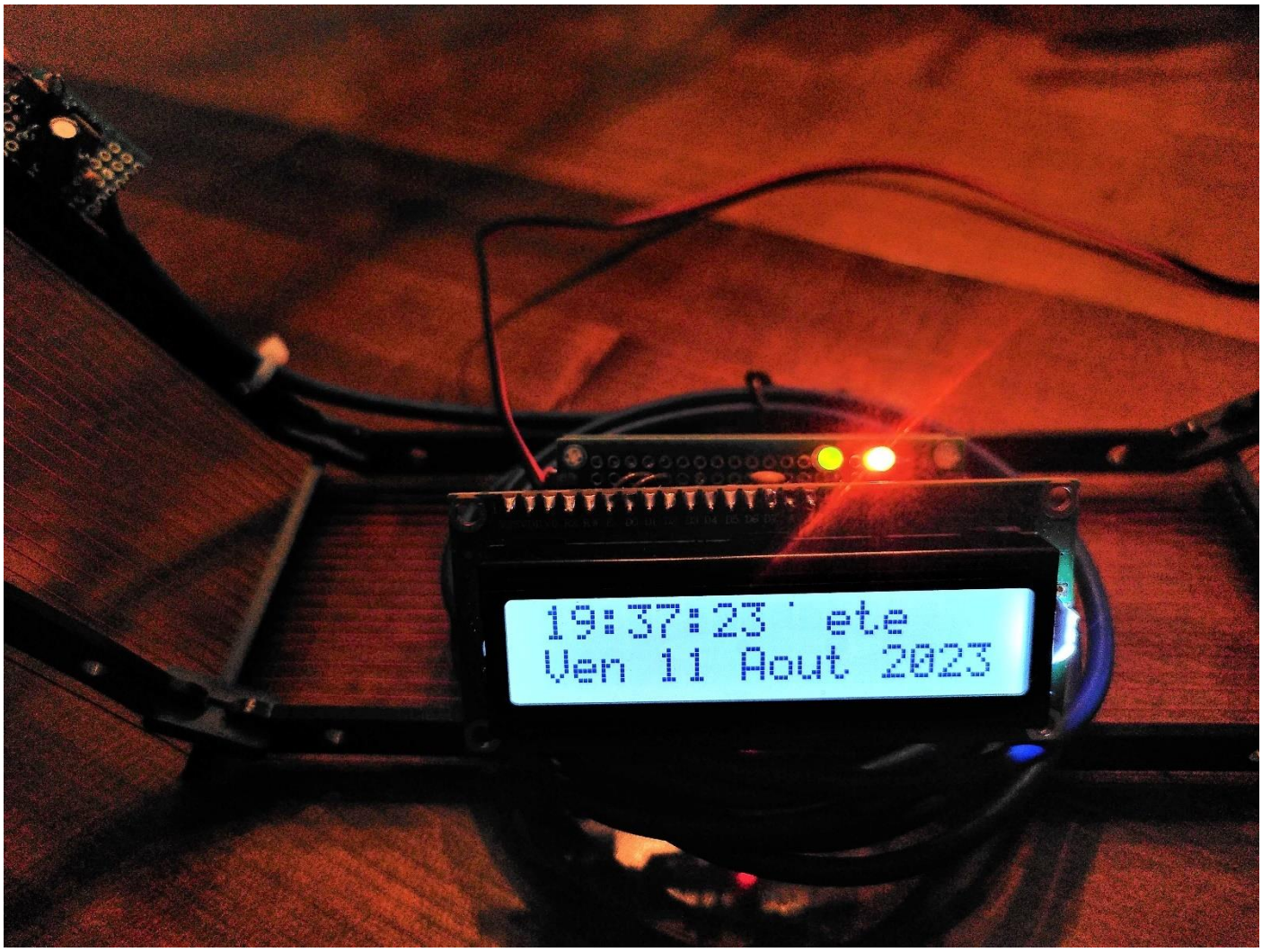
ALS162 an alternative to DCF77 time radio with Arduino Uno



Author : Philippe de Craene dcphilippe@yahoo.fr

August 2023

Manual version : 1.0
Program version : 0.9

Before the world of the internet and GPS, the way to set the same time throughout a country was to use a radio station. There were at least 5 of these special radio stations in Europe until the 1990s, of which only 2 are still operating today: the very popular DCF77 in Germany and the rather unknown ALS162 in France.

DCF77 is a 50 kW, 77.5 kHz transmitter based at Mainflingen in Germany, detail can be found here: https://en.wikipedia.org/wiki/DCF77

ALS162 is a 800 kW, 162 kHz transmitter based at Allouis in France, detail are available here:
in English https://en.wikipedia.org/wiki/ALS162_time_signal
or in French https://syrte.obspm.fr/spip/services/ref-temps/article/mise-a-disposition-du-temps-legal-par-le-signal-als162

Both use similar time signal code sequence.

Most descriptions of time radio devices made or sold are based on DCF77. To make things easier for the do-it-yourselfer, there are also ready-made receiver modules for DCF77, made in China and very, very cheap...

In fact, I've only found one description of a time radio receiver based on ALS162, here: https://www.rvq.fr/tech/fi.php , from which I got some ideas and a lot of inspiration for this article.

Also, ALS162 is much less documented than DCF77, and can be considered more difficult to build, as it is a phase-modulated signal, while DCF77 is a very simple AM signal.

However, if you want to build your own receiver instead of using these ready-made modules, you will understand how hard it is to get the little LED to blink every second. Then it looks much easier to build a 162 kHz direct amplified receiver, as the power of the radio transmitter gives the ALS162 the advantage.

As DCF77 is modulated in AM, the first thought is that it is easy to handle, but it is also very sensitive to electromagnetic noise. So sensitive, in fact, that it is almost necessary to move the microprocessor used to decode and display the time away from the receiver, at the risk of interfering with the reception of the signal.

Instead, the ALS162, with its phase modulation method (very similar to FM), gains a very great advantage in terms of immunity to electromagnetic noise. Even all those modern LED lamps with their buck converters, as well as any device that requires a voltage converter power supply, are no longer a threat.

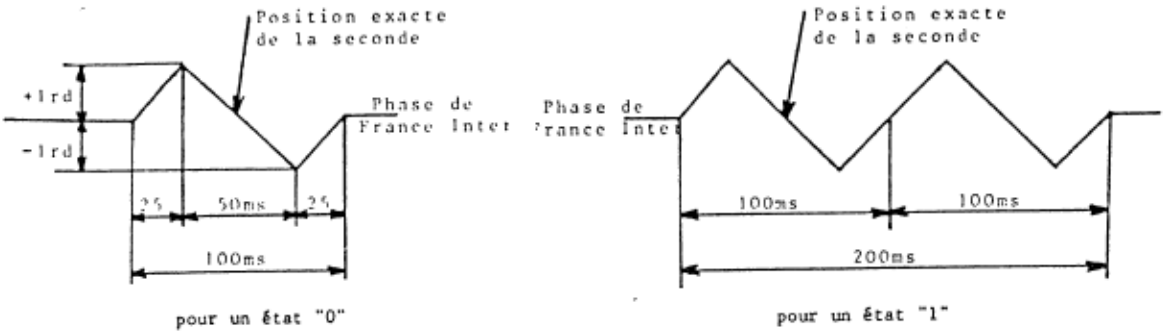
This manual explains step by step how to build an ALS162 receiver with its time decoder in the simplest way possible.

Table of content

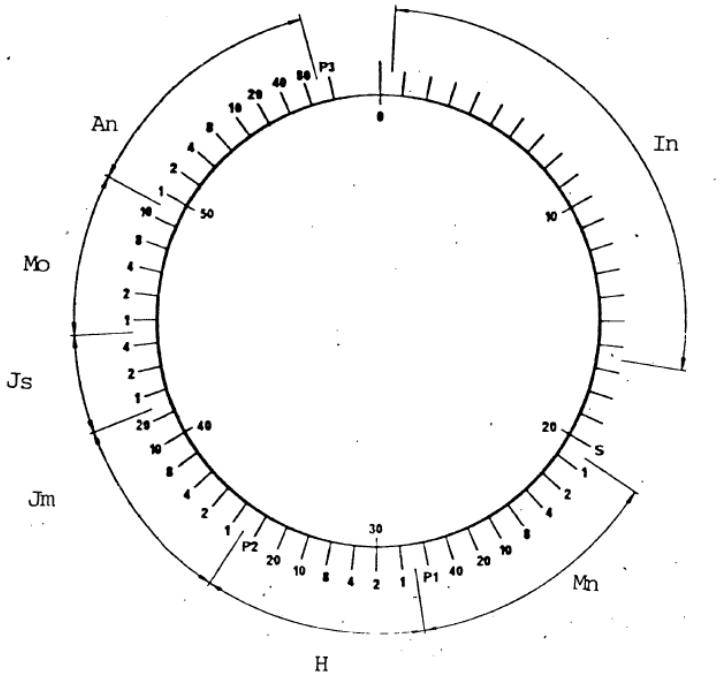
Table of content 3
Introduction 4
List of materials 5
Radio receiver 5
Time decoder 5
The 162 kHz radio receiver 6
The receiver diagrams 6
How to tune LC circuit to 162 kHz 7
The demodulation 9
Which method chosen to demodulate 9
How it works 10
Verify demodulation 11
The time decoder 12
1- Timer1 interrupts every 10ms 12
2- Signal detection 12
3- 59th second detection 13
4- First and second “bit” data 13
5- Time decoding sequence 13
6- Display result 15
The final diagram 16
The final program 17
Illustrations 23

Introduction

The time data code is similar to DCF77, one "bit" is sent every second. The difference is that DCF77 uses short or long "bits" for "0" or "1" respectively. Here we have 1 bit for "0" and 2 bits for "1". All this takes place in the first 200ms of each second. Except for the 59th second, where there is no "bit" at all. This second 59 allows the synchronisation for the decoding of the time data.



The time data sequence gives the time and date every minute as follows:



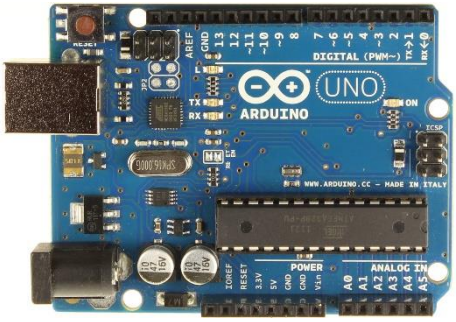
- Second 14 indicates whether it is a national holiday day (1st January, 1st May, 14th July…)
- Seconds 16 to 18 indicate the winter/summer time.
- Seconds 21 to 27 give the minutes
- second 28 is a parity check of the minutes
- Seconds 29 to 34 indicate the hour
- Seconds 35 is a parity check of the hour
- Seconds 36 to 41 indicate the day of the month
- Seconds 42 to 44 are the day of the week
- Seconds 45 to 49 are the month
- Seconds 50 to 57 are the year
- Second 58 is a parity check of the date

From seconds 21 to 57 the encoding for ALS162 data is identical to DCF77

List of materials

Radio receiver

1. Antenna loop
2. 1x BC639 or equivalent
3. 2x BF245B or equivalent
4. 1x 1mH coil,
5. 1x 100pF adjustable capacitor,
6. CD4046 PLL,
7. Few resistors, capacitors….



Time decoder

1- 1x Arduino Uno with a DIP28 Atmega328p MPU
2- 1x LCD1602 with I2C
3- 1x crystal quartz of 10.368 MHz
4- 1x BB910 varactor diode
5- 1x 10pF adjustable capacitor
6- Few resistors, capacitors….

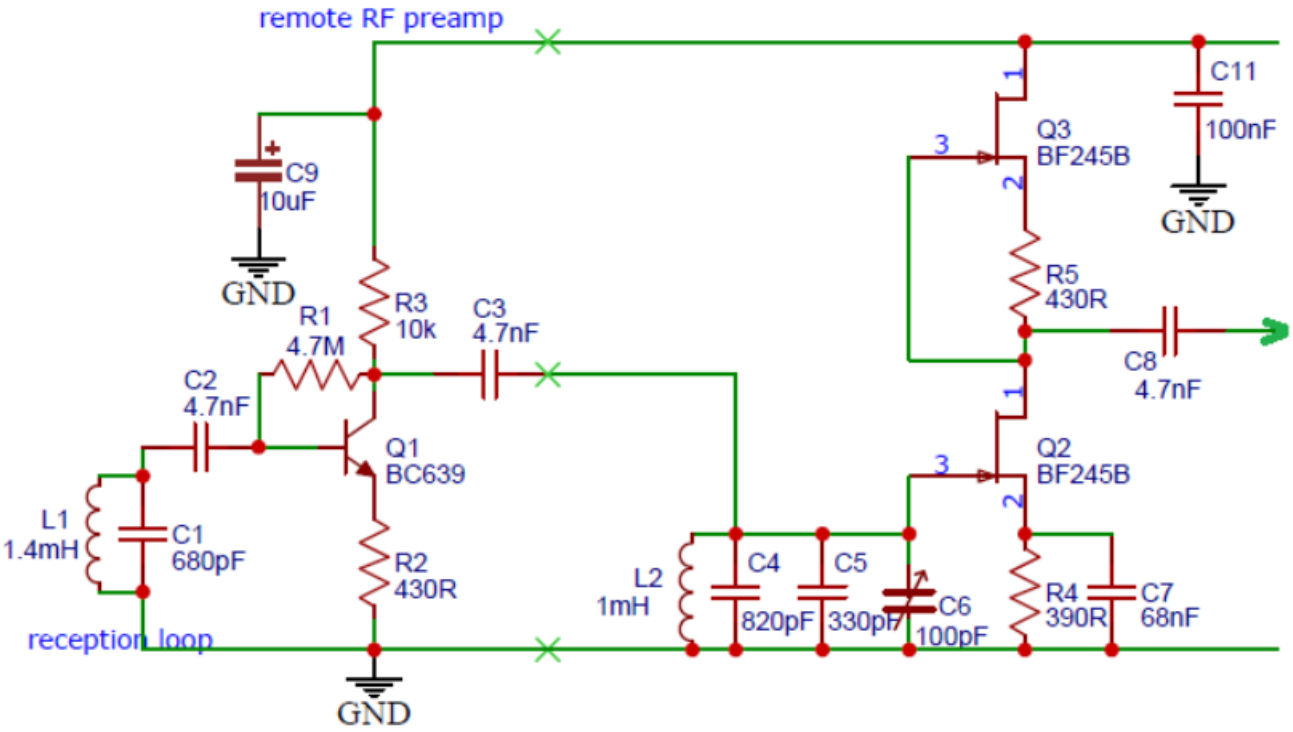
The 162 kHz radio receiver

The receiver diagrams

The receiver is based on "bottom of drawer" components. It is a 2-stage amplifier.

The first stage is based on an NPN bipolar transistor in a classic common emitter scheme.

The second stage is based on a FET SRPP, which offers very high gain and very high input impedance, necessary for a high Q of the LC filter. The best selectivity is also the best to immunise the receiver against any electromagnetic noise.

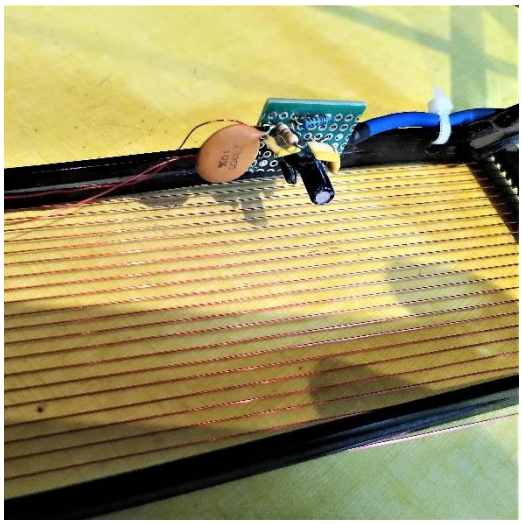


The disadvantage of an SRPP is that it requires a relatively high supply voltage: at least 9V.

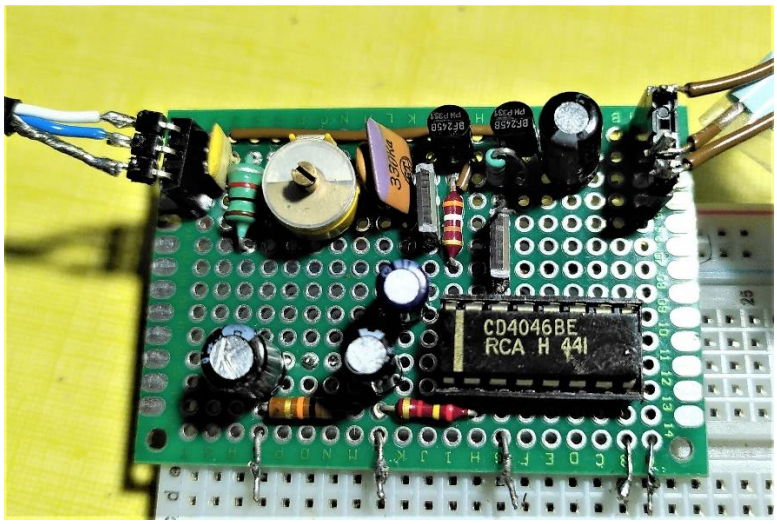
For more gain it is possible to reduce R2 to 180R, but care must be taken to avoid self-oscillation. Decoupling with a capacitor is also not recommended.

If the antenna loop is remote, the first stage should be placed close to it. It is of course possible to replace the antenna loop with a ferrite, or a simple wire antenna with a coil or RF transformer.

Here are 2 photos of this maquette:



Loop antenna with its first stage amplifier. The 680pF capacitor is very old and very big!



Second stage amplifier with the two BF245B. We can notice the 4046 PLL that we will study in the demodulation part.

The most difficult task here is to fine-tune the two LC resonant circuits to a frequency of 162 kHz.

How to tune LC circuit to 162 kHz

For this operation we need a reference signal of 162 kHz. And if we do not have expensive devices that can synthesise radio frequencies, what better than the very cheap Arduino Uno?

Remember that the 162 kHz frequency is obtained by dividing 10.368 MHz by 64. Timer2 will be used to perform this function. The idea is to replace the usual 16 MHz crystal with a 10.268 MHz one. This can be done with a DIP28 Atmega328p. Once the code is uploaded, all we have to do is remove the MPU from its socket and place it on a homemade shield.

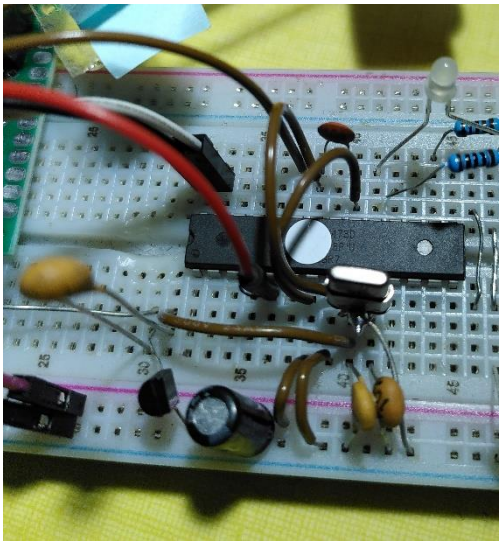
Here is the list of operations:

1- From the Arduino IDE, upload the following code to the Arduino Uno:

```
/*
 * 162kHz generator
Materials: ATmega328p with a 10368.000kHz Crystal Oscillator

 *
 *                               ATmega328p
 *
 *                    (RESET) PC6  1    28 PC5 (ADC5/SCL)
 *                      (RXD) PD0  2    27 PC4 (ADC4/SDA)
 *                      (TXD) PD1  3    26 PC3 (ADC3)
 *                     (INT0) PD2  4    25 PC2 (ADC2)
 *    freqOutPin = 11  (OC2B) PD3  5    24 PC1 (ADC1)
 *                   (XCK/T0) PD4  6    23 PC0 (ADC0)
 *                            VCC  7    22 AGND
 *                            GND  8    21 AREF
 *     XTAL1    (XTAL1/TOSC1) PB6  9    20 AVCC
 *     XTAL2    (XTAL2/TOSC2) PB7 10    19 PB5
 *                      (T1) PD5 11    18 PB4
 *                    (AIN0) PD6 12    17 PB3 (OC2A)
 *                    (AIN1) PD7 13    16 PB2
 *                    (ICP1) PB0 14    15 PB1 (OC1A)
 */
```

```
#define freqOut      162000L     // 162Khz clock generated for radio demodulation
#define freqCPU    10368000L     // note F_CPU still defined as 16000000 by Arduino IDE

#define ledPin            13     // buitin led on Arduino Uno

//
// setup
//_____________________________________________________________________________________

void setup() {

  DDRB = 0xFF;    // set all portB as output = PB0 to PB5, I/O 8 to 13, pins 14 to 19
  DDRD = 0xFF;    // set all portD as output = PD0 to PD5

// Set the timers
//----------------------------------------------------------------------

  noInterrupts();

// set timer 2 to get 162kHz clock on output 3
// from exampler: http://www.gammon.com.au/timers
  TCCR2A = 0;
  TCCR2B = 0;
  TCCR2A = bit(WGM20) | bit(WGM21) | bit(COM2B1);  // fast PWM, clear OC2A on compare
  TCCR2B = bit(WGM22) | bit(CS20);                 // fast PWM, no prescaler
  OCR2A = (freqCPU / freqOut) - 1;          // set the output signal frequency
  OCR2B = ((OCR2A + 1) / 2) - 1;            // 50% duty cycle

  interrupts();

}

//
// loop
//_____________________________________________________________________________________

void loop() {

  digitalWrite( ledPin, HIGH );    // just to see that something is done.
  delay(500);                      // the led will blink ~1.5 slower with the 10.368MHz crystal
  digitalWrite( ledPin, LOW );
  delay(500);

}        // end of loop
```

2- Once uploaded, the built-in led should blink with a period of 1 second.



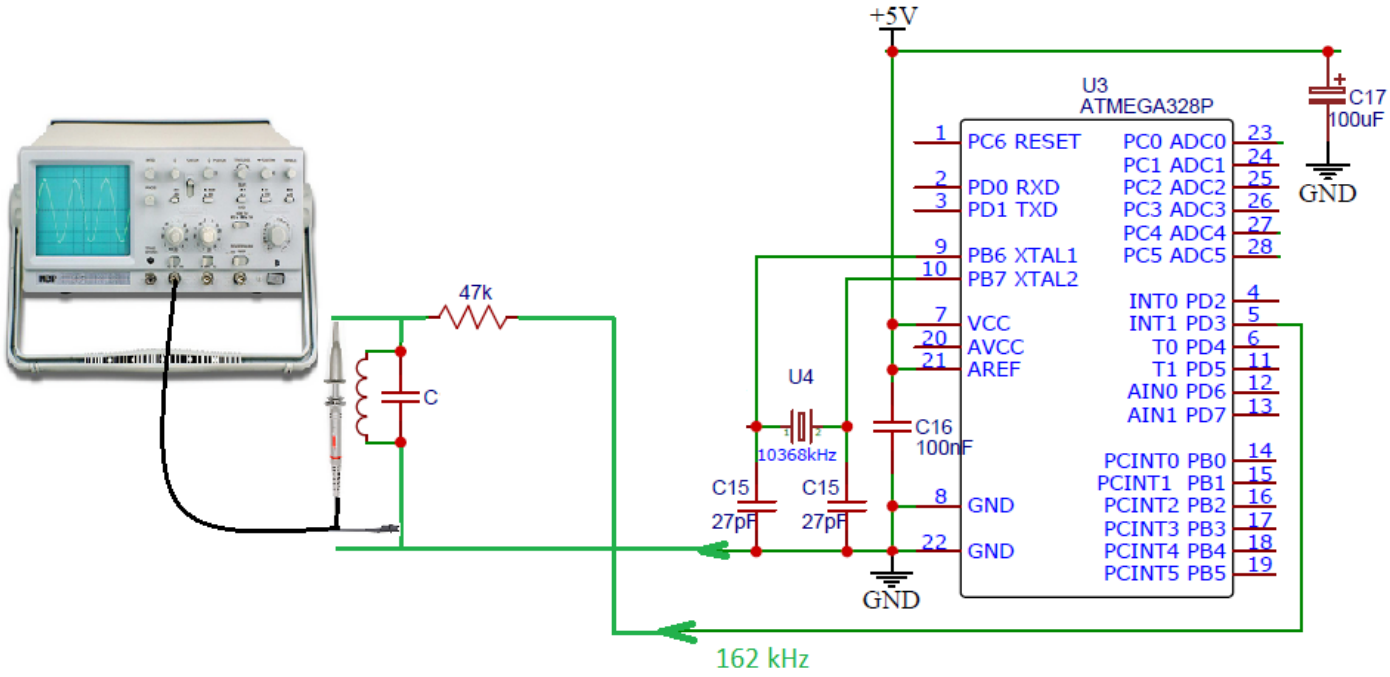
3- Remove the ATmega328p DIP28 from the Arduino Uno and make a small shield with the 10.368 MHz crystal.

The photo on the right shows the crystal with its 2x 27pF capacitors. The output frequency of 162 kHz is the red wire.

If you place a LED pin 19 of the ATmega328p you should notice that now it is blinking 1.5x slower than before.

4- Once this frequency reference is ready, using a coil L (1mH or more), it is possible to adjust the correct value of the capacitor C to obtain the resonance at 162 kHz: This is the moment when the voltage is at its maximum.

The use of an oscilloscope will help....

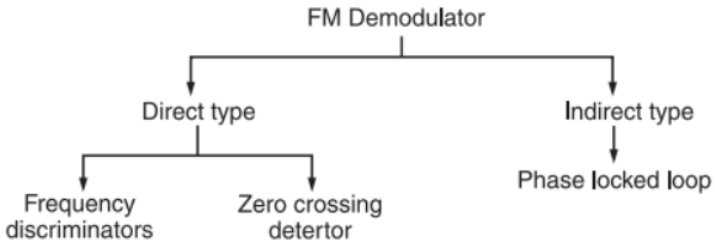


5- This operation must be carried out on both LC circuits: Antenna loop with C1, then L2 C4 (and C5). An adjustable capacitor C6 has been added to fine-tune the resonance frequency once the final unit is in operation.

Once resonant LC circuit tuned and the shield ready, depending on the orientation of the antenna loop, an RF output signal of at least 0.7Vcc should be obtained.

The demodulation

Which method chosen to demodulate



A phase modulation of +/- 1 radian corresponds to a frequency shift of +/- 6 Hz around the 162 000 Hz transmitter frequency.

This can be compared with the +/- 75 kHz for radiophonic transmitters on FM (87 to 108 MHz) to imagine that the decoding method must be much more sensitive than the traditional Foster-Seeley discriminator used for FM radio receivers in Europe or the Balanced Slope Detector in the USA.

There are 2 indirect ways of FM/phase demodulation using a PLL (Phase Lock Loop):

Either the 162 kHz radio signal is mixed with a 160 kHz local oscillator, the 2 kHz obtained is amplified and filtered and then demodulated with a PLL (phase lock loop), for example the CD4046. In this case the VCO for Voltage Controlled Oscillator, is set to a free frequency of 2kHz in our example. And the VCO voltage follows the data signal: This is the PLL demodulation by VCO.

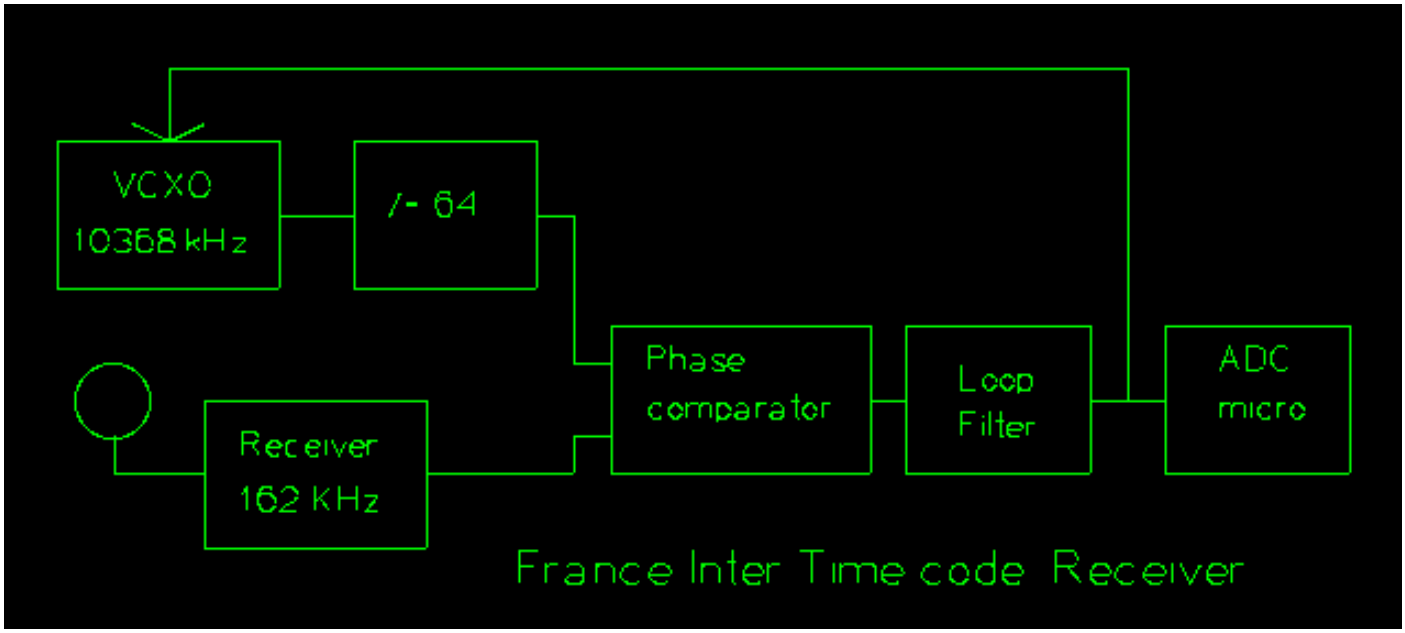
Either this 162 kHz radio signal is compared directly to a 162 kHz reference frequency, this reference frequency is generated by a VCXO. A VCXO is a Voltage Controlled Crystal Oscillator, which consists of a crystal oscillator with a varactor diode and supporting circuitry. When a DC control voltage is applied to the crystal, the oscillation frequency changes. The error between the two is then the data. This is PLL demodulation by VCXO.

To resume the characteristics between the two PLL methods:

VCO	VCXO
Oscillation Frequency Generated by an Electronic Circuit	Oscillation Frequency Generated based on Crystal Properties
Wide Frequency Range	Narrow Frequency Range
Lower Stability	Higher Stability
Higher Phase Noise	Lower Phase Noise

For the best stability offered and as we do not need a wide frequency range, we will choose the 2nd solution, i.e. PLL demodulation by VCXO.

And with the help of the ATmega328p: instead of the usual 16 MHz quartz on the Arduino Uno, the MPU will run with a 10.368 MHz quartz. 10.368 MHz divided by 64 gives 162 kHz. 😊



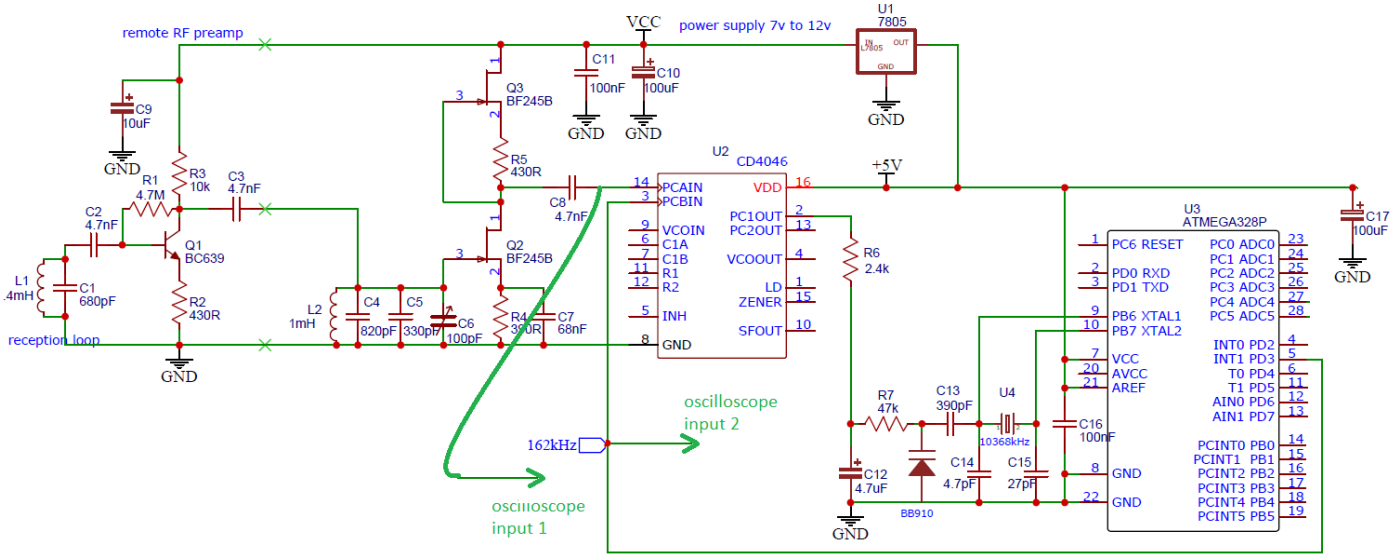
How it works

The demodulation consists of a phase comparator between the RF output of the radio receiver and a frequency reference of 162 kHz built up by the ATmega328p Timer2. The phase comparator used is the CD4046 phase comparator I: an exclusive OR network.

The output of the phase comparator is a square-wave signal corresponding to the level of the phase comparison between the two input frequencies. This square-wave signal is filtered at a very low frequency - between 1 and 2 Hz - and then becomes a fairly continuous voltage derived from the average of the phase comparisons.

This voltage drives the varactor of the VCXO: this varactor, the BB910 is specified for 25pF at 4V and 35pF at 1V, can modify the clock frequency of the Atmega328p, i.e. the 162 kHz generated by the ATmega328p.

As long as there is no modulation on the RF signal, the 162 kHz generated by the Atmega328p is locked to the RF signal and the voltage on the varactor does not move. But as soon as there is a data signal on the RF, the "fairly continuous voltage" reflects this variation, the varactor modifies the clock frequency, and so on: the data is detected.



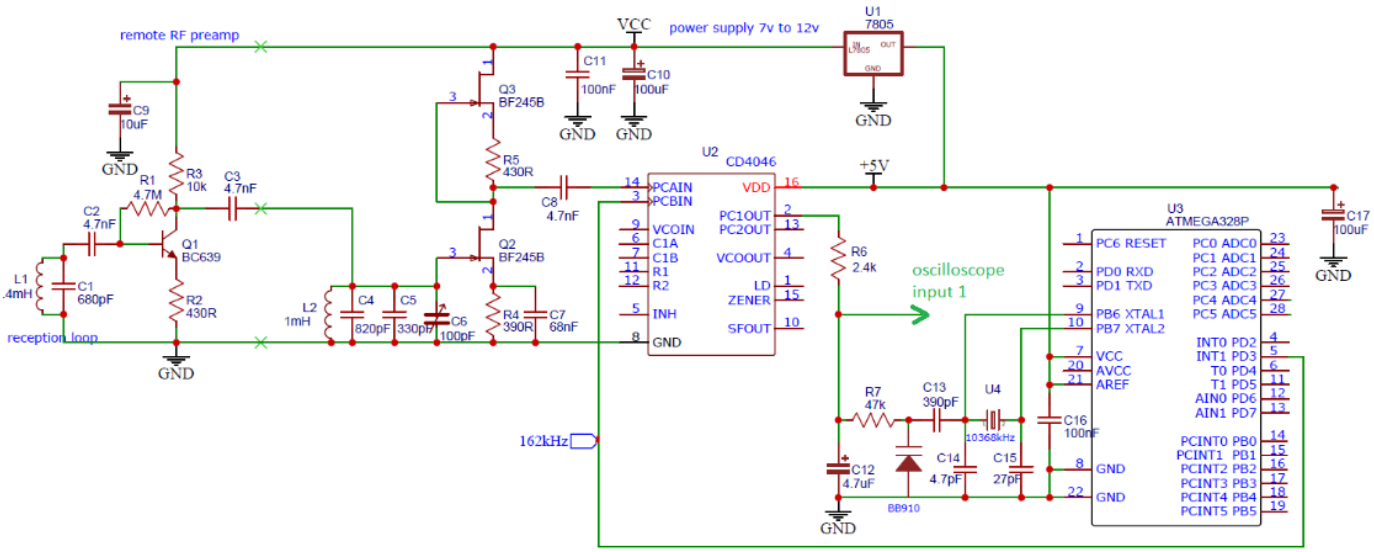
Verify demodulation

According to the diagram above, the oscilloscope can show the good phase lock between the 2 frequencies. The 2 inputs must remain synchronised regardless of the data signal. If not, adjust C13 C14:

- Check that the voltage on the varactor is around 2.5V (middle of the 5V supply). C14 helps to phase lock without data signal.
- C13 helps to limit frequency variations. Too much variation in the clock frequency will interfere with the operation of the LCD1602 display.

Note: Finally, an adjustable capacitor C15 of 10pF has been added in parallel with C14 (see final diagram) to allow easy adjustment of the phase lock.

The diagram below shows where it is possible to analyse the signal data. You will see that there is a lot of traffic, not just the "bit" data during the first 200ms of each second. Unfortunately, there is no information available about this data, which is transmitted in addition to the time signals. The only thing that is easily detectable is the "silent" signal during a full second, which is transmitted every minute: it is the 59th second.



The time decoder

What we need to know is:

1- To know when to synchronise the time decoder sequence.

Either a "bit" (at least) is received during a "while", or not. If not, it could be the 59th second. This no-signal second starts the time decoding sequence for the next minute. So, at second 0 of the next minute, all the data should be received and well understood to display the real time.

2- To receive the "bit" data every second

Either the "bit" received during the first 100ms is followed by a second "bit" during the second 100ms of each second, except the 59th second (see above), or not. This sequence of 0 and 1 received during 59s of each minute contains the hour data of the next minute.

The demodulated signal arrives at the analogue input A0 of the ATmega328p.

The process is as follows:

1- Timer1 interrupts every 10ms

Timer1 is set to generate an interrupt every 10ms (according to the 10.368 MHz crystal of course)

```
// set timer 1 to interrupt every 10ms (100Hz) used for ADC sampling rate
// Clear Timer on Compare Match (CTC) Mode
// precaler setting : freqCPU/8, N = 8
  TCCR1A = 0;
  TCCR1B = 0;
  TCCR1B = bit(WGM12) | bit(CS11);
  OCR1A  = ( freqCPU / 8L / 100L ) - 1;
  TIMSK1 = bit(OCIE1A);
```

Theses interrupts generate 2 counters: 10ms counter, 100ms.

```
ISR(TIMER1_COMPA_vect) {
// set times counters
  counter10ms ++;                          // count from 0 to ~6040 during 1 minute
  counter100ms = (counter10ms / 10) % 10;  // count from 0 to 9 for each second
  }
```

There is a 3rd counter for counting every second. It is updated at the same time the display is updated – in the void loop():

```
      secondsCounter++;
      blinkLed = !blinkLed;
      BlinkingLed( blinkLed, synchro );
```

2- Signal detection

The ADC is performed every 10ms: each falling front of the analogue input is detected. At the same time, the noActivityCounter is incremented every 10ms if there is no signal, or reset if there is a signal.

```
ISR(TIMER1_COMPA_vect) {

// check demodulated radio signal
  memo_timeBit = timeBit;
  memo_dataIn = dataIn;
  dataIn = analogRead( dataInPin );
  if(dataIn < memo_dataIn - offset) {
    timeBit = HIGH;
    noActivityCounter = 0;
  }
  else {
    timeBit = LOW;
    noActivityCounter++;
  }
```

3- 59th second detection

If no “bit” is detected for 100x 10ms, the next “bit” detected will be the second 0 for the next minute. All counters are reset:

```
// detect a full second with no activity = 59th second to start teh synchro
  if( noActivityCounter > 99 ) activity = false;
  if( !activity && (noActivityCounter == 0)) {
    activity = true;
    synchro = true;
    counter10ms = 0;
    secondsCounter = 0;
  }    // end of test !activity
```

4- First and second “bit” data

The first "bit" within the first 100ms is only detected to flash the activity LED. We only look at the second "bit" during the second 100ms:

```
// translate signal to bits
  if( counter100ms == 0 ) {
    if( timeBit ) {
      ActivityLed(HIGH);
    }
    else ActivityLed(LOW);
  }
  else if( counter100ms == 1 ) {
    if( timeBit ) {
      ActivityLed(HIGH);
      if( !memo_timeBit ) bitHigh = HIGH;  // only here there is the interesting data
    }
    else ActivityLed(LOW);
  }
```

5- Time decoding sequence

Second by second, the sequence extracts the time data according to the value of this second "bit".
And performs the binary to decimal conversion by bitwise OR and shift operations.

```
       switch( secondsCounter ) {
        case 14:
          holiday = bitHigh;
          break;
        case 17:
          summer = bitHigh;
```

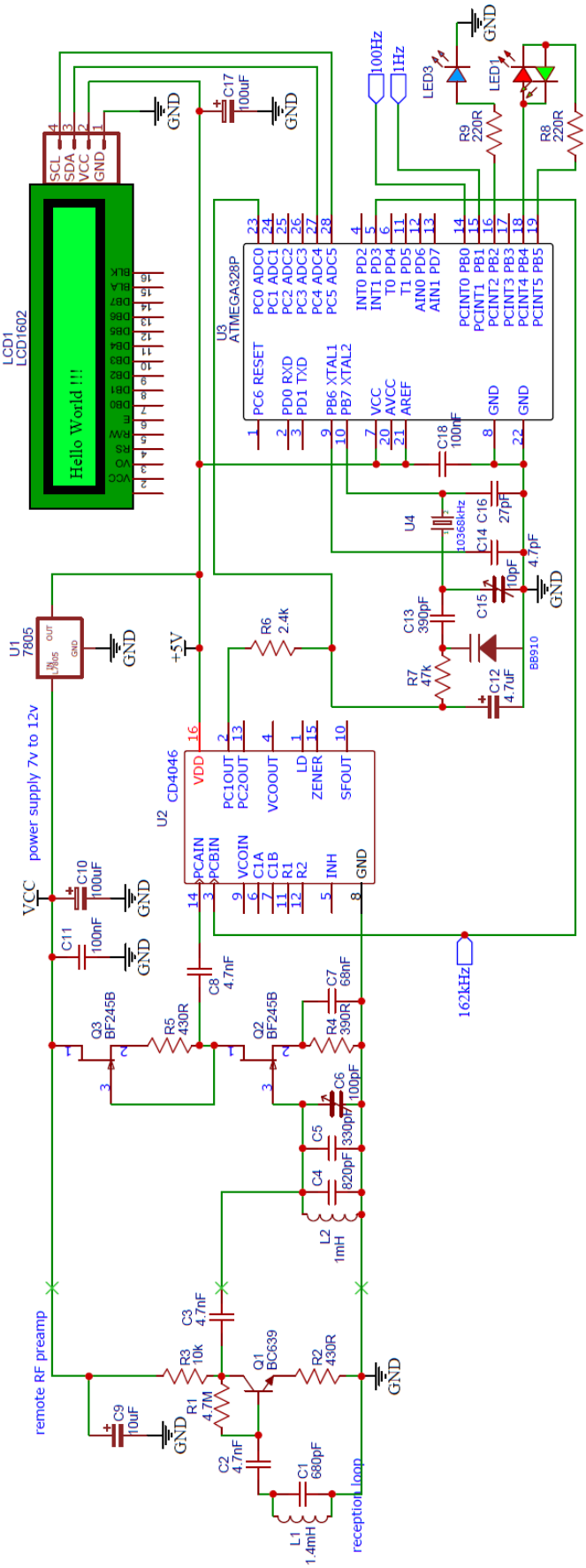
```
    break;
  case 21:
  case 22:
  case 23:
  case 24:
    minuteU = minuteU | bitHigh << ( secondsCounter - 21 );
    if( bitHigh ) bitHighCounter++;
    break;
  case 25:
  case 26:
  case 27:
    minuteD = minuteD | bitHigh << ( secondsCounter - 25 );
    if( bitHigh ) bitHighCounter++;
    break;
  case 28:
    if((bitHighCounter %2) != bitHigh ) parity1 = false;
    bitHighCounter = 0;
    break;
  case 29:
  case 30:
  case 31:
  case 32:
    hourU = hourU | bitHigh << ( secondsCounter - 29 );
    if( bitHigh ) bitHighCounter++;
    break;
  case 33:
  case 34:
    hourD = hourD | bitHigh << ( secondsCounter - 33 );
    if( bitHigh ) bitHighCounter++;
    break;
  case 35:
    if((bitHighCounter %2) != bitHigh ) parity2 = false;
    bitHighCounter = 0;
    break;
  case 36:
  case 37:
  case 38:
  case 39:
    dayU = dayU | bitHigh << ( secondsCounter - 36 );
    if( bitHigh ) bitHighCounter++;
    break;
  case 40:
  case 41:
    dayD = dayD | bitHigh << ( secondsCounter - 40 );
    if( bitHigh ) bitHighCounter++;
    break;
  case 42:
  case 43:
  case 44:
    wday = wday | bitHigh << ( secondsCounter - 42 );
    if( bitHigh ) bitHighCounter++;
    break;
  case 45:
  case 46:
  case 47:
  case 48:
    monthU = monthU | bitHigh << ( secondsCounter - 45 );
    if( bitHigh ) bitHighCounter++;
    break;
  case 49:
    monthD = bitHigh;
    if( bitHigh ) bitHighCounter++;
    break;
  case 50:
  case 51:
  case 52:
  case 53:
    yearU = yearU | bitHigh << ( secondsCounter - 50 );
    if( bitHigh ) bitHighCounter++;
    break;
  case 54:
  case 55:
  case 56:
  case 57:
    yearD = yearD | bitHigh << ( secondsCounter - 54 );
    if( bitHigh ) bitHighCounter++;
    break;
  case 58:
    if(( bitHighCounter %2) != bitHigh) parity3 = false;
    break;
}  // end of switch
```

6- Display result

Every 0 seconds, the time is displayed on an LCD1602 controlled by the I2C protocol. We are lucky that I2C can work regardless of the MPU's clock speed. However, it does not like large clock variations when running.

```
        switch( secondsCounter ) {
         case  0:
           if(displayReady) {
             lcd.setCursor(0, 0);
             if(parity2) { lcd.print(hourD); lcd.print(hourU); }
             else          lcd.print("--");
             lcd.print(":");
             if(parity1) { lcd.print(minuteD); lcd.print(minuteU); }
             else          lcd.print("--");
             lcd.print(":");
             lcd.setCursor(9, 0);
             if( summer )  lcd.print(" ete   ");
             else          lcd.print(" hiver ");
             lcd.setCursor(13, 0);
             if( holiday ) lcd.print("/fe");
             lcd.setCursor(0, 1);
             lcd.print("                ");
             lcd.setCursor(0, 1);
             if(parity3) {
               if( --wday >= 0 ) lcd.print(weekDay[wday]); lcd.print(" ");
               lcd.print((dayD *10) + dayU); lcd.print(" ");
               int monthN = (monthD *10) + monthU;
               if( --monthN >= 0 ) lcd.print(months[monthN]); lcd.print(" ");
               lcd.print((yearD *10) + yearU + 2000);
               if( dayD == 0 ) lcd.print(" ");  // to insure to cover the 16 characters of the
line
             }
           }
           else {
             lcd.setCursor(0, 0);
             lcd.print("      00        ");
             lcd.print(secondsCounter);
             lcd.setCursor(0, 1);
             lcd.print("                ");
           }
           minuteU = 0; minuteD = 0; hourU = 0; hourD = 0;
           wday = 0; dayU = 0; dayD = 0; monthU = 0; monthD = 0; yearU = 0; yearD = 0;
           parity1 = true; parity2 = true; parity3 = true;
           bitHighCounter = 0;
           break;
         case  1:
           if( synchro ) testFullMinute = true;
           else          testFullMinute = false;
           break;


         case 59:
           synchro = false;
           if( testFullMinute ) displayReady = true;
           else                 displayReady = false;
           break;
        }  // end of switch
```

The final diagram



The final program

Remember that the code is uploaded from the Arduino IDE to an Arduino Uno running with a 16 MHz quartz. Then the ATmega328p is plugged into our circuit and runs with a 10.368 MHz quartz.

Note: if you uncomment `//#define DEBUG 1 // debug mode only`

you will NOT get the time but the 3 counter states. This also allows 10ms and 1s signal sync on IC pins 15 and 14 respectively to synchronise the oscilloscope to be able to see the useful part of the radio signal data.

```
/*
 * ALS162 - Allouis (France) time broadcasted radio station decoder

  ____________________________________________________________________
  |                                                                  |
  |       author : Philippe de Craene <dcphilippe@yahoo.fr           |
  |       Free of use - Any feedback is welcome                      |
  ____________________________________________________________________

  for ALS162 detail information: https://en.wikipedia.org/wiki/ALS162_time_signal
 *

Materials:
 - 162kHz radio receiver
 - CD4046 PLL
 - ATmega328p with a 10368.000kHz Crystal Oscillator
 - LCD1602 with I2C

Compiled with Arduino IDE v1.8.16


 *
 *                                ATmega328p
 *
 *                     (RESET) PC6  1    28 PC5 (ADC5/SCL) = lcd1602 SCL
 *                       (RXD) PD0  2    27 PC4 (ADC4/SDA) = lcd1602 SDA
 *                       (TXD) PD1  3    26 PC3 (ADC3)
 *                      (INT0) PD2  4    25 PC2 (ADC2)
 *    freqOutPin = 11   (OC2B) PD3  5    24 PC1 (ADC1)
 *                    (XCK/T0) PD4  6    23 PC0 (ADC0)  A0 = dataInPin
 *                             VCC  7    22 AGND
 *                             GND  8    21 AREF
 *     XTAL1    (XTAL1/TOSC1) PB6  9    20 AVCC
 *     XTAL2    (XTAL2/TOSC2) PB7 10    19 PB5          13 = blink led
 *                       (T1) PD5 11    18 PB4          12 = blink led
 *                     (AIN0) PD6 12    17 PB3 (OC2A)
 *                     (AIN1) PD7 13    16 PB2          10 = activity led
 *  SecondSynchro = 8  (ICP1) PB0 14    15 PB1 (OC1A)    9 = MilliSynchro


Code versions
-------------

V0.9       11 Aug 2023     first full working version


 */


// parameters
#define VERSION         0.90     // code version
//#define DEBUG              1     // debug mode only

#define freqOut      162000L     // 162Khz clock generated for radio demodulation
#define freqCPU    10368000L     // note F_CPU still defined as 16000000 by Arduino IDE
#define offset             6     // serial data read offset: define sensibility

// pin definition
```

```
#define dataInPin          0     // analog read the instantaneous voltage of serial time data
activity

// libraries call
#include <Wire.h>                // i2c communication
#include <LiquidCrystal_I2C.h>   // https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-
library
LiquidCrystal_I2C lcd(0x27, 16, 2);

// variables
int dataIn = 512, memo_dataIn = 512;
unsigned int counter10ms = 0;          // 10ms counter during one minute
unsigned int counter100ms = 0;         // 100ms counter during one second
unsigned int noActivityCounter = 1;    // no activity 10ms counter to find the 59th second
synchro
unsigned int secondsCounter = 0;       // simple counter for seconds
bool activity = true;                  // flag to detect the 59th second
bool memo_timeBit, timeBit = false;    // binary data
volatile bool synchro = false;         // 59th second detected
volatile bool bitHigh = LOW;           // flag HIGH data during the 2nd 100ms of each second
volatile bool displayEnable = false;   // flag to enable update display
bool blinkLed = false;                 // blinking led
bool runOnce = true;
byte minuteU = 0, minuteD = 0, hourU = 0, hourD = 0;
byte wday = 0, dayU = 0, dayD = 0, monthU = 0, monthD = 0, yearU = 0, yearD = 0;
String weekDay[] = { "Lun", "Mar", "Mer", "Jeu", "Ven", "Sam", "Dim" };
String months[] = { "Janv", "Fevr", "Mars", "Avri", "Mai ", "Juin",
                    "Juil", "Aout", "Sept", "Octo", "Nove", "Dece" };
bool holiday = false, summer = false;
bool parity1 = true, parity2 = true, parity3 = true;
byte bitHighCounter = 0;
bool testFullMinute = 0;
bool displayReady = false;

// special characters for LCD
uint8_t low[8]  = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1};   // single dot bottom right
uint8_t high[8]  = {0x1, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0};  // single dot top right

#ifdef DEBUG
 volatile unsigned int memo_counter10ms = 0;
 volatile unsigned int memo_secondsCounter = 0;
 volatile unsigned int minutesCounter = 0;
#endif

// port access definition
#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif
#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif


//
// setup
//_____________________________________________________________________________________

void setup() {

  DDRB = 0xFF;    // set all portB as output = PB0 to PB5, I/O 8 to 13, pins 14 to 19
  DDRD = 0xFF;    // set all portD as output = PD0 to PD5

// Set the timers
//----------------------------------------------------------------------

  noInterrupts();

// set timer 1 to interrupt every 10ms (100Hz) used for ADC sampling rate
// Clear Timer on Compare Match (CTC) Mode
// precaler setting : freqCPU/8, N = 8
  TCCR1A = 0;
  TCCR1B = 0;
  TCCR1B = bit(WGM12) | bit(CS11);
  OCR1A  = ( freqCPU / 8L / 100L ) - 1;
  TIMSK1 = bit(OCIE1A);

// set timer 2 to get 162kHz clock on output 3
// from examples: http://www.gammon.com.au/timers
  TCCR2A = 0;
  TCCR2B = 0;
  TCCR2A = bit(WGM20) | bit(WGM21) | bit(COM2B1);  // fast PWM, clear OC2A on compare
  TCCR2B = bit(WGM22) | bit(CS20);                 // fast PWM, no prescaler
  OCR2A = (freqCPU / freqOut) - 1;          // set the output signal frequency
  OCR2B = ((OCR2A + 1) / 2) - 1;            // 50% duty cycle
```

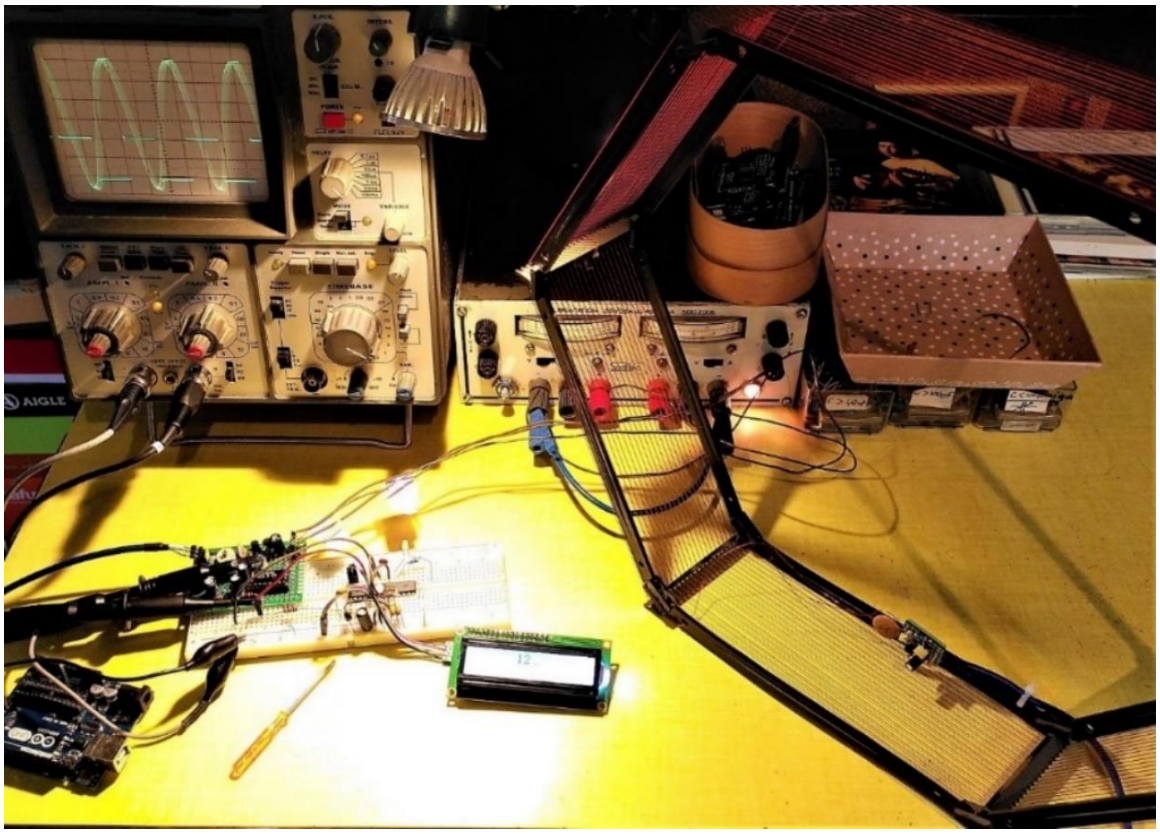
```
  interrupts();

// change the ADC prescaler to 16 instead of 128 by default : 16x faster
// from examples: https://zestedesavoir.com/billets/2068/arduino-accelerer-analogread/
  sbi(ADCSRA, ADPS2);
  cbi(ADCSRA, ADPS1);
  cbi(ADCSRA, ADPS0);

// start the LCD display
  Wire.begin();
  Wire.setClock(400000);        // Change clock speed from 100k(default) to 400kHz
  lcd.begin();
  lcd.clear();
  lcd.createChar(0, low);       // special character declaration
  lcd.createChar(1, high);
  lcd.setCursor(0, 0);
  lcd.print("horloge ALS162");
  lcd.setCursor(0, 1);
  lcd.print("version v");
  lcd.print(VERSION);
  delay(1000);
  lcd.clear();
}

//
// loop
//_____________________________________________________________________________________

void loop() {

// once per second: display sequence
  if( displayEnable ) {
    if( runOnce ) {
      runOnce = false;
      #ifdef DEBUG
       SecondSynchro(HIGH);               // make a debug signal each second I/O 8 pin 14
       lcd.setCursor(0, 0);
       lcd.print("                ");    // very much faster than lcd.clear()
       lcd.setCursor(0, 0);
       lcd.print(secondsCounter);
       lcd.setCursor(4, 0);
       lcd.print(bitHigh);
       lcd.setCursor(10, 0);
       lcd.print(memo_secondsCounter);
       lcd.setCursor(0, 1);
       lcd.print("                ");
       lcd.setCursor(0, 1);
       lcd.print(minutesCounter);
       lcd.setCursor(10, 1);
       lcd.print(memo_counter10ms);
      #else
       lcd.setCursor(6, 0);
       if( secondsCounter < 10 ) lcd.print("0");
       lcd.print(secondsCounter);
       lcd.setCursor(8, 0);
       if(bitHigh) lcd.write(1);
       else        lcd.write(0);
       switch( secondsCounter ) {
        case  0:
          if(displayReady) {
            lcd.setCursor(0, 0);
            if(parity2) { lcd.print(hourD); lcd.print(hourU); }
            else          lcd.print("--");
            lcd.print(":");
            if(parity1) { lcd.print(minuteD); lcd.print(minuteU); }
            else          lcd.print("--");
            lcd.print(":");
            lcd.setCursor(9, 0);
            if( summer )  lcd.print(" ete   ");
            else          lcd.print(" hiver ");
            lcd.setCursor(13, 0);
            if( holiday ) lcd.print("/fe");
            lcd.setCursor(0, 1);
            lcd.print("                ");
            lcd.setCursor(0, 1);
            if(parity3) {
              if( --wday >= 0 ) lcd.print(weekDay[wday]); lcd.print(" ");
              lcd.print((dayD *10) + dayU); lcd.print(" ");
              int monthN = (monthD *10) + monthU;
              if( --monthN >= 0 ) lcd.print(months[monthN]); lcd.print(" ");
              lcd.print((yearD *10) + yearU + 2000);
              if( dayD == 0 ) lcd.print(" ");  // to insure to cover the 16 characters of the
line
```

```
    }
  }
  else {
    lcd.setCursor(0, 0);
    lcd.print("      00        ");
    lcd.print(secondsCounter);
    lcd.setCursor(0, 1);
    lcd.print("                ");
  }
  minuteU = 0; minuteD = 0; hourU = 0; hourD = 0;
  wday = 0; dayU = 0; dayD = 0; monthU = 0; monthD = 0; yearU = 0; yearD = 0;
  parity1 = true; parity2 = true; parity3 = true;
  bitHighCounter = 0;
  break;
case  1:
  if( synchro ) testFullMinute = true;
  else          testFullMinute = false;
  break;
case 14:
  holiday = bitHigh;
  break;
case 17:
  summer = bitHigh;
  break;
case 21:
case 22:
case 23:
case 24:
  minuteU = minuteU | bitHigh << ( secondsCounter - 21 );
  if( bitHigh ) bitHighCounter++;
  break;
case 25:
case 26:
case 27:
  minuteD = minuteD | bitHigh << ( secondsCounter - 25 );
  if( bitHigh ) bitHighCounter++;
  break;
case 28:
  if((bitHighCounter %2) != bitHigh ) parity1 = false;
  bitHighCounter = 0;
  break;
case 29:
case 30:
case 31:
case 32:
  hourU = hourU | bitHigh << ( secondsCounter - 29 );
  if( bitHigh ) bitHighCounter++;
  break;
case 33:
case 34:
  hourD = hourD | bitHigh << ( secondsCounter - 33 );
  if( bitHigh ) bitHighCounter++;
  break;
case 35:
  if((bitHighCounter %2) != bitHigh ) parity2 = false;
  bitHighCounter = 0;
  break;
case 36:
case 37:
case 38:
case 39:
  dayU = dayU | bitHigh << ( secondsCounter - 36 );
  if( bitHigh ) bitHighCounter++;
  break;
case 40:
case 41:
  dayD = dayD | bitHigh << ( secondsCounter - 40 );
  if( bitHigh ) bitHighCounter++;
  break;
case 42:
case 43:
case 44:
  wday = wday | bitHigh << ( secondsCounter - 42 );
  if( bitHigh ) bitHighCounter++;
  break;
case 45:
case 46:
case 47:
case 48:
  monthU = monthU | bitHigh << ( secondsCounter - 45 );
  if( bitHigh ) bitHighCounter++;
  break;
case 49:
  monthD = bitHigh;
```

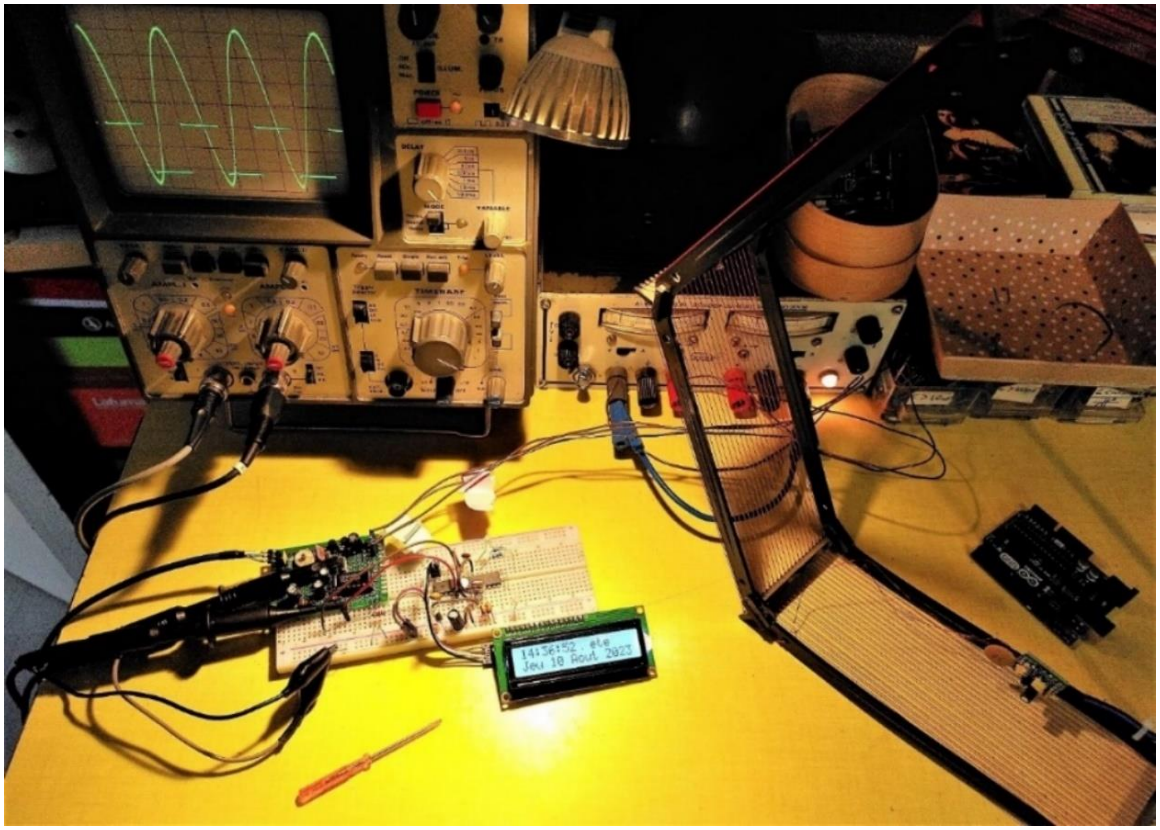
```
          if( bitHigh ) bitHighCounter++;
          break;
        case 50:
        case 51:
        case 52:
        case 53:
          yearU = yearU | bitHigh << ( secondsCounter - 50 );
          if( bitHigh ) bitHighCounter++;
          break;
        case 54:
        case 55:
        case 56:
        case 57:
          yearD = yearD | bitHigh << ( secondsCounter - 54 );
          if( bitHigh ) bitHighCounter++;
          break;
        case 58:
          if(( bitHighCounter %2) != bitHigh) parity3 = false;
          break;
        case 59:
          synchro = false;
          if( testFullMinute ) displayReady = true;
          else                 displayReady = false;
          break;
       }  // end of switch
      #endif

      secondsCounter++;
      blinkLed = !blinkLed;
      BlinkingLed( blinkLed, synchro );
    }    // end test runOnce
  }      // end test displayEnable

  else  {
    runOnce = true;
    #ifdef DEBUG
     SecondSynchro(LOW);
    #endif
  }
}        // end of loop

//================================================================================================
// list of functions
//================================================================================================

//
// Interrupt function run every 10ms
//________________________________________________________________________________________________

ISR(TIMER1_COMPA_vect) {

// set times counters
  counter10ms ++;                          // count from 0 to ~6040 during 1 minute
  counter100ms = (counter10ms / 10) % 10;  // count from 0 to 9 for each second

// check demodulated radio signal
  memo_timeBit = timeBit;
  memo_dataIn = dataIn;
  dataIn = analogRead( dataInPin );
  if(dataIn < memo_dataIn - offset) {
    timeBit = HIGH;
    noActivityCounter = 0;
  }
  else {
    timeBit = LOW;
    noActivityCounter++;
  }

// translate signal to bits
  if( counter100ms == 0 ) {
    if( timeBit ) {
      ActivityLed(HIGH);
    }
    else ActivityLed(LOW);
  }
  else if( counter100ms == 1 ) {
    if( timeBit ) {
      ActivityLed(HIGH);
      if( !memo_timeBit ) bitHigh = HIGH;  // only here there is the interesting data
    }
    else ActivityLed(LOW);
  }

// update display enable moment
```

```
  else if( counter100ms == 2 ) displayEnable = true;
  else {
    bitHigh = LOW;
    displayEnable = false;
  }

// detect a full second with no activity = 59th second to start teh synchro
  if( noActivityCounter > 99 ) activity = false;
  if( !activity && (noActivityCounter == 0)) {
    activity = true;
    synchro = true;
    #ifdef DEBUG
     memo_counter10ms = counter10ms;
     memo_secondsCounter = secondsCounter;
     minutesCounter++;
    #endif
    counter10ms = 0;
    secondsCounter = 0;
  }    // end of test !activity

// make a debug signal each 10ms I/O 9 pin 15
  #ifdef DEBUG
   static bool half = false;
   half = !half;
   if( half ) MilliSynchro(HIGH);
   else       MilliSynchro(LOW);
  #endif
}

//
// ActivityLed() = function to drive activity led
//_____________________________________________________________________________

void ActivityLed( bool s ) {
  if( s ) PORTB |= B000100;    // output 10 to HIGH
  else    PORTB &= B111011;    // output 10 to LOW
}

//
// SecondSynchro() = gives second syncho for debugging
//_____________________________________________________________________________

void SecondSynchro( bool s ) {
  if( s ) PORTB |= B000001;    // output 8 to HIGH
  else    PORTB &= B111110;    // output 8 to LOW
}

//
// MilliSynchro() = gives synchro every 20ms
//_____________________________________________________________________________

void MilliSynchro( bool s ) {
  if( s ) PORTB |= B000010;    // output 9 to HIGH
  else    PORTB &= B111101;    // output 9 to LOW
}

//
// BlinkingLed() = function to drive blinking led
//_____________________________________________________________________________

void BlinkingLed( bool s, bool c ) {
  if( !s ) {
    PORTB &= B011111;      // output 13 to LOW
    PORTB &= B101111;      // output 12 to LOW
  }
  else {
    if( c ) {
      PORTB |= B100000;    // output 13 to HIGH
      PORTB &= B101111;    // output 12 to LOW
    }
    else {
      PORTB |= B010000;    // output 12 to HIGH
      PORTB &= B011111;    // output 13 to LOW
    }
  }
}
```

Illustrations



When starting up : only the seconds are displayed, waiting after the 59th second to start decoding the time.



This shows the time (and date, and summer/winter time).
Note on the oscilloscope: the perfect synchronisation between the 162 kHz signal generated (square wave) and the radio signal (sinusoidal), which clearly shows the phase-locked loop.