params1.pos_curFreq_gene_ADF43--;}
555.                           pas_frq_gene_ADF43=1;
556.                           for    (n=1;    n<params1.pos_curFreq_gene_ADF43;    n++)
          {      pas_frq_gene_ADF43 *=10;   }
557.                    }
558.                    if ((SW_mode == mode_WOBU_dB) || (SW_mode == mode_WOBU_mV))
   // WOBULATION
559.                    {
560.                           if      (params1.pos_curFreq_wobu_ADF43      >      1)
   {params1.pos_curFreq_wobu_ADF43--;}
561.                           pas_frq_wobu_ADF43=1;
562.                           for    (n=1;    n<params1.pos_curFreq_wobu_ADF43;    n++)
          {      pas_frq_wobu_ADF43 *=10;   }
563.                    }
564.
565.             }
566.      }
567.
568.
569.      void inc_pas_curseur_wob()
570.      {
571.             uint8_t cur_max;
572.             if (SW_carte == Crt_ADF4351) { cur_max = 12;}
573.             if (SW_carte == Crt_AD9850) { cur_max = 9;}
574.
575.             if (params1.pos_curseur_wob > cur_max) {params1.pos_curseur_wob =
   cur_max;}
576.             if  (params1.pos_curseur_wob  <  cur_max)  {params1.pos_curseur_wob+
   + ;}
577.
578.      }
```

```
579.
580.      void dec_pas_curseur_wob()
581.      {
582.            if (params1.pos_curseur_wob > 0) {params1.pos_curseur_wob--;}
583.      }
584.
585.
586.      void affiche_freq_marqueur()
587.      {
588.            Affi_10.affiche_float(F_mark, 6, 2, "");
589.            TFT480.setColor(100, 100, 100);
590.      }
591.
592.
593.      void affi_marqueur()
594.      {
595.            uint16_t memo_pos_mark;
596.            uint16_t y_i, y2;
597.            memo_pos_mark = params1.pos_mark;
598.
599.            if((memo_pos_mark % 47)==0) { TFT480.setColor(120, 120,120); } else
   { TFT480.setColor(0, 0, 0); }
600.            if(memo_pos_mark == 235) { TFT480.setColor(255, 100,100); }
601.            TFT480.drawLine(params1.pos_mark,  81,  params1.pos_mark,  318);//
   efface le marqueur avant de le deplacer (et retrace le réticule en jouant sur
   les couleurs)
602.
603.      //re-quadrillage H (juste les points qui ont été effacés)
604.            for (int i=1; i<10; i++)
605.            {
606.                  y2= 81 + 24*i;
607.                  TFT480.setColor(60, 60, 60);
608.                  TFT480.drawPixel(memo_pos_mark, y2);
609.            }
610.
611.            if (etat == 1)      {     if      (params1.pos_mark      >      1)
   {params1.pos_mark --;}     }
612.            else   {     if ( params1.pos_mark  <= 469) {params1.pos_mark ++;}
         if ( params1.pos_mark > 470) {params1.pos_mark = 470;}      }
613.            TFT480.setColor(200, 0, 255);
614.            TFT480.drawLine(params1.pos_mark,  81,  params1.pos_mark,  318);  //
   retrace le marqueur à sa nouvelle position
615.
616.            TFT480.setColor(0, 255, 0);
617.
```

```
618.            y_i=tableau_acq[memo_pos_mark];
619.            TFT480.drawPixel(memo_pos_mark, y_i);  // <===  retrace le signal
    effacé
620.
621.            affiche_freq_marqueur();
622.      }
623.
624.
625.      void affi_min_max()
626.      {
627.            TFT480.setFont(SmallFont);
628.      //RAPPEL: void      printNumF(double num, byte dec, int x, int y, char
    divider='.', int length=0, char filler=' ');
629.            TFT480.setColor(0, 180, 255);
630.            TFT480.printNumF(F_affi_min/100.0, 2, 2, 304, '.', 8, ' ');
631.            TFT480.printNumF(F_affi_max/100.0, 2, 400, 304, '.', 8, ' ');
632.      }
633.
634.
635.      ISR (INT0_vect)
636.      {
637.            cli();
638.      //interruption sur front descendant sur l'entree Int0 déclenchée par la
    rotation du codeur_ROT (1) -> FRQ
639.
640.            etat =(PIND & pin_rot0) == pin_rot0;
641.
642.            switch (SW_mode)
643.            {
644.                  case mode_GENE :
645.                  {
646.                        if(SW_carte == Crt_AD9850) {     if(etat     ==     0)
        {inc_FRQ_AD9850();} else   {dec_FRQ_AD9850();} }
647.                        if(SW_carte == Crt_ADF4351) {    if(etat     ==     0)
        {inc_FRQ_ADF4351();} else  {dec_FRQ_ADF4351();} }
648.                  }
649.                  break;
650.
651.
652.                  case mode_WOBU_dB :
653.                  case mode_WOBU_mV :
654.                  {
655.                        switch (SW_saisie)
656.                        {
657.                              case md_saisie_Hz_div :
```

```
658.                                {
659.                                       affi_marqueur();
660.                                }
661.                                break;
662.
663.                                case  md_saisie_frq :
664.                                {
665.                                       setCouleur(couleur_gain, 0, 255, 0);
666.                                       if(SW_carte == Crt_AD9850) {     if(etat
   == 0)  {inc_FRQ_AD9850();} else   {dec_FRQ_AD9850();} }
667.                                       if(SW_carte == Crt_ADF4351) {    if(etat
   == 0)  {inc_FRQ_ADF4351();} else  {dec_FRQ_ADF4351();} }
668.                                }
669.                                break;
670.
671.                                case md_saisie_dB_div :
672.                                {
673.                                       switch (SW_mode)
674.                                       {
675.                                              case mode_WOBU_dB : //saisie
   OFFSET - gain dB/div
676.                                              {
677.                                                     setCouleur(couleur_gain, 0,
   255, 255);
678.                                                     if(etat == 0){params1.gain
   += 1;} else   {params1.gain -= 1;}
679.                                                     if(params1.gain<0)
   {params1.gain=0;}
680.                                                     if(params1.gain>3)
   {params1.gain=3;}
681.                                                     dB_div =1;
682.                                                     dB_div                   *=
   facteurs_mult[params1.gain];
683.
          Affi_11.affiche_valeur(dB_div, 2, "");
684.                                              }
685.                                              break;
686.
687.
688.                                              case mode_WOBU_mV : //saisie
   OFFSET - gain mV/div
689.                                              {
690.                                                     setCouleur(couleur_gain, 0,
   255, 255);
```

```
691.                                                    if(etat == 1){params1.gain
   += 1;} else  {params1.gain -= 1;}
692.                                                    if(params1.gain<3)
   {params1.gain=3;}
693.                                                    if(params1.gain>12)
   {params1.gain=12;}
694.                                                    mV_div =1;
695.                                                    mV_div                  *=
   facteurs_mult[params1.gain];
696.                                                    if (params1.gain == 3)
697.                                                    {
698.
          Affi_11.setCouleurTxt(bleu);
699.
          Affi_11.affiche_texte("min");
700.                                                           _delay_ms(300);
701.                                                    }
702.
          Affi_11.setCouleurTxt(jaune);
703.                                                    affiche_V_div();
704.                                              }
705.                                              break;
706.                                        }
707.                                        break;
708.                                  }
709.                                  break;
710.                            }
711.                            break;
712.                      }
713.              }
714.              envoyer_data = 1;
715.              stop =1;
716.              //raffraichir=1;
717.              afficher_freq=1;
718.              sei();
719.      }
720.
721.
722.
723.      ISR (INT1_vect)
724.      {
725.              cli();
726.      //interruption sur front descendant sur l'entree Int1 déclenchée par la
   rotation du codeur_ROT (2)
727.      //     uint8_t n;
```

```
728.            etat =(PIND & pin_rot1) == pin_rot1;
729.
730.            if (SW_mode == mode_GENE)
731.            {
732.                  if(etat == 1){     inc_pos_curseur_frq();     }     else
        {     dec_pos_curseur_frq();}
733.                  envoyer_data = 1;
734.            }
735.            else // MODES WOBULATEUR
736.            {
737.                  switch (SW_saisie)
738.                  {
739.                        case md_saisie_dB_div :
740.                              if(etat == 0){     inc_offset();} else {
        dec_offset();}
741.                        break;
742.
743.
744.                        case md_saisie_Hz_div :
745.                        {
746.                              if(etat == 0){inc_pas_curseur_wob();}     else
        {dec_pas_curseur_wob();}
747.                              if (SW_carte == Crt_AD9850) {kHz_div=1;}
748.                              if (SW_carte == Crt_ADF4351) {kHz_div=10;}
749.                              kHz_div                                    *=
   facteurs_mult[params1.pos_curseur_wob];
750.
751.
752.                              if (params1.pos_curseur_wob == 0)
753.                              {
754.                                    Affi_9.setCouleurTxt(bleu);
755.                                    Affi_9.affiche_texte("min");
756.                                    _delay_ms(300);
757.                              }
758.                              if (params1.pos_curseur_wob == 12)
759.                              {
760.                                    Affi_9.setCouleurTxt(rouge);
761.                                    Affi_9.affiche_texte("MAX");
762.                                    _delay_ms(300);
763.                              }
764.                        }
765.                        break;
766.
767.
768.                        case md_saisie_frq :
```

```
769.                          {
770.                                 if(etat == 1){inc_pos_curseur_frq();    }  else
         {dec_pos_curseur_frq();}
771.                                 envoyer_data = 1;
772.                          }
773.                          break;
774.                   }
775.             }
776.
777.             envoyer_data = 1;
778.             stop =1; // ==> provoque la sortie de la boucle de balayage (voir
   "BOUCLE DE BALAYAGE" dans fonction "loop", ligne 1552 environ)
779.             afficher_freq=1;
780.             sei();
781.
782.      }
783.
784.
785.      ISR(TIMER3_COMPA_vect)
786.      { }
787.
788.
789.
790.      void acquisition_data_WOB()
791.      {
792.             uint16_t acqH;
793.             uint16_t val = 0;
794.
795.             PORTF |=  0b00000010; // signal test
796.             ADCSRA |= _BV(ADSC);    //Start conversion - resolution 10bits
797.             while  (ADCSRA  &  _BV(ADSC)  )  {;}       //  attend  la  fin  de  la
   conversion
798.             PORTF &= ~0b00000010;
799.             val = ADCL;
800.             acqH = ADCH; // passage à 16 bits pour pouvoir décaller
801.             val += (acqH << 8);
802.             valeur_lue = val;
803.
804.      /** pour afficher le résultat, pour test (en connectant une tension
   connue sur l'entrée ADC)
805.             TFT480.setFont(BigFont);
806.             TFT480.setColor(255, 255, 100);
807.             TFT480.print("ADC", 20, 180);
808.             TFT480.printNumI(valeur_lue, 145, 180, 4, '0');
809.             // résultat [0...1023] pour [0V .. 2.54V]
```

```
	**/

	}


	void affiche_entete()
	{
		TFT480.setColor(64, 64, 64);
		TFT480.fillRect(0, 0, 479, 13); // bandeau en haut

		TFT480.setColor(30, 30, 30);
		TFT480.fillRect(0, 300, 479, 319);  // bandeau en bas

		TFT480.setColor(255,255,255);
		TFT480.setFont(SmallFont);
		TFT480.print("GENE-Wobu  0.1Hz  ..  4400MHz  -  AD9850  +  ADF4351", CENTER, 1);

		//  TFT480.setBackColor(64, 64, 64);
		TFT480.setColor(255,255,255);
		TFT480.print(version, 5, 300);
		TFT480.setColor(255,255,255);
		TFT480.print("ATmega2560 - ", 100, 300);
		TFT480.setColor(255,255,255);
		TFT480.print("Silicium628", 210, 300);
	}


	void init_afficheurs_wob()
	{
		if (SW_carte == Crt_AD9850)
		{
			Affi_9.init(220,  40,  120,  35,"  Hz/div");
	Affi_9.setCouleurTxt(bleu_pale);
			Affi_10.init(220, 0, 140, 35,"Mark");
	Affi_10.setCouleurTxt(violet);

			if (SW_mode == mode_WOBU_dB)
			{
				Affi_11.init(355,  0,  110,  35,"dB/div");
	Affi_11.setCouleurTxt(jaune);
			}
			if (SW_mode == mode_WOBU_mV)
			{
```

```
851.                         Affi_11.init(375,      0,      90,      35,"      V/dv");
   Affi_11.setCouleurTxt(jaune);
852.                   }
853.                   Affi_12.init(355,      40,      110,      35,"Offset");
         Affi_12.setCouleurTxt(vert);
854.       }
855.
856.
857.            if (SW_carte == Crt_ADF4351)
858.            {
859.                   Affi_9.init(220,     40,     120,     35,"     Hz/div");
         Affi_9.setCouleurTxt(bleu_pale);
860.                   Affi_10.init(220, 0, 120, 35,"Mark");
         Affi_10.setCouleurTxt(violet);
861.
862.                   if (SW_mode == mode_WOBU_dB)
863.                   {
864.                         Affi_11.init(355,      0,      110,      35,"dB/div");
   Affi_11.setCouleurTxt(jaune);
865.                   }
866.                   if (SW_mode == mode_WOBU_mV)
867.                   {
868.                         Affi_11.init(355,     0,     110,     35,"     V/div");
   Affi_11.setCouleurTxt(jaune);
869.                   }
870.                   Affi_12.init(355,      40,      110,      35,"Offset");
         Affi_12.setCouleurTxt(vert);
871.       }
872.      }
873.
874.
875.
876.      void affiche_pas_DIV(uint8_t couleur)
877.      {
878.            if (kHz_div<1000)
879.            {
880.                   Affi_9.init(220, 40, 120, 35,"kHz/div");
881.                   if (couleur == 0)
882.                   {
883.                         Affi_9.setCouleurTxt(blanc);
884.                         Affi_9.affiche_valeur(kHz_div, 6, "");
885.                   }
886.                   else
887.                   {
888.                         Affi_9.setCouleurTxt(jaune);
```

```
889.                         Affi_9.affiche_valeur(kHz_div, 6, "");
890.                   }
891.             }
892.             else
893.             {
894.                   Affi_9.init(220, 40, 120, 35,"MHz/div");
895.                   if (couleur == 0)
896.                   {
897.                         Affi_9.setCouleurTxt(blanc);
898.                         Affi_9.affiche_valeur(kHz_div/1000, 6, "");
899.                   }
900.                   else
901.                   {
902.                         Affi_9.setCouleurTxt(jaune);
903.                         Affi_9.affiche_valeur(kHz_div/1000, 6, "");
904.                   }
905.             }
906.       }
907.
908.
909.       void affiche_V_div()
910.       {
911.             if (SW_carte == Crt_AD9850)
912.             {
913.                   if (mV_div<1000)
914.                   {
915.                         Affi_11.init(375, 0, 90, 35,"mV/dv");
916.                         Affi_11.affiche_valeur(mV_div, 5, "");
917.                   }
918.                   else
919.                   {
920.                         Affi_11.init(375, 0, 90, 35," V/div");
921.                         Affi_11.affiche_valeur(mV_div/1000, 5, "");
922.                   }
923.             }
924.
925.             if (SW_carte == Crt_ADF4351)
926.             {
927.                   if (mV_div<1000)
928.                   {
929.                         Affi_11.init(355, 0, 110, 35,"mV/div");
930.                         Affi_11.affiche_valeur(mV_div, 5, "");
931.                   }
932.                   else
933.                   {
```

```
934.                          Affi_11.init(375, 0, 110, 35," V/div");
935.                          Affi_11.affiche_valeur(mV_div/1000, 5, "");
936.                    }
937.              }
938.
939.       }
940.
941.
942.       void init_afficheurs_var_ADF()
943.       {
944.                    uint16_t xp1 = 145;
945.                    uint16_t yp1 = 120;
946.                    uint8_t dy1 = 60;
947.                    Affi_5.init(xp1, yp1, 75, 50,"Div");
          Affi_5.setCouleurTxt(orange); Affi_5.setCouleurCadre(gris);
948.                    Affi_6.init(xp1,       yp1+dy1,       75,       50,"FRAC");
          Affi_6.setCouleurTxt(orange); Affi_6.setCouleurCadre(gris);
949.                    Affi_7.init(xp1,       yp1+dy1*2,       75,       50,"MOD");
          Affi_7.setCouleurTxt(orange); Affi_7.setCouleurCadre(gris);
950.
951.                    Affi_8.init(xp1+80,       yp1,       85,       50,"INTA");
    Affi_8.setCouleurTxt(orange); Affi_8.setCouleurCadre(gris);
952.
953.                    uint16_t xp2 = 385;
954.                    uint16_t yp2 = 40;
955.                    uint8_t dy2 = 40;
956.                    Affi_R0.init(xp2,    yp2,                    80,    35,"R0");
    Affi_R0.setCouleurTxt(jaune_sombre); Affi_R0.setCouleurCadre(gris);
957.                    Affi_R1.init(xp2,      yp2+dy2,           80,      35,"R1");
    Affi_R1.setCouleurTxt(jaune_sombre); Affi_R1.setCouleurCadre(gris);
958.                    Affi_R2.init(xp2,      yp2+dy2*2,           80,      35,"R2");
    Affi_R2.setCouleurTxt(jaune_sombre); Affi_R2.setCouleurCadre(gris);
959.                    Affi_R3.init(xp2,       yp2+dy2*3,       80,       35,"R3");
    Affi_R3.setCouleurTxt(jaune_sombre); Affi_R3.setCouleurCadre(gris);
960.                    Affi_R4.init(xp2,       yp2+dy2*4,       80,       35,"R4");
    Affi_R5.setCouleurTxt(jaune_sombre); Affi_R4.setCouleurCadre(gris);
961.                    Affi_R5.init(xp2,       yp2+dy2*5,       80,       35,"R5");
    Affi_R5.setCouleurTxt(jaune_sombre); Affi_R5.setCouleurCadre(gris);
962.       }
963.
964.
965.       void efface_ecran()
966.       {
967.              TFT480.setColor(0, 0, 0);
968.       //     TFT480.fillRect(0, 14, 479, 309); // efface
```

```
969.              TFT480.fillRect(0, 0, 479, 319); // efface
970.      }
971.
972.
973.      void trace_ecran()
974.      {
975.              if (SW_mode == mode_GENE)
976.              {
977.                      efface_ecran();
978.                      affiche_entete();
979.
980.                      TFT480.setBackColor(0, 0, 0);
981.                      TFT480.setColor(180,180,180);
982.                      TFT480.setFont(BigFont);
983.
984.                      Affi_1a.init(5, 20, 358, 80, "FREQ AD9850");
985.                      Affi_1b.init(5,     165,     210,     80,"FREQ     ADF4351");
   Affi_1b.setCouleurTxt(bleu); Affi_1b.setCouleurCadre(gris);
986.                      Affi_1a.setCouleurTxt(bleu);
987.                      Affi_1b.setCouleurTxt(bleu);
988.
989.                      TFT480.setFont(BigFont);
990.                      TFT480.setBackColor(20, 20, 20);
991.                      TFT480.setColor(0, 255, 0);
992.                      TFT480.print("AD9850", 376, 20);
993.
994.                      Affi_2a.init(5,    110,    130,    40,"Lambda        ");
          Affi_2a.setCouleurTxt(jaune);  Affi_2a.setCouleurCadre(gris);
995.                      Affi_3a.init(140,       110,       130,       40,"Lambda/2");
   Affi_3a.setCouleurTxt(jaune);  Affi_3a.setCouleurCadre(gris);
996.                      Affi_4a.init(275,       110,       130,       40,"Lambda/4");
   Affi_4a.setCouleurTxt(jaune);  Affi_4a.setCouleurCadre(gris);
997.
998.                      TFT480.setColor(0, 255, 0);
999.                      TFT480.drawLine(0, 160, 479, 160);
1000.
1001.     // init_afficheurs_var_ADF();
1002.                     TFT480.setFont(BigFont);
1003.                     TFT480.setBackColor(20, 20, 20);
1004.                     TFT480.setColor(0, 255, 255);
1005.                     TFT480.print("ADF4351", 365, 165);
1006.
1007.
1008.                     Affi_2b.init(5,    250,    130,    40,"Lambda        ");
          Affi_2b.setCouleurTxt(jaune);  Affi_2b.setCouleurCadre(gris);
```

```
1009.                   Affi_3b.init(140,       250,       130,       40,"Lambda/2");
   Affi_3b.setCouleurTxt(jaune);  Affi_3b.setCouleurCadre(gris);
1010.                   Affi_4b.init(275,       250,       130,       40,"Lambda/4");
   Affi_4b.setCouleurTxt(jaune);  Affi_4b.setCouleurCadre(gris);
1011.
1012.           }
1013.           if ((SW_mode == mode_WOBU_dB) || (SW_mode == mode_WOBU_mV))
1014.           {
1015.                   TFT480.setColor(10, 10, 5);
1016.                   TFT480.fillRect(0, 0, 479, 319); // efface tout
1017.
1018.                   if(SW_carte == Crt_AD9850)
1019.                   {
1020.                           Affi_1c.init(0,     0,     210,     80,"Fo     AD9850");
   Affi_1c.setCouleurTxt(bleu); Affi_1c.setCouleurCadre(gris);
1021.                   }
1022.                   if(SW_carte == Crt_ADF4351)
1023.                   {
1024.                           Affi_1c.init(0,     0,     210,     80,"Fo     ADF4351");
   Affi_1c.setCouleurTxt(bleu); Affi_1c.setCouleurCadre(gris);
1025.                   }
1026.
1027.                   init_afficheurs_wob();
1028.                   trace_pannel_graph();
1029.           }
1030.           affi_helps();
1031.     }
1032.
1033.
1034.     void trace_pannel_graph()
1035.     {
1036.           TFT480.setColor(0, 0, 0);
1037.           TFT480.fillRect(0, 80, 479, 319);
1038.           TFT480.setColor(180, 180, 180);
1039.           TFT480.drawRect(0, 80, 479, 319);
1040.
1041.     //quadrillage V
1042.           uint16_t n;
1043.           uint16_t x;
1044.           for ( n=1; n<10; n++)
1045.           {
1046.                   x= 47*n;
1047.                   if   (n==5)   {   TFT480.setColor(120,   120,120);   }   else
   { TFT480.setColor(60, 60, 60); }
1048.                   TFT480.drawLine(x, 81, x, 318);
```

```
1049.           }
1050.
1051.     //quadrillage H
1052.           uint16_t y;
1053.           for (n=1; n<10; n++)
1054.           {
1055.                 y= 81 + 24*n;
1056.                 if   (n==5)   {   TFT480.setColor(120,   120,120);   }   else
   { TFT480.setColor(60, 60, 60); }
1057.                 TFT480.drawLine(1, y, 478, y);
1058.           }
1059.     }
1060.
1061.
1062.     void efface_centre_graph()
1063.     {
1064.           TFT480.setColor(0, 0, 0);
1065.           TFT480.fillRect(101, 101, 349, 299);
1066.           TFT480.setColor(0, 200, 200);
1067.           TFT480.drawRect(100, 100, 350, 300);
1068.
1069.           TFT480.setColor(200, 0, 0);
1070.           TFT480.drawLine(235, 101, 235, 299);
1071.
1072.           TFT480.setColor(200,200,200);
1073.           TFT480.setFont(SmallFont);
1074.           TFT480.print("Mode balayage rapide", 150, 306);
1075.     }
1076.
1077.
1078.     void trace_sinusoide()
1079.     {
1080.           TFT480.setColor(0, 255, 0);
1081.           for (uint16_t n=0; n<470; n++)
1082.           {
1083.                 float x, y;
1084.                 x= (float)n/470.0*2.0*M_PI;
1085.                 y=190-80*sin(5*x);
1086.                 TFT480.drawPixel(n, y);
1087.           }
1088.     }
1089.
1090.
1091.
1092.
```

```
1093.     void TFT_affiche_chiffre_7seg(uint8_t num, uint16_t x_i, uint16_t y_i )
1094.     {
1095.           TFT480.setFont(SevenSegNumFont);
1096.           TFT480.printNumI(num, x_i, y_i);
1097.     }
1098.
1099.
1100.
1101.     void  TFT_aff_nb_form3  (uint32_t  valeur,  uint8_t  nb_chiffres,  uint8_t
   curseur, uint8_t R, uint8_t G, uint8_t B, uint8_t SB, uint16_t x_i, uint16_t
   y_i)
1102.     {
1103.     //affiche un nombre en représentation decimale en séparant les groupes de
   3 chiffres
1104.           unsigned char r ;
1105.           char tbl[10];
1106.           uint8_t i;
1107.
1108.           x_7seg = x_i;
1109.           y_7seg = y_i;
1110.
1111.           curseur=nb_chiffres +1 -curseur;
1112.
1113.           for (i=1; i<=nb_chiffres; i++)
1114.           {
1115.                 r=valeur % 10; // modulo (reste de la division)
1116.                 valeur /= 10; // quotient
1117.                 tbl[i]=r;
1118.           }
1119.           for (i=1; i<= nb_chiffres; i++)
1120.           {
1121.                 TFT480.setColor(R,G,B);
1122.
1123.                 if ((i==curseur) && (SB == 1))
1124.                 {
1125.                       TFT480.setColor(0,250,250);
1126.                 }
1127.                 TFT_affiche_chiffre_7seg(tbl[nb_chiffres  +1  -i],  x_7seg,
   y_7seg);
1128.                 x_7seg+=30;
1129.
1130.                 uint8_t m, k, u, d;
1131.                 m = nb_chiffres-7;
1132.                 k = nb_chiffres-4;
1133.                 u = nb_chiffres-1;
```

```
1134.                   d = nb_chiffres;
1135.
1136.                   if (i== m)
1137.                   {
1138.                         TFT480.setFont(BigFont);
1139.                         TFT480.setColor(180,180,180);
1140.                         TFT480.print("M",x_7seg, y_7seg+30);
1141.                         x_7seg+=15;
1142.                   }
1143.
1144.                   if (i== k)
1145.                   {
1146.                         TFT480.setFont(BigFont);
1147.                         TFT480.setColor(180,180,180);
1148.                         TFT480.print("k",x_7seg, y_7seg+30);
1149.                         x_7seg+=15;
1150.                   }
1151.
1152.                   if (i== u)
1153.                   {
1154.                         TFT480.setFont(BigFont);
1155.                         TFT480.setColor(220,220,220);
1156.                         TFT480.print(".",x_7seg, y_7seg+30);
1157.                         x_7seg+=15;
1158.                   }
1159.
1160.                   if (i== d)
1161.                   {
1162.                         TFT480.setFont(BigFont);
1163.                         TFT480.setColor(220,220,220);
1164.                         TFT480.print("Hz",x_7seg, y_7seg+30);
1165.                   }
1166.             }
1167.       }
1168.
1169.
1170.       void  TFT_aff_nb_form3b  (uint32_t  valeur,  uint8_t  nb_chiffres,  uint8_t
      nb_dec,  uint8_t curseur, uint8_t R, uint8_t G, uint8_t B, uint8_t SB, uint16_t
      x_i, uint16_t y_i)
1171.       {
1172.       //affiche  un  nombre  en  représentation  décimale  avec  séparateur  'M'  à
      gauche des 2 chiffres de droite
1173.
1174.             unsigned char r ;
1175.             char tbl[7];
```

```
1176.           uint8_t i, m;
1177.
1178.           x_7seg = x_i;
1179.           y_7seg = y_i;
1180.
1181.           curseur=nb_chiffres +1 -curseur;
1182.
1183.           for (i=1; i<=nb_chiffres; i++)
1184.           {
1185.                 r=valeur % 10; // modulo (reste de la division)
1186.                 valeur /= 10; // quotient
1187.                 tbl[i]=r;
1188.           }
1189.           for (i=1; i<= nb_chiffres; i++)
1190.           {
1191.                 TFT480.setColor(R,G,B);
1192.
1193.                 if ((i==curseur) && (SB == 1) )
1194.                 {
1195.                       TFT480.setColor(0,250,250); // surbrillance du digit
   sélectionné
1196.                 }
1197.                 TFT_affiche_chiffre_7seg(tbl[nb_chiffres  +1  -i],  x_7seg,
   y_7seg);
1198.                 x_7seg+=30;
1199.
1200.                 m = nb_chiffres - nb_dec;
1201.                 if (i==m)
1202.                 {
1203.                       TFT480.setFont(BigFont);
1204.                       TFT480.setColor(200,200,200);
1205.                       TFT480.print("M",x_7seg,  y_7seg+30); // affichage du
   separateur 'M' (M comme Megahertz)
1206.                       x_7seg+=15;
1207.                 }
1208.           }
1209.     }
1210.
1211.
1212.
1213.     //
   ================================================================================
   ================================================================
1214.
1215.     void setup()
```

```
1216.     {
1217.           init_ports();
1218.           init_variables();
1219.           scrute_switches();
1220.
1221.           TFT480.InitLCD();
1222.           TFT480.setFont(SmallFont);
1223.           TFT480.clrScr();
1224.
1225.           int_EXT_setup();
1226.           CarteADF4351.init();
1227.           CarteADF4351.mute(0);
1228.
1229.           AffiV_01.init(469, 110);  AffiV_01.setCouleur(noir);
1230.           AffiV_02.init(469, 305); AffiV_02.setCouleur(noir);
1231.
1232.     /* POUR INFO - ces lignes peuvent être commentées */
1233.           //int free_RAM;
1234.           //free_RAM = freeRam();
1235.           //TFT480.setFont(BigFont);
1236.           //TFT480.setColor(255, 255, 100);
1237.           //TFT480.print(version, 20, 160);
1238.           //TFT480.setColor(0, 200, 200);
1239.           //TFT480.print("free_RAM", 20, 180);
1240.           //TFT480.printNumI(free_RAM, 145, 180, 6, ' ');
1241.           //TFT480.print("octets", 250, 180);
1242.           //_delay_ms(2000);
1243.     /* ******************************************** */
1244.
1245.           trace_pannel_graph();
1246.     // trace_sinusoide();
1247.     // _delay_ms(1000);
1248.
1249.           trace_ecran();
1250.           if (SW_rapide == 1) {      efface_centre_graph();}
1251.
1252.           //init_leds_switches();
1253.           InitADC();
1254.
1255.           sei(); // enable interruptions
1256.           envoyer_data = 1;
1257.     }
1258.
1259.
1260.
```

```
1261.     void affiche_valeurs_registres()
1262.     {
1263.           Affi_R0.affiche_HEXA(CarteADF4351.registers[0]);
1264.           Affi_R1.affiche_HEXA(CarteADF4351.registers[1]);
1265.           Affi_R2.affiche_HEXA(CarteADF4351.registers[2]);
1266.           Affi_R3.affiche_HEXA(CarteADF4351.registers[3]);
1267.           Affi_R4.affiche_HEXA(CarteADF4351.registers[4]);
1268.           Affi_R5.affiche_HEXA(CarteADF4351.registers[5]);
1269.     }
1270.
1271.
1272.     void affiche_lambda_a()
1273.     {
1274.           if (params1.frequence_AD9850 >= 60000)
1275.           {
1276.                 Affi_2a.affiche_float(lambda_a/100.0, 4, 2, "m  ");
1277.                 Affi_3a.affiche_float(lambda_a/200.0, 4, 2, "m  ");
1278.                 Affi_4a.affiche_float(lambda_a/400.0, 4, 1, "m  ");
1279.
1280.           }
1281.           else
1282.           {
1283.                 Affi_2a.affiche_texte("-------");
1284.                 Affi_3a.affiche_texte("-------");
1285.                 Affi_4a.affiche_texte("-------");
1286.           }
1287.     }
1288.
1289.
1290.     void affiche_lambda_b()
1291.     {
1292.           if (params1.frequence_ADF4351 >= 300 )
1293.           {
1294.                 if(lambda_b <= 100)
1295.                 {
1296.                       Affi_2b.affiche_float(lambda_b, 4, 1, "cm");
1297.                       Affi_3b.affiche_float(lambda_b/2, 4, 1, "cm");
1298.                       Affi_4b.affiche_float(lambda_b/4, 4, 1, "cm");
1299.                 }
1300.                 else
1301.                       {
1302.                       Affi_2b.affiche_float(lambda_b/100.0, 4, 2, "m ");
1303.                       Affi_3b.affiche_float(lambda_b/200.0, 4, 2, "m ");
1304.                       Affi_4b.affiche_float(lambda_b/400.0, 4, 2, "m ");
1305.                 }
```

```
1306.           }
1307.           else
1308.           {
1309.                 Affi_2b.affiche_texte("-------");
1310.                 Affi_3b.affiche_texte("-------");
1311.                 Affi_4b.affiche_texte("-------");
1312.           }
1313.     }
1314.
1315.     void affiche_params()
1316.     {
1317.           DIV  = CarteADF4351.DIV;   Affi_5.affiche_valeur(DIV,  4, "  ");
1318.           FRAC = CarteADF4351.FRAC;  Affi_6.affiche_valeur(FRAC, 4, "  ");
1319.           MOD  = CarteADF4351.MOD;   Affi_7.affiche_valeur(MOD,  4, "  ");
1320.           INTA = CarteADF4351.INTA;  Affi_8.affiche_valeur(INTA, 5, "  ");
1321.     }
1322.
1323.
1324.     void init_leds()
1325.     {
1326.           LED_b1.init(388, 70, 255,0,0,10, "ON" );
1327.           LED_b2.init(388, 200, 255,0,0,10, "ON" );
1328.
1329.           if (SW_carte == Crt_ADF4351)
1330.           {
1331.                 LED_a.init(388, 230, 0,255,0,10, "LOCK" );
1332.           }
1333.     }
1334.
1335.
1336.     void init_leds_switches()
1337.     {
1338.           LED0.init(452, 8, 255,0,0,5, "" );
1339.           LED1.init(452, 18, 255,255,0,5, "" );
1340.           LED2.init(462, 8, 0,255,0,5, "" );
1341.           LED3.init(472, 8, 0,255,255,5, "" );
1342.           LED4.init(472, 18, 0,0,255,5, "" );
1343.
1344.     }
1345.
1346.
1347.     void calcul_lambda_a()
1348.     {
1349.           lambda_a = 3e11/params1.frequence_AD9850; // en cm
1350.     }
```

```
1351.
1352.
1353.     void calcul_lambda_b()
1354.     {
1355.           lambda_b = 3e6/params1.frequence_ADF4351; // en cm
1356.     }
1357.
1358.
1359.     void affi_helps()  // Ecriture verticales au bord droit de l'écran pour
   rappeler la fonction des boutons + coloration des afficheurs
1360.     {
1361.           if (SW_mode == mode_GENE) //GENE
1362.           {
1363.                 AffiV_01.affiche_texte_V(" FREQ");
1364.                 AffiV_02.affiche_texte_V(" DIGIT");
1365.                 if (SW_carte == Crt_AD9850)
1366.                 {
1367.                       Affi_1a.setCouleurCadre(jaune_sombre);
1368.                       Affi_1a.flashFond(jaune_sombre);
1369.                       Affi_1b.setCouleurCadre(gris);
1370.                 }
1371.                 if (SW_carte == Crt_ADF4351)
1372.                 {
1373.                       Affi_1b.setCouleurCadre(jaune_sombre);
1374.                       Affi_1b.flashFond(jaune_sombre);
1375.                       Affi_1a.setCouleurCadre(gris);
1376.                 }
1377.
1378.           }
1379.
1380.           if     ((SW_mode     ==     mode_WOBU_dB)     ||     (SW_mode     ==
   mode_WOBU_mV) )//WOBULATEUR
1381.           {
1382.                 if (SW_saisie == md_saisie_dB_div)
1383.                 {
1384.     //saisie OFFSET
1385.                       setCouleur(couleur_gain, 0, 255, 255);
1386.                       if          (SW_mode          ==          mode_WOBU_dB)
   { AffiV_01.affiche_texte_V(" Gain");}
1387.                       if          (SW_mode          ==          mode_WOBU_mV)
   { AffiV_01.affiche_texte_V(" dB/div");}
1388.                       AffiV_02.affiche_texte_V(" OFFSET");
1389.                       Affi_11.setCouleurCadre(jaune_sombre);
   Affi_11.flashFond(jaune_sombre);
```

```
1390.                         Affi_12.setCouleurCadre(jaune_sombre);
   Affi_12.flashFond(jaune_sombre);
1391.                         Affi_9.setCouleurCadre(gris);
1392.                         Affi_10.setCouleurCadre(gris);
1393.                 }
1394.
1395.                 if( SW_saisie == md_saisie_Hz_div)
1396.                 {
1397.     //saisie PAS
1398.                                 AffiV_01.affiche_texte_V(" MARQUEUR");
1399.                                 AffiV_02.affiche_texte_V(" Hz/div");
1400.                                 Affi_10.setCouleurCadre(jaune_sombre);
   Affi_10.flashFond(jaune_sombre);
1401.                                 Affi_9.setCouleurCadre(jaune_sombre);
   Affi_9.flashFond(jaune_sombre);
1402.                                 Affi_11.setCouleurCadre(gris);
1403.                                 Affi_12.setCouleurCadre(gris);
1404.                 }
1405.
1406.                 if (SW_saisie == md_saisie_frq)
1407.                 {
1408.     //saisie FREQ
1409.                         AffiV_01.affiche_texte_V(" FREQ");
1410.                         AffiV_02.affiche_texte_V(" DIGIT");
1411.                         Affi_9.setCouleurCadre(gris);
1412.                         Affi_10.setCouleurCadre(gris);
1413.                         Affi_11.setCouleurCadre(gris);
1414.                         Affi_12.setCouleurCadre(gris);
1415.                 }
1416.           }
1417.     }
1418.
1419.
1420.     void scrute_switches()
1421.     {
1422.     //Le pin est tiré à 1 (VCC) par R tirage interne de l'ATmega; Il est
   forcé à 0 par la fermeture du contact.
1423.           SW_mode=2; // SW_mode -> 3 positions
1424.           if ((portPIN_switch0 & pin_switch0) == 0)       {SW_mode = 1;}
1425.           if ((portPIN_switch1 & pin_switch1) == 0)       {SW_mode = 3;}
1426.
1427.           SW_saisie=md_saisie_Hz_div;  // SW_saisie -> 3 positions
1428.           if ((portPIN_switch3 & pin_switch3) == 0)          {SW_saisie     =
   md_saisie_dB_div;}
```

```
1429.           if ((portPIN_switch4 & pin_switch4) == 0)          {SW_saisie     =
   md_saisie_frq;}
1430.
1431.           SW_rapide=0;
1432.           if ((portPIN_switch2 & pin_switch2) == 0)       {SW_rapide = 1;}
1433.
1434.           SW_polarite=0;
1435.           if ((portPIN_switch5 & pin_switch5) == 0)       {SW_polarite = 1;}
1436.           if (SW_polarite == 0) {polarite = -1;} else {polarite = 1;} // la
   variable "polarite" est utilisée comme facteur dans le calcul de l'acquisition
1437.
1438.           SW_carte=Crt_AD9850;
1439.           if ((portPIN_switch6 & pin_switch6) == 0)           {SW_carte     =
   Crt_ADF4351;}
1440.
1441.           memo_switches = switches;
1442.
1443.           switches = 0;
1444.           switches |= SW_mode;
1445.           switches |= SW_saisie << 2;
1446.           switches |= SW_rapide << 4 ;
1447.           switches |= SW_polarite << 5 ;
1448.           switches |= (SW_carte-1) << 6 ;
1449.
1450.
1451.           if ((switches & 0b01000000) != (memo_switches & 0b01000000)) // si
   SW_carte a changé
1452.           {
1453.                 stop = 1;
1454.                 efface_ecran();
1455.                 init_variables();
1456.                 raffraichir=1;
1457.                 afficher_freq = 1;
1458.           }
1459.
1460.           if ((switches & 0b01111111) != (memo_switches & 0b01111111))
1461.           {
1462.                 trace_ecran();
1463.                 envoyer_data = 1;
1464.                 raffraichir=1;
1465.                 afficher_freq = 1;
1466.           }
1467.
1468.     }
1469.
```

```
1470.
1471.     double recadre(double valeur_i)
1472.     {
1473.           double y2;
1474.           double y_min, y_max;
1475.           if (SW_mode == mode_WOBU_dB) // pour détecteur log
1476.           {
1477.     /**
1478.     - la réponse du détecteur log AD8318 = 25mV/dB
1479.     - l'ADC de l'ATmega convertit 2.56V -> 1024 pas (0..1023) soit
   1024/2560mV = 0.4pas/mV (voir fonction InitADC() -> Internal 2.56V Voltage
   Reference)
1480.     - 1dB => 25mV => 25x0.4pas => 10pas
1481.     - echelle de la valeur lue = Ech = 10pas/dB
1482.     - le panel graphique fait 320-80 = 240px en Y
1483.     - ce panel graphique est divisé en 10 divisions qui font chacune 240/10 =
   24 pixels
1484.     - en position 1dB/div le gain doit être 1dB => 10pasADC => 1divY => 24px
1485.     - gain(1dB/div) = G1 = 24/10 px/pasADC = 2.4
1486.     **/
1487.                 if (SW_rapide == 0) {     y_min=83;   y_max =317;} else {
          y_min=101;  y_max =299;}
1488.                 y2 = 200 + (-1.0) * (params1.offset_y  + (2.4 / dB_div) *
   polarite * valeur_i );
1489.                 if (y2<y_min) {y2=y_min;}
1490.                 if (y2>y_max) {y2=y_max;}
1491.                 return y2;
1492.           }
1493.
1494.           if (SW_mode == mode_WOBU_mV) // pour détecteur linéaire
1495.           {
1496.     /**
1497.     - l'ADC de l'ATmega convertit 2.56V -> 1024 pas (0..1023) soit
   1024/2560mV = 0.4pas/mV
1498.     - le panel graphique fait 320-80 = 240px en Y
1499.     - ce panel graphique est divisé en 10 divisions qui font chacune 240/10 =
   24 pixels
1500.     - pour obtenir un affichage de 1mV/div soit 0.4pas/div soit 0.4pas/24pix
   le facteur de proportion = 24/0.4 = 60 pix/pas
1501.
1502.     **/
1503.                 if (SW_rapide == 0) {     y_min=83;   y_max =317;} else {
          y_min=101;  y_max =299;}
1504.                 y2 = 200 + polarite * ( (params1.offset_y  + (60.0 / mV_div)
   * valeur_i ) );
```

```
1505.                   if (y2<y_min) {y2=y_min;}
1506.                   if (y2>y_max) {y2=y_max;}
1507.                   return y2;
1508.             }
1509.     }
1510.
1511.
1512.
1513.      void loop()
1514.     {
1515.             double Fd0, Fd1, Fmin, Fmax, increment_F;
1516.     //      double Fcentrale;
1517.             double y_float;
1518.             uint16_t y, memo_y, y_repos, y2;
1519.             uint8_t i;
1520.             uint8_t SB;
1521.             uint8_t c;
1522.
1523.             raffraichir = 1;
1524.             afficher_freq = 1;
1525.             while(1)
1526.             {
1527.                   scrute_switches();
1528.
1529.     //-------------------------------------------------------------------------
   ----------------------------------------
1530.     //
          GENERATEUR SIMPLE
1531.     //-------------------------------------------------------------------------
   ----------------------------------------
1532.                   if (SW_mode == mode_GENE)
1533.                   {
1534.                               if(envoyer_data > 0)
1535.                               {
1536.                                       envoyer_data = 0;
1537.                                       calcul_lambda_a();
1538.                                       calcul_lambda_b();
1539.
1540.                                       Fd0 = (double) params1.frequence_AD9850;
          // pour l'AD9850 - Fd0 = fréquence en 1/10 Hz c.a.d 10x fréquence en Hz
1541.                                       affiche_lambda_a();
1542.
1543.                                       Fd1 = params1.frequence_ADF4351 / 100.0;
          // pour l'ADF4351 - Fd1 = fréquence en MHz (avec decimales, c'est un réel
   (double))
```

```
1544.                                       affiche_lambda_b();
1545.
1546.                                       init_leds();
1547.                                       LED_b1.setEtat(1);
1548.                                       LED_b2.setEtat(1);
1549.
1550.                                       CarteAD9850.out_F(Fd0); // le paramètre
   = fréquence en 1/10 Hz c.a.d à 10x fréquence en Hz, ce qui permet une précision
   du 1/10 Hz
1551.                                       _delay_us(10);
1552.                                       Affi_1a.setCouleurTxt(bleu);
   //Affi_1a.setCouleurCadre(jaune_sombre);
1553.                                       SB=0  ;  if  (SW_carte  ==  Crt_AD9850)
   {SB=1;}
1554.
          Affi_1a.affiche_frequence_9(params1.frequence_AD9850,  SB);  //  (avec  de
   grands chiffres 7 segments)
1555.
1556.                                       CarteADF4351.setFreq(Fd1);
1557.                                       CarteADF4351.Write_All_Register();
1558.                                       _delay_us(10);
1559.                                       if  ((PIN_ADF4351_A  &  pin_MUXOUT)  ==
   pin_MUXOUT) { LED_a.setEtat(1); } else { LED_a.setEtat(0);}
1560.                                       //affiche_params();
1561.                                       //affiche_valeurs_registres();
1562.                                       Affi_1b.setCouleurTxt(bleu);
   //Affi_1b.setCouleurCadre(jaune_sombre);
1563.                                       SB=0  ;  if  (SW_carte  ==  Crt_ADF4351)
   {SB=1;}
1564.                                       c = params1.pos_curFreq_gene_ADF43;
1565.
          Affi_1b.affiche_frequence_6(params1.frequence_ADF4351,  2,  c,  SB);  //
   (avec de grands chiffres 7 segments)
1566.
1567.                                       save_params_to_EEPROM();
1568.                                  }
1569.                  }
1570.     //-------------------------------------------------------------------------
   ----------------------------------------
1571.     //
          WOBULATEUR
1572.     //-------------------------------------------------------------------------
   ----------------------------------------
1573.                  else  //  mode  wobulation    (SW_mode  ==  mode_WOBU_dB)  ||
   (SW_mode == mode_WOBU_mV)
```

```
1574.                   {
1575.                         if (raffraichir == 1)
1576.                         {
1577.                               raffraichir=0;
1578.                               if(SW_carte == Crt_AD9850)
1579.                               {
1580.                                     Affi_1c.init(0, 0, 210, 78,"Fo AD9850");
1581.                                     Affi_1c.setCouleurTxt(bleu);
   Affi_1c.setCouleurCadre(jaune_sombre);
1582.                               }
1583.                               if(SW_carte == Crt_ADF4351)
1584.                               {
1585.                                     Affi_1c.init(0,     0,     210,     78,"Fo
   ADF4351");
1586.                                     Affi_1c.setCouleurTxt(bleu);
   Affi_1c.setCouleurCadre(jaune_sombre);
1587.
1588.                               }
1589.                               init_afficheurs_wob();
1590.
1591.                               save_params_to_EEPROM();
1592.                         }
1593.
1594.
1595.                         if (SW_rapide == 1) {      efface_centre_graph();}
1596.
1597.                         if (afficher_freq == 1)
1598.                         {
1599.                               afficher_freq=0;
1600.                               if (SW_saisie == md_saisie_frq)
1601.                               {
1602.                                     if(SW_carte == Crt_AD9850)
1603.                                     {
1604.                                           //Affi_1c.setCouleurTxt(bleu);
   Affi_1c.setCouleurCadre(jaune_sombre);
1605.                                           c                                   =
   params1.pos_curFreq_wobu_AD98;
1606.
          Affi_1c.affiche_frequence_5(params1.frequence_AD9850/10000, 3, c, 1);   //
   surbrilllance 1 digit
1607.                                     }
1608.                                     if(SW_carte == Crt_ADF4351)
1609.                                     {
1610.                                           //Affi_1c.setCouleurTxt(bleu);
   Affi_1c.setCouleurCadre(jaune_sombre);
```

```
1611.                                                   c                                   =
   params1.pos_curFreq_wobu_ADF43;
1612.
          Affi_1c.affiche_frequence_6(params1.frequence_ADF4351,  2,  c,  1);    //
   surbrilllance 1 digit
1613.                                       }
1614.                                 }
1615.                                 else // modes saisie paramètres de wobulation
   (SW_saisie == md_saisie_Hz_div) ||(SW_saisie == md_saisie_dB_div)
1616.                                 {
1617.                                       if(SW_carte == Crt_AD9850)
1618.                                       {
1619.                                             Affi_1c.setCouleurTxt(bleu);
   Affi_1c.setCouleurCadre(gris);
1620.                                             c                                   =
   params1.pos_curFreq_wobu_AD98;
1621.
          Affi_1c.affiche_frequence_5(params1.frequence_AD9850/10000, 3, c, 1);  //
   surbrilllance 1 digit
1622.
1623.                                       }
1624.                                       if(SW_carte == Crt_ADF4351)
1625.                                       {
1626.                                             Affi_1c.setCouleurTxt(bleu);
   Affi_1c.setCouleurCadre(gris);
1627.                                             c                                   =
   params1.pos_curFreq_wobu_ADF43;
1628.
          Affi_1c.affiche_frequence_6(params1.frequence_ADF4351, 2, c, 0);  // pas
   de chiffre en surbrilllance
1629.                                       }
1630.                                 }
1631.                           }
1632.
1633.                           if(SW_carte == Crt_AD9850)
1634.                           {
1635.                                 affiche_pas_DIV(1);
1636.
1637.                                 frq_centrale  =  params1.frequence_AD9850;  //
   fréquence en 1/10 Hz c.a.d 10x fréquence en Hz
1638.
1639.                                 increment_F = 1e5 * kHz_div / 470.0;
1640.
1641.                                 Fmin = frq_centrale-235*increment_F;
1642.                                 Fmax = frq_centrale+235*increment_F;
```

```
1643.
1644.                                F_mark =(Fmin + ((double) params1.pos_mark *
   increment_F))/1e4;
1645.
1646.                                F_affi_min=Fmin/1e2;
1647.                                if (F_affi_min < frq_min_AD98) {F_affi_min =
   frq_min_AD98;}
1648.
1649.                                F_affi_max=Fmax/1e2;
1650.                                if (F_affi_max > frq_max_AD98) {F_affi_max =
   frq_max_AD98;}
1651.                           }
1652.
1653.
1654.                           if(SW_carte == Crt_ADF4351)
1655.                           {
1656.                                affiche_pas_DIV(0);
1657.
1658.                                //Fcentrale   =   params1.frequence_ADF4351   /
   100.0;
1659.                                increment_F = kHz_div / 470.0;
1660.
1661.                                Fmin        =        params1.frequence_ADF4351-
   235*increment_F;
1662.                                Fmax                                          =
   params1.frequence_ADF4351+235*increment_F;
1663.
1664.                                F_mark =(Fmin + ((double) params1.pos_mark *
   increment_F)) / 100.0;
1665.
1666.                                F_affi_min=Fmin;
1667.                                if (F_affi_min < frq_min_ADF43) {F_affi_min =
   frq_min_ADF43;}
1668.
1669.                                F_affi_max=Fmax;
1670.                                if (F_affi_max > frq_max_ADF43) {F_affi_max =
   frq_max_ADF43;}
1671.                           }
1672.
1673.     //                    acquisition_data_WOB();
1674.     //                    y=valeur_lue;
1675.
1676.     // ****************************************
1677.                           affiche_freq_marqueur();
```

```
1678.                         if          (SW_mode          ==          mode_WOBU_dB)
   {Affi_11.affiche_valeur(dB_div, 2, "");}
1679.                         if (SW_mode == mode_WOBU_mV) {affiche_V_div();}
1680.                         Affi_12.affiche_valeur(params1.offset_y, 5, "");
1681.
1682.     // BOUCLE DE BALAYAGE - incrémente la fréquence et affiche le graphique
   en temps réel
1683.
1684.     //     for (int n=0; n<470; n++)
1685.
1686.                         uint8_t dn=1;if (SW_rapide == 1) {dn =5;}
1687.                         int n=0; if (SW_rapide == 1) {n =100;}
1688.                         uint16_t  n_max  =470;  if  (SW_rapide  ==  1)  {n_max
   =350;}
1689.                         stop=0;
1690.                         while ((n<n_max) && (stop ==0)) // stop set à 1 par
   les INT0 et INT1
1691.                         {
1692.                               if(raffraichir==1)
1693.                               {
1694.                                     raffraichir=0;
1695.                                     affiche_pas_DIV(0);
1696.                                     Affi_1c.setCouleurTxt(bleu);
1697.
1698.                                     if (SW_saisie == md_saisie_frq)
1699.                                     {
1700.                                           if (SW_carte == Crt_AD9850 )
1701.                                           {
1702.                                                 c                         =
   params1.pos_curFreq_wobu_AD98;
1703.
         Affi_1c.affiche_frequence_6(params1.frequence_AD9850/1000,  4,  c,  1);  //
   (grands chiffres + surbrillance 1 digit)
1704.                                           }
1705.                                           if (SW_carte == Crt_ADF4351 )
1706.                                           {
1707.                                                 c                         =
   params1.pos_curFreq_wobu_AD98;
1708.
         Affi_1c.affiche_frequence_6(params1.frequence_ADF4351,   2,   c,   1);   //
   (grands chiffres + surbrillance 1 digit)
1709.                                           }
1710.                                     }
1711.                                     else
1712.                                     {
```

```
1713.                                                 if (SW_carte == Crt_AD9850 )
1714.                                                 {
1715.                                                       c                         =
   params1.pos_curFreq_wobu_AD98;
1716.
          Affi_1c.affiche_frequence_6(params1.frequence_AD9850/1000,  4,  c,  0);  //
   (grands chiffres, pas de surbrillance digit )
1717.                                                       F_affi_min=Fmin;
1718.                                                       if      (F_affi_min      <
   frq_min_AD98) {F_affi_min = frq_min_AD98;}
1719.                                                       F_affi_max=Fmax;
1720.                                                       if      (F_affi_max      >
   frq_max_AD98) {F_affi_max = frq_max_AD98;}
1721.                                                 }
1722.
1723.                                                 if (SW_carte == Crt_ADF4351 )
1724.                                                 {
1725.                                                       c                         =
   params1.pos_curFreq_wobu_AD98;
1726.
          Affi_1c.affiche_frequence_6(params1.frequence_ADF4351,   2,   c,   0);   //
   (grands chiffres, pas de surbrillance digit )
1727.                                                       F_affi_min=Fmin;
1728.                                                       if      (F_affi_min      <
   frq_min_ADF43) {F_affi_min = frq_min_ADF43;}
1729.                                                       F_affi_max=Fmax;
1730.                                                       if      (F_affi_max      >
   frq_max_ADF43) {F_affi_max = frq_max_ADF43;}
1731.                                                 }
1732.                                           }
1733.
1734.                                           affiche_pas_DIV(0);
1735.                                           affiche_freq_marqueur();
1736.                                           affi_min_max();
1737.                                     }
1738.
1739.                                     depassement =0;
1740.
1741.     // GENERATION DU SIGNAL
1742.
1743.                                     if (SW_carte == Crt_AD9850 )
1744.                                     {
1745.                                           Fd1 = (Fmin + n * increment_F);
1746.                                           if   (Fd1   <   frq_min_AD98)   {Fd1   =
   frq_min_AD98; depassement =1;}
```

```
1747.                                   if   (Fd1   >   frq_max_AD98)   {Fd1   =
   frq_max_AD98; depassement =1;}
1748.                                   //Fd1 = Fd1 / 10.0;
1749.                                   CarteAD9850.out_F(Fd1); // le paramètre
   = fréquence en 1/10 Hz c.a.d à 10x fréquence en Hz, ce qui permet une précision
   du 1/10 Hz
1750.                               }
1751.
1752.                               if (SW_carte == Crt_ADF4351 )
1753.                               {
1754.                                   Fd1 = (Fmin + n * increment_F);
1755.                                   if   (Fd1   <   frq_min_ADF43)   {Fd1   =
   frq_min_ADF43; depassement =1;}
1756.                                   if   (Fd1   >   frq_max_ADF43)   {Fd1   =
   frq_max_ADF43; depassement =1;}
1757.                                   Fd1 = Fd1 / 100.0; // Fd1 en MHz (avec 2
   décimales)
1758.                                   CarteADF4351.setFreq(Fd1);
1759.                               }
1760.
1761.                               _delay_ms(1);
1762.
1763.                               acquisition_data_WOB();
1764.
1765.                               memo_y=y;
1766.                               y_float   =   recadre((float)valeur_lue);   //
   règlage de offset et amplitude pour entrer dans le cadre d'affichage
1767.                               y = (uint16_t)y_float;
1768.                               if (y>317) {y=317;}
1769.
1770.                               tableau_acq[n]=y; // mémorisation de la courbe
   (recadrée) en SRAM
1771.
1772.                               if(SW_rapide == 0)
1773.                               {
1774.                                   TFT480.setColor(255, 255, 255);
1775.                                   if(n<465)   {TFT480.drawLine(n+2,   y-3,
   n+2,  y+3);}  //  petit  tiret  vertical  pour  faire  joli  (spot)  et  montrer
   l'avancement
1776.                                   if((n  %  47)==0)  {  TFT480.setColor(60,
   60, 60); } else { TFT480.setColor(0, 0, 0); }
1777.                                   if(n  ==  235)  {  TFT480.setColor(255,
   100,100); }
1778.                                   if        (n==        params1.pos_mark)
   { TFT480.setColor(200, 0, 255); }
```

```
1779.
1780.                                       TFT480.drawLine(n,   81,   n,   318);   // efface toute une ligne verticale tout en redessinant le reticule V et le marqueur (en jouant sur les couleurs)
1781.                                       TFT480.setColor(100, 100, 100);
1782.
1783.     //quadrillage H
1784.                                       for (int i=1; i<10; i++)
1785.                                       {
1786.                                             y2= 81 + 24*i;
1787.                                             if  (i==5)  {  TFT480.setColor(120, 120,120); } else { TFT480.setColor(60, 60, 60); }
1788.                                             TFT480.drawPixel(n, y2);
1789.                                       }
1790.                                 }
1791.
1792.                                 TFT480.setColor(0, 255, 0);
1793.                                 if (depassement == 0) // en fréquence
1794.                                 {
1795.                                       if(n>(20*dn))     {TFT480.drawLine(n-dn, memo_y, n, y);}  // <===  trace le signal normal *****
1796.                                 }
1797.                                 else
1798.                                 {
1799.                                       TFT480.setColor(255, 0, 0);
1800.                                       TFT480.drawLine(n-1,  318,  n,  318);  // trace une ligne rouge H
1801.                                       TFT480.drawLine(n-1, 310, n, 310);
1802.                                 }
1803.                                 if ((n==80) ||      (n==460))                {
          affi_min_max();     }
1804.
1805.                                 n+=dn;
1806.
1807.                                 scrute_switches();
1808.                                 if ((switches & 0b00001111) != (memo_switches & 0b00001111)) {     stop=1;      }
1809.                           }
1810.     // _delay_ms(1000); // pause pour pouvoir prendre des photos facilement;
1811.                 }
1812.           }
1813.     }
1814.
```

```
1815.     /**
   ********************************************************************************
   ***
1816.                   CLASS AffiRect  // affiche un nombre ou un petit texte dans
   un rectangle
1817.     ********************************************************************************
   *************/
1818.
1819.     // Constructeur
1820.     AffiRect::AffiRect()
1821.     {
1822.
1823.     }
1824.
1825.
1826.     void AffiRect::init(uint16_t x, uint16_t y, uint16_t dx, uint16_t dy,
   char txt_etiquette_i[12])
1827.     {
1828.           x0 = x;
1829.           y0 = y;
1830.           dx0 = dx;
1831.           dy0 = dy;
1832.
1833.           for (int i=0; i<12; i++) {txt_etiquette[i]=txt_etiquette_i[i];}
1834.           txt_etiquette[12]='\0';  // zero terminal
1835.
1836.           TFT480.setBackColor(0, 0, 0);
1837.           TFT480.setColor(10, 10, 5);
1838.
1839.           TFT480.setColor(Rcadre, Vcadre, Bcadre);
1840.           TFT480.drawRect(x0, y0+5, x0+dx0, y0+dy0);
1841.
1842.           traceCadre();
1843.           affi_etiquette();
1844.
1845.     }
1846.
1847.     void AffiRect::traceCadre()
1848.     {
1849.           TFT480.setColor(Rcadre, Vcadre, Bcadre);
1850.           TFT480.drawRect(x0, y0+5, x0+dx0, y0+dy0);
1851.     }
1852.
1853.
1854.     void AffiRect::affi_etiquette()
```

```
1855.     {
1856.           TFT480.setBackColor(0, 0, 0);
1857.           TFT480.setFont(BigFont);
1858.           TFT480.setColor(180,180,180); // couleur de l'étiquette
1859.           TFT480.print(txt_etiquette, x0, y0); // Etiquette
1860.     }
1861.
1862.
1863.     void AffiRect::setCouleurTxt(uint8_t *couleur_i)
1864.     {
1865.           Rtxt=couleur_i[0];
1866.           Vtxt=couleur_i[1];
1867.           Btxt=couleur_i[2];
1868.     }
1869.
1870.     void AffiRect::setCouleurCadre(uint8_t *couleur_i)
1871.     {
1872.           Rcadre=couleur_i[0];
1873.           Vcadre=couleur_i[1];
1874.           Bcadre=couleur_i[2];
1875.
1876.           TFT480.setColor(Rcadre, Vcadre, Bcadre);
1877.           TFT480.drawRect(x0, y0+5, x0+dx0, y0+dy0);
1878.           TFT480.setBackColor(0, 0, 0);
1879.           TFT480.setFont(BigFont);
1880.           TFT480.setColor(180,180,180); // couleur de l'étiquette
1881.           TFT480.print(txt_etiquette,  x0,  y0);  //  Etiquette  (pour  écrire
    volaontairement par dessus le cadre))
1882.     }
1883.
1884.     void AffiRect::setCouleurFond(uint8_t *couleur_i)
1885.     {
1886.           Rfond=couleur_i[0];
1887.           Vfond=couleur_i[1];
1888.           Bfond=couleur_i[2];
1889.     }
1890.
1891.     void AffiRect::flashFond(uint8_t *couleur_i)
1892.     {
1893.           Rfond=couleur_i[0];
1894.           Vfond=couleur_i[1];
1895.           Bfond=couleur_i[2];
1896.           TFT480.setColor(Rfond, Vfond, Bfond);
1897.           TFT480.fillRect(x0, y0, x0+dx0, y0+dy0);
1898.           _delay_ms(10);
```

```
1899.            TFT480.setColor(10, 10, 5);
1900.            TFT480.fillRect(x0, y0, x0+dx0, y0+dy0);
1901.            traceCadre();
1902.            affi_etiquette();
1903.      }
1904.
1905.
1906.      void   AffiRect::affiche_frequence_5(uint32_t   F_i,   uint8_t   nb_dec_i,
   uint8_t pos_cur, uint8_t SB_i) // sur 5 digits ; SB = surbrillance 1 digit
1907.      {
1908.            uint16_t x_7seg;
1909.            uint16_t y_7seg;
1910.            uint8_t p2;
1911.
1912.            x_7seg = x0+10;
1913.            y_7seg = y0+20;
1914.            TFT480.setBackColor(0, 0, 0);
1915.            TFT_aff_nb_form3b  (F_i,  5,  nb_dec_i,  pos_cur,  Rtxt,  Vtxt,  Btxt,
   SB_i, x_7seg, y_7seg); // 5 chiffres significatifs
1916.      }
1917.
1918.      void   AffiRect::affiche_frequence_6(uint32_t   F_i,   uint8_t   nb_dec_i,
   uint8_t pos_cur, uint8_t SB_i) // sur 6 digits ; SB = surbrillance 1 digit
1919.      {
1920.            uint16_t x_7seg;
1921.            uint16_t y_7seg;
1922.            uint8_t p2;
1923.
1924.            x_7seg = x0+10;
1925.            y_7seg = y0+20;
1926.            TFT480.setBackColor(0, 0, 0);
1927.            TFT_aff_nb_form3b  (F_i,  6,  nb_dec_i,  pos_cur,  Rtxt,  Vtxt,  Btxt,
   SB_i, x_7seg, y_7seg); // 6 chiffres significatifs
1928.      }
1929.
1930.
1931.      void AffiRect::affiche_frequence_9(uint32_t F_i, uint8_t SB_i) // sur 9
   digits (8 + 1 décimale);  SB_i = surbrillance 1 digit
1932.      {
1933.            uint16_t x_7seg;
1934.            uint16_t y_7seg;
1935.            uint8_t p2;
1936.
1937.            x_7seg = x0+10;
1938.            y_7seg = y0+20;
```

```
1939.           p2 = params1.pos_curFreq_gene_AD98;
1940.           TFT480.setBackColor(0, 0, 0);
1941.
1942.           TFT_aff_nb_form3 (F_i, 9, p2, Rtxt, Vtxt, Btxt, SB_i, x_7seg,
   y_7seg);
1943.     }
1944.
1945.
1946.
1947.     void AffiRect::affiche_valeur(uint32_t valeur, uint8_t nb_chiffres, char
   txt_unite_i[3])
1948.     {
1949.           for (int i=0; i<3; i++) {txt_unite[i]=txt_unite_i[i];}
1950.           txt_unite[3]='\0';  // zero terminal
1951.
1952.           TFT480.setBackColor(0, 0, 0);
1953.           TFT480.setColor(Rtxt, Vtxt, Btxt);
1954.           TFT480.setFont(BigFont);
1955.           TFT480.printNumI(valeur, x0+5,y0+18, nb_chiffres, ' ');
1956.           TFT480.setFont(BigFont);
1957.           TFT480.print(txt_unite, x0+80,y0+18); // ex : mm, kHz, etc...
1958.     }
1959.
1960.
1961.
1962.     void AffiRect::affiche_float(float valeur, uint8_t nb_chiffres, uint8_t
   nb_dec, char txt_unite_i[3])
1963.     {
1964.           for (int i=0; i<3; i++) {txt_unite[i]=txt_unite_i[i];}
1965.           txt_unite[3]='\0';  // zero terminal
1966.
1967.           TFT480.setBackColor(0, 0, 0);
1968.           TFT480.setColor(Rtxt, Vtxt, Btxt);
1969.           TFT480.setFont(BigFont);
1970.
1971.     //RAPPEL: void      printNumF(double num, byte dec, int x, int y, char
   divider='.', int length=0, char filler=' ');
1972.           TFT480.printNumF(valeur, nb_dec, x0+5,y0+18, ',',   nb_chiffres, '
   ');
1973.
1974.           TFT480.setFont(BigFont);
1975.           TFT480.print(txt_unite, x0+82,y0+18); // ex : mm, kHz, etc...
1976.     }
1977.
1978.
```

```
1979.     void AffiRect::affiche_HEXA(uint32_t valeur)
1980.     {
1981.     // affiche un nombre en representation hexadécimale
1982.     // 16 nb_signes hexa max
1983.
1984.           uint8_t r;
1985.           uint8_t i;
1986.
1987.           char tbl[9];
1988.           char signes[17] = "0123456789ABCDEF";
1989.
1990.           for (i=0; i<8; i++)
1991.           {
1992.                 r= valeur % 16; // modulo (reste de la division)
1993.                 valeur /= 16; // quotient
1994.                 tbl[7-i]=signes[r];
1995.           };
1996.           tbl[8]='\0';
1997.
1998.           TFT480.setBackColor(0, 0, 0);
1999.           TFT480.setColor(0,255,255);
2000.           TFT480.setFont(SmallFont);
2001.           TFT480.print(tbl, x0+10,y0+15);
2002.     }
2003.
2004.
2005.     void AffiRect::affiche_texte(char txt_i[10])
2006.     {
2007.           for (int i=0; i<10; i++) {texte[i]=txt_i[i];}
2008.           texte[10]='\0';  // zero terminal
2009.
2010.           TFT480.setBackColor(0, 0, 0);
2011.           TFT480.setColor(Rtxt, Vtxt, Btxt);
2012.           TFT480.setFont(BigFont);
2013.           TFT480.print(texte, x0+5,y0+18);
2014.     }
2015.
2016.
2017.     /**
    ********************************************************************************
    ***
2018.                 CLASS AffiV  // affiche verticalement sur fond gris

2019.     ********************************************************************************
    **************/
```

```
2020.
2021.     // Constructeur
2022.     AffiV::AffiV()
2023.     {
2024.
2025.     }
2026.
2027.     void AffiV::init(uint16_t x, uint16_t y)
2028.     {
2029.           x0 = x;
2030.           y0 = y;
2031.     }
2032.
2033.     void AffiV::setCouleur(uint8_t *couleur_i)
2034.     {
2035.           R=couleur_i[0];
2036.           V=couleur_i[1];
2037.           B=couleur_i[2];
2038.     }
2039.
2040.     void  AffiV::affiche_texte_V(char txt_i[10])
2041.     {
2042.
2043.           for (int i=0; i<10; i++) {texte[i]=txt_i[i];}
2044.           texte[10]='\0';  // zero terminal
2045.
2046.           TFT480.setBackColor(200,200,200);
2047.           TFT480.setColor(R, V, B);
2048.           TFT480.setFont(SmallFont);
2049.           TFT480.print(texte, x0,y0, -90);
2050.     }
2051.
2052.
2053.
2054.     /**
    ********************************************************************************
    ***
2055.                                                 CLASS LED

2056.     ********************************************************************************
    **************/
2057.     // Constructeur
2058.     LED::LED()
2059.     {
2060.
```

```
2061.     }
2062.
2063.
2064.     void LED::init(uint16_t x_i, uint16_t y_i, uint8_t R_i, uint8_t G_i,
   uint8_t B_i, uint_fast8_t taille_i, char txt_etiquette[5])
2065.     {
2066.           x=x_i;
2067.           y=y_i;
2068.
2069.           R=R_i;
2070.           G=G_i;
2071.           B=B_i;
2072.
2073.           taille = taille_i;
2074.
2075.           TFT480.setColor(40,40,40); // gris
2076.           TFT480.fillCircle(x, y, taille); // dessine la led éteinte (cercle
   plein grisé)
2077.
2078.           TFT480.setColor(180,180,180);
2079.           TFT480.setFont(BigFont);
2080.           if (txt_etiquette[0] != '\0') {TFT480.print(txt_etiquette, x+15, y-
   8);} // Etiquette
2081.     }
2082.
2083.
2084.
2085.     void LED::setEtat(uint8_t etat_i)
2086.     {
2087.           if(etat_i == 0)
2088.           {
2089.                 TFT480.setColor(40,40,40); // gris
2090.                 TFT480.fillCircle(x, y, taille);
2091.           }
2092.           if(etat_i == 1)
2093.           {
2094.                 TFT480.setColor(R,G,B); // couleur choisie
2095.                 TFT480.fillCircle(x, y, taille);
2096.           }
2097.
2098.     }
2099.
```

Plik Gene_Wobul_ADF4351_v10_2.h

```
1. //================================
2. // version "v10.2"
3. //================================
4.
5. #ifndef GENE_WOBUL_ADF4351_V10_2_H
6. #define GENE_WOBUL_ADF4351_V10_2_H
7.
8. #include <stdint.h>
9. #include "Arduino.h"
10.
11.
12. /**
    ********************************************************************************
    ***
13.              CLASS AffiRect   // affiche un nombre ou un petit texte dans un
    rectangle
14. ********************************************************************************
    *******/
15.
16. class AffiRect
17. {
18. public:
19.
20.       uint16_t x0; //position
21.       uint16_t y0;
22.       uint16_t dx0; //dimension
23.       uint16_t dy0;
24.
25. //couleurs
26.       uint8_t Rfond, Vfond, Bfond;
27.       uint8_t Rcadre, Vcadre, Bcadre;
28.       uint8_t Rtxt, Vtxt, Btxt;
29.
30.       AffiRect();
31.
32.       void  init(uint16_t  x,  uint16_t  y,  uint16_t  dx,  uint16_t  dy,char
    txt_etiquette[10]);
33.       void setCouleurCadre(uint8_t *couleur_i);
34.       void setCouleurTxt(uint8_t *couleur_i);
35.       void setCouleurFond(uint8_t *couleur_i);
36.       void flashFond(uint8_t *couleur_i);
37.       void affiche_frequence_5(uint32_t F_i, uint8_t nb_dec_i, uint8_t pos_cur,
    uint8_t SB_i); // sur 5 digits ; SB = surbrillance 1 digit
38.       void affiche_frequence_6(uint32_t F_i, uint8_t nb_dec_i, uint8_t pos_cur,
    uint8_t SB_i); //sur 6 digits ; SB = surbrillance 1 digit
```

```
39.       void affiche_frequence_9(uint32_t F_i, uint8_t SB_i); //sur 9 digits ; SB
   = surbrillance 1 digit
40.       void   affiche_valeur(uint32_t   valeur,   uint8_t   nb_chiffres,   char
   txt_unite_i[3]);
41.       void  affiche_float(float  valeur,  uint8_t  nb_chiffres,  uint8_t  nb_dec,
   char txt_unite_i[3]);
42.       void affiche_HEXA(uint32_t valeur);
43.       void affiche_texte(char txt_i[5]);
44.
45.
46. private:
47.       char txt_etiquette[13];
48.       char txt_unite[4];
49.       char texte[11];
50.
51.       void traceCadre();
52.       void affi_etiquette();
53.
54. };
55.
56. /**
   *****************************************************************************
   ***
57.              CLASS AffiV  // affiche verticalement sur fond gris

58. *****************************************************************************
   *******/
59. class AffiV
60. {
61. public:
62.
63.       uint16_t x0;
64.       uint16_t y0;
65.       uint8_t R, V, B;
66.
67.       AffiV();
68.
69.       void init(uint16_t x, uint16_t y);
70.       void setCouleur(uint8_t *couleur_i);
71.       void  affiche_texte_V(char txt_i[10]);
72.
73.
74. private:
75.       char texte[11];
76. };
```

```
77.
78.
79. /**
    ********************************************************************************
    ***
80.                                         CLASS LED
81. ********************************************************************************
    *******/
82.
83. class LED
84. {
85. public:
86.
87.       uint16_t x;
88.       uint16_t y;
89.
90.       uint8_t R;
91.       uint8_t G;
92.       uint8_t B;
93.
94.       uint8_t taille; // = dimensions x et y
95.
96.   LED();
97.
98.       void init(uint16_t x_i, uint16_t y_i, uint8_t R_i, uint8_t G_i, uint8_t
    B_i, uint_fast8_t taille, char txt_etiquette[5]);
99.       void setEtat(uint8_t etat_i);  // = 0 ou 1
100.
101.      private:
102.            char txt_etiquette[5];
103.
104.      };
105.
106.
107.      /***** autres declarations ******/
108.
109.
110.      #define      PIN_ADF4351_A PINF
111.      #define      PIN_ADF4351_B PINK
112.
113.
114.      // types
115.      struct parametres
116.      {
117.            uint8_t cle;
```

```
118.            uint32_t frequence_AD9850;
119.            uint32_t frequence_ADF4351;
120.            uint8_t  pos_curFreq_gene_AD98;  //  position  du  curseur  (pour
    affichage freq) ->  AD9850
121.            uint8_t  pos_curFreq_gene_ADF43;  //  position  du  curseur  (pour
    affichage freq)     ->     ADF4351
122.            uint8_t  pos_curFreq_wobu_AD98;  //  position  du  curseur  (pour
    affichage freq de wobulation)
123.            uint8_t  pos_curFreq_wobu_ADF43;  //  position  du  curseur  (pour
    affichage freq de wobulation)
124.            uint8_t pos_curseur_wob;  // position du curseur (pour le pas de
    wobulation)
125.            uint16_t pos_mark;
126.            int32_t offset_y;
127.            int16_t gain;
128.      };
129.
130.
131.      // fonctions
132.
133.      void InitADC();
134.      void init_ports ();
135.      void init_variables();
136.      void init_leds_switches();
137.      void init_afficheurs_wob();
138.      void init_leds();
139.      void init_leds_switches();
140.
141.      void save_params_to_EEPROM();
142.      void load_params_from_EEPROM();
143.      int freeRam();
144.      void int_EXT_setup();
145.      void setCouleur(uint8_t *couleur_i, uint8_t R, uint8_t V,uint8_t  B);
146.      void inc_FRQ_AD9850();
147.      void dec_FRQ_AD9850();
148.      void inc_FRQ_ADF4351();
149.      void dec_FRQ_ADF4351();
150.      void borne_frequence_AD9850();
151.      void borne_frequence_ADF4351();
152.      void inc_offset();
153.      void dec_offset();
154.      void inc_pos_curseur_frq();
155.      void dec_pos_curseur_frq();
156.      void inc_pas_curseur_wob();
157.      void dec_pas_curseur_wob();
```

```
158.      void affiche_entete();
159.      void affiche_freq_marqueur();
160.      void affi_marqueur();
161.      void affi_min_max();
162.
163.      void acquisition_data_WOB();
164.      void scrute_switches();
165.      void calcul_lambda_a();
166.      void calcul_lambda_b();
167.      double recadre(double valeur_i);
168.
169.      void affi_helps();
170.      void trace_sinusoide();
171.      void efface_ecran();
172.      void trace_ecran();
173.      void trace_pannel_graph();
174.      void trace_afficheurs_wob();
175.      void efface_centre_graph();
176.
177.      void affiche_valeurs_registres();
178.      void affiche_lambda_a();
179.      void affiche_lambda_b();
180.      void affiche_params();
181.      void affiche_freq_marqueur();
182.      void affiche_pas_DIV(uint8_t couleur);
183.      void affiche_V_div();
184.      void TFT_affiche_chiffre_7seg(uint8_t num, uint16_t x_i, uint16_t y_i );
185.      void  TFT_aff_nb_form3   (uint32_t  valeur,  uint8_t  nb_chiffres,  uint8_t
   curseur,  uint8_t  R,  uint8_t  G,  uint8_t  B,  uint8_t  SB,  uint16_t  x_i,  uint16_t
   y_i);
186.      void  TFT_aff_nb_form3b  (uint32_t  valeur,  uint8_t  nb_chiffres,  uint8_t
   curseur,  uint8_t  R,  uint8_t  G,  uint8_t  B,  uint8_t  SB,  uint16_t  x_i,  uint16_t
   y_i);
187.
188.
189.      void setup();
190.
191.
192.      #endif
193.
```

17 Klasa AD9850 wersja 1

Kod źródłowy en C

```
/**
        AD9850-628v1.cpp
        Firmware pour piloter un circuit DDS : AD9850
        avec un AVR atmega2560
**/

/*
        Ce fichier source "AD9850-628v1.cpp" est libre, vous pouvez le redistribuer et/ou le modifier selon
        les termes de la Licence Publique Générale GNU.

        Silicium628 - octobre 2018
*/

#include <avr/io.h>
#include <util/delay.h>
#include "AD9850-628v1.h"

#define bitSet(value, bit) ((value) |= (1UL << (bit)))   // 1UL signifie la valeur 1 au format Unsigned Long (32 bits)
#define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
#define bitWrite(value, bit, bitvalue) (bitvalue ? bitSet(value, bit) : bitClear(value, bit))

// Constructeur
AD9850::AD9850() { }


void AD9850::reset()
{
 _delay_ms(1);
  PORT_AD9850 |=  pin_RESET;
 _delay_ms(10);
  PORT_AD9850 &= ~pin_RESET; // l'operateur "~" donne le complement (inverse des bits). ne pas confondre avec l'opérateur "!"
 _delay_ms(10);
}


void AD9850::impulse_clk_W()
{
 _delay_us(5);
  PORT_AD9850 |=  pin_WCL;
```

```
 _delay_us(5);
  PORT_AD9850 &= ~pin_WCL; // l'operateur "~" donne le complement (inverse des bits). ne pas confondre avec l'opérateur "!"
 _delay_us(5);
}


void AD9850::impulse_FQ_U()
{
 _delay_us(5);
  PORT_AD9850 |=  pin_FQU;
 _delay_us(5);
  PORT_AD9850 &= ~pin_FQU; // l'operateur "~" donne le complement (inverse des bits). ne pas confondre avec l'opérateur "!"
 _delay_us(5);
}


/*************************************************************************
f_out = FTW x CLKIN / 2³² nous dit le datasheet
en en déduit :

FTW = f_out * 2³² / CLKIN
ici CLKIN = 125MHz / 2 = 62.5 MHz
125MHZ c'est la fréquence du quartz
cette fréquence est /2 en interne

FTW = f_out * (2³² / 62.5E6 ) = 68.71947674 x f_out

*************************************************************************/

void AD9850::calcul_FTW()
{
      float y;
      y = 6.871947674 * f_out;
      FTW = (uint32_t) y;

}


void  AD9850::out_F(uint32_t  f_out_i)   // le paramètre f_out_i est = à la fréquence en 1/10 Hz c.a.d à 10x fréquence en Hz, ce qui permet une précision du 1/10 Hz
```

```
{
// cette fonction envoie 40 bits vers AD9850; voir le datasheet
      f_out = f_out_i;
      calcul_FTW();

      uint8_t n;

//envoi de la frequence
  uint32_t masque;
  phase =0;

      masque = 1;
  for (n=0;  n<= 31; n++)   // on sort le LSB (W0) en premier
      {
            masque += masque;   // revient à x2 le masque le '1' se deplacant de droite a gauche, 32 bits en tout
            if ( (FTW & masque) != 0) {PORT_AD9850 |=  pin_D7;}  else { PORT_AD9850 &= ~pin_D7; }

                  impulse_clk_W();
            }

       PORT_AD9850 &= ~pin_D7;                     // (W32 toujours = 0)
       impulse_clk_W();

       PORT_AD9850 &= ~pin_D7;        // (W33 toujours = 0)
       impulse_clk_W();

       PORT_AD9850 &= ~pin_D7;        // (W34 = Power-Down = 0)
       impulse_clk_W();

     // envoi de la phase (5 bits)

       for (n=0;  n<=4; n++)    // on sort le LSB (W35) en premier et le MSB en dernier.
            {
                  masque = (1 << n);
                  if  (phase  &  masque)  {PORT_AD9850  |=    pin_D7;}  else { PORT_AD9850 &= ~pin_D7; }

                  impulse_clk_W();
            }
            impulse_FQ_U();
     }

```

```
124.
125.
126.
127.
```

Plik AD9850-628v1.h

Kod źródłowy en C

```
1.  /**
2.          AD9850-628v1.h
3.
4.          Firmware pour piloter un circuit DDS : AD9850
5.          avec un AVR atmega2560
6.
7.  **/
8.
9.
10. #ifndef       AD9850_628V1_H
11. #define       AD9850_628V1_H
12.
13. #pragma       once
14.
15. #define       PORT_AD9850   PORTK
16.
17. #define       pin_WCL             0b01000000          // (sortie)
18. #define       pin_D7        0b10000000          // (sortie)
19. #define       pin_FQU             0b00100000          // (sortie)
20. #define       pin_RESET     0b00010000          // (sortie)
21.
22.
23. class AD9850
24. {
25.
26.
27. public:
28.
29.   uint32_t f_out; // fréquence du signal de sortie sinusoïdal
30.   uint32_t FTW;
31.   uint8_t phase;
32.
33.         double f_outMin = 1; // 1Hz;
34.         double f_outMax = 50000000; // 50MHz
35.
36.
37.         uint8_t DIV;
```

```
38.
39.         AD9850();
40.
41.         void reset();
42.
43.         void out_F(uint32_t f_out_i);
44.
45. private:
46.
47.
48.         void calcul_FTW();
49.         void impulse_clk_W();
50.         void impulse_FQ_U();
51.
52.
53. };
54.
55. #endif
56.
```

18 La klasa ADF4351 wersja 5

Kod źródłowy en C

```
1.  /**
2.         ADF4351-628v5.cpp
3.         Firmware pour piloter un circuit DDS : ADF4351
4.         avec un AVR atmega2560
5.  **/
6.
7.  /*
8.         Ce  fichier  source  "ADF4351-628v5.cpp"  est  libre,  vous  pouvez  le
    redistribuer et/ou le modifier selon
9.         les termes de la Licence Publique Générale GNU.
10.
11.        Un grand merci à Alain Fort F1CJN feb 2,2016
12.        pour son travail :
13.
14.        "ADF4251 and Arduino
15.        update march 7, 2016 (ROBOT V1.1 and V1.0)"
16.        -http://f6kbf.free.fr/html/ADF4351%20and%20Arduino_Fr_Gb.htm
17.        -http://f6kbf.free.fr/html/ADF4351_LCD_07032016.zip
```

```
18.
19.       dont je me suis inspiré pour créer cette objet (class) en C++
20.       J'ai aussi utilisé mon propre travail effectué pour les DDS AD9850 et
   AD9951
21.
22.       Silicium628 - Juillet 2018
23. */
24.
25. #include <avr/io.h>
26. #include <util/delay.h>
27. #include "ADF4351-628v5.h"
28.
29. #define bitSet(value, bit) ((value) |= (1UL << (bit)))   // 1UL signifie la
   valeur 1 au format Unsigned Long (32 bits)
30. #define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
31. #define bitWrite(value, bit, bitvalue) (bitvalue ? bitSet(value, bit) :
   bitClear(value, bit))
32.
33. // Constructeur
34. ADF4351::ADF4351() { }
35.
36.
37. void ADF4351::PORT_ADF_set_bit( uint8_t masque_bit) // masque_bit doit être du
   type 0b00001000 (un seul bit à 1)
38. {
39.       PORT_ADF4351 |= masque_bit;
40. }
41.
42.
43. void ADF4351::PORT_ADF_clear_bit( uint8_t masque_bit) // masque_bit doit être du
   type 0b00001000 (un seul bit à 1)
44. {
45.       PORT_ADF4351 &= ~masque_bit;
46. }
47.
48.
49.
50. void ADF4351::init()
51. {
52.       PORT_ADF_set_bit(pin_LE);
53.       Write_All_Register();
54.       PFDRFout=25;
55.       RFint=7000;
56.       RFout = RFint/100 ; // fréquence de sortie
57.       OutputChannelSpacing = 0.01; // Pas de fréquence = 10kHz
```

```
58. }
59.
60.
61. void ADF4351::mute(uint8_t valeur)
62. {
63.       if     (valeur     ==     1)     {PORT_ADF_clear_bit(pin_CE);}     else
    {PORT_ADF_set_bit(pin_CE);}
64. }
65.
66.
67. void ADF4351::impulse_CLK()
68. {
69.       PORT_ADF_set_bit(pin_CLK);
70.       //_delay_us(1);                                          // le fait de
    supprimer ces tempos augmente grandement la réactivité du circuit.
71.       PORT_ADF_clear_bit(pin_CLK);
72.       //_delay_us(1);
73. }
74.
75.
76. void ADF4351::impulse_LE()
77. {
78.       PORT_ADF_set_bit(pin_LE);
79.       _delay_us(1);
80.       PORT_ADF_clear_bit(pin_LE);
81.       //_delay_us(1);
82. }
83.
84.
85. void ADF4351::out1octet(uint8_t octet_i)
86. {
87. // MSB first (= mode par defaut)
88.       uint8_t i;
89.   uint8_t masque;
90.
91.       for (i=0;  i < 8; i++)
92.       {
93.             masque = 0b10000000 >> i; // le '1' se déplace de gauche a droite
94.             if ( (octet_i & masque) != 0) {PORT_ADF_set_bit(pin_DATA); }
    else {PORT_ADF_clear_bit(pin_DATA); }
95.             impulse_CLK();
96.       }
97. }
98.
99.
```

```
100.      void ADF4351::Write_Register32(uint8_t reg_num)   //Programme un registre
   32bits
101.      {
102.            uint8_t n; // numéro du registre (0..5) codé par les trois bits de
   poids faible
103.            uint8_t octet_i;
104.            uint32_t data;
105.
106.            data = registers[reg_num];
107.            n = data && 0b00000111;
108.            if (data != memo_registers[n]) // on ne programme le registre que
   si son contenu doit changer. (cela accelère l'execution du programme)
109.            {
110.                  memo_registers[n] = data;
111.                  PORT_ADF_clear_bit(pin_LE);
112.
113.                  for (int i = 3; i >= 0; i--) // boucle sur 4 x 8bits
114.                  {
115.                        octet_i=(data >> 8 * i) & 0xFF; // décalage et
   masquage de l'octet
116.                        out1octet(octet_i);
117.                  }
118.                  impulse_LE();
119.      // LE = Load Enable. When LE goes high, the data stored in the 32-bit
   shift register is loaded
120.      // into the register that is selected by the three control bits (ce sont
   les trois bits de poids faible)
121.            }
122.      }
123.
124.
125.      void ADF4351::Write_All_Register()   // Programme tous les registres de
   l'ADF4351
126.      {
127.            for (int i = 5; i >= 0; i--)   // programmation ADF4351 en
   commencant par R5
128.            {
129.                  Write_Register32(i);
130.            }
131.      }
132.
133.
134.      void ADF4351::setFreq(double frequence)  // calcule la valeur des
   registres mais n'envoie rien au circuit ADF4351
135.      {
```

```
136.        RFout = frequence;
137.
138.            registers[4] = 0x8C803C; // cette valeur n'est pas arbitraire. En
    particulier les bits 20,21,22 doivent être =0.
139.      // ce qui est le cas : 0x8C803C = 0b 1000 1100 1000 0000 0011 1100
140.            if (RFout >= 2200)              { DIV = 1; }
141.            else if (RFout >= 1100)  { DIV = 2;           registers[4]      +=
    (1UL<<20); } // 1UL signifie la valeur 1 au format Unsigned Long en memoire
    (uint32_t -> 4octets = 32 bits)
142.            else if (RFout >= 550)   { DIV = 4;           registers[4]      +=
    (2UL<<20); }
143.            else if (RFout >= 275)   { DIV = 8;           registers[4]      +=
    (3UL<<20); }
144.            else if (RFout >= 137.5) { DIV = 16;          registers[4]      +=
    (4UL<<20); }
145.            else if (RFout >= 68.75) { DIV = 32;          registers[4]      +=
    (5UL<<20); }
146.
          else  { DIV = 64;          registers[4] += (6UL<<20); }
147.
148.            INTA = (RFout * DIV) / PFDRFout;
149.            MOD = (PFDRFout / OutputChannelSpacing);
150.            FRACF = (((RFout * DIV) / PFDRFout) - INTA) * MOD;
151.            FRAC = round(FRACF); // On arrondit le résultat
152.
153.            registers[0] = 0;
154.            registers[0] = INTA << 15; // décalage de 15 rangs pour ranger
    cette valeur (16 bits) à sa place dans R0 ; voir p:15 du pdf
155.            registers[0] = registers[0] + (FRAC << 3); // decalage de 3 rangs
    vers la gauche pour le rentrer dans les bits [DB3..DB14]
156.
157.            registers[1] = 0;
158.            registers[1] = MOD << 3; // decalage de 3 pour le rentrer dans les
    bits [DB3..DB14]
159.            registers[1] = registers[1] + 1 ; // ajout de l'adresse "001"
160.            bitSet (registers[1], 27); // PR1-> Prescaler = 8/9 ; voir p:16 du
    pdf
161.
162.            bitSet (registers[2], 28); // Digital lock == "110" sur b28 b27 b26
163.            bitSet (registers[2], 27); // digital lock
164.            bitClear (registers[2], 26); // digital lock
165.
166.      // bitSet (registers[4], 10); //MTLD: Supply current to the RF output
    stage to be shut down until the part achieves lock
167.
```

```
168.            Write_All_Register();
169.      }
170.
171.
172.
173.
```

Plik ADF4351-628v5.h

Kod źródłowy en C

```
1.  /**
2.        ADF4351-628v5.h
3.
4.        Firmware pour piloter un circuit DDS : ADF4351
5.        avec un AVR atmega2560
6.
7.  **/
8.
9.
10. #ifndef      ADF4351_628V5_H
11. #define      ADF4351_628V5_H
12.
13. #pragma      once
14.
15. #define      PORT_ADF4351 PORTF
16.
17. #define      pin_CLK                  0b10000000          // (sortie)
18. #define      pin_DATA           0b01000000          // (sortie)
19. #define      pin_LE             0b00100000          // (sortie)
20. #define      pin_MUXOUT   0b00010000          // (*ENTREE*)
21. #define      pin_CE             0b00001000          // (sortie)
22.
23. class ADF4351
24. {
25.
26. public:
27.
28.       uint32_t  registers[6]  =  {0x4580A8,  0x80080C9,  0x4E42,  0x4B3,  0xBC803C,
    0x580005};
29.       uint32_t memo_registers[6] = {0, 0, 0, 0, 0, 0};
30.
31.       uint32_t RFint, RFintold, INTA, RFcalc, PDRFout, MOD, FRAC;
32.
```

```
33.       double RFout, REFin, PFDRFout, OutputChannelSpacing, FRACF;
34.
35.       double RFoutMin = 35; // 35 MHz
36.       double RFoutMax = 4400; // 4400MHz = 4.4 GHz
37.       double REFinMax = 250;
38.
39.       uint8_t DIV;
40.
41.       ADF4351();
42.
43.       void init();
44.       void mute(uint8_t valeur);
45.       void setFreq(double frequence);  // en MHz (avec decimales) -> calcule la
   valeur des registres mais n'envoie rien au circuit ADF4351
46.       void  Write_Register32(uint8_t  reg_num);     //Programme  physiquement  un
   registre 32bits. SetFreq() doit avoir été appelé avant.
47.       void  Write_All_Register(); // Programme  physiquement  tous  les  registres
   de l'ADF4351
48.
49. private:
50.
51.       bool _powerdown, _auxEnabled, _rfEnabled, _feedbackType;
52.       void PORT_ADF_set_bit( uint8_t masque_bit);
53.       void PORT_ADF_clear_bit( uint8_t masque_bit);
54.       void impulse_CLK();
55.       void impulse_LE();
56.       void out1octet(uint8_t octet_i);
57.
58. };
59.
60. #endif
61.
```

Wielkie podziękowania dla Alain Fort F1CJN luty 2,2016

za jego pracę:

„ADF4251 i Arduino

aktualizacja 7 marca 2016 r. (ROBOT V1.1 i V1.0) ”

-http: //f6kbf.free.fr/html/ADF4351%20and%20Arduino_Fr_Gb.htm

-http: //f6kbf.free.fr/html/ADF4351_LCD_07032016.zip

którego użyłem do utworzenia tego obiektu (klasy) w C ++

Korzystałem również z własnej pracy wykonanej dla DDS AD9850 i AD9951

Krzem 628 - lipiec 2018

19 Evolution

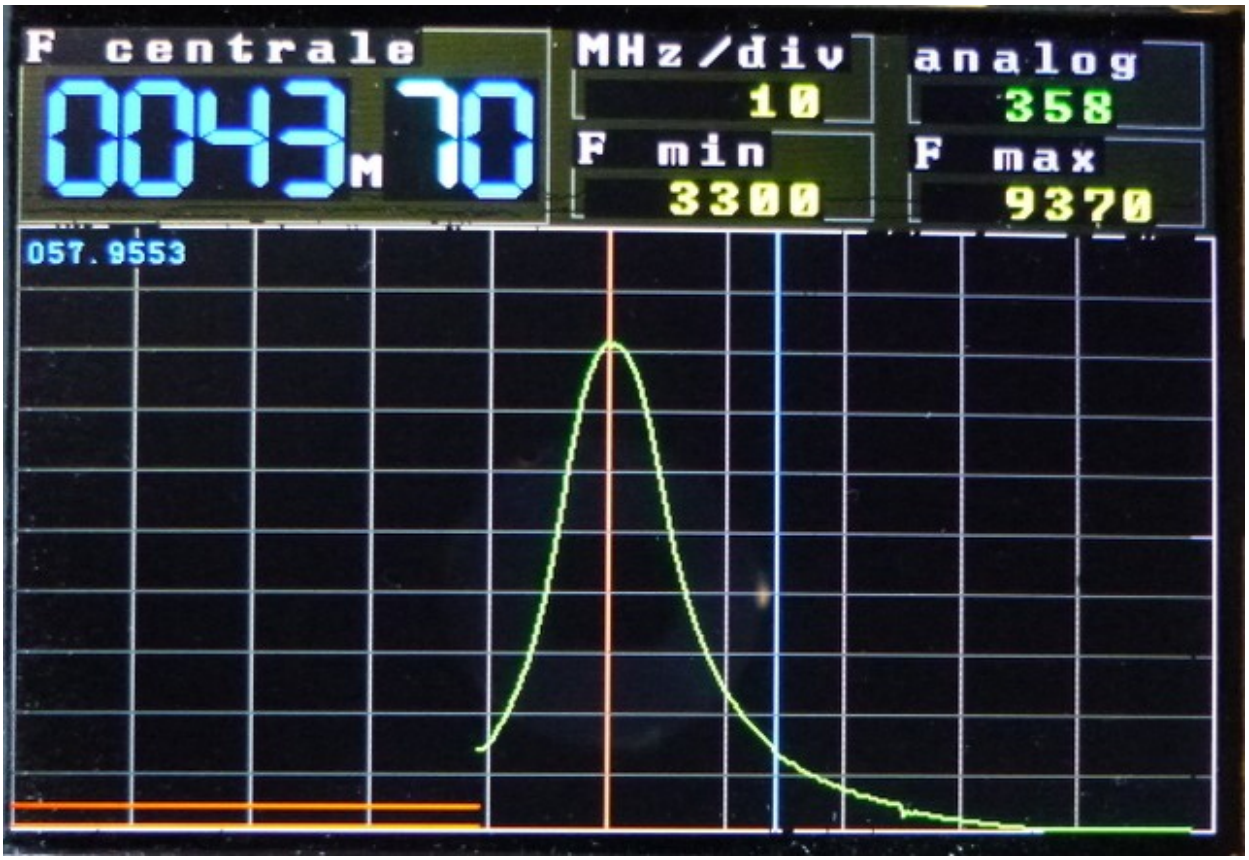
3 lipca 2018: codziennie aktualizuję to urządzenie i publikuję aktualizacje programu wielodniowego ...

Odwróciłem więc wyświetlanie krzywej (było „do góry nogami”) i dodałem dokładny znacznik częstotliwości, nałożony, ruchomy i bardzo dokładny (100 Hz).

Należy również zauważyć, że na razie nie gwarantuję wartości składników części analogowej, opublikuję nowy diagram wkrótce, gdy te wartości będą ostateczne.

20 Réponse fréquentielle d'un circuit LC

20 Pasmo przenoszenia obwodu LC



Jest to obwód LC (indukcyjność w / na kondensatorze) zwany „korkiem obwodu” w rzeczywistości RLC, ponieważ doskonałe komponenty nie istnieją i obwód jest atakowany przez opór 1K. ..

Obwód ten mówi się o drugim rzędzie, tj. Jest opisany równaniem różniczkowym drugiego rzędu. Przestudiowaliśmy ten obwód: OBWÓD RLC

Na zdjęciu zauważamy:

pionowa niebieska linia (przy 58 MHz) jest markerem

wyświetlanie częstotliwości znacznika (małymi niebieskimi znakami)

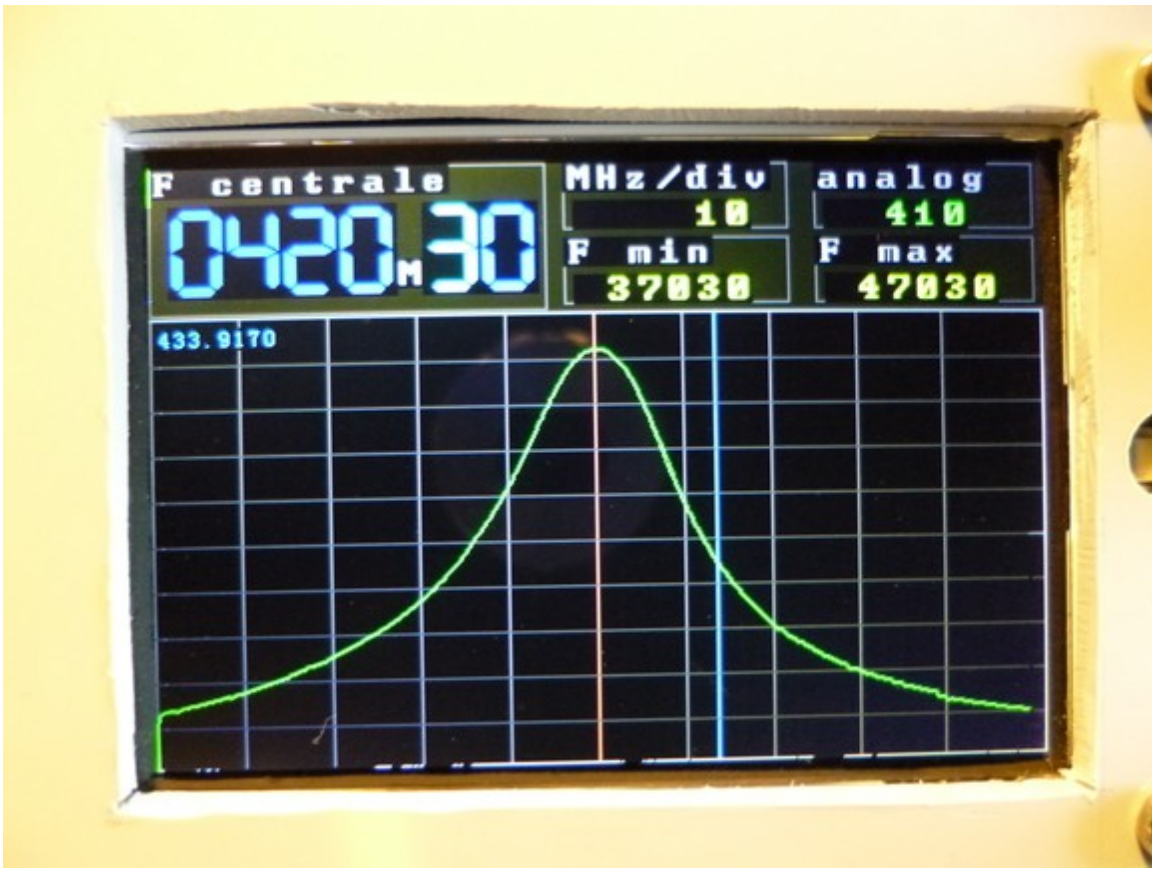
czerwona pozioma linia na dole wskazująca strefę niepokrytych częstotliwości (<33MHz)

Etykieta „No WOB” została zastąpiona przez „MHz / div”, co jest bardziej wymowne.

Dlatego zbadamy tutaj domenę najniższych częstotliwości dostępnych dla urządzenia. Zobaczymy, co stanie się w UHF poza GHz.

21 Tests en UHF

21 testów UHF



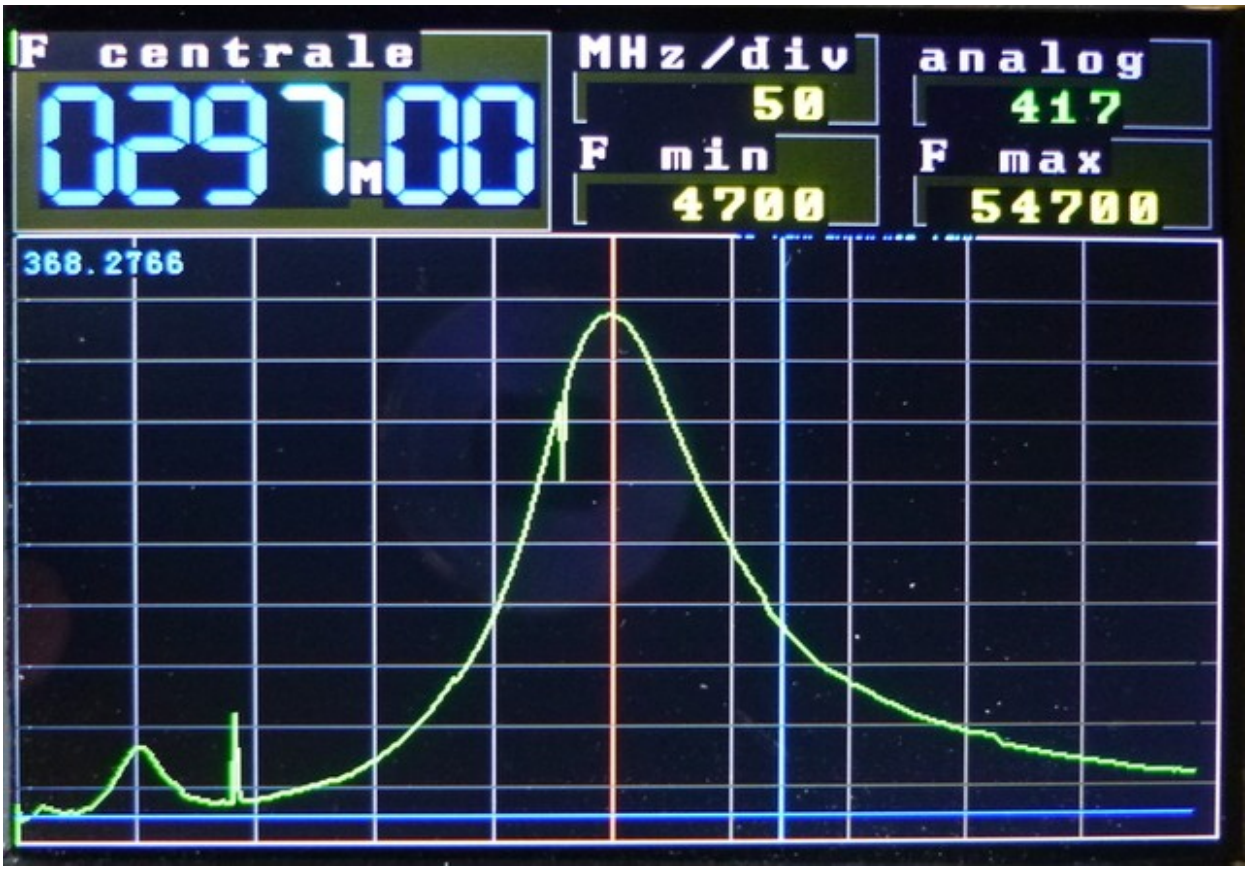
Jak widać na pokazie slajdów u góry tego artykułu, przesunąłem testy obwodu LC do prawie 700 MHz (z detektorem diodowym 1N6263) Po 500 MHz krzywa odpowiedzi rozszerza się i oczywiście z dala od oczekiwanej krzywej dzwonowej. Należy powiedzieć, że dla tych częstotliwości rozproszone indukcyjności i pojemności stają się przeważające w porównaniu z tymi składnikami. (Użyłem dławików 0802 cm, ale małe, ale nie cmowe kondensatory ceramiczne i obcięcie końcówek kondensatora skróciły częstotliwość o prawie 100 MHz!). A potem zbliżamy się do granicznej częstotliwości używanych diod detekcyjnych (1 GHz). Studiuję rozwiązanie, aby osiągnąć 4 GHz.

2 suite...

22 więcej ...

1 sierpnia 2018:

Dodałem nowe funkcje

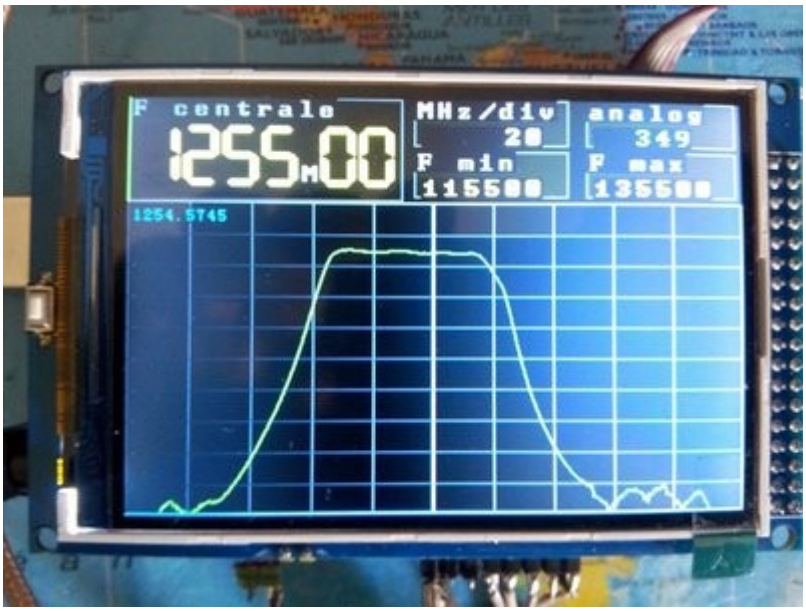


OM, który niektórzy z was znają na pewno, Laurent de F4FDW, któremu dziękuję tutaj, widząc, że ta strona skontaktowała się ze mną i od tego czasu pomaga mi ulepszyć to urządzenie, aby było przydatne dla amatorów i znaleźć rozwiązania problem wykrywania w SHF. Co więcej, eksperymentował z obecną realizacją i wysłał mi obiecujące obrazy pokazujące reakcję filtrów do ponad 2 GHz. Laurent, po swojej stronie użył już ADF4351, jak widać na jego stronie:

http://f4fdw.free.fr/

23 Les résultats obtenus par Laurent de F4FDW

23 Wyniki uzyskane przez Laurenta de F4FDW



26 sierpnia 2018 r .:

Oto wynik z 23 cm filtrem pasmowym (1300 MHz)

Inne równie obiecujące wyniki zostały uzyskane przez Laurenta, na przykład w paśmie 13 cm (2,3 GHz)

Ale zapraszam do odkrycia tego wszystkiego bardziej szczegółowo na dedykowanej stronie jego strony:

http://f4fdw.free.fr/gene_wobul.html

24 Version (v7_3) – Couleurs

24 Version (v7_3) – Kolory

28 sierpnia 2018 r .:

Dodałem możliwość regulacji wzmocnienia i przesunięcia y podczas pracy, działając na enkodery obrotowe.

Dostęp jest możliwy po przełączeniu nowego (trzeciego tak) przełącznika.

Mówię o strojeniu wzmocnienia, powinienem powiedzieć współczynnik wzmocnienia

oprogramowania regulacyjnego, ponieważ działa on w skali liniowej, a nie logarytmicznej. I zmniejszając wartość bezwzględną, osiągamy zero i przekraczamy ją, aby uzyskać wartości ujemne (-1, -2, ...), które są pożądane i mają wpływ na odwrócenie wykresu (symetria wzdłuż osi Y)) co stało się konieczne, ponieważ obecnie testujemy F4FDW i I, różne detektory liniowe lub logarytmiczne, których odpowiedzi mają różne polaryzacje.

Wyświetlane są nowe wskazania:

1 Trzy małe kolorowe „diody” wskazujące pozycje 3 przełączników

2 Przypomnienia, w pozycji pionowej na prawej krawędzi ekranu, funkcja enkoderów obrotowych, biorąc pod uwagę aktualną pozycję przełączników. Wiemy w ten sposób, co zmienimy, zanim zmienimy pryszcze

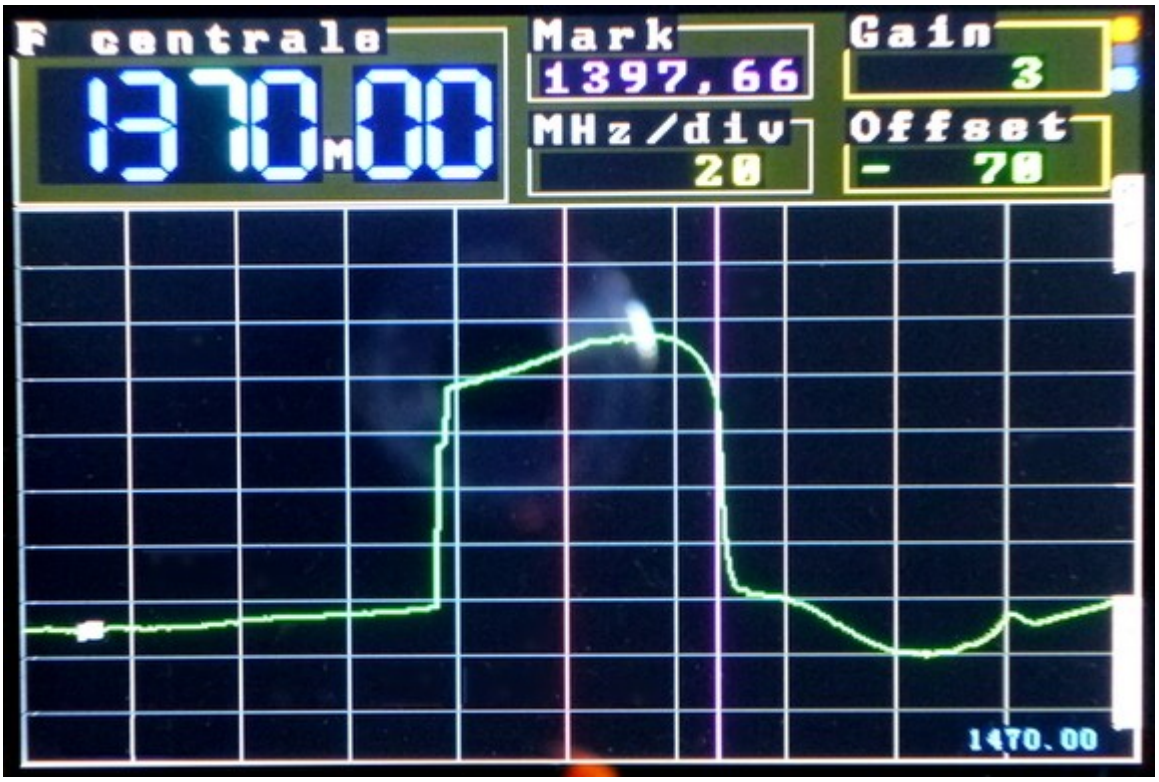
3 Małe numeryczne ramki wyświetlania zmieniają kolor w zależności od położenia przełączników i migają krótko podczas zmiany trybu, aby łatwo zlokalizować edytowane.

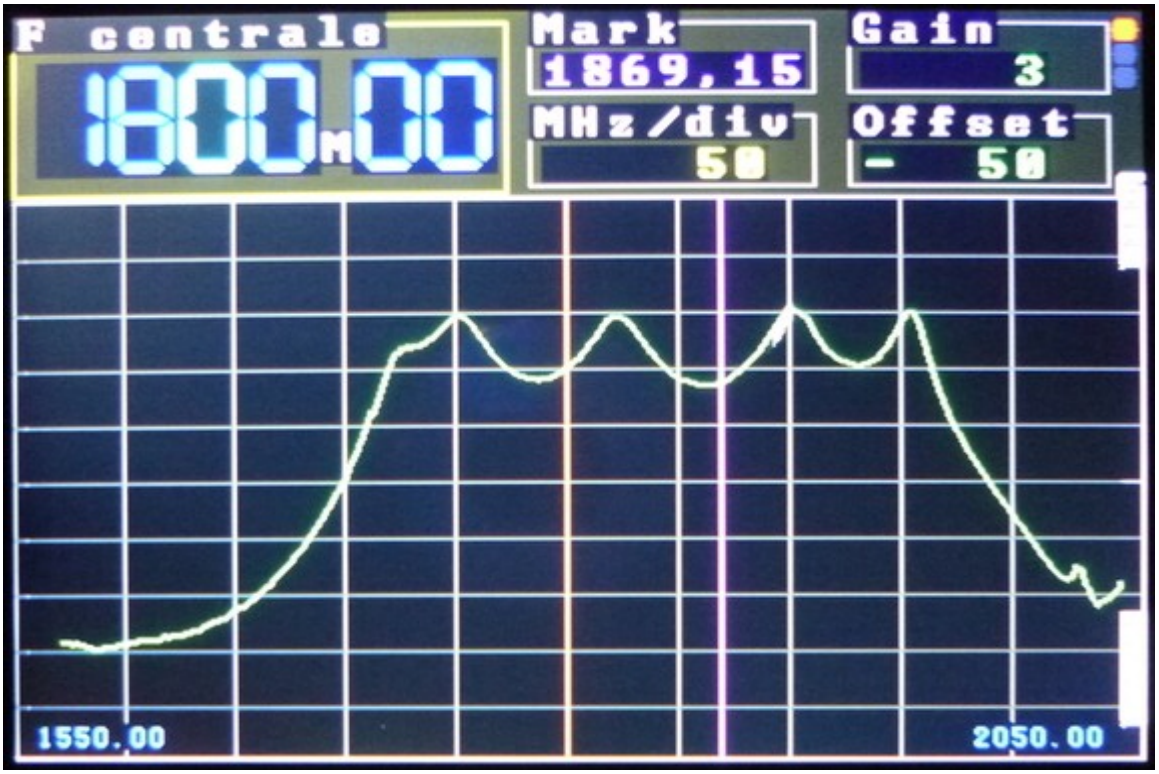
Korzystałem z okazji, aby całkowicie przepisać niektóre funkcje, zwłaszcza przetwarzanie przerwań sprzętowych.

Otrzymałem także nowe komponenty do wykrywania mikrofal (detektor logów AD8318), które umieściłem na diagramie i które zaczynają dawać mi interesujące wyniki.

25 Résultats obtenus avec le détecteur logarithmique AD8318

25 Wyniki uzyskane za pomocą detektora logarytmicznego AD8318





4 września 2018 r .:

Oto po lewej stronie krzywa prześledzona podczas analizy filtra „poważnego” (typ

„Filtr linii grzebieniowej” marki RLC ELECTRONICS) po graniu przez jedną godzinę za pomocą 7 małych śrub regulacyjnych dla aby sprawić, by działał w amatorskim zespole 23 cm, bez naprawdę udanego sukcesu, nie wydaje się, aby był on zawzięty tuż po zakończeniu tego zespołu. Nasz wobulator doskonale nadaje się do tego rodzaju operacji. Będę jednak starał się zwiększyć szybkość skanowania, aby uczynić jego użytkowanie bardziej wygodnym i zaoszczędzić czas podczas takiej operacji ustawiania.

Zdjęcie po prawej pokazuje ten sam zestaw filtrów w oryginalnym paśmie (od 1800 do 1900 MHz). Ten filtr odzyskiwania, który znaleźliśmy w eBayu jako jego bliźniacy, prawdopodobnie służył w przekaźniku GSM.

27 Version (v8_4) - Switch trois positions en remplace deux switches

27 Wersja (v8_4) - Przełącznik trójpozycyjny do wymiany dwóch przełączników

2 września 2018 r .:

Jak widać na schemacie, zastąpiłem dwa przełączniki używane do wyboru trybu wejścia tylko jednym do 3 poz. Te dwa przełączniki dwupołożeniowe mogą logicznie przyjąć cztery różne stany, podczas gdy są tylko trzy tryby wprowadzania:

częstotliwość napadów

przejąć pozycje znacznika i MHz / div

Wejście wzmocnienia i przesunięcia

Skąd szybko zabrano głowę !!

Przełącznik z trzema stabilnymi pozycjami mechanicznymi jest znacznie lepszy i rozsądnie ułożony odzwierciedla wyświetlanie wybranego trybu.

To główna modyfikacja tej nowej wersji v8_4.

Ale wiele ulepszeń pozostaje do zrobienia, takich jak:

kalibracja i skalowanie w dB ekranu wobulacji (do użycia z logarytmicznym

detektorem obwiedni sygnału, takim jak AD8318)

Wyświetlanie całego znacznika podczas ruchu, bez kasowania wykresu krzywej (wymaga zapamiętania krzywej w pamięci RAM między dwoma skanami, zobaczę, czy 8 K bajtów SRAM ATmega 2560 są wystarczające (oprócz zmiennych programu), a priori tak, głównie.)

28 Version (v8_5) - Mémorisation de la courbe en SRAM – Marqueur

Zgodnie z oczekiwaniami, przestrzeń SRAM w AT2560 była wystarczająco duża, aby uchwycić wszystkie punkty tworzące wobulator, w ten sposób umieszczając (i przesuwając) ślady na krzywej bez jej uszkodzenia. czasami nazywamy „sprite” lub warstwy warstwowe. W superpozycji jest to przerysowanie tła, gdy obiekt na pierwszym planie się poruszył.

Najpierw skorzystaj z tego: fioletowa pionowa linia symbolizująca znacznik częstotliwości może przemieszczać się pionowo po całym ekranie bez wymazywania części krzywej po przesunięciu w bok.

Nawiasem mówiąc, to ustawienie w pamięci pozwoli:

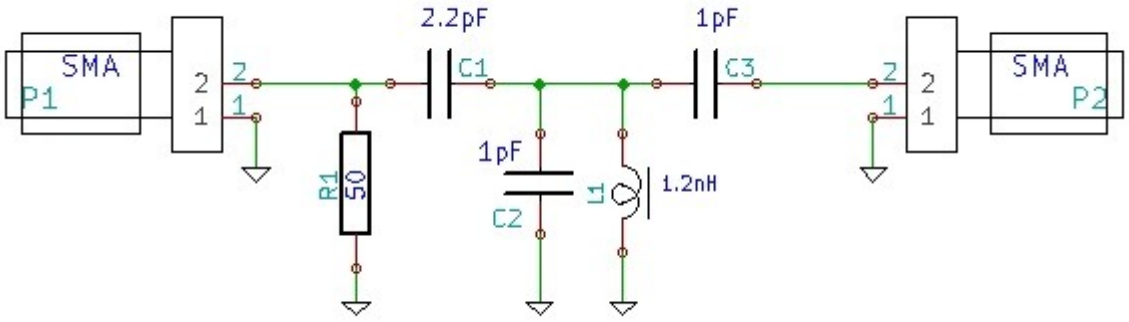
przywołać zapamiętaną krzywą odpowiedzi dla porównania z bieżącą.

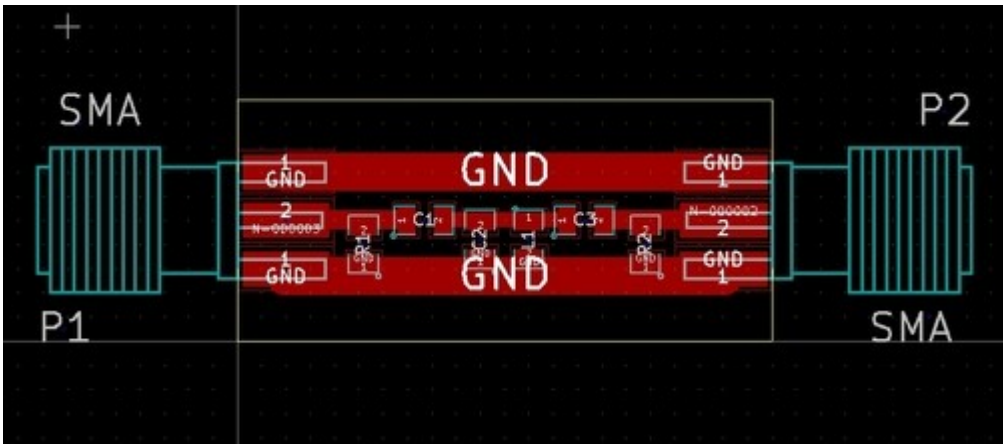
wysyłaj dane ze skanowania na komputer (lub gadżet z systemem Android) za pomocą szybkiego łącza szeregowego USB do przeglądania, analizy, drukowania ...

To wszystko, jest więcej niż ...

29 Filtre en Pi & Utilisation d'un détecteur linéaire à diode schottky

29 Filtr pi i wykorzystanie liniowego detektora diody Schottky'ego



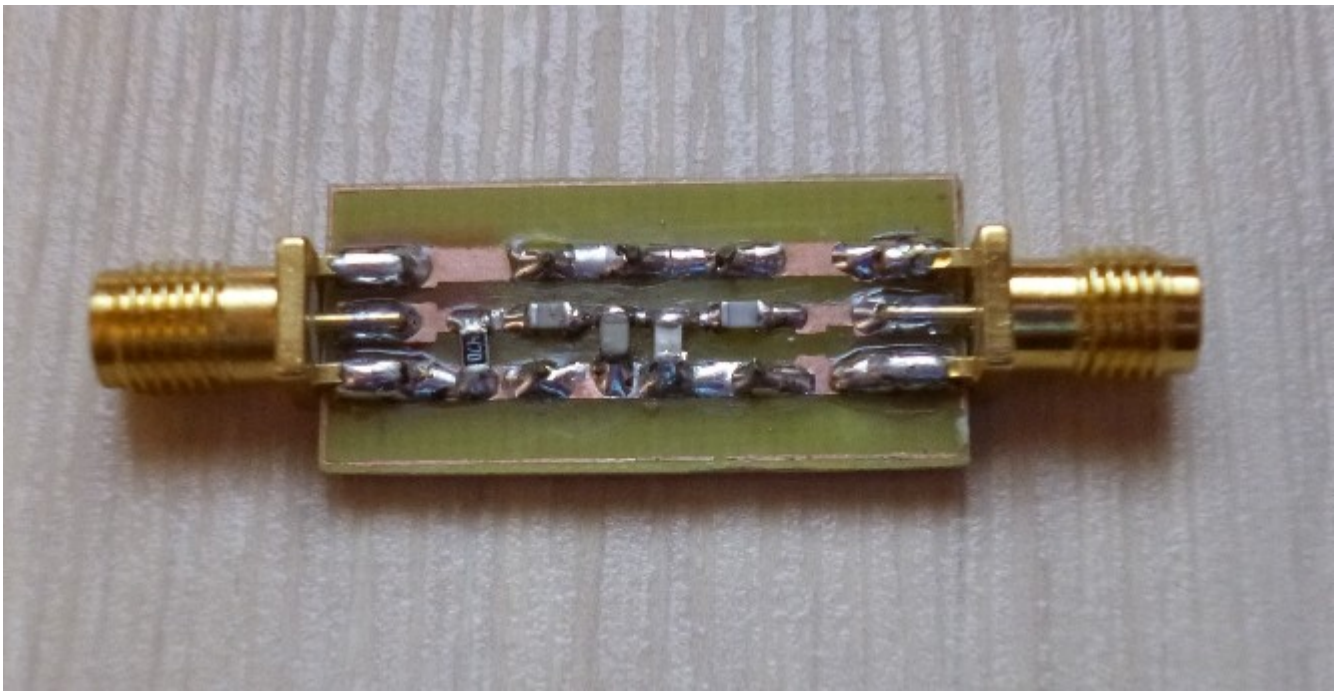


4 września 2018:

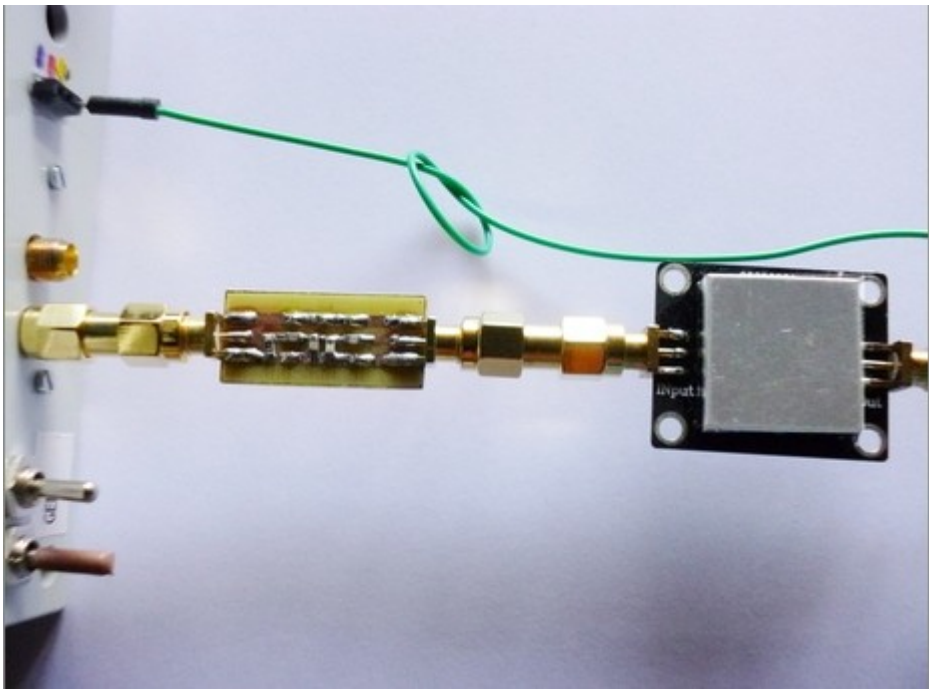
Przetestowałem wobulator za pomocą pasywnego detektora liniowej diody Schottky'ego zakupionego w serwisie eBay. Działa prawidłowo do ponad 3 GHz. Jego sygnał wyjściowy nie jest zatem logarytmiczny, dlatego jest znacznie mniej czuły niż AD8318. W części przeciwnej nie wymaga zasilania. Jeśli chodzi o niższą czułość, może to być zaletą, gdy testowanie obwodów nie jest lub niewystarczająco ekranowane eksperymentalnie, takie jak ten domowy filtr w pi, wykonany na dwustronnej płytce drukowanej z elementami cms. W tym przypadku AD8318 „powąchał” wiele bardzo niskich poziomów, co sprawiło, że ogniskowanie było niezwykle trudne. Jednak detektor logarytmiczny używany z profesjonalnym sprzętem będzie wykazywał wyższość.

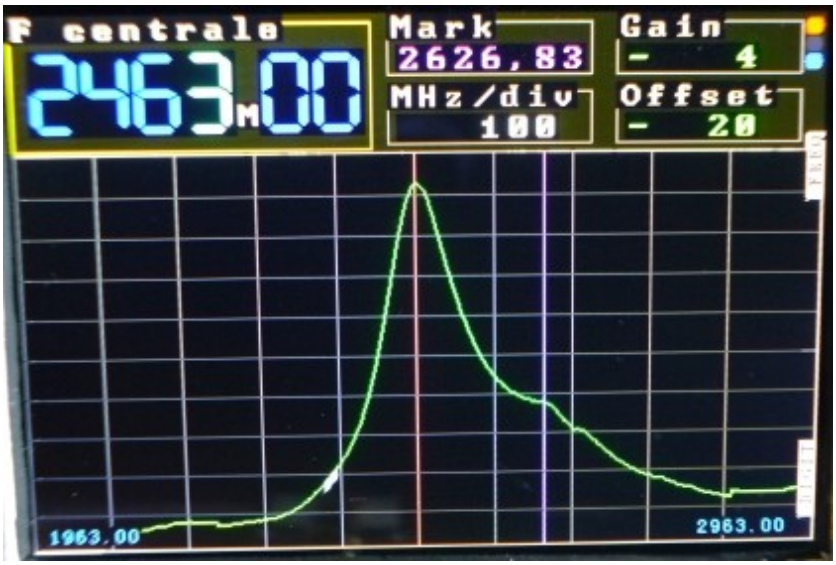
30 Détails de ce filtre SHF en composants cms

30 Szczegóły tego filtra SHF w komponentach cms



Ten filtr odmówił poprawnej pracy, dopóki nie dodałem funkcji via łączącej ścieżki GND z płaszczyzną uziemienia drugiej strony. Przy tych częstotliwościach, co może wydawać się tylko szczegółem, jest w rzeczywistości konieczne i nieuniknione.





est to prawdopodobnie granica tego, co można zrobić z dyskretnymi składnikami, nawet jeśli są to cms, indukcyjność jest rzędu nanorurek i kondensatora rzędu pF. To było po prostu „zobaczyć”! Oczywiście zintegrowane komponenty na chipie mogą zamontować o rząd wielkości wyższe, co można łatwo zauważyć (np. Systemy wideo 5,8 GHz lub radar zbliżeniowy 10 GHz, LNB do odbioru satelitarnego, nie wspominając o naszych małych syntezatorach częstotliwość ...)

32 Version (v9_0) - échelle logarithmique en dB/div

Oto funkcja obiecana przez długi czas: Ustawienie „wzmocnienia” staje się ustawieniem liczby dB / podziału.

Postęp ustawiania jest następujący:

-1 dB / div

-2 dB / div

-5 dB / div

-10 dB / div

Ten wyświetlacz dB ma sens tylko przy użyciu detektora mocy shf, takiego jak AD8318 (25mV / dB). Gdy inna czujka ma inną czułość, będziesz musiał edytować wartość w programie głównym (w funkcji „crop ()”)

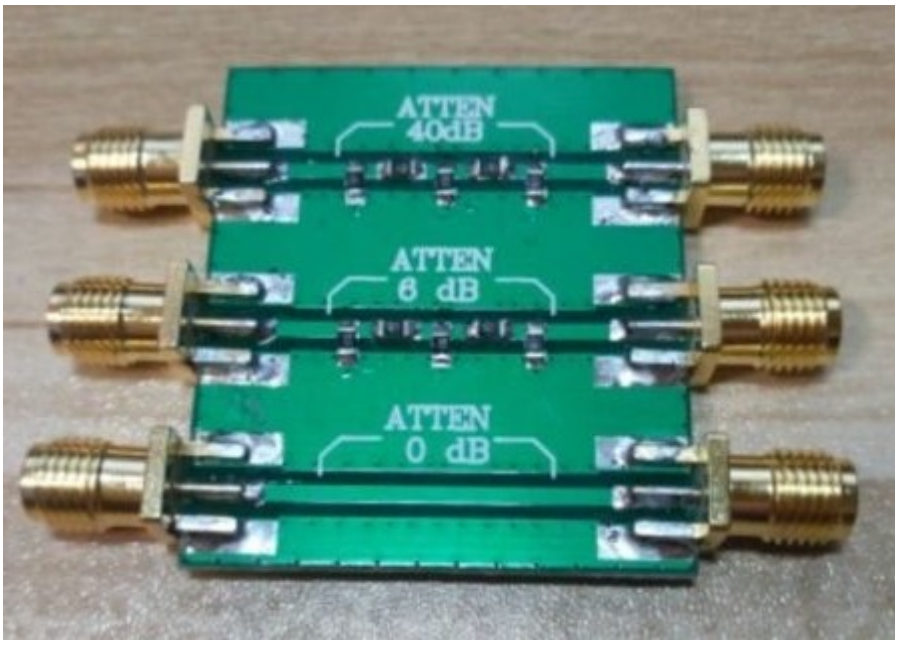
Z drugiej strony musiałem rozszerzyć zakres regulacji przesunięcia z -2000 na +2000, tak abyśmy mogli powiększyć szczegół, będąc na wysokiej czułości (na przykład 1 dB / div). Widzimy małe kamyki na szczycie wielkiej góry, gdzie nawet małe źdźbło trawy u jej stóp.

Zmienna („polatite”) na samym początku programu głównego pozwala (przed kompilacją) odwrócić krzywą zgodnie z biegunowością sygnału wyjściowego detektora.

W przypadku użycia detektora liniowego (diody Schottky'ego) sygnały będą wyświetlane poprawnie, po prostu jednostka „dB / div” nie będzie miała znaczenia.

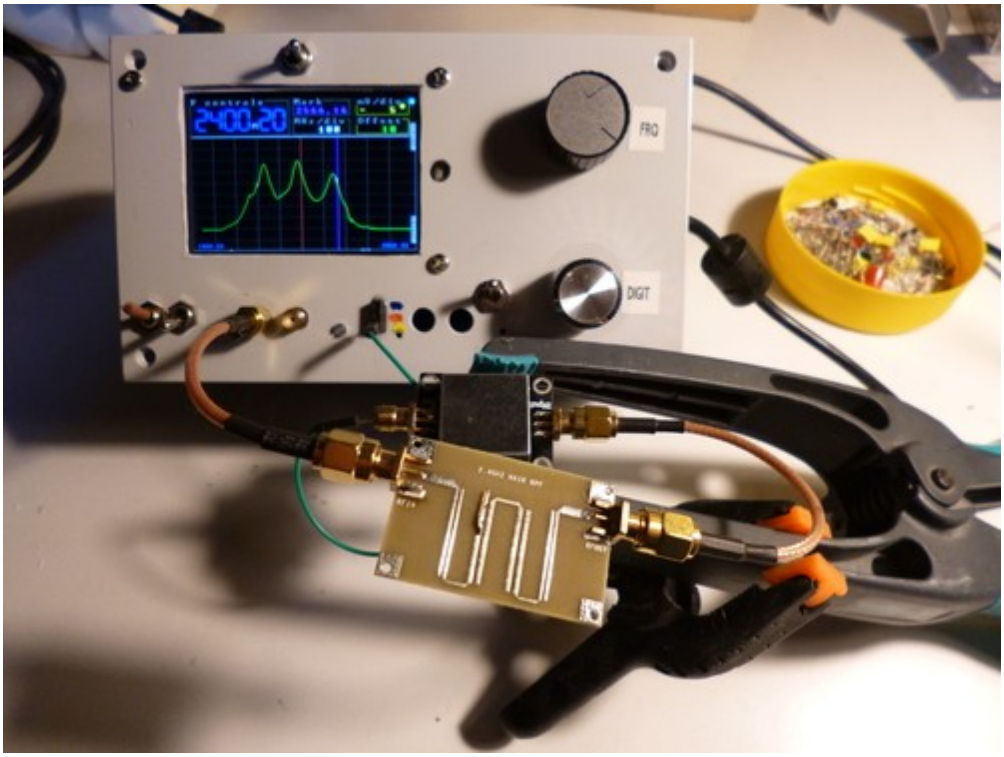
Co jeszcze mogę powiedzieć? Cóż, musimy to wszystko przetestować! (Zwłaszcza dokładność podziałki, coś dla OM, które ... Tęsknię za tym, że teraz jest niezawodnym tłumikiem 10dB w shf, to jest dowodzenie)

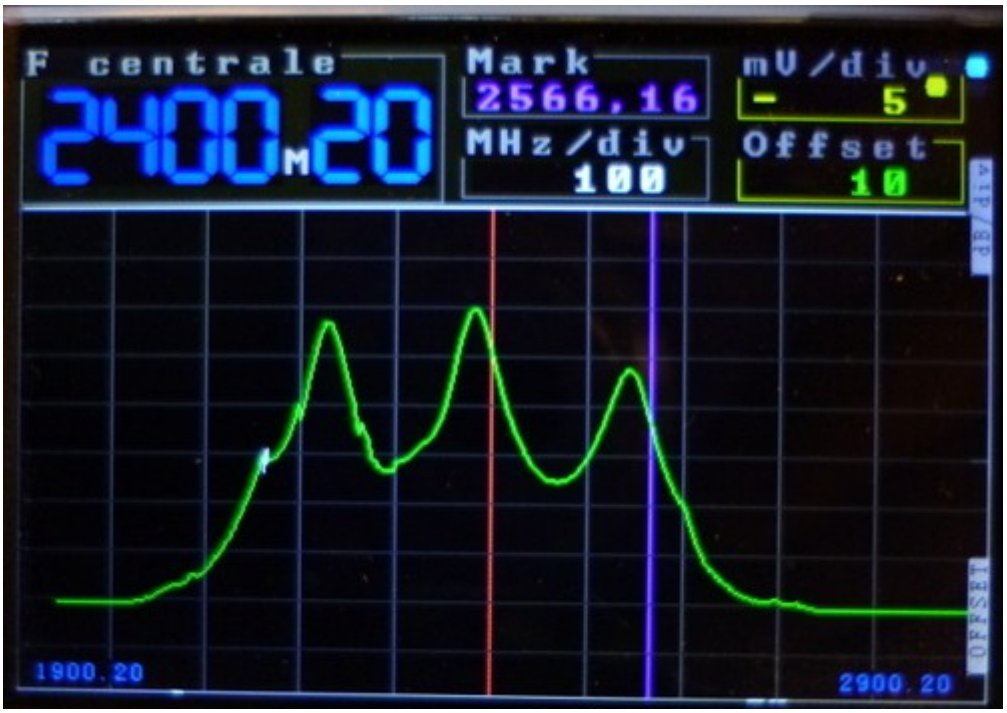
Myślę, że pierwszą rzeczą, jaką zrobię, gdy ją otrzymam, jest przekształcenie dolnej (0 dB ... dzięki) w 10dB lub 20dB.



33 Version (v9_1) - Deux types de détecteurs

33 Wersja (v9_1) - Dwa typy czujek

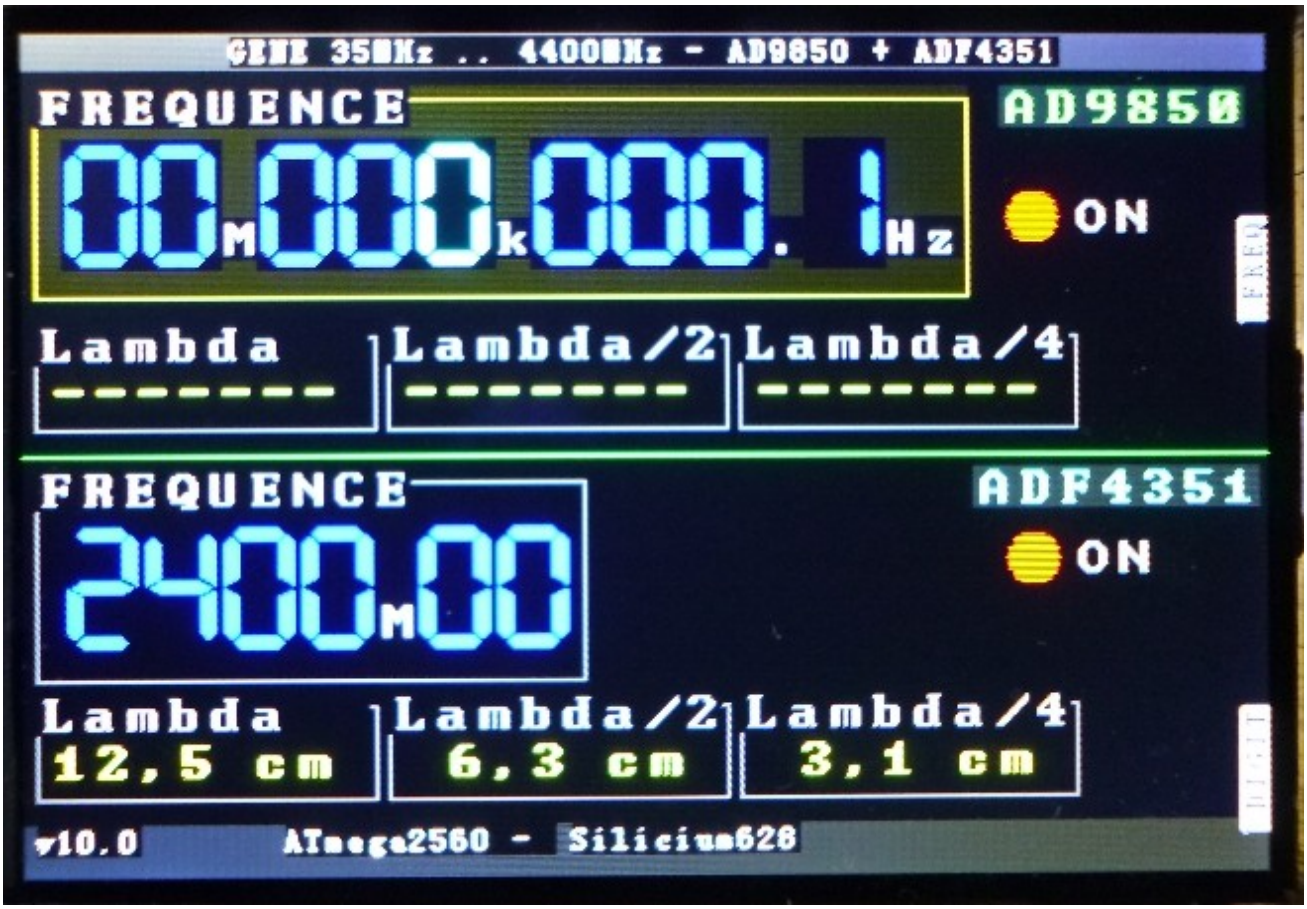




Dwa typy detektorów (liniowy i logarytmiczny) z powiązanym przetwarzaniem numerycznym i wyświetlaniem (w dB / działce i mV / div) są teraz brane pod uwagę. Powyżej przedstawiono odpowiedź filtra pasmowego z centralnym pasmem 2,4 GHz podłączonego do liniowego detektora pasywnego.

34 Version (v10_1) Ajout d'un DDS AD9850

34 Wersja (v10_1) Dodawanie DDS AD9850



20 października 2018:

Dodanie obwodu DDS (bezpośrednia synteza sygnału sinusoidalnego) AD9850 oprócz ADF4351. Generowane częstotliwości wahają się obecnie od 33 MHz do 4,4 GHz, ale od 0,1 Hz do 4,4 GHz.

Od dawna opisuję AD9850 na tej stronie, w tym artykule. Ale dla tego nowego zastosowania przepisałem kod w postaci „klasy” C ++, tak aby cały kod był bardziej spójny, co ułatwi późniejsze wykorzystanie tego komponentu.

Na razie tylko ADF4351 może być wobulowany. W przypadku AD9850 nie będzie to długo.

śledzić ...

35 Version (v10_2) Wobulation permise également p

35 Wersja (v10_2) Wobulacja dozwolona również dla AD9850

28 października 2018:

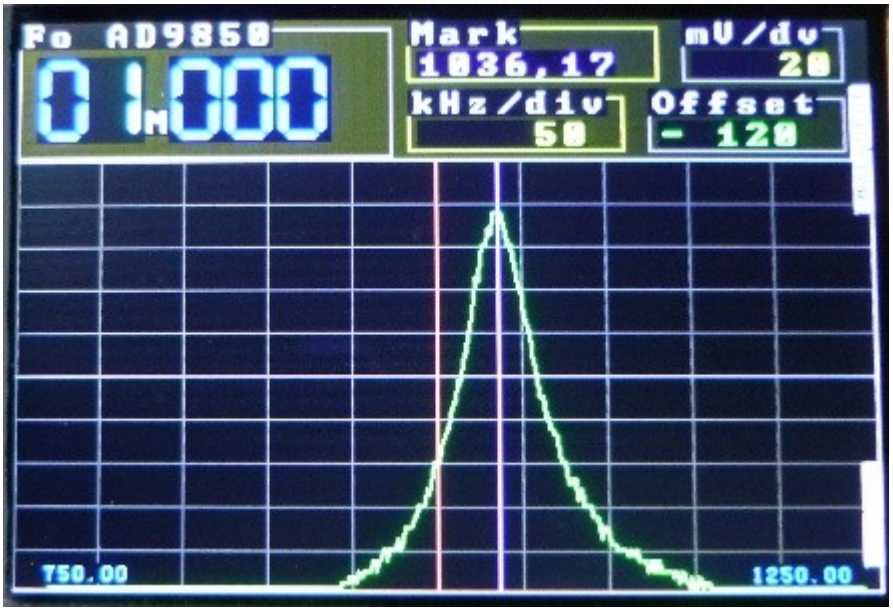
AD9850, który pokrywa od 0,1 Hz do 50 MHz, jest w końcu chwiejny jak jego starszy brat ADF4351.

Zauważmy, że AD9850, który jest DDS, generuje sygnał sinusoidalny, którego forma jest bardziej precyzyjna, ponieważ częstotliwość jest niska. Wręcz przeciwnie, ADF4351 generuje sygnał sinusoidalny tylko w paśmie 2,2 GHz przy 4,4 GHz pochodzący bezpośrednio z jego wewnętrznego VCO, ale niższe częstotliwości (od 33 MHz do 2,2 GHz) są uzyskiwane przez logiczne dzielniki, w postaci sygnałów prostopadle. kwadrat! c.a.d dostatecznie dostarczane w harmonicznych, towary widoczne dla analizatora widma. Kwadratowy kształt jest widoczny tylko przy pochyleniu do kilkuset MHz, znalazłem, to kwadrat kwadratowy.

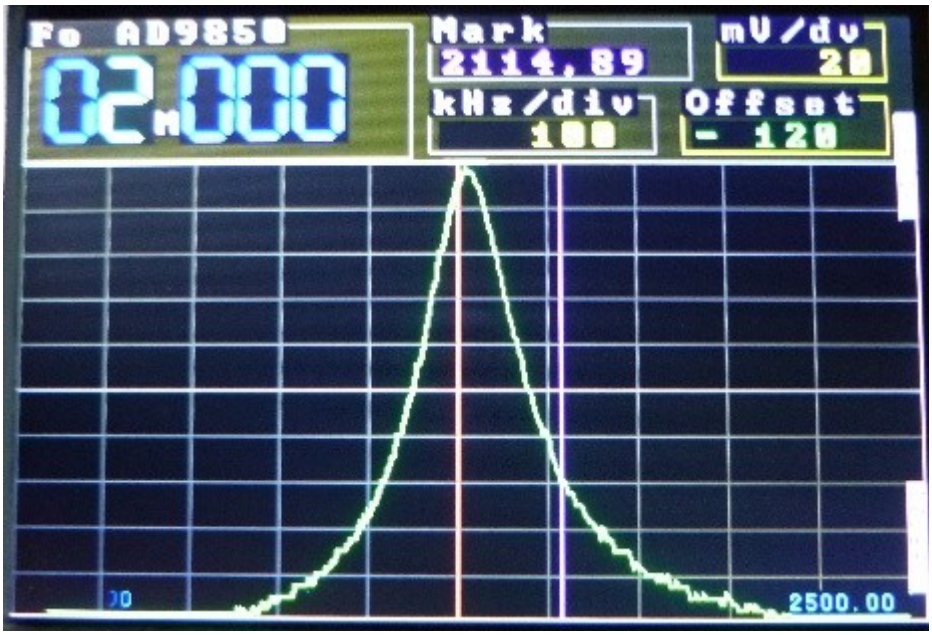
Powiemy, że te dwa obwody się uzupełniają.

6 Tests de wobulation en mode AD9850

6 testów wobulacji w trybie AD9850



LC-56uH-390pF

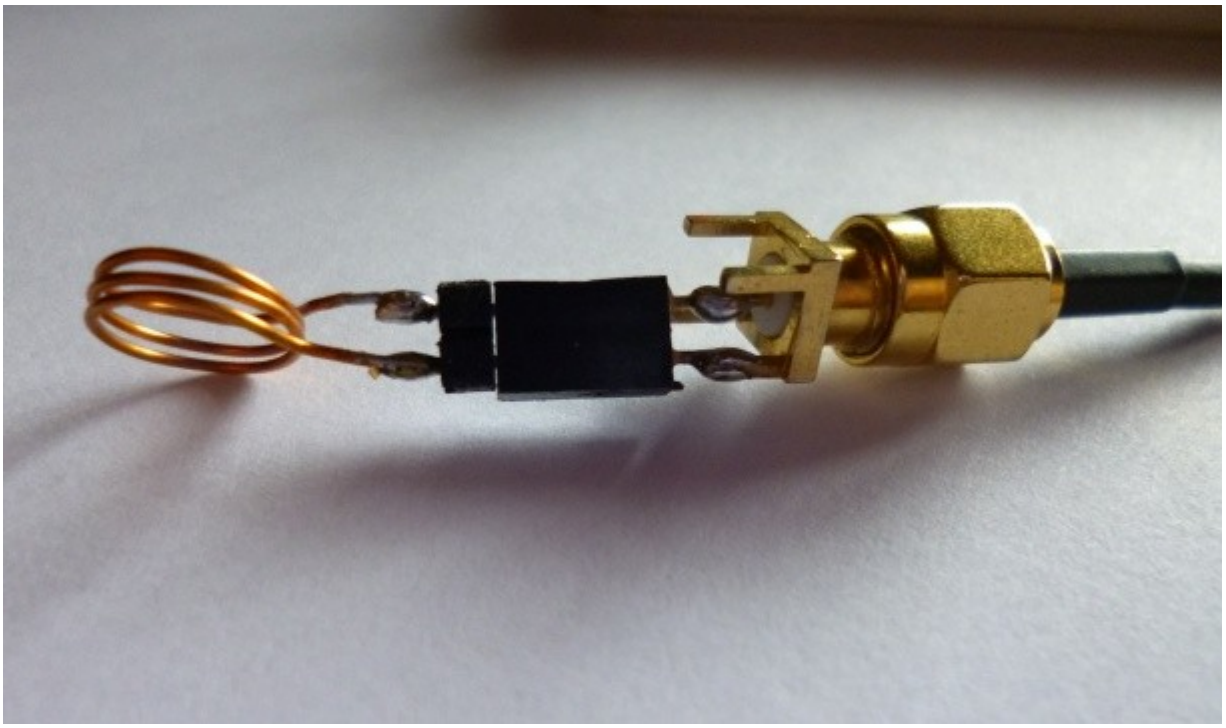


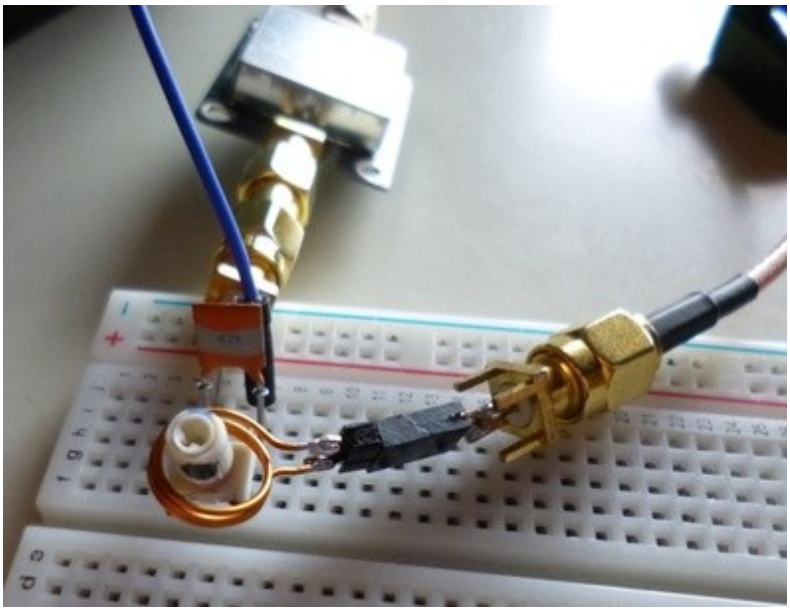
LC-15uH-390pF

2 listopada 2018:

Oto wyniki uzyskane z różnymi obwodami LC obejmującymi zakres od 1 MHz do 20 MHz przy użyciu części urządzenia AD9850.

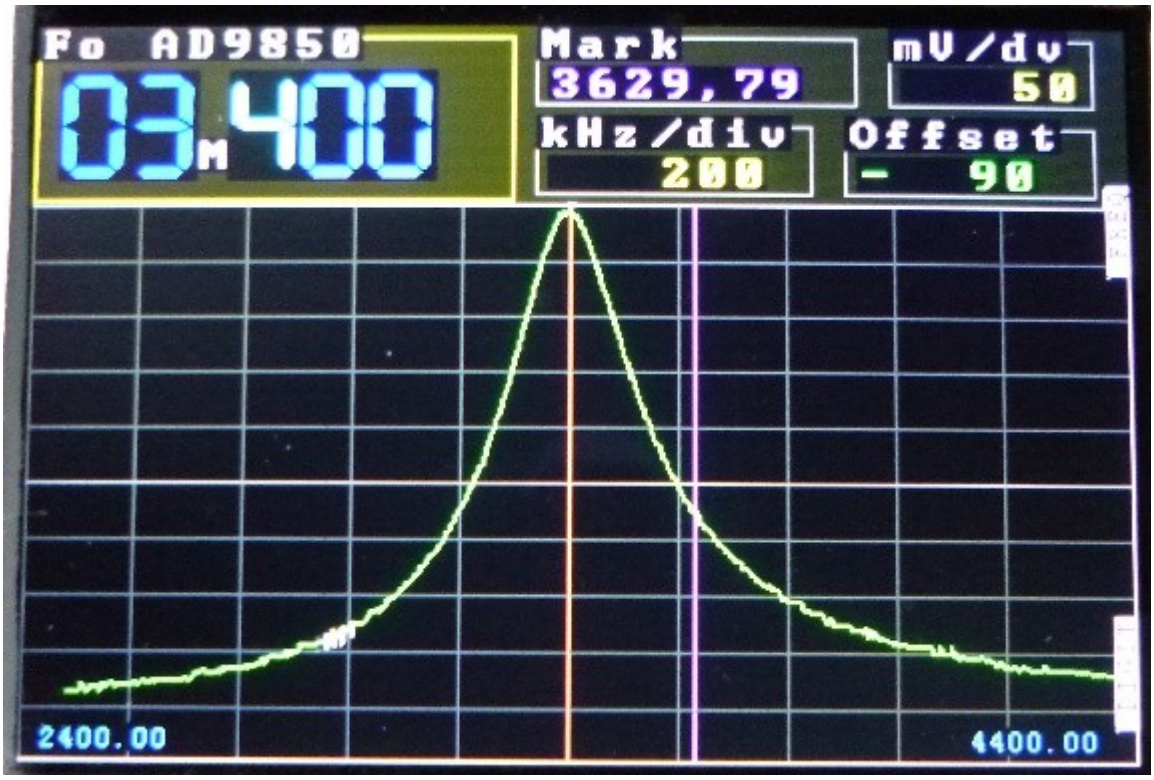
37 La bobine d'excitation



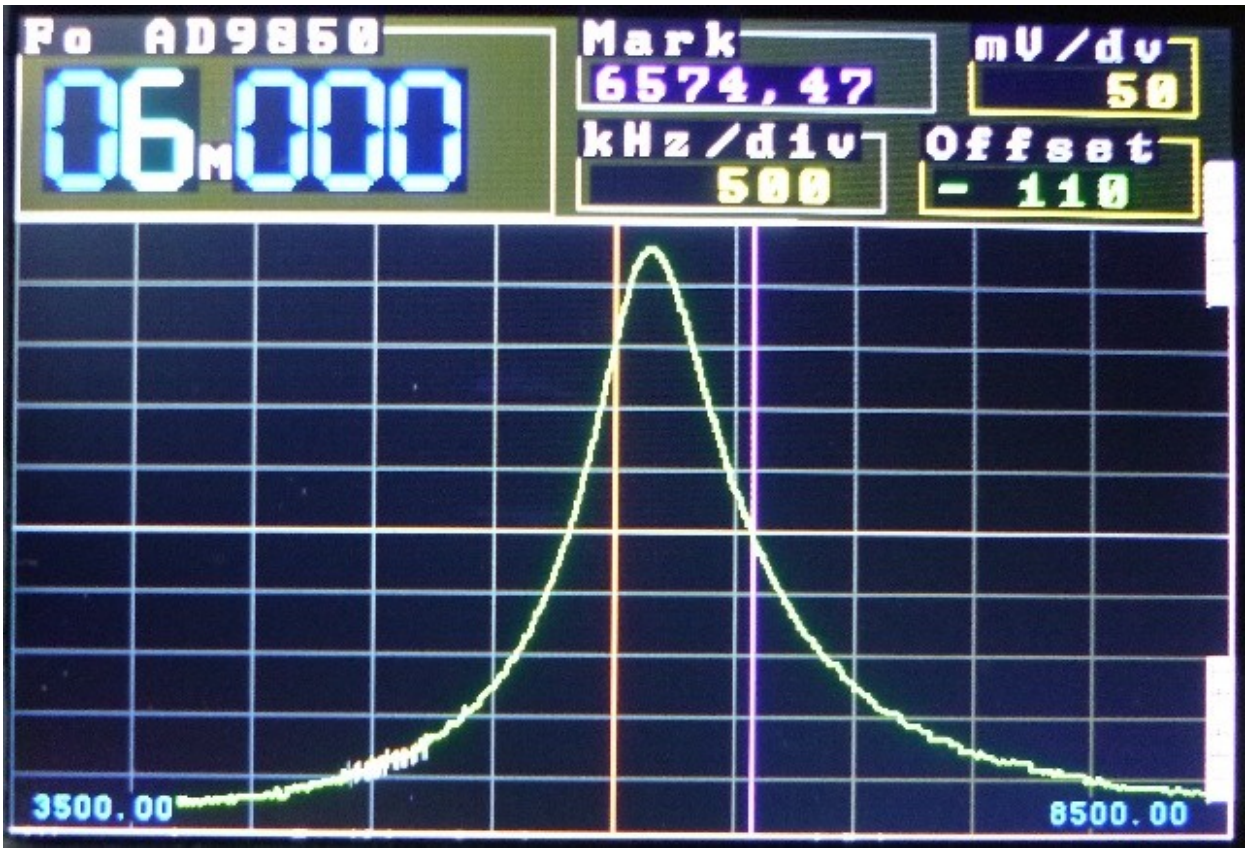


Równoległy obwód LC, zwany „wtyczką”, nadaje rezonansowi bardzo dużą impedancję (funkcję tłumienia przez R in //, promieniowanie, straty w szeregu R induktora, straty dielektryczne w C ... wszystkie rzeczy, które maleją ta impedancja, w przeciwnym razie byłaby po prostu nieskończona). W każdym razie jest to w zasadzie bardzo duże (kilka k ohm patrz M ohm). W związku z tym napotykamy na problem pobudzenia go przez wyjście AD9850, które, przeciwnie, ma bardzo niską impedancję, zwykle 50 omów. Z tego powodu bezpośrednie połączenie rezystancyjne w szeregowym (lepszym) kondensatorze nie jest wcale optymalne. Dlatego zdecydowałem się na inne rozwiązanie, aby dostosować te impedancje: cewkę wzbudzającą składającą się z kilku zwojów emaliowanego drutu miedzianego, który po prostu magnetycznie łączy się ze sobą, na przykład poprzez stylizację jak na zdjęcie powyżej.

38 –

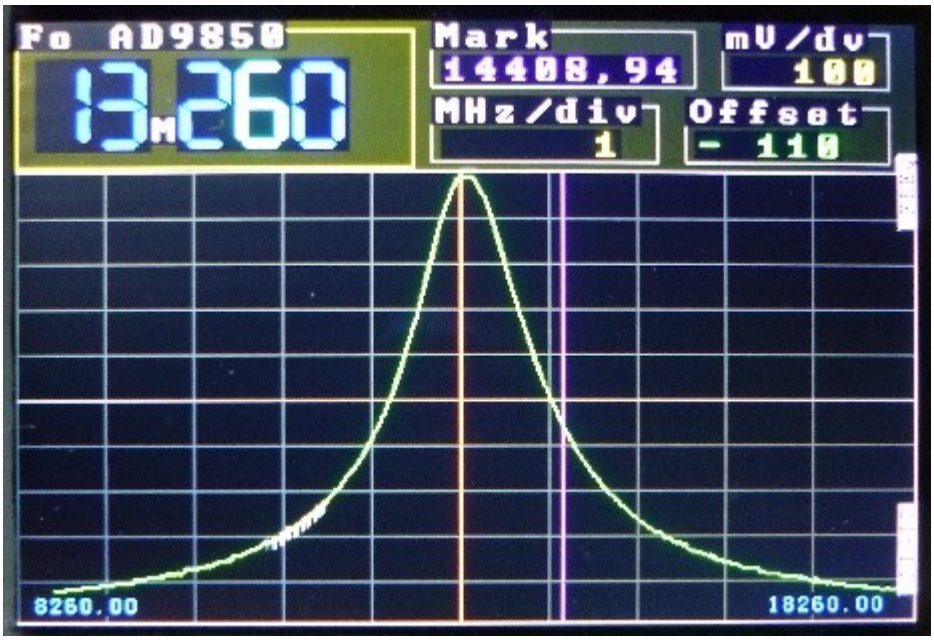


LC-5,6uH-390pF.jpg

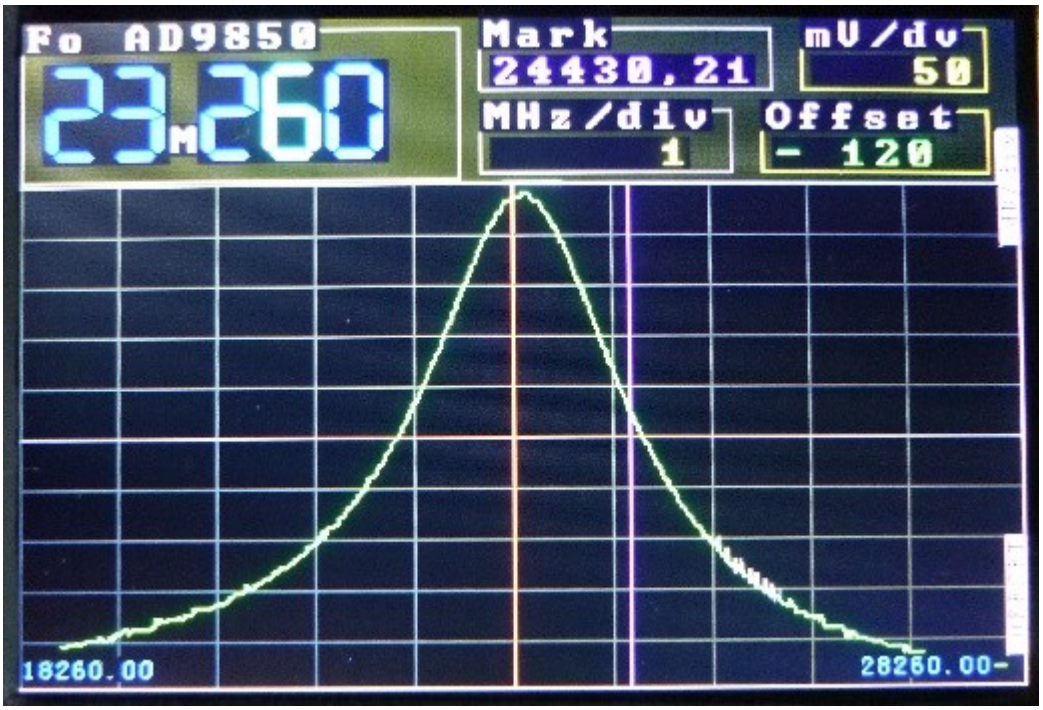


LC-5,6uH-120pF

39 ---



LC-1,2uH-120pF

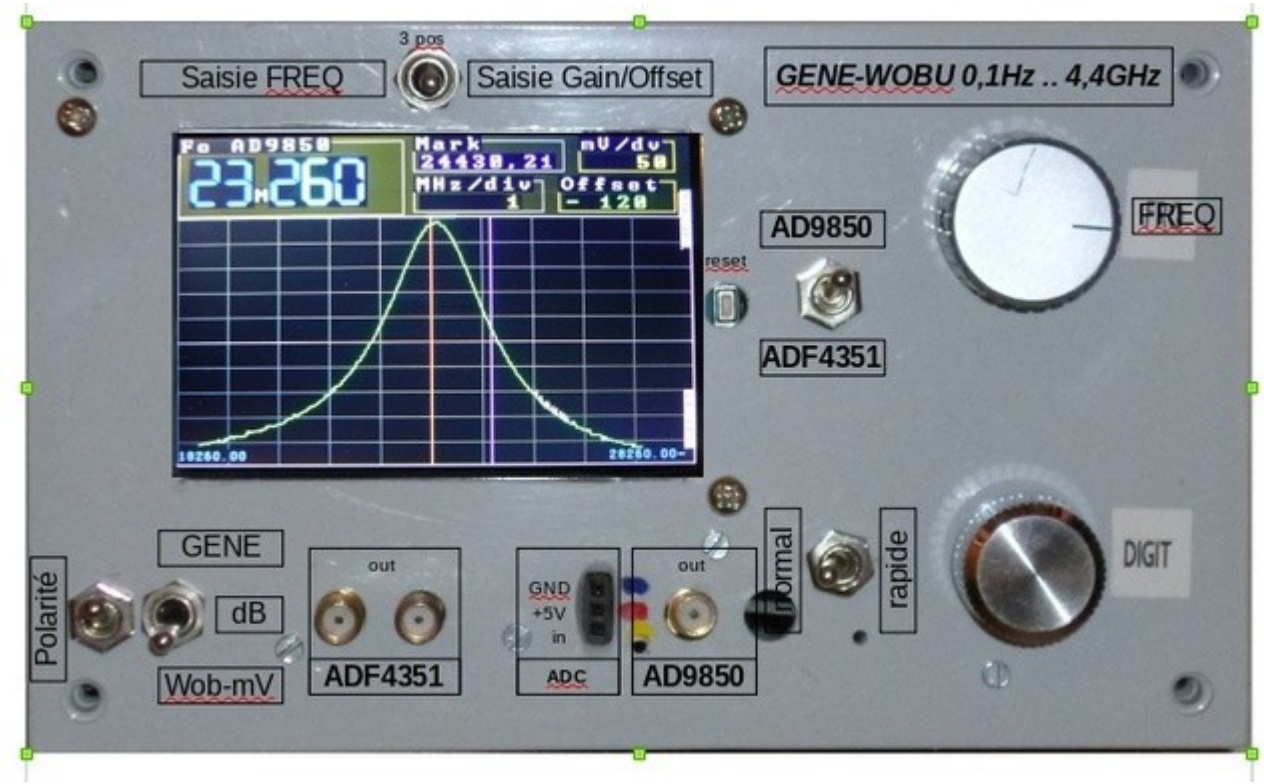


LC-390nH-120pF

Stwierdzamy więc, że AD9850 jest dobrze wobulowany i ma odpowiednią

częstotliwość (do twoich kalkulatorów !! formuła Thomsona Fo = 1 / (2pi root (LC))

40 Une tit' face avant...



Rysunek w OpenOffice Draw. Istnieje kilka nakładających się warstw.

Wydrukowałem działkę na białym papierze, a następnie laminowałem ją laminatorem termicznym.

Ekran, przełączniki i przyciski są tutaj tylko obrazem na dole pracy.

oto wynik, umieszczony na urządzeniu.

- Tam wszystko jest prawdziwe

- Tak, sir, przy okazji, to utknęło!

- tss! pierwsze ostrzeżenie!

41 La réalisation d'un OM (F1CHM)

06 luty 2019:

F1CHM, który miał wyświetlacz inny od mojego, oparty na układzie graficznym ILI-9486, był w stanie z powodzeniem zaprogramować kartę mega2560 ze specjalnie skompilowanego pliku .hex do obsługi tego układu. A ponieważ chcę przynieść korzyści wszystkim, dodałem tę wersję .hex do dokumentów dostępnych poniżej na tej stronie (w .tar.gz).

42 –

Filtre centré sur 145 MHz.

Filtre centré sur 145 MHz.

> F1CHM:

Bardzo zadowolony z tego montażu

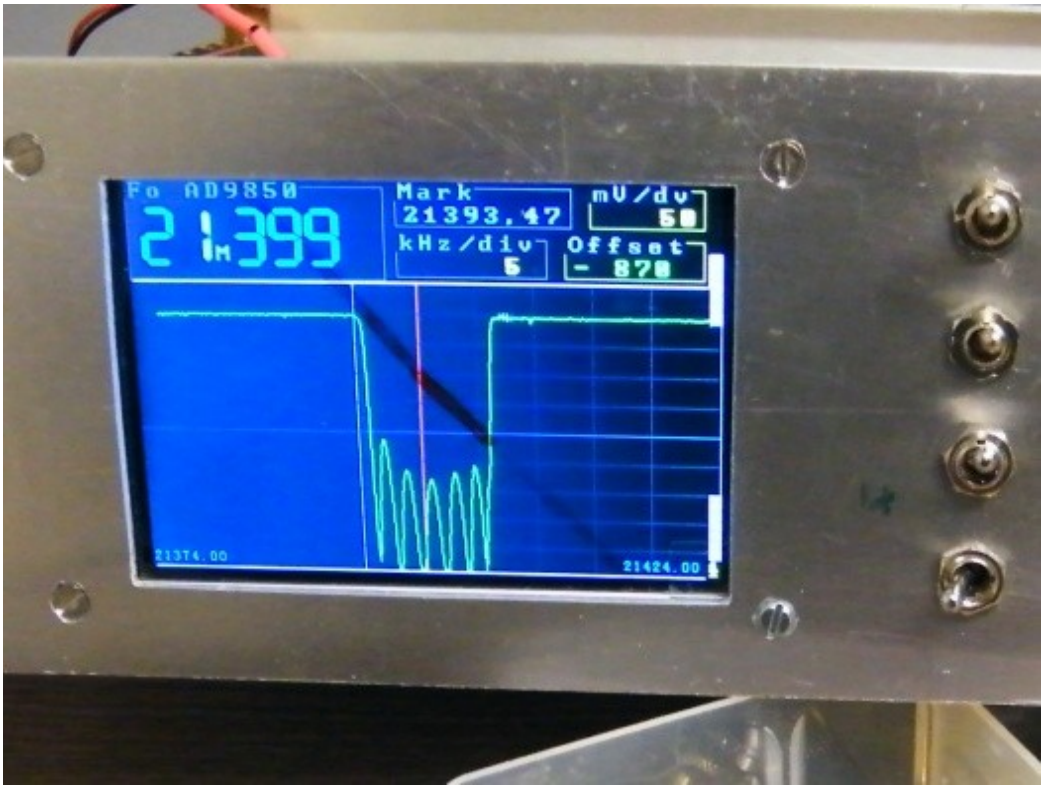
zdjęcie filtra vhf na 144 megapikselach

73 serdeczny

> Silicon628:

Dziękuję, publikuję ten wynik pomiaru w trybie logarytmicznym.

43 –



Filtr 21,4 MHz, którego reakcja wyraźnie nasyca wyświetlacz. Przy niższym wzmocnieniu i / lub detektorze dziennika używanym w trybie dziennika wszystko powinno być w porządku.

Ty, OM, jesteś nieskończenie bardziej kompetentny ode mnie w obsłudze tych stworzeń, więc będziesz nas informował o tym, co powinno być ulepszone, jeśli to

konieczne.

44 Documents techniques :

44 Dokumenty techniczne:

https://www.silicium628.fr/articles/92/documents/Firmware_C_v10_3.tar.gz

i inne „biblioteka”

Napisałem i skompilowałem ten program za pomocą edytora Geany pod Linux Mint 19 Tara 64-bit.

25 stycznia 2019:

Dodałem plik .hex w folderze źródeł. (W rzeczywistości skompilowano kilka .hex dla kilku wariantów wyświetlacza TFT przy użyciu różnych układów graficznych).

45 Liens :

45 linków:

Wysoka częstotliwość:

http://fr.wikipedia.org/wiki/Haute_fr%C3%A9quence

http://jredoutey.free.fr/Radiocom/Poly%20Radiocom%202010.pdf

http://fr.wikipedia.org/wiki/Spectre_radiofr%C3%A9quence

http://fr.wikipedia.org/wiki/Spectre_radiofr%C3%A9quence

http://www.mp-i.fr/wp-content/uploads/2011/09/1187111871_1239971965application.pdf

http://www.syscope.net/elec/A15.pdf

https://www.silicium628.fr/www.jacquesdubois.com/pdf/introduction/blindage_electromagnetique.pdf

regulacja:

http://www.arcep.fr/uploads/tx_gsavis/00-1364.pdf

określenie warunków użytkowania amatorskich urządzeń radiowych (w szczególności załącznik 3: parametry techniczne, których należy przestrzegać podczas korzystania z amatorskiej instalacji radiowej). To jest bardzo restrykcyjne. OSTRZEŻENIE: dotyczy to amatorów radia, dla innych osób (osób) każdej transmisji, wszelkie

promieniowanie na VHF jest surowo zabronione.

Filtres :

http://fr.wikipedia.org/wiki/Filtre_de_Butterworth

http://fr.wikipedia.org/wiki/Filtre_de_Tchebychev

http://fr.wikipedia.org/wiki/Filtre_elliptique

http://fr.wikipedia.org/wiki/Filtre_de_Bessel

obwody typu mikropaskowa (mikropaskowa)

https://paginas.fe.up.pt/~hmiranda/etele/microstrip/

https://paginas.fe.up.pt/~hmiranda/etele/microstrip/hairpin_1.pdf

http://home.sandiego.edu/~ekim/e194rfs01/filterek.pdf

https://hackaday.com/2017/07/25/rapidly-prototyping-rf-filters/

http://f6csx.free.fr/techni/LIGNES%20MICROSTRIP.pdf

Oprogramowanie do obliczania i symulowania filtrów:

-RF SIM 99 (pod Linuksem z winem):

http://pagesperso-orange.fr/f6crp/ba/rfsim.htm

-Qucs (bezpośrednio pod Linuksem):

http://qucs.sourceforge.net/screenshots.html

https://doc.ubuntu-fr.org/qucs

http://qucs.sourceforge.net/install.html#install_ubuntu

https://launchpad.net/~fransschreuder1/+archive/ubuntu/qucs?field.series_filter=xenial

Cewki indukcyjne: -

http://www.carnets-tsf.fr/bobine

http://www.tavernier-c.com/bobinages.htm

http://oernst.f5lvg.free.fr/div/bob/bob.html

Teoria dzielników częstotliwości - przerzutniki RS, T, D, JK, wewnętrzne obwody bramek logicznych:

http://www.sonelec-musique.com/electronique_bases_diviseurs_frequence.html

http://www.liafa.jussieu.fr/~carton/Enseignement/Architecture/Cours/Gates/

Mnożenie częstotliwości - przetwornice częstotliwości:

https://intranet.insa-toulouse.fr/view/431/content/fr/rfc_freq_converter_FR.html

http://home.nordnet.fr/~fthobois/MRA-10.htm

http://doctsf.com/histoire.html

46 –

Aby zobaczyć również ...

http://www.silicium628.fr/article_i.php?id=46