BIBLIOTEKA
POLSKIEGO KRÓTKOFALOWCA 21

Krzysztof Dąbrowski
OE1KDA

Arduino w krótkofalarstwie
Tom 2

Wiedeń 2013

Arduino w krótkofalarstwie

Tom 2

Krzysztof Dąbrowski OE1KDA

Wydanie 1
Wiedeń, listopad 2013

Spis treści

Wstęp

Obfitość materiału związanego z krótkofalarskimi zastosowaniami Arduino spowodowała konieczność podziału skryptu na dwa tomy. Na treść tomu drugiego składają się przykłady programów sterujących scalone syntezery cyfrowe, generujacych przy ich użyciu sygnały różnych emieji cyfrowych, programów odczytujących i dekodujących wzorcowe sygnały czasu lub przydatne do celów pomiarowych. Liczba programów przydatnych dla krótkofalowców będzie z pewnością szybko rosła co zaowocuje dalszymi tomami niniejszego skryptu.
Programy przytoczone są w całości co wprawdzie oznacza, że pewne ich części powtarzają się ale oszczędza to zmudnego poszukiwania ich w innych rozdziałach. Ze względu na to, że są one przeznaczone do bezpośredniego kopiowania do edytora Arduino w komentarzach zrezygnowano z polskich liter tak jak w progamach zamieszczonych w tomie 1.
Przed uruchomieniem skopiowanych programów konieczne jest nie tylko wprowadzenie do nich własnych danych jak znaki wywwołwacze itp. ale także dopasowanie wyświetlanych lub nadawanych tekstów i komunikatów.

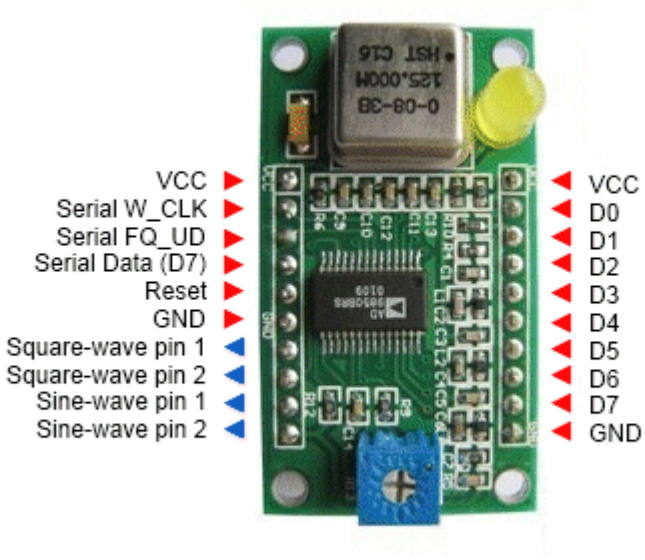
Krzysztof Dąbrowski OE1KDA
Wiedeń
Listopad 2013

Sterowanie syntezerów cyfrowych

Zasada pracy syntezerów cyfrowych została opisana w tomie pierwszym skryptu. Najważniejszą różnicą w stosunku do przedstawionych tam rozwiązań jest zakres częstotliwości wyjściowych. Szybkość pracy procesora Arduino pozwala na bezpośrednią programową syntezę sygnałów jedynie w zakresie niskich częstotliwości – częstotliwości akustycznych. Dla zakresu w.cz. konieczne jest podłączenie do procesora zewnętrznego układu syntezera zawierającego na przykład obwody scalone z serii AD98xx lub AD99xx. Arduino służy wówczas do sterowania syntezerem, kluczowania lub modulacji sygnału wyjściowego i zapewnienia wygody obsługi przez użytkownika.

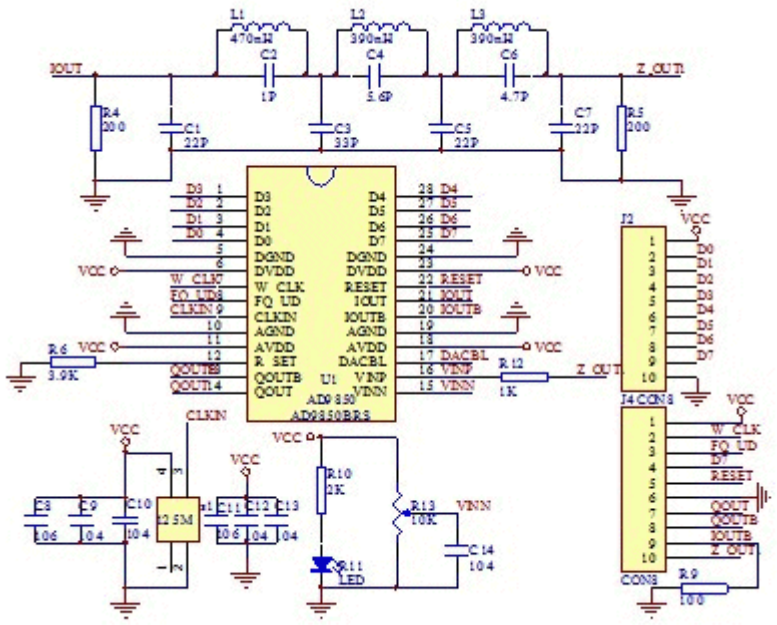
Chiński moduł z obwodem AD9850

Na rynku dostępnych jest szereg gotowych modułów syntezerów cyfrowych z obwodami scalonymi z serii AD98xx i AD99xx. Jednym z nich jest opracowany przez amerykański klub QRP moduł DDS-60 oparty na AD9851 a drugim (popularny ze względu na cenę) moduł produkcji chińskiej wyposażony w syntezer AD9850. Posiada on wyjścia sygnału sinusoidalnego (*sin-wave pin 1* i *sin-wave pin2*) i prostokątnego (*square-wave pin 1* i *square-wave pin 2*) a użyteczna częstotliwość wyjściowa wynosi ok. 40 MHz dzięki czemu pokrywa on cały zakres fal krótkich. Oprócz układu syntezera zawiera on generator częstotliwości odniesienia (zegarowej) i wymaga jedynie połączenia z mikroprocesorem. Sterowanie syntezerem przez procesor może odbywać się za pomocą złącza szeregowego (kontakty *Serial W_CLOCK*, *Serial FQ_UD*, *Serial data D7*) lub równoległego o szerokości 8 bitów (kontakty D0 – D7). Oprócz tego wymaga on jedynie podłączenia napięcia zasilania 5 V.

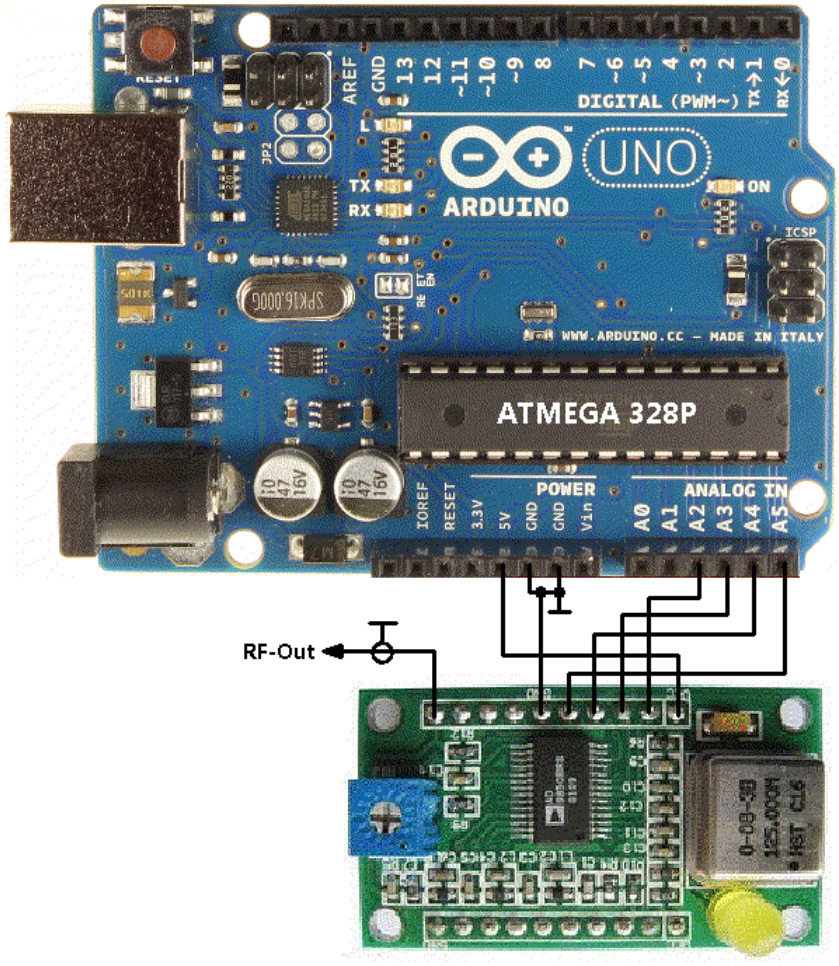


Rys. 1.1. Wygląd płytki syntezera i jej wyprowadzenia

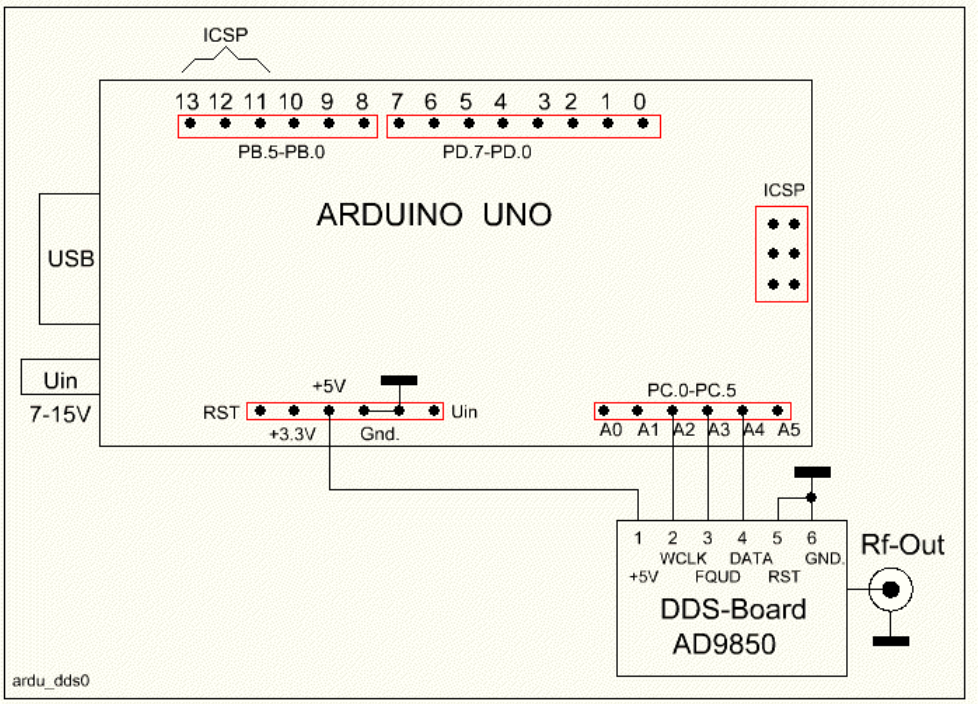
Jak wynika z przedstawionego na rys. 1.2 schematu ideowego układ zawiera obwód scalony AD9850, kwarcowy generator odniesienia o częstotliwości 125 MHz i niezbędny w rozwiązaniach tego typu filtr dolnoprzepustowy na wyjściu sygnału sinusoidalnego. Jest to filtr trzysegmentowy o charakterystyce eliptycznej (Cauera).
Scalone syntezery cyfrowe posiadają najczęściej jedno lub dwa wyjścia prądowe, które należy obciążyć opornikiem o wartości podanej w katalogu. Na schemacie z rysunku 1.2 jest to opornik o wartości 200 Ω. Drugie z wyjść może w zależności od typu syntezera dostarczać przykładowo sygnału kwadraturowego.
Również z zależności od typu syntezera możliwa jest modulacja amplitudy sygnału i fazy wyjściowego.



Rys. 1.2. Schemat ideowy syntezera.

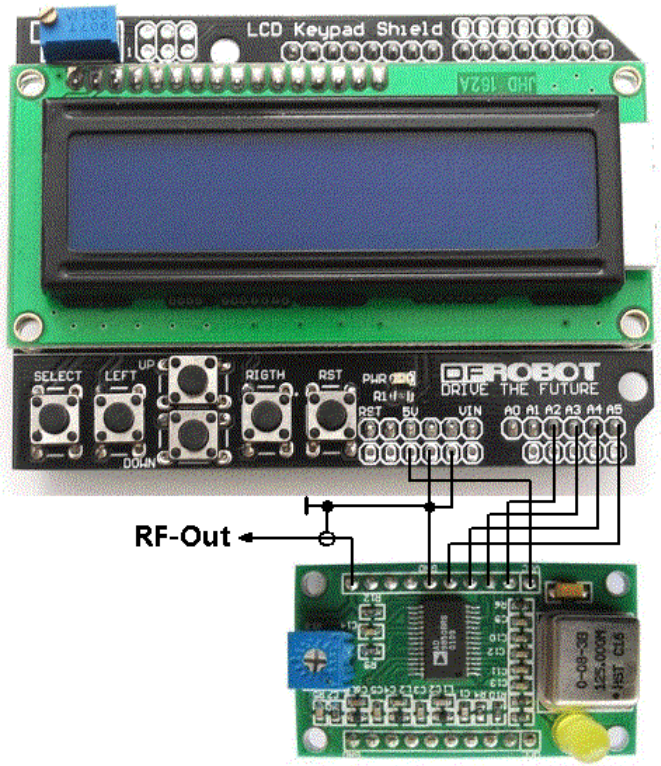


Rys. 1.3. Sposób podłączenia syntezera do Arduino UNO



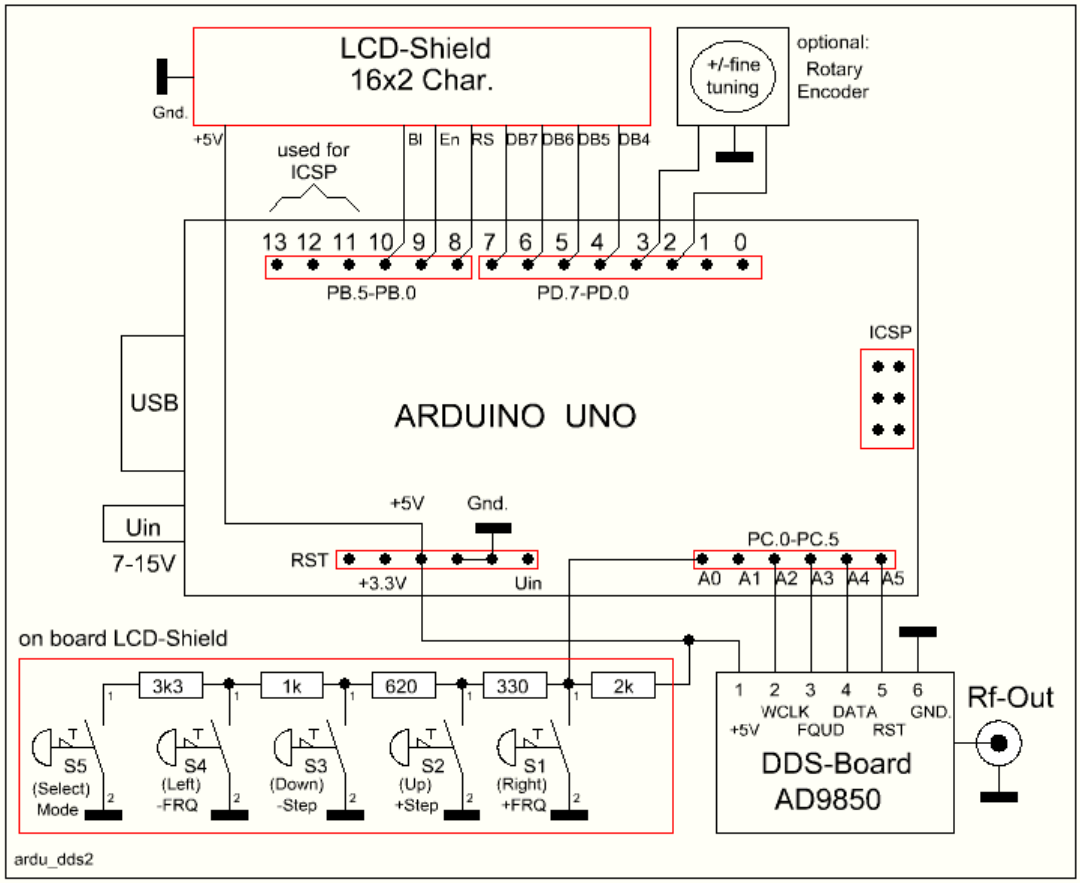
Rys.1.4. Schemat połączeń między płytkami syntezera i Arduino

Generator na zakres 0 – 40 MHz z AD9850



Rys. 1.5. Sposób podłączenia wyświetlacza z klawiaturą

Układ generatora pracującego w zakresie 0 – 40 MHz zawiera dodatkowo do Aurduino i właściwego syntezera wyświetlacz ciekłokrystaliczny wyposażony w pięć przycisków wykorzystanych do jego sterowania i strojenia. Alternatywnym rozwiązaniem może być użycie gałki z koderem impulsowym. Przyciski S3 i S4 służą do wyboru kroku przestrajania a S1 i S4 do zmiany częstotliwości. S5 służy do wywołania jednej z 9 zaprogramowanych częstotliwości.
W odróżnieniu od większości przytoczonych w skrypcie programów przykładowy program sterujący napisany jest w języku Bascom. Po skompilowaniu musi on zostać załadowany do procesora przez złącze ICSP. Archiwum programu jest dostępne w internecie pod adresem *http://www.kh-gps.de/dds.zip*.



Rys. 1.6. Schemat połączeń

Program sterowany klawiszami

```
'DDS-Ardu_Basic  plytka ARDUINO z wyswietlaczem LCD i klawiatura
'                              Program generuje pojedyncza czestotliwosc
'
$regfile = "m328def.dat"                    'Typ procesora: ATMEGA328
$crystal = 16000000                         'Czestotliwosc zegarowa procesora w Hz
$hwstack = 32                               'Stos sprzetowy
$swstack = 8                                'Stos programowy
$framesize = 20                             'parametr „Framsize”

'******************************
'    Prosty probny program obslugi syntezera DDS
'   wykorzystujacy pomysly Nielsa i JA9TTT
'  DDS AD9850   Czestotliwosc zegarowa :125MHZ
'******************************
'Polaczenia
```

```
'Linia C.2 = DDS WCLK
'Linia C.3 = DDS FQUD
'Linia C.4 = DDS DATA
'Linia C.5 = DDS RST

'Linia D.4 = LCD DB4 ( nie konieczne )
'Linia D.5 = LCD DB5 ( nie konieczne )
'Linia D.6 = LCD DB6 ( nie konieczne )
'Linia D.7 = LCD DB7 ( nie konieczne )

'Linia B.0 = LCD Rs  ( nie konieczne )
'Linia B.1 = LCD En  ( nie konieczne )
'Linia B.2 = LCD podswietlenie ( nie konieczne )

Dim Dds_cmd As Byte
Dim Freq_data As Long
Dim Freq_accu As Single
Dim F_in As Single

Dim Frq As Single
Dim Frq_str As String * 10

Const Clk_in = 125000000                    'Czestotliwosc zegarowa syntezera
Const Braincells = &H100000000

Ddrb = &B11111111
Ddrc = &B11111111
Portc = &H00

Config Lcdpin = Pin , Db7 = Portd.7 , Db6 = Portd.6 , Db5 = Portd.5 , Db4 = Portd.4 , Rs = Portb.0 , E
= Portb.1
Config Lcd = 16 * 2
Cls
Cursor Off

Portb.2 = 1
Portc.5 = 0

Cls
Locate 1 , 1
Lcd " DDS-Generator "
Locate 2 , 1
Lcd " Basic Version "
Wait 3

'Czestotliwosc wyjsciowa w Hz ( dostosowac do potrzeb )
F_in = 7080000                              '7080000 Hz = 7080 KHz

Freq_accu = F_in * Braincells
Freq_accu = Freq_accu / Clk_in
Freq_data = Freq_accu

Dds_cmd = &B00000000

Gosub Init_dds
```

```
Waitms 100
Gosub Dds_output

Frq = F_in / 1000
Frq_str = Fusing(frq , "#.##")

Cls
Locate 1 , 1
Lcd "F: " ; Frq_str ; " kHz"
Locate 2 , 1
Lcd "H: " ; Freq_data

End

'********** Inicjalizacja syntezera ***********
Init_dds:
Reset Portc.4                              'DATA
Reset Portc.2                              'WCLK
Waitus 10
Set Portc.2                               'WCLK
Waitus 10
Reset Portc.2                              'WCLK
Waitus 10
Set Portc.3                               'FQUD
Waitus 10
Reset Portc.3                              'FQUD
Waitus 10
Return

'********** DATA > DDS *********
Dds_output:
Shiftout Portc.4 , Portc.2 , Freq_data , 3 , 32 , 0        'DATA,WCLK
Shiftout Portc.4 , Portc.2 , Dds_cmd , 3 , 8 , 0           'DATA,WCLK
Waitus 10
Set Portc.3                               'FQUD
Waitus 10
Reset Portc.3                              'FQUD
Return
```

Program korzystający z gałki

```
'Generator-syntezer 5
'
'Wersja dla ARDUINO z koderem obrotowym
'
'strojenie precyzyjne wylacznie z krokiem 10 Hz
'
'******************************
'    Prosty probny program obslugi syntezera DDS
'  oparty na pomyslach Kato, JA9TTT, Nielsa i Stefana Hoffmanna
'  DDS AD9850   Czestotliwosc zegarowa:125MHZ
'******************************
'Polaczenia
'Linia D.2 = wyjscie obnizajace kodera
'Linia D.3 = wyjscie podwyzszajace kodera
```

```
'Linia C.0 = analogowe odpytanie przyciskow

'Linia C.2 = DDS WCLK
'Linia C.3 = DDS FQUD
'Linia C.4 = DDS DATA
'Linia C.5 = DDS RST

'Linia D.4 = LCD DB4
'Linia D.5 = LCD DB5
'Linia D.6 = LCD DB6
'Linia D.7 = LCD DB7

'Linia B.0 = LCD Rs
'Linia B.1 = LCD En
'Linia B.2 = LCD podswietlenie

$regfile = "M328def.dat"
$crystal = 16000000
$hwstack = 32
$swstack = 10
$framesize = 40
$eeprom
'$sim

Config Lcdpin = Pin , Db7 = Portd.7 , Db6 = Portd.6 , Db5 = Portd.5 , Db4 = Portd.4 , Rs = Portb.0 , E = Portb.1
Config Lcd = 16 * 2
Cls
Cursor Off

Config Adc = Single , Prescaler = Auto , Reference = Avcc   'Wybor przetwornika analogowego

Enable Interrupts

Declare Sub Buttonscan
Declare Sub Rast
Declare Sub Frq_upd
Declare Sub Modi
Declare Sub Mem_store
Declare Sub Mem_read
Declare Sub Write_dds

Dim Ana As Word
Dim Offen As Bit
Dim Auswahl As Bit
Dim Links As Bit
Dim Down As Bit
Dim Up As Bit
Dim Rechts As Bit
Dim Mm As Bit
Dim Spacing As Long
Dim Valu As String * 5
Dim Cnt As Long
Dim Frq As Long
```

```
Dim Frq_strt As Long
Dim Frq_old As Long
Dim Frq2 As Single
Dim Frqstr As String * 15
Dim Raster As Byte
Dim Modicnt As Byte
Dim Oldmodicnt As Byte
Dim Modi2 As String * 8
Dim Mem2 As String * 1
Dim Oldfrq As Long
Dim Ch As Byte
Dim Xx As Long

Dim Dds_cmd As Byte
Dim Freq_data As Long
Dim Freq_accu As Single

Dim Timerstartwert As Word
Dim Zustandswechsel As Byte

Ddrb = &B11111111
Ddrc = &B11111110                    'Linie logiczne: 0 = wejscie; 1 = wyjscie
Ddrd = &B11110011

Portb = &B00000100
Portc = &B00000001
Portd = &B00001100                   'Linie logiczne: 1 =  wlaczony opornik podtrzymujacy

Encoder_a Alias Pind.2
Encoder_b Alias Pind.3

Const Clk_in = 125000000             'Czestotliwosc zegarowa syntezera
Const Braincells = &H100000000

'Licznik dla kodera obrotowego
Config Timer1 = Timer , Prescale = 64
On Timer1 Drehencoder
Enable Timer1
Enable Interrupts

Timerstartwert = 65411
Timer1 = Timerstartwert

Wait 1

Raster = 2                           'R3 = 100 Hz
Modicnt = 0
Ch = 1

Gosub Init_dds                       'Inicjalizacja syntezera

Locate 1 , 1                         'Wyswietlanie informacji na wyswietlaczu LCD
Lcd " DDS-GENERATOR  "
Locate 2 , 1
Lcd "(C)KHH 2013 V1.5"
```

```
Wait 2
Cls

Gosub Mem_store
Frq_strt = 14070000                         'Czestotliwosc poczatkowa w Hz
Frq = Frq_strt

'glowna petla programu
'-----------------------------------------------------------------------
Do
If Frq < 100000 Or Frq > 40000000 Then Frq = Frq_strt
Ana = Getadc(0)
Gosub Buttonscan
Gosub Rast
Gosub Frq_upd
Gosub Modi

If Modicnt = 0 Then Modi2 = "VFO"
If Modicnt > 0 Then Modi2 = " M"

'If Modicnt = 0 Then Gosub Frq_upd
If Modicnt > 0 And Mm = 1 Then Readeeprom Frq , Xx

'Wyswietlanie ustawien
Locate 1 , 1
Frq2 = Frq / 1000                           '/ 1000
Frqstr = Fusing(frq2 , "#.##")
Lcd "F: " ; Frqstr ; "KHz  "

Locate 2 , 1
If Spacing > 999 Then Valu = "kHz" Else Valu = "Hz"
If Spacing > 999 Then Spacing = Spacing / 1000
Lcd "R: " ; Spacing ; Valu ; "    "

'If Modicnt = 0 Then Lcd "R: " ; Spacing ; Valu ; "    "
'If Modicnt > 0 Then Lcd "                "

Locate 2 , 14
If Modicnt = 0 Then Lcd Modi2 ; "    "
If Modicnt > 0 Then Lcd Modi2 ; Modicnt ; "  "

If Frq <> Frq_old Then Gosub Write_dds

Fin:
Frq_old = Frq
Waitms 10
Loop

End

'Odczyt przyciskow
Sub Buttonscan
   If Ana > 882 Then Offen = 1 Else Offen = 0
   If Ana > 624 And Ana =< 882 Then Auswahl = 1 Else Auswahl = 0
   If Ana > 417 And Ana =< 624 Then Links = 1 Else Links = 0
```

```
    If Ana > 236 And Ana =< 417 Then Down = 1 Else Down = 0
    If Ana > 72 And Ana =< 236 Then Up = 1 Else Up = 0
    If Ana =< 72 Then Rechts = 1 Else Rechts = 0
End Sub
'-------------------------------------------------
'zmiana kroku strojenia
Sub Rast
Local Oldrast As Byte
Oldrast = Raster
  If Up = 1 Then Raster = Raster + 1
  If Down = 1 Then Raster = Raster -1
  If Raster > 6 Then Raster = 1
  If Raster < 1 Then Raster = 6
  If Raster = 1 Then Spacing = 10
  If Raster = 2 Then Spacing = 100
  If Raster = 3 Then Spacing = 1000
  If Raster = 4 Then Spacing = 10000
  If Raster = 5 Then Spacing = 100000
  If Raster = 6 Then Spacing = 1000000
  If Raster <> Oldrast Then Waitms 200
  Waitms 10
End Sub
'-------------------------------------------------
'Tryby pracy syntezera
Sub Modi
  Local Oldmodicnt As Byte
  Oldmodicnt = Modicnt
  If Auswahl = 1 Then Modicnt = Modicnt + 1
  If Modicnt > 9 Then Modicnt = 0
  If Modicnt = 1 Then Xx = &H08
  If Modicnt = 2 Then Xx = &H18
  If Modicnt = 3 Then Xx = &H28
  If Modicnt = 4 Then Xx = &H38
  If Modicnt = 5 Then Xx = &H48
  If Modicnt = 6 Then Xx = &H58
  If Modicnt = 7 Then Xx = &H68
  If Modicnt = 8 Then Xx = &H78
  If Modicnt = 9 Then Xx = &H88
  If Modicnt <> Oldmodicnt Then Mm = 1 Else Mm = 0
  If Modicnt <> Oldmodicnt Then Waitms 300
  Waitms 10
End Sub
'-------------------------------------------------
'Zmiana czestotliwosci
  Sub Frq_upd
  Oldfrq = Frq
  If Links = 1 Then Frq = Frq - Spacing
  If Rechts = 1 Then Frq = Frq + Spacing
  If Frq <> Oldfrq Then Waitms 200
  Waitms 10
End Sub
'-------------------------------------------------
'nowa czestotliwosc do syntezera
Sub Write_dds
  Freq_accu = Frq * Braincells
```

```
  Freq_accu = Freq_accu / Clk_in
  Freq_data = Freq_accu
  Dds_cmd = &B00000000
  'Gosub Init_dds
  Gosub Dds_output
  Waitms 10
End Sub
'-------------------------------------------------------
'Zapis w pamieci EEPROM
Sub Mem_store
  Local Bb As Long
  Bb = 136000
  Writeeeprom Bb , &H08
  Bb = 474200
  Writeeeprom Bb , &H18
  Bb = 1800000
  Writeeeprom Bb , &H28
  Bb = 3600000
  Writeeeprom Bb , &H38
  Bb = 7080000
  Writeeeprom Bb , &H48
  Bb = 10070000
  Writeeeprom Bb , &H58
  Bb = 14070000
  Writeeeprom Bb , &H68
  Bb = 21070000
  Writeeeprom Bb , &H78
  Bb = 28070000
  Writeeeprom Bb , &H88
End Sub
'-------------------------------------------------------------
'inicjalizacja syntezera
Init_dds:
  Reset Portc.4                             'DATA
  Reset Portc.2                             'WCLK
  Waitus 10
  Set Portc.2                              'WCLK
  Waitus 10
  Reset Portc.2                             'WCLK
  Waitus 10
  Set Portc.3                              'FQUD
  Waitus 10
  Reset Portc.3                             'FQUD
  Waitus 10
Return
'-------------------------------------------------------------
' wydanie polecenia dla syntezera
Dds_output:
  Shiftout Portc.4 , Portc.2 , Freq_data , 3 , 32 , 0        'DATA,WCLK
  Shiftout Portc.4 , Portc.2 , Dds_cmd , 3 , 8 , 0           'DATA,WCLK
  Waitus 10
  Set Portc.3                              'FQUD
  Waitus 10
  Reset Portc.3                             'FQUD
Return
```

```
'-------------------------------------------------------------
' Odczyt kodera i interpretacja wyniku
Drehencoder:
  Timer1 = Timerstartwert
  Zustandswechsel.0 = Encoder_a
  Zustandswechsel.1 = Encoder_b

  Select Case Zustandswechsel
    Case &B0000_0010 : Frq = Frq + 10
    Case &B0000_0001 : Frq = Frq - 10
  End Select

  Zustandswechsel.4 = Zustandswechsel.0
  Zustandswechsel.5 = Zustandswechsel.1
Return
```

Sterowanie AD9850/9851 przez złącze szeregowe

Program w języku Arduino ilustruje sposób sterowania scalonego syntezera AD9850/9851 przez złącze szeregowe. Przykład ten i następne mogą być wykorzystane w innych bardziej rozbudowanych programach.

```
/**********************************************************************
**  Uklad scalony: AD9850/9851                                       **
**  Plik:   EF_AD9850_Serial.pde                                     **
**                                                                   **
**  Autor ElecFreaks Robi.W /28 pazdziernika 2011                    **
**                                                                   **
**  Opis:                                                            **
**  Przyklad sterowania AD9850/9851 przez Arduino za pomoca          **
**  zlacza szeregowego. Fala prostokatna i sinusoida wyswietlane na  **
**  LXARDOSCOPE bezplatnym programie oscyloskopu dla Arduino         **
**                                                                   **
**  Program przykladowy moze byc bezplatnie rozpowszechniany         **
**  na zasadach licencji GNU                                         **
**                                                                   **
**  Copyright (C) 2011 ElecFreaks Corp.                              **
**                                                                   **
**  http://www.elecfreaks.com                                        **
**********************************************************************/

//Definicje dla AD9850
#define REST      11
#define FQUP      10
#define CLK       9
#define BitData   8

//Definicje dla LXARDOSCOPE
int sensorValue = 0;        // wartosc odczytywana z potencjometru
byte lb;
byte hb;

void AD9850_Init(){
  pinMode(REST, OUTPUT);              // programowanie wyjsc Arduino sluzacych do
  pinMode(FQUP, OUTPUT);              // sterowania syntezerem
```

```
  pinMode(CLK , OUTPUT);
  pinMode(BitData, OUTPUT);
  digitalWrite(REST, 0);                    // sygnaly stanu spoczynkowego na zlaczu
  digitalWrite(FQUP, 0);
  digitalWrite(CLK, 0);
  digitalWrite(BitData, 0);
}

void AD9850_Reset_Serial(){                 // zerowanie
 digitalWrite(CLK, 0);
 digitalWrite(FQUP, 0);
 digitalWrite(REST, 0);                    // sygnal zerowania
 digitalWrite(REST, 1);
 digitalWrite(REST, 0);
 digitalWrite(CLK, 0);                     //sygnal zegarowy
 digitalWrite(CLK, 1);
 digitalWrite(CLK, 0);
 digitalWrite(FQUP, 0);                   //sygnal Fq-up
 digitalWrite(FQUP, 1);
 digitalWrite(FQUP, 0);
}

void AD9850_WR_Serial(unsigned char w0,double frequence){    //zapis danych do syntezera
 unsigned char i,w;
 long int y;
 double x;
 //Obliczanie czestotliwosci
 x=4294967295/125;//dla kwarcu 125 MHz
 frequence=frequence/1000000;
 frequence=frequence*x;
 y=frequence;

 //slowo sterujace w4
 w=(y>>=0);
 for(i=0; i<8; i++)
 {
  digitalWrite(BitData, (w>>i)&0x01);
  digitalWrite(CLK, 1);
  digitalWrite(CLK, 0);
 }
 //slowo sterujace w3
 w=(y>>8);
 for(i=0; i<8; i++)
 {
  digitalWrite(BitData, (w>>i)&0x01);
  digitalWrite(CLK, 1);
  digitalWrite(CLK, 0);
 }
 //slowo sterujace w2
 w=(y>>16);
 for(i=0; i<8; i++)
 {
  digitalWrite(BitData, (w>>i)&0x01);
  digitalWrite(CLK, 1);
  digitalWrite(CLK, 0);
```

```
  }
  //slowo sterujace w1
  w=(y>>24);
  for(i=0; i<8; i++)
  {
    digitalWrite(BitData, (w>>i)&0x01);
    digitalWrite(CLK, 1);
    digitalWrite(CLK, 0);
  }
  //slowo sterujace w0
  w=w0;
  for(i=0; i<8; i++)
  {
    digitalWrite(BitData, (w>>i)&0x01);
    digitalWrite(CLK, 1);
    digitalWrite(CLK, 0);
  }
  digitalWrite(FQUP, 1);
  digitalWrite(FQUP, 0);
}

void setup(){
  AD9850_Init();
  AD9850_Reset_Serial();
  AD9850_WR_Serial(0x00, 200); //500Hz
  // inicjalizacja zlacza szeregowego z szybkoscia 115200 bit/s:
  Serial.begin(115200);
}

void loop(){
  // odczyt wejscia analogowego A0:
  sensorValue = analogRead(A0);
// przesuniecie wartosci o 3 bity i wybor wyzszego bajtu
  hb=highByte(sensorValue<<3);
// ustawienie 3 najwyzszych bitow i nadanie
  Serial.print(hb|224,BYTE);
// wybor nizszego bajtu i wyzerowanie 3 najwyzszych bitow
  lb=(lowByte(sensorValue))&31;
// ustawienie bitow 5 i 6 i nadanie
  Serial.print(lb|96,BYTE);
// odczyt wejscia analogowego A1
  sensorValue = analogRead(A1);
// przesuniecie wartosci o 3 bity i wybor wyzszego bajtu
  hb=highByte(sensorValue<<3);
// ustawienie bitow 5 i 6 i nadanie
  Serial.print(hb|96,BYTE);
// wybor nizszego bajtu i wyzerowanie 3 najwyzszych bitow
  lb=(lowByte(sensorValue))&31;
// ustawienie bitow 5 i 6 i nadanie
  Serial.print(lb|96,BYTE);
}
```

Sterowanie AD9850/9851 przez złącze równoległe

```
/*********************************************************************
**  Uklad scalony: AD9850/9851                                      **
**  Plik:   EF_AD9850_Parallel.pde                                  **
**                                                                  **
**  Autor ElecFreaks Robi.W /28 pazdziernika 2011                   **
**                                                                  **
**  Opis:                                                           **
**  Przyklad sterowania AD9850/9851 przez Arduino za pomoca         **
**  zlacza rownoleglego. Fala prostokatna i sinusoida wyswietlane na**
**  LXARDOSCOPE bezplatnym programie oscyloskopu dla Arduino        **
**                                                                  **
**  Program przykladowy moze byc bezplatnie rozpowszechniany        **
**  na zasadach licencji GNU                                        **
**                                                                  **
**  Copyright (C) 2011 ElecFreaks Corp.                             **
**                                                                  **
**  http://www.elecfreaks.com                                       **
*********************************************************************/

//Definicje dla AD9850
#define REST      11
#define FQUP      10
#define CLK       9

#define BitData_Port PORTD
#define BitData_DIR  DDRD
#define BitData_IN   PIND

//Definicje dla LXARDOSCOPE
int sensorValue = 0;        // odczyt wartosci z potencjometru
byte lb;
byte hb;

void AD9850_Init(){
  pinMode(REST, OUTPUT);                  // programowanie wyjsc Arduino
  pinMode(FQUP, OUTPUT);                  // sluzacych do sterowania syntezerem
  pinMode(CLK , OUTPUT);
  digitalWrite(REST, 0);                  // stan spoczynkowy na zlaczu
  digitalWrite(FQUP, 0);
  digitalWrite(CLK, 0);

  BitData_DIR =  0xFF;    // ustawienie „IO OUTPUT”
  BitData_Port = 0x0;     // inicjalizacja „IO”
}

void AD9850_Reset(){
 digitalWrite(CLK, 0);
 digitalWrite(FQUP, 0);
 //Sygnal zerowania
 digitalWrite(REST, 0);
 digitalWrite(REST, 1);
 digitalWrite(REST, 0);
}
```

```
void AD9850_WR(unsigned char w0,double frequence){          //wpisanie danych do syntezera
  unsigned char i,w;
  long int y;
  double x;

  //Obliczenie czestotliwosci
  x=4294967295/125;//dla kwarcu 125 MHz
  frequence=frequence/1000000;
  frequence=frequence*x;
  y=frequence;
  //slowo sterujace w0
  w = w0;
  BitData_Port = w;
  digitalWrite(CLK, 1);
  digitalWrite(CLK, 0);
  //slowo sterujace w1
  w=(y>>24);
  BitData_Port = w;
  digitalWrite(CLK, 1);
  digitalWrite(CLK, 0);
  //slowo sterujace w2
  w=(y>>16);
  BitData_Port = w;
  digitalWrite(CLK, 1);
  digitalWrite(CLK, 0);
  //slowo sterujace w3
  w=(y>>8);
  BitData_Port = w;
  digitalWrite(CLK, 1);
  digitalWrite(CLK, 0);
  //slowo sterujace w4
  w=(y>>=0);
  BitData_Port = w;
  digitalWrite(CLK, 1);
  digitalWrite(CLK, 0);

  //wylaczenie wejscia
  digitalWrite(FQUP, 1);
  digitalWrite(FQUP, 0);
}

void setup(){
  AD9850_Init();
  AD9850_Reset();
  AD9850_WR(0x00, 200); //500 Hz
  // inicjalizacja zlacza szeregowego z szybkoscia 115200 bit/s:
  Serial.begin(115200);
}

void loop(){
  // odczyt wejscia analogowego A0:
  sensorValue = analogRead(A0);
// przesuniecie wartosci o 3 bity i wybor wyzszego bajtu
  hb=highByte(sensorValue<<3);
```

```
// ustawienie 3 najwyzszych bitow i nadanie
  Serial.print(hb|224,BYTE);
// wybor nizszego bajtu i wyzerowanie 3 najwyzszych bitow
  lb=(lowByte(sensorValue))&31;
// ustawienie bitow 5 i 6 i nadanie
  Serial.print(lb|96,BYTE);
// Odczyt wejscia nalogowego A1
  sensorValue = analogRead(A1);
// przesuniecie wartosci o3 bity i wybor wyzszego bajtu
  hb=highByte(sensorValue<<3);
// ustawienie bitow 5 i 6 i nadanie
  Serial.print(hb|96,BYTE);
// wybor nizszego bajtu i zerowanie 3 najwyzszych bitow
  lb=(lowByte(sensorValue))&31;
// ustawienie bitow 5 i 6 i nadanie
  Serial.print(lb|96,BYTE);
}
```

Biblioteka EF_AD9850

Sposób tworzenia własnych bibliotek dla Arduino został omówiony w dodatkach do tomu 1. Poniżej przedstawony jest przykład biblioteki ułatwiającej sterowanie scalonymi syntezerami cyfrowymi z rodziny AD9850.

Plik EF_AD9850.h

```
/*************************************************************************
**  Uklad scalony: AD9850/9851                                          **
**  Plik:   EF_AD9850.h                                                 **
**                                                                      **
**  Autor ElecFreaks Robi.W /28 pazdzierrnika 2011                      **
**                                                                      **
**  Opis:                                                               **
**  Przyklad sterowania AD9850/9851 przez Arduino za pomoca             **
**  zlacza szeregowego. Fala prostokatna i sinusoida wyswietlane na     **
**  LXARDOSCOPE bezplatnym programie oscyloskopu dla Arduino            **
**                                                                      **
**  Program przykladowy moze byc bezplatnie rozpowszechniany            **
**  na zasadach licencji GNU                                            **
**                                                                      **
**  Copyright (C) 2011 ElecFreaks Corp.                                 **
**                                                                      **
**  http://www.elecfreaks.com                                           **
*************************************************************************/

#ifndef __EF_AD9850_H__
#define __EF_AD9850_H__

#include "WProgram.h"
#include <avr/pgmspace.h>

class EF_AD9850{
      public:
            EF_AD9850(int D_CLK, int D_FQUP, int D_REST, int D_BitData);
    void init(void);
```

```
    void reset(void);
    void wr_serial(unsigned char w0,double frequence);
    void wr_parrel(unsigned char w0,double frequence);

  private:
    int BitData, FQUP, REST, Mode, CLK;

};

#endif
```

Plik EF_AD9850.cpp

```
/*************************************************************************
**  Uklad scalony: AD9850/9851                                          **
**  Plik:   EF_AD9850.cpp                                               **
**                                                                      **
**  Autor ElecFreaks Robi.W /28 pazdzierrnika 2011                      **
**                                                                      **
**  Opis:                                                               **
**  Przyklad sterowania AD9850/9851 przez Arduino za pomoca             **
**  zlacza szeregowego. Fala prostokatna i sinusoida wyswietlane na     **
**  LXARDOSCOPE bezplatnym programie oscyloskopu dla Arduino            **
**                                                                      **
**  Program przykladowy moze byc bezplatnie rozpowszechniany            **
**  na zasadach licencji GNU                                            **
**                                                                      **
**  Copyright (C) 2011 ElecFreaks Corp.                                 **
**                                                                      **
**  http://www.elecfreaks.com                                           **
*************************************************************************/

#include "EF_AD9850.h"


EF_AD9850::EF_AD9850(int D_CLK, int D_FQUP, int D_REST, int D_BitData)
{
  pinMode(D_REST, OUTPUT);                    // programowanie wyjsc Arduino
  pinMode(D_FQUP, OUTPUT);                    // sluzacych do sterowania syntezerem
  pinMode(D_CLK , OUTPUT);
  pinMode(D_BitData, OUTPUT);

  CLK = D_CLK;
  FQUP= D_FQUP;
  REST= D_REST;
  BitData = D_BitData;
}

void EF_AD9850::init(void)
{
  digitalWrite(REST, 0);                      // stan spoczynkowy na zlaczu
  digitalWrite(FQUP, 0);
  digitalWrite(CLK, 0);
  digitalWrite(BitData, 0);
}
```

```
void EF_AD9850::reset(void)
{
  digitalWrite(CLK, 0);                                  // zerowanie syntezera
  digitalWrite(FQUP, 0);
  //sygnal zerujacy
  digitalWrite(REST, 0);
  digitalWrite(REST, 1);
  digitalWrite(REST, 0);
  //sygnal zegarowy
  digitalWrite(CLK, 0);
  digitalWrite(CLK, 1);
  digitalWrite(CLK, 0);
  //sygnal Fq-up
  digitalWrite(FQUP, 0);
  digitalWrite(FQUP, 1);
  digitalWrite(FQUP, 0);
}

void EF_AD9850::wr_serial(unsigned char w0,double frequence)  // wpisanie danych do syntezera
{
  unsigned char i,w;
  long int y;
  double x;

  //obliczenie czestotliwosci
  x=4294967295/125;//dla kwarcu 125 MHz
  frequence=frequence/1000000;
  frequence=frequence*x;
  y=frequence;

  //slowo sterujace w4
  w=(y>>=0);
  for(i=0; i<8; i++)
  {
    digitalWrite(BitData, (w>>i)&0x01);
    digitalWrite(CLK, 1);
    digitalWrite(CLK, 0);
  }
  //slowo sterujace w3
  w=(y>>8);
  for(i=0; i<8; i++)
  {
    digitalWrite(BitData, (w>>i)&0x01);
    digitalWrite(CLK, 1);
    digitalWrite(CLK, 0);
  }
  //slowo sterujace w2
  w=(y>>16);
  for(i=0; i<8; i++)
  {
    digitalWrite(BitData, (w>>i)&0x01);
    digitalWrite(CLK, 1);
    digitalWrite(CLK, 0);
  }
```

```
  //slowo sterujace w1
  w=(y>>24);
  for(i=0; i<8; i++)
  {
    digitalWrite(BitData, (w>>i)&0x01);
    digitalWrite(CLK, 1);
    digitalWrite(CLK, 0);
  }
  //slowo sterujace w0
  w=w0;
  for(i=0; i<8; i++)
  {
    digitalWrite(BitData, (w>>i)&0x01);
    digitalWrite(CLK, 1);
    digitalWrite(CLK, 0);
  }
  digitalWrite(FQUP, 1);
  digitalWrite(FQUP, 0);
}

void EF_AD9850::wr_parrel(unsigned char w0,double frequence)
{

}
```

Plik keywords.txt

```
EF_AD9850    KEYWORD1

init         KEYWORD2
reset        KEYWORD2
wr_serial    KEYWORD2
wr_parrel    KEYWORD2
```

Sterowanie AD9851 przez złącze szeregowe wersja 2

```
// Program sluzy do sterowania syntezera AD9851 DDS a jego rozwiazanie jest oparte na pracach:
// Mike‘a Bowthorpe, http://www.ladyada.net/rant/2007/02/cotw-ltc6903/ i
// http://www.geocities.com/leon_heller/dds.html
// Autor Peter Marks http://marxy.org

#define DDS_CLOCK 180000000

byte LOAD = 8;
byte CLOCK = 9;
byte DATA = 10;
byte LED = 13;

void setup()
{
  pinMode (DATA, OUTPUT); // programowanie wyjsc Arduino
  pinMode (CLOCK, OUTPUT);
  pinMode (LOAD, OUTPUT); //
  pinMode (LED, OUTPUT);
}

void loop()
{
  // przemiatanie czestotliwosci w Hz
  for(unsigned long freq = 10000000; freq < 10001000; freq++)
  {
    sendFrequency(freq);
    delay(2);
  }
}

void sendFrequency(unsigned long frequency)       // wpisanie danych do syntezera
{
  unsigned long tuning_word = (frequency * pow(2, 32)) / DDS_CLOCK;
  digitalWrite (LOAD, LOW); // niski poziom na linii „load“

  for(int i = 0; i < 32; i++)
  {
    if ((tuning_word & 1) == 1)
      outOne();
    else
      outZero();
    tuning_word = tuning_word >> 1;
  }
  byte_out(0x09);

  digitalWrite (LOAD, HIGH); // ponownie wysoki poziom na linii „load“
}

void byte_out(unsigned char byte)    // wydanie bajtu danych
{
  int i;

  for (i = 0; i < 8; i++)
```

```
  {
   if ((byte & 1) == 1)
     outOne();
   else
     outZero();
   byte = byte >> 1;
  }
}

void outOne()                                   // wydanie jedynki
{
 digitalWrite(CLOCK, LOW);
 digitalWrite(DATA, HIGH);
 digitalWrite(CLOCK, HIGH);
 digitalWrite(DATA, LOW);
}

void outZero()                                  // wydanie zera
{
 digitalWrite(CLOCK, LOW);
 digitalWrite(DATA, LOW);
 digitalWrite(CLOCK, HIGH);
}
```

Sterowanie AD9851 przez złącze szeregowe wersja 3

```
/ Program lekko zmodyfikowany przez George‘a Smarta, M1GEO - 12 lutego 2012.
// http://www,george-smart.co.uk/
//
// Autor kodu wyjsciowego Peter Marks http://marxy.org
// i http://blog.marxy.org/2008/05/controlling-ad9851-dds-with-arduino.html

// Czestotliwosc zegarowa syntezera DDS w Hz (nalezy uwzglednic PLL).
#define DDS_REF 180000000

// Polaczenia syntezera z Arduino
#define DDS_LOAD  8
#define DDS_CLOCK 9
#define DDS_DATA  10
#define LED       13

void setup()
{
 pinMode (DDS_DATA,  OUTPUT);               // Programowanie wyjsc
 pinMode (DDS_CLOCK, OUTPUT);
 pinMode (DDS_LOAD,  OUTPUT);
 pinMode (LED,       OUTPUT);
}

void loop()
{
 frequency(50000000);
 digitalWrite (LED, HIGH);
 delay(250);
```

```
  frequency(51000000);
  digitalWrite (LED, LOW);
  delay(250);
}

void frequency(unsigned long frequency) {          // wpisanie czestotliwosci do syntezera
  unsigned long tuning_word = (frequency * pow(2, 32)) / DDS_REF;
  digitalWrite (DDS_LOAD, LOW); // poziom niski na linii „load|

  for(int i = 0; i < 32; i++) {
    if ((tuning_word & 1) == 1)
      outOne();
    else
      outZero();
    tuning_word = tuning_word >> 1;
  }
  byte_out(0x09);
  digitalWrite (DDS_LOAD, HIGH); // ponownie poziom wysoki na linii „load“
}

void byte_out(unsigned char byte) {  // wydanie bajtu danych
  int i;

  for (i = 0; i < 8; i++) {
    if ((byte & 1) == 1)
      outOne();
    else
      outZero();
    byte = byte >> 1;
  }
}

void outOne() {                                    // wydanie jedynki
  digitalWrite(DDS_CLOCK, LOW);
  digitalWrite(DDS_DATA, HIGH);
  digitalWrite(DDS_CLOCK, HIGH);
  digitalWrite(DDS_DATA, LOW);
}

void outZero() {                                    // wydanie zera
  digitalWrite(DDS_CLOCK, LOW);
  digitalWrite(DDS_DATA, LOW);
  digitalWrite(DDS_CLOCK, HIGH);
}
```

Radiolatarnia RTTY z syntezerem cyfrowym AD9851

Kod źródłowy

```
// Program nadawczy RTTY z wykorzystaniem syntezera AD9851
// George Smart, M1GEO.
//  http://www,george-smart.co.uk/wiki/Arduino_RTTY
//
// Kod AD9851
// autorstwa Petera Marksa
//  http://blog.marxy.org/2008/05/controlling-ad9851-dds-with-arduino.html
//
// Kod transmisji RTTY Baudota
// Tim Zaman
// http://www.timzaman.nl/?p=138&lang=en

// Czestotliwosc zegarowa syntezera DDS w Hz (nalezy uwzglednic PLL).
#define DDS_REF 180000000

// Czestotliwosc wyjsciowa RTTY (50 MHz)
#define RTTY_TXF  50.100e6

// Dewiacja czestotliwosci RTTY (170 Hz)
#define RTTY_OSET 170

// Polaczenia miedzy syntezerem i Arduino
#define DDS_LOAD  8
#define DDS_CLOCK 9
#define DDS_DATA  10
#define LED       13

#include <stdint.h>

#define ARRAY_LEN 32
#define LETTERS_SHIFT 31
#define FIGURES_SHIFT 27
#define LINEFEED 2
#define CARRRTN  8

#define is_lowercase(ch)    ((ch) >= 'a' && (ch) <= 'z')
#define is_uppercase(ch)    ((ch) >= 'A' && (ch) <= 'Z')

unsigned long time;
 // Tabele alfabetu Bodota – litery i cyfry
char letters_arr[33] = "\000E\nA SIU\rDRJNFCKTZLWHYPQOBG\000MXV\000";
char figures_arr[33] = "\0003\n- \a87\r$4',!:(5\")2#6019?&\000./;\000";

void setup()
{
  pinMode (DDS_DATA,  OUTPUT);                  // Programowanie wyjsc
  pinMode (DDS_CLOCK, OUTPUT);
  pinMode (DDS_LOAD,  OUTPUT);
  pinMode (LED,       OUTPUT);
  Serial.begin(9600);                           // Inicjalizacja zlacza szeregowego RS232
}
```

```
void loop()
{
  // wlaczenie nadajnika z wyprzedzeniem w stosunku do transmisji
  delay(100);
  frequency(RTTY_TXF);
  delay(1000);

  time=millis();
                                                    // przykladowe teksty nadawane. Zmienic na wlasne
  rtty_txstring("\r\n      \r\n");
  rtty_txstring("George Smart, M1GEO.\r\n");
  rtty_txstring("http://www.george-smart.co.uk/\r\n");
  rtty_txstring("Arduino RTTY Test!\r\n\r\n");
  rtty_txstring("\"An expert is a person who has made all the mistakes that can be made in a very narrow
field\"  Nils Bohr\r\n\r\n");
  rtty_txstring("\"Most of the important things in the world have been accomplished by people who have
kept on trying when there seemed to be no hope at all\"  Dale Carnegie\r\n\r\n");
  rtty_txstring("\"If others would think as hard as I did, then they would get similar results\"
Newton\r\n\r\n");
  rtty_txstring("\"If you can’t explain what you are doing to a nine-year-old, then either you still don’t
understand it very well, or it’s not all that worthwhile in the first place.\"  Albert Einstein\r\n\r\n");
  rtty_txstring("-----\r\n");

  time=millis()-time;
  Serial.println(time);

  digitalWrite (LED, HIGH);
  delay(100);
  digitalWrite (LED, LOW);
  delay(1000);

  frequency(0);
  delay(1000);
}

void frequency(unsigned long frequency) {                    // wpisanie czestotliwosci do syntezera
  unsigned long tuning_word = (frequency * pow(2, 32)) / DDS_REF;
  digitalWrite (DDS_LOAD, LOW); // niski poziom na linii „load“

  for(int i = 0; i < 32; i++) {
    if ((tuning_word & 1) == 1)
      outOne();
    else
      outZero();
    tuning_word = tuning_word >> 1;
  }
  byte_out(0x09);
  digitalWrite (DDS_LOAD, HIGH); // Ponownie wysoki poziom na linii „load“
}

void byte_out(unsigned char byte) {   // wydanie bajtu danych
  int i;

  for (i = 0; i < 8; i++) {
```

```
    if ((byte & 1) == 1)
      outOne();
    else
      outZero();
    byte = byte >> 1;
  }
}

void outOne() {                         // wydanie jedynki do syntezera
  digitalWrite(DDS_CLOCK, LOW);
  digitalWrite(DDS_DATA, HIGH);
  digitalWrite(DDS_CLOCK, HIGH);
  digitalWrite(DDS_DATA, LOW);
}

void outZero() {                        // wydanie zera do AD9851
  digitalWrite(DDS_CLOCK, LOW);
  digitalWrite(DDS_DATA, LOW);
  digitalWrite(DDS_CLOCK, HIGH);
}

uint8_t char_to_baudot(char c, char *array)   // przekodowanie na kod Baudota
{
  int i;
  for (i = 0; i < ARRAY_LEN; i++)
  {
    if (array[i] == c)
      return i;
  }

  return 0;
}

void rtty_txbyte(uint8_t b)
{
  int8_t i;

  rtty_txbit(0);

  /* dla odwrotnego porzadku transmisji bitow */
  /* for (i = 4; i >= 0; i--) */
  for (i = 0; i < 5; i++)
  {
    if (b & (1 << i))
      rtty_txbit(1);
    else
      rtty_txbit(0);
  }

  rtty_txbit(1);
}

enum baudot_mode                                // rejestry kodu
{
  NONE,
```

```
  LETTERS,
  FIGURES
};

void rtty_txstring(char *str)                    // nadanie tekstu alfabetem Bodota
{
  enum baudot_mode current_mode = NONE;
  char c;
  uint8_t b;

  while (*str != '\0')
  {
    c = *str;
    /* niektore znaki sa identyczne w zbiorach liter i cyfr */
    if (c == '\n')
    {
      rtty_txbyte(LINEFEED);                     // nowa linia
    }
    else if (c == '\r')
    {
      rtty_txbyte(CARRRTN);                      // powrot wozka
    }
    else if (is_lowercase(*str) || is_uppercase(*str))
    {
      if (is_lowercase(*str))
      {
        c -= 32;
      }

      if (current_mode != LETTERS)
      {
        rtty_txbyte(LETTERS_SHIFT);              // zmiana rejestru na litery
        current_mode = LETTERS;
      }

      rtty_txbyte(char_to_baudot(c, letters_arr));
    }
    else
    {
      b = char_to_baudot(c, figures_arr);

      if (b != 0 && current_mode != FIGURES)
      {
        rtty_txbyte(FIGURES_SHIFT);              // zmiana rejestru na cyfry
        current_mode = FIGURES;
      }

      rtty_txbyte(b);
    }

    str++;
  }
}
```

```
// nadanie bitu jako znak („mark”) lub odstep („space”)
void rtty_txbit (int bit) {
  if (bit) {
    // poziom wysoki – znak („mark“)
    //digitalWrite(2, HIGH);
    //digitalWrite(3, LOW);

    frequency(RTTY_TXF + RTTY_OSET);

  } else {
    // poziom niski – odstep („space“)
    //digitalWrite(3, HIGH);
    //digitalWrite(2, LOW);

    frequency(RTTY_TXF);

  }

  // Odstep czasu 22 ms dla szybkosci 45,45 boda.
  delay(22); //ustala szybkosc transmisji
  //delayMicroseconds(250);
}
```

Radiolatarnia QRSS z syntezerem cyfrowym AD9851

```
// Program nadawczy QRSS z wykorzystaniem AD9851
// George Smart, M1GEO.
//  http://www,george-smart.co.uk/wiki/Arduino_QRSS
//
// Kod AD9851
// autorstwa Petera Marksa
//  http://blog.marxy.org/2008/05/controlling-ad9851-dds-with-arduino.html
//

// Czestotliwosc zegarowa syntezera w Hz. (nalezy uwzglednic PLL).
#define DDS_REF   180000000

// Dewiacja czestotliwosci w Hz
#define DDS_OSET  404

// Czestotliwosc wyjsciowa QRSS 10,1400040 MHz
#define QRSS_TX  10.140040e6

// Dewiacja czestotliwosci QRSS
#define QRSS_OSET 5

// Zaleznosci czasowe QRSS w milisekundach
#define QRSS_DIT  6000   //5000
#define QRSS_DAH  (QRSS_DIT*3)  //18000

// Czas narastania sygnalu trojkatnego w milisekundach
#define TRI_TIME  (2*QRSS_DIT)

// Polaczenia syntezera z Arduino
#define DDS_LOAD  8
#define DDS_CLOCK 9
#define DDS_DATA  10
#define LED       13
#define QRSS_TXF  (QRSS_TX+DDS_OSET)
#include <stdint.h>

void setup()
{
  // Programowanie wyjsc
  pinMode (DDS_DATA,  OUTPUT);
  pinMode (DDS_CLOCK, OUTPUT);
  pinMode (DDS_LOAD,  OUTPUT);
  pinMode (LED,       OUTPUT);

  // Inicjalizacja zlacza szeregowego RS232
  Serial.begin(9600);

  // Wlaczenie nadajnika zawczasu przed rozpoczeciem transmisji
  delay(100);
  frequency(QRSS_TXF);
  waitDit();
}
```

```
void loop()
{
  Serial.println("*** Starting Cycle");

  waitDah();                          // odstep o dlugosci kreski

  Serial.print("Sending Morse: ");
  sendPattern("--/.----/--././---"); // Znak wywolawczy M1GEO nadany telegrafia. Zmienic na wlasny.
  Serial.println(" OK");

  Serial.print("Sending 3 Triangles: ");
  // Nadanie trzech zboczy narastajacych
  for(int i=0;i<3;i++) {
    Serial.print(i+1);
    digitalWrite(LED, HIGH);   // sygnalizacja swietlna
    sendUpRamp();
    digitalWrite(LED, LOW);    // sygnalizacja swietlna
    sendDownRamp();
    Serial.print(" ");
  }
  Serial.println("OK");

  waitDah();                    // odstepy o dlugosci 3 kresek
  waitDah();
  waitDah();
}

void frequency(unsigned long frequency) {                    // wydanie czestotliwosci do syntezera
  unsigned long tuning_word = (frequency * pow(2, 32)) / DDS_REF;
  digitalWrite (DDS_LOAD, LOW); // niski poziom na linii „load“

  for(int i = 0; i < 32; i++) {
    if ((tuning_word & 1) == 1)
      outOne();
    else
      outZero();
    tuning_word = tuning_word >> 1;
  }
  byte_out(0x09);
  digitalWrite (DDS_LOAD, HIGH); // Ponownie wysoki poziom na linii „load“
}

void byte_out(unsigned char byte) {   // wydanie bajtu
  int i;

  for (i = 0; i < 8; i++) {
    if ((byte & 1) == 1)
      outOne();
    else
      outZero();
    byte = byte >> 1;
  }
}
```

```
void outOne() {                                  // wydanie jedynki do syntezera
  digitalWrite(DDS_CLOCK, LOW);
  digitalWrite(DDS_DATA, HIGH);
  digitalWrite(DDS_CLOCK, HIGH);
  digitalWrite(DDS_DATA, LOW);
}

void outZero() {
  digitalWrite(DDS_CLOCK, LOW);                  // wydanie zera do syntezera
  digitalWrite(DDS_DATA, LOW);
  digitalWrite(DDS_CLOCK, HIGH);
}

void sendDit() {                                 // nadanie kropki telegrafia
  frequency(QRSS_TXF + QRSS_OSET);
  waitDit();
  frequency(QRSS_TXF);
  waitDit();  // zawsze odstep o dlugosci kropki
}

void sendDah() {                                 // nadanie kreski telegrafia
  frequency(QRSS_TXF + QRSS_OSET);
  waitDah();
  frequency(QRSS_TXF);
  waitDit();  // zawsze odstep o dlugosci kropki
}

void waitDit() {                                 // odstep o dlugosci kropki
  int a = 0;
  for (a=0;a<QRSS_DIT;a+=100) {
    digitalWrite(LED, HIGH);
    delay(25);
    digitalWrite(LED, LOW);
    delay(75);
  }
}

void waitDah() {                                 // odstep o dlugosci kreski
  int a = 0;
  for (a=0;a<QRSS_DAH;a+=100) {
    digitalWrite(LED, HIGH);
    delay(75);
    digitalWrite(LED, LOW);
    delay(25);
  }
}

void sendPattern(String str) {                   // wyswietlenie „komunikatu“
  int strLen = str.length();                     // w oknie monitora Arduino
  Serial.print(strLen);
  Serial.print(" elements: ");

  int i=0;
  for(i=0;i<strLen;i++) {
    switch (str.charAt(i)) {                     // wyswietlenie i nadanie dozwolonych znakow
```

```
      case '-':                                          // kreski
        Serial.print("-");
        sendDah();
        break;
      case '.':
        Serial.print(".");                               // kropki
        sendDit();
        break;
      case '/':
      case ' ':
        Serial.print("/");                               // kreski
        waitDah();
        break;
      default:                                           // niedozwolone
        Serial.print("(I didn't recognise '");
        Serial.print(str.charAt(i));
        Serial.print("'.  Permitted Chars are -  .  or  /)");
    }
  }
}

void sendDownRamp() {                                    // nadanie zbocza opadajacego
  for (int i=QRSS_OSET; i>=0;i--) {
    frequency(QRSS_TXF + i);
    delay(TRI_TIME/QRSS_OSET);
  }
}

void sendUpRamp() {                                      // nadanie zbocza narastajacego
  for (int i=0; i<=QRSS_OSET;i++) {
    frequency(QRSS_TXF + i);
    delay(TRI_TIME/QRSS_OSET);
  }
}
```

Radiolatarnia Hella z syntezerem cyfrowym AD9851

```
// Program nadawczy dalekopisowy w normie „Hellschreiber“ z uzyciem AD9851
// George Smart, M1GEO.
//  http://www,george-smart.co.uk/wiki/Arduino_Hellschreiber
//
// Kod AD9851
// autorstwa Petera Marksa
// http://blog.marxy.org/2008/05/controlling-ad9851-dds-with-arduino.html
//

// Czestotliwosc zegarowa dla syntezera DDS w Hz. (nalezy uwzglednic PLL).
#define DDS_REF   180000000
#define DDS_OSET  0  //DDS #2 offset

// Czestotliwosc wyjsciowa WSPR
#define HELL_TX    10.138000e6  // Dolna granica podzakresu. Stacje pracuja w jej poblizu.
#define HELL_TXF   HELL_TX+DDS_OSET  // Dolna granica podzakresu. Stacje pracuja w jej
poblizu.

// Polaczenia syntezera z Arduino
#define DDS_LOAD  8
#define DDS_CLOCK 9
#define DDS_DATA  10
#define LED       13

// Biblioteki
#include <stdint.h>

typedef struct glyph {
   char ch ;
   word col[7] ;
} Glyph ;                                   // struktura dla znakow i ich elementow

Glyph glyphtab[] PROGMEM = {
{' ', {0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'A', {0x07fc, 0x0e60, 0x0c60, 0x0e60, 0x07fc, 0x0000, 0x0000}},
{'B', {0x0c0c, 0x0ffc, 0x0ccc, 0x0ccc, 0x0738, 0x0000, 0x0000}},
{'C', {0x0ffc, 0x0c0c, 0x0c0c, 0x0c0c, 0x0c0c, 0x0000, 0x0000}},
{'D', {0x0c0c, 0x0ffc, 0x0c0c, 0x0c0c, 0x07f8, 0x0000, 0x0000}},
{'E', {0x0ffc, 0x0ccc, 0x0ccc, 0x0c0c, 0x0c0c, 0x0000, 0x0000}},
{'F', {0x0ffc, 0x0cc0, 0x0cc0, 0x0c00, 0x0c00, 0x0000, 0x0000}},
{'G', {0x0ffc, 0x0c0c, 0x0c0c, 0x0ccc, 0x0cfc, 0x0000, 0x0000}},
{'H', {0x0ffc, 0x00c0, 0x00c0, 0x00c0, 0x0ffc, 0x0000, 0x0000}},
{'I', {0x0ffc, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'J', {0x003c, 0x000c, 0x000c, 0x000c, 0x0ffc, 0x0000, 0x0000}},
{'K', {0x0ffc, 0x00c0, 0x00e0, 0x0330, 0x0e1c, 0x0000, 0x0000}},
{'L', {0x0ffc, 0x000c, 0x000c, 0x000c, 0x000c, 0x0000, 0x0000}},
{'M', {0x0ffc, 0x0600, 0x0300, 0x0600, 0x0ffc, 0x0000, 0x0000}},
{'N', {0x0ffc, 0x0700, 0x01c0, 0x0070, 0x0ffc, 0x0000, 0x0000}},
{'O', {0x0ffc, 0x0c0c, 0x0c0c, 0x0c0c, 0x0ffc, 0x0000, 0x0000}},
{'P', {0x0c0c, 0x0ffc, 0x0ccc, 0x0cc0, 0x0780, 0x0000, 0x0000}},
{'Q', {0x0ffc, 0x0c0c, 0x0c3c, 0x0ffc, 0x000f, 0x0000, 0x0000}},
{'R', {0x0ffc, 0x0cc0, 0x0cc0, 0x0cf0, 0x079c, 0x0000, 0x0000}},
{'S', {0x078c, 0x0ccc, 0x0ccc, 0x0ccc, 0x0c78, 0x0000, 0x0000}},
```

```
{'T', {0x0c00, 0x0c00, 0x0ffc, 0x0c00, 0x0c00, 0x0000, 0x0000}},
{'U', {0x0ff8, 0x000c, 0x000c, 0x000c, 0x0ff8, 0x0000, 0x0000}},
{'V', {0x0ffc, 0x0038, 0x00e0, 0x0380, 0x0e00, 0x0000, 0x0000}},
{'W', {0x0ff8, 0x000c, 0x00f8, 0x000c, 0x0ff8, 0x0000, 0x0000}},
{'X', {0x0e1c, 0x0330, 0x01e0, 0x0330, 0x0e1c, 0x0000, 0x0000}},
{'Y', {0x0e00, 0x0380, 0x00fc, 0x0380, 0x0e00, 0x0000, 0x0000}},
{'Z', {0x0c1c, 0x0c7c, 0x0ccc, 0x0f8c, 0x0e0c, 0x0000, 0x0000}},
{'0', {0x07f8, 0x0c0c, 0x0c0c, 0x0c0c, 0x07f8, 0x0000, 0x0000}},
{'1', {0x0300, 0x0600, 0x0ffc, 0x0000, 0x0000, 0x0000, 0x0000}},
{'2', {0x061c, 0x0c3c, 0x0ccc, 0x078c, 0x000c, 0x0000, 0x0000}},
{'3', {0x0006, 0x1806, 0x198c, 0x1f98, 0x00f0, 0x0000, 0x0000}},
{'4', {0x1fe0, 0x0060, 0x0060, 0x0ffc, 0x0060, 0x0000, 0x0000}},
{'5', {0x000c, 0x000c, 0x1f8c, 0x1998, 0x18f0, 0x0000, 0x0000}},
{'6', {0x07fc, 0x0c66, 0x18c6, 0x00c6, 0x007c, 0x0000, 0x0000}},
{'7', {0x181c, 0x1870, 0x19c0, 0x1f00, 0x1c00, 0x0000, 0x0000}},
{'8', {0x0f3c, 0x19e6, 0x18c6, 0x19e6, 0x0f3c, 0x0000, 0x0000}},
{'9', {0x0f80, 0x18c6, 0x18cc, 0x1818, 0x0ff0, 0x0000, 0x0000}},
{'*', {0x018c, 0x0198, 0x0ff0, 0x0198, 0x018c, 0x0000, 0x0000}},
{'.', {0x001c, 0x001c, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'?', {0x1800, 0x1800, 0x19ce, 0x1f00, 0x0000, 0x0000, 0x0000}},
{'!', {0x1f9c, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
{'(', {0x01e0, 0x0738, 0x1c0e, 0x0000, 0x0000, 0x0000, 0x0000}},
{')', {0x1c0e, 0x0738, 0x01e0, 0x0000, 0x0000, 0x0000, 0x0000}},
{'#', {0x0330, 0x0ffc, 0x0330, 0x0ffc, 0x0330, 0x0000, 0x0000}},
{'$', {0x078c, 0x0ccc, 0x1ffe, 0x0ccc, 0x0c78, 0x0000, 0x0000}},
{'/', {0x001c, 0x0070, 0x01c0, 0x0700, 0x1c00, 0x0000, 0x0000}},
} ;                                                 // tabela znakow z rozkladem na elementy

#define NGLYPHS        (sizeof(glyphtab)/sizeof(glyphtab[0]))

void setup()
{
  char sz[50];

  pinMode (DDS_DATA,  OUTPUT);                  // Programowanie wyjsc
  pinMode (DDS_CLOCK, OUTPUT);
  pinMode (DDS_LOAD,  OUTPUT);
  pinMode (LED,       OUTPUT);

  Serial.begin(9600);                                 // Inicjalizacja zlacza szeregowego RS232

  sprintf(sz, "\nM1GEO Compiled %s %s", __TIME__, __DATE__);
  Serial.println(sz);

  Serial.print("\n\nDDS Reset   ");
  delay(900);
  frequency(0);
  delay(100);
  Serial.println("OK");

  Serial.print("\n\nATU TUNE   ");
  frequency(HELL_TXF);                              // dostrojenie do czestotliwosci pracy
  delay(5000);
  frequency(0);
  delay(1000);
```

```
  Serial.println("OK");
}

void loop()
{                                                    // Tekst nadawany. Zmienic na wlasny
  encode("M1GEO / HELL TEST / LOCATOR JO01CN / POWER 500 MW / ") ;
}

void frequency(unsigned long frequency) {       // wydanie czestotliwosci do syntezera
  unsigned long tuning_word = (frequency * pow(2, 32)) / DDS_REF;
  digitalWrite (DDS_LOAD, LOW); // niski poziom na linii „load“

  for(int i = 0; i < 32; i++) {
    if ((tuning_word & 1) == 1)
      outOne();
    else
      outZero();
    tuning_word = tuning_word >> 1;
  }
  byte_out(0x09);
  digitalWrite (DDS_LOAD, HIGH); // ponownie wysoki poziom na linii „load“
}

void byte_out(unsigned char byte) {   // wydanie bajtu
  int i;

  for (i = 0; i < 8; i++) {
    if ((byte & 1) == 1)
      outOne();
    else
      outZero();
    byte = byte >> 1;
  }
}

void outOne() {                                 // wydanie jedynki do syntezera
  digitalWrite(DDS_CLOCK, LOW);
  digitalWrite(DDS_DATA, HIGH);
  digitalWrite(DDS_CLOCK, HIGH);
  digitalWrite(DDS_DATA, LOW);
}

void outZero() {                                // wydanie zera do syntezera
  digitalWrite(DDS_CLOCK, LOW);
  digitalWrite(DDS_DATA, LOW);
  digitalWrite(DDS_CLOCK, HIGH);
}

void
encodechar(int ch)
{
    int i, x, y, fch ;
    word fbits ;

    /* Poszukiwanie kontynuowane nawet po znalezieniu znaku
```

```
    * ale zapewnia to staly czas wykonywania tej czesci programu
    * i ulatwia zachowanie rownomiernej szybkosci transmisji
    */
   for (i=0; i<NGLYPHS; i++) {
      fch = pgm_read_byte(&glyphtab[i].ch) ;
      if (fch == ch) {
         for (x=0; x<7; x++) {
            fbits = pgm_read_word(&(glyphtab[i].col[x])) ;
            for (y=0; y<14; y++) {
               if (fbits & (1<<y)) {
                  digitalWrite(LED, HIGH);
                  frequency (HELL_TXF) ;
               } else {
                  digitalWrite(LED, LOW);
                  frequency (0) ;
               }
               //delayMicroseconds(4045L) ;
               delayMicroseconds(3362L) ;  // zbyt dlugie czasy powoduja nachylenie pisma w gore.
            }
         }
      }
   }
}

void encode(char *ch)
{
   while (*ch != '\0')
      encodechar(*ch++) ;
}
```

Radiolatarnia WSPR z syntezerem cyfrowym AD9851

```
// Program nadawczy WSPR z wykorzystaniem AD9851
// George Smart, M1GEO.
//  here: http://www,george-smart.co.uk/wiki/Arduino_WSPR
//
// Kod AD9851 Code
// autorstwa Petera Marksa
// http://blog.marxy.org/2008/05/controlling-ad9851-dds-with-arduino.html
//
// kod GPS z TinyGPS
// http://arduiniana.org/libraries/tinygps/
//

// Czestotliwosc zegarowa syntezera DDS w Hz (nalezy uwzglednic PLL).
#define DDS_REF   180000000

// Dewiacja czestotliwosci DDS w Hz
#define DDS_OSET  164  //DDS #2

// Czestotliwosc wyjsciowa WSPR
#define WSPR_TX_A    10.140100e6  // Dolna granica pasma. Stacje pracuja w jej poblizu.
#define WSPR_TX_B     7.040000e6  // Dolna granica pasma. Stacje pracuja w jej poblizu.

#define WSPR_DUTY  3 // transmisja co N odcinkow czasowych.

// Tekst: GB0SNB 20dbm (100mw). Zmienic na wlasny
// Zakodowana kolejnosc tonow WSPR (0-161) dla danego tekstu.
static byte WSPR_DATA_BUNKER[] =
{1,1,2,2,0,2,0,0,1,0,0,0,3,1,3,0,0,2,3,0,2,1,0,3,1,1,3,2,0,0,2,2,2,
           2,1,2,0,1,2,1,2,0,2,0,0,2,1,2,1,3,2,2,1,3,2,1,0,0,0,3,1,2,1,0,2,2,0,1,1,0,3,2,3,2,1,0,
           1,0,0,3,2,2,3,0,3,1,0,0,0,1,3,0,3,0,1,0,0,0,1,0,2,0,2,2,3,2,2,3,0,2,3,3,1,0,3,1,0,0,3,
           3,2,1,2,0,0,3,3,1,0,2,0,0,2,1,0,1,0,0,1,1,2,2,0,0,0,2,2,3,3,0,1,0,1,3,0,2,2,3,1,0,2,2}; // 162 bity

//Tekst: M1GEO 23dbm (200mw).  Zmienic na wlasny
static byte WSPR_DATA_HOME[] = {3,3,0,0,2,0,2,0,1,0,2,0,1,3,1,2,2,2,3,0,2,1,2,3,1,1,1,2,0,0,0,2,0,
         2,3,0,0,1,2,1,2,2,0,0,2,2,3,0,1,1,2,2,3,3,2,1,2,2,0,1,3,0,3,2,2,0,2,3,3,0,3,0,3,0,3,0,
         1,0,2,3,2,2,3,2,3,1,0,2,2,1,1,2,1,0,3,0,2,2,1,2,0,0,0,0,1,2,2,1,0,2,3,3,1,0,3,3,0,2,1,
         1,0,3,0,0,2,3,3,3,2,0,0,0,0,3,0,3,2,2,3,1,0,2,0,0,2,2,2,3,3,2,3,2,3,3,2,0,0,3,1,0,2,2}; // 162 bity

#define WSPR_DATA WSPR_DATA_HOME

// Polaczenia syntezera z Arduino
#define DDS_LOAD  8
#define DDS_CLOCK 9
#define DDS_DATA  10
#define LED       13
#define GPS_LED   12

// Naglowki dla bibliotek
#include <stdint.h>
#include <SoftwareSerial.h>
#include <TinyGPS.h>

// Obsluga GPS
```

```
TinyGPS gps;
SoftwareSerial nss(3, 4);  //GPS RX 3, TX 4

// Deklaracje zmiennych
unsigned long  WSPR_TXF = 0;
unsigned long startT = 0, stopT = 0;
int year;
byte month, day, hour, minute, second, hundredths, Nsatellites, ret, duty, band;
unsigned long fix_age, fail;
char sz[32];

void setup()
{
  pinMode (DDS_DATA,  OUTPUT);                    // Programowanie wyjsc
  pinMode (DDS_CLOCK, OUTPUT);
  pinMode (DDS_LOAD,  OUTPUT);
  pinMode (LED,       OUTPUT);
  pinMode (GPS_LED,   OUTPUT);

  Serial.begin(4800);                             // Inicjalizacja zlacza szeregowego RS232
  //nss.begin(4800);

  sprintf(sz, "\nM1GEO Compiled %s %s", __TIME__, __DATE__);
  Serial.println(sz);

  Serial.print("\n\nDDS Reset   ");
  delay(900);
  frequency(0);
  delay(100);
  Serial.println("OK");

  duty = 0;
}

void loop()
{
  fail++;
  ret = feedgps();

  if (fail == 60000) {
    digitalWrite (GPS_LED, LOW);
    Serial.println("GPS: No Data.");
  }

  if (ret>0) {
    Nsatellites = gps.satellites();               // liczba odbieranych satelitow
    gps.crack_datetime(&year, &month, &day, &hour, &minute, &second, &hundredths, &fix_age);
    if (fix_age == TinyGPS::GPS_INVALID_AGE) {    // badanie aktualnosci danych GPS
      Serial.println("GPS: No Fix.");
      digitalWrite (GPS_LED, LOW);
    } else {
      digitalWrite (GPS_LED, HIGH);
      sprintf(sz, "Date %02d/%02d/%02d, Time %02d:%02d:%02d (Sats: %02d, WSPR Duty: %d, GPS
Age: %lu ms, GPS Feed: %lu).", day, month, year, hour, minute, second, Nsatellites, (duty %
WSPR_DUTY), fix_age, fail);
```

```
    Serial.println(sz);

    if(band % 2 == 0) {
      WSPR_TXF = (WSPR_TX_A+DDS_OSET) + random(10, 190); // wybor czestotliwosci
modyfikowany przez pRNG.
    } else {
      WSPR_TXF = (WSPR_TX_B+DDS_OSET) + random(10, 190); // wybor czestotliwosci
modyfikowany przez pRNG.
    }

    if ( (minute % 2 == 0) && (second >= 1) && (second <= 4) ) {  // poczatek transmisji pomiedzy 1 a
4 sekunda po parzystej minucie
      if (duty % WSPR_DUTY == 0) {
        Serial.print("Beginning WSPR Transmission on ");
        Serial.print(WSPR_TXF-DDS_OSET);
        Serial.println(" Hz.");
        wsprTX();
        duty++;
        band++;
        Serial.println(" Transmission Finished.");
      } else {
        duty++;
        digitalWrite (LED, LOW);
        digitalWrite (GPS_LED, LOW);
        delay(5000); // opuszczenie okna aby rozpoczac nadawanie.
        flash_led(Nsatellites, GPS_LED); // wyswietlenie liczby odbieranych satelitow GPS – miganie
diody odpowiednia ilosc razy.
        flash_led(WSPR_DUTY, LED); // wspolczynnik czestotliwosci transmisji WSPR – miganie
diody odpowiednia ilosc razy.
      }
    }
    digitalWrite (LED, HIGH);
    delay(250);
    digitalWrite (LED, LOW);
    delay(250);
  }
  fail = 0;
 }
}

void frequency(unsigned long frequency) {                    // wydanie czestotliwosci do syntezera
 unsigned long tuning_word = (frequency * pow(2, 32)) / DDS_REF;
 digitalWrite (DDS_LOAD, LOW); // niski poziom na linii „load“

 for(int i = 0; i < 32; i++) {
  if ((tuning_word & 1) == 1)
    outOne();
  else
    outZero();
  tuning_word = tuning_word >> 1;
 }
 byte_out(0x09);
 digitalWrite (DDS_LOAD, HIGH); // Ponownie wysoki poziom na linii „load“
}
```

```
void byte_out(unsigned char byte) {                    // wydanie bajtu
  int i;

  for (i = 0; i < 8; i++) {
    if ((byte & 1) == 1)
      outOne();
    else
      outZero();
    byte = byte >> 1;
  }
}

void outOne() {                                        // wydanie jedynki do syntezera
  digitalWrite(DDS_CLOCK, LOW);
  digitalWrite(DDS_DATA, HIGH);
  digitalWrite(DDS_CLOCK, HIGH);
  digitalWrite(DDS_DATA, LOW);
}

void outZero() {                                       // wydanie zera do syntezera
  digitalWrite(DDS_CLOCK, LOW);
  digitalWrite(DDS_DATA, LOW);
  digitalWrite(DDS_CLOCK, HIGH);
}

void flash_led(unsigned int t, int l) {                // informacja wyswietlana za pomoca
  unsigned int i = 0;                                  // migotania diody swiecacej
  if (t > 25) {
    digitalWrite(l, HIGH);
    delay(2000);
    digitalWrite(l, LOW);
  } else {
    for (i=0;i<t;i++) {
      digitalWrite(l, HIGH);
      delay(250);
      digitalWrite(l, LOW);
      delay(250);
    }
  }
}

void wsprTXtone(int t) {                               // wybor tonu
  if ((t >= 0) && (t <= 3) ) {
    frequency((WSPR_TXF + (t * 2))); // powinny byc odstepy 1,4648 Hz a nie 2.
  } else {
    Serial.print("Tone #");
    Serial.print(t);
    Serial.println(" is not valid.  (0 <= t <= 3).");
  }
}

void wsprTX() {                                // nadanie komunikatu WSPR
  int i = 0;

  digitalWrite(LED, HIGH);
```

```
  for (i=0;i<162;i++) {
    wsprTXtone( WSPR_DATA[i] );
    delay(682);
  }
  frequency(0);
  digitalWrite(LED, LOW);
}

static bool feedgps()
{
  while (Serial.available()) {
    if (gps.encode(Serial.read())) {
      return true;
    }
  }
  return false;
}
```

Radiolatarnia PSK31 z syntezerem cyfrowym AD9835

```
//////////////////////////////////////////////////////////////
// AD9835_PSK31
//
// Prosty program radiolatarni PSK31 na 20 m nadajacy wspolrzedne GPS
// pracuje na ATMega328 i AD9835 DDS
//
// autor Thomas Krahn KT5TK / DL4MDW (2012)
//
// Program jest bezplatnie dostepny na zasadach licencji GNU
//

// Kod oparty o dokument http://www.sparkfun.com/products/9169:
//
// autorstwa Marka J. Blaira, NF6X
//
// i czesciowo o komentarz tjfreebo z 22 marca 2011
//

#include <SPI.h>
#include <math.h>
#include <avr/power.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <util/crc16.h>
#include "config.h"
#include "varicode.h"   // tabela kodu „varicode“
#include "gps.h"

volatile int wd_counter;    // zmienna ulotna
bool gpsIsOn;
bool newPositionStillUnknown;
int sentence_id = 1;

const unsigned int LUTsize = 1<<8; // Tabela probek. Jej dlugosc powinna wyrazac sie za pomoca
potegi 2 aby ulatwic obliczanie adresu modulo.
int8_t sintable[LUTsize] PROGMEM = { // wartosci przesuniete o +127
  127,130,133,136,139,143,146,149,152,155,158,161,164,167,170,173,
  176,179,182,184,187,190,193,195,198,200,203,205,208,210,213,215,
  217,219,221,224,226,228,229,231,233,235,236,238,239,241,242,244,
  245,246,247,248,249,250,251,251,252,253,253,254,254,254,254,254,
  255,254,254,254,254,254,253,253,252,251,251,250,249,248,247,246,
  245,244,242,241,239,238,236,235,233,231,229,228,226,224,221,219,
  217,215,213,210,208,205,203,200,198,195,193,190,187,184,182,179,
  176,173,170,167,164,161,158,155,152,149,146,143,139,136,133,130,
  127,124,121,118,115,111,108,105,102,99,96,93,90,87,84,81,
  78,75,72,70,67,64,61,59,56,54,51,49,46,44,41,39,
  37,35,33,30,28,26,25,23,21,19,18,16,15,13,12,10,
  9,8,7,6,5,4,3,3,2,1,1,0,0,0,0,0,
  0,0,0,0,0,0,1,1,2,3,3,4,5,6,7,8,
  9,10,12,13,15,16,18,19,21,23,25,26,28,30,33,35,
  37,39,41,44,46,49,51,54,56,59,61,64,67,70,72,75,
  78,81,84,87,90,93,96,99,102,105,108,111,115,118,121,124
};
```

```
String beacon_text;

const int timerPrescale=1<<9;

struct Psk31Encoder
{
  int8_t send_ok;
  int8_t doing_char;
  int8_t bit_wait;
  int8_t bit_send;
  int8_t mbit_send;
  int16_t b_count;
  int8_t last_bit;
  char v_byte;
  unsigned int idx;
  int8_t mbit_flag;
  int8_t OUT_BIT;
  unsigned long fracfreq;
} psk1;

struct oscillator
{
 uint32_t phase;
 int32_t phase_increment;
 int32_t frequency_increment;
 int16_t amplitude;
 int16_t amplitude_increment;
 uint32_t framecounter;
 uint32_t phase_counter;
} o1;

const int fractionalbits = 16; // 16-bitowy adres ulamkowy fazy
// obliczenie skoku fazy w zaleznosci od czestotliwosc
unsigned long phaseinc(float frequency_in_Hz)
{
  return LUTsize *(1l<<fractionalbits)* frequency_in_Hz/(F_CPU/timerPrescale);
}

// Powyzsze obliczenia wymagaja arytmetyki zmiennoprzecinkowej i zapewniaja
// wystarczajaca dokladnosc w szerokim zakresie wartosci parametrow
// Dla dobranego zakresu parametrow mozliwe jest wykonywanie obliczen
// znacznie szybciej w oparciu o arytmetyke liczb calkowitych (zmiennych typu int).
// obliczenia sa sprawdzane tak aby uniknac dzielenia przez zero i utraty rozdzielczosci
//
 // uzywane przygotowane wyniki dzielenia tak, ze (pow(2,predivide) dzieli F_CPU, a wiec obliczenia
pasuja do 4 MHz (1,7 V), 8 MHz, 12 MHz (3,3 V) i 16 MHz lub 20 MHz
 // i nalezy zwrocic uwage aby czestotliwosc w Hz nie byla za duza. Praktycznie mozna korzystac z
pasma ok. 16 kHz przy wykorzystaniu licznikow Arduino
const int predivide = 8;
unsigned long phaseinc_from_fractional_frequency(unsigned long frequency_in_Hz_times_256)
{
   return (1l<<(fractionalbits-predivide))*
((LUTsize*(timerPrescale/(1<<predivide))*frequency_in_Hz_times_256)/(F_CPU/(1<<predivide)));
}
```

```
// Czestotliwosc zegarowa syntezera DDS (Hz)
#define FCLK 50000000

// Kontrola obliczen fazy i czestotliwosci
// #define DEBUG_CALC

// Wydawanie bezposrednio na magistrale SPI za pomoca funkcji SPIwrite() zamiast za pomoca
digitalWrite()?
#define FAST_IO

// Stale dla programowania licznikow

#if defined(__AVR_ATmega8__)

// Na plytkach starszego typu z ATmega8 wyjscie na linii 11
#define PWM_PIN       11
#define PWM_VALUE_DESTINATION     OCR2
#define PWM_INTERRUPT TIMER2_OVF_vect
#elif defined(__AVR_ATmega1280__)

#define PWM_PIN       3
#define PWM_VALUE_DESTINATION     OCR3C
#define PWM_INTERRUPT TIMER3_OVF_vect
#else

// Na plytkach nowszego typu z ATmega168 wyjscie na linii 3
#define PWM_PIN       3
#define PWM_VALUE_DESTINATION     OCR2B
#define PWM_INTERRUPT TIMER2_OVF_vect
#endif

// Zapis w rejestrze AD9835 przez magistrale SPI
void SPIwrite(int byte1, int byte2) {
   // niski poziom na linii FSYNC dla wyboru obwodu:
#ifdef FAST_IO
   PORTB &= ~0x02;
#else
   digitalWrite(FSYNCpin, LOW);
#endif

   //  nadanie adresu i wartosci przez magistrale SPI:
   SPI.transfer(byte1);
   SPI.transfer(byte2);

   // wysoki poziom na linii FSYNC nie-adresowanie obwodu syntezera:
#ifdef FAST_IO
   PORTB |= 0x02;
#else
   digitalWrite(FSYNCpin, LOW);
#endif
}

// Obliczenie slowa sterujacego syntezer dla 32-bitowego rejestru fazy
// czestotliwosc wyjsciowa i zegarowa syntezera podane w Hz
unsigned long calcFTW32(unsigned long freq, unsigned long fclk) {
```

```
   unsigned long FTW;
   double FTWf;

   // Pogorszenie rozdzielczości ponieważ na Arduino obliczenia dla zmiennych
   // typu double przeprowadzane sa w rzeczywistosci na zmiennych typu float
   // a wiec na przyklad
   // calcFTW32(10000000,50000000) powinno dac wynik  0x33333333, ale zamiast tego
   // otrzymywany jest na Arduino wynik 0x33333340.
   // w przyszlosci nalezaloby skorygowac obliczenia tak aby zapewnic pelna dokladnosc
   // 32-bitowa dla pozadanego kroku np. 1 Hz
   FTWf = pow(2,32) * (double)freq / (double)fclk;
   FTW = (unsigned long) FTWf;

#ifdef DEBUG_CALC
   Serial.print("calcFTW32(");
   Serial.print(freq);
   Serial.print(", ");
   Serial.print(fclk);
   Serial.print(") = 0x");
   Serial.println(FTW, HEX);
#endif

   return FTW;
}

// Obliczenie skoku fazy dla syntezera przy ożyciu 12-bitowego rejestru skoku fazy
// Pozadana faza podawana jest w stopniach (0-359)
unsigned int calcPTW12d(unsigned int deg) {

   unsigned int PTW;
   double PTWf;

   PTWf = ((double)(deg % 360) / 360.0) * pow(2,12);
   PTW = (unsigned int) PTWf;

#ifdef DEBUG_CALC
   Serial.print("calcPTW12d(");
   Serial.print(deg);
   Serial.print(") = 0x");
   Serial.println(PTW, HEX);
#endif

   return PTW;
}

// zapis w wybranym rejestrze czestotliwosci (0-1) AD9835
void writeFTW(unsigned long FTW, int reg) {
   int regaddr;
   regaddr = (reg & 0x01) << 2;

   SPIwrite(0x33 + regaddr, ((FTW & 0xFF000000) >> 24));
   SPIwrite(0x22 + regaddr, ((FTW & 0x00FF0000) >> 16));
   SPIwrite(0x31 + regaddr, ((FTW & 0x0000FF00) >> 8));
   SPIwrite(0x20 + regaddr, ((FTW & 0x000000FF)));
```

```
}

// Zapis w wybranym rejestrze fazy (0-3) AD9835
void writePTW(unsigned int PTW, int reg) {
   int regaddr;
   regaddr = (reg & 0x03) << 1;

   SPIwrite(0x19 + regaddr, ((PTW & 0x0F00) >> 8));
   SPIwrite(0x08 + regaddr, ((PTW & 0x00FF)));
}

// funkcja ponizsza wspolpracuje z podprogramem przerwania (ISR) i funkcja send_varichar() i
przelacza stan bitu sygnalizujacego transmisje danych
void bitsend(void)
{
  if(psk1.send_ok) {                    // sprawdzenie czy czeka bit danych do nadania
    //Serial.println("send_ok");
    if(!psk1.bit_send)  {               // tak – czy to jest zero?
      //Serial.println("bit_send");
      psk1.OUT_BIT = !psk1.OUT_BIT;     // zmiana stanu bitu wyjsciowego (oznacza zero)
      if (psk1.OUT_BIT){
        //Serial.println("OUT_BIT HIGH");
        //digitalWrite(PskOutPin, HIGH);
        digitalWrite(PSEL1pin, HIGH);
        //Serial.println("OUT_BIT written");
      }
      else {
        //Serial.println("OUT_BIT LOW");
        //digitalWrite(PskOutPin, LOW);
        digitalWrite(PSEL1pin, LOW);
        //Serial.println("OUT_BIT written");
      }
    }
    psk1.send_ok = 0;                   // wyzerowanie sygnalizacji nadawania
    //Serial.println("send_ok reset");
    psk1.bit_wait = 1;                  // oznacza zakonczenie transmisji bitu...
    //Serial.print("bit_wait = ");
    //Serial.println(psk1.bit_wait, DEC);
  }
}

// ponizsza funkcja przesuwa dane i wydaje kolejny bit w 'bit_send'
void sendbit(void)
{
  psk1.bit_wait = 0;                    // zerowanie wskaznika bitu
  psk1.last_bit = psk1.bit_send;        // pobranie poprzednego bitu
  psk1.bit_send = bitRead(psk1.v_byte, 7);      // pobranie biezacego bitu do nadania
  //Serial.print("bit_send = ");
  //Serial.print(psk1.bit_send, DEC);
  //Serial.print("; ");
  if(!psk1.mbit_flag) {                 // kopiowanie biezacego bitu o ile nie zostalo juz dokonane...
    psk1.mbit_flag = 1;
    psk1.mbit_send = psk1.bit_send;     // rzeczywiste kopiowanie bitu...
  }
  psk1.b_count++;                       // zliczanie bitow
```

```
  //Serial.print("b_count = ");
  //Serial.println(psk1.b_count, DEC);
  psk1.v_byte <<= 1;                    // przesuniecie bajtu „varicode” w lewo o jedna pozycje
  if((!psk1.last_bit) && (!psk1.bit_send))  {  //sprawdzenie czy poprzednie dwa bity mialy wartosc zero
    //Serial.println("00");
    psk1.doing_char = 0;                // sygnalizacja, ze to ostatni bit znaku
  }
  if(psk1.b_count > 7)   {              // jezeli tak, pobranie nastepnego znaku...
    psk1.idx++;                         // pobranie nastepnego bitu
    psk1.v_byte = varicode[psk1.idx];         // wczytanie pozostalych bitow „varicode“
    //Serial.print("2nd byte = ");
    //Serial.println(psk1.v_byte, BIN);
    psk1.b_count = 0;                   // zerowanie licznika bitow
  }
  psk1.mbit_flag = 0;                   // zerowanie sygnalizatora jako przygotowanie do nastepnego bitu
}

void initializeTimer() {
 // Czestotliwosc zegarowa PWM  Clock/256 (np.. 31,25 kHz dla Arduino 16 MHz;
 // i tryb dokladnej fazy

#if defined(__AVR_ATmega8__)
  // ATmega8 rozni sie rejestrami
  TCCR2 = _BV(WGM20) | _BV(COM21) | _BV(CS20);
  TIMSK = _BV(TOIE2);
#elif defined(__AVR_ATmega1280__)
  TCCR3A = _BV(COM3C1) | _BV(WGM30);
  TCCR3B = _BV(CS30);
  TIMSK3 = _BV(TOIE3);
#else
  TCCR2A = _BV(COM2B1) | _BV(WGM20);
  TCCR2B = _BV(CS20);
  TIMSK2 = _BV(TOIE2);
#endif
//  pinMode(PWM_PIN,OUTPUT);
}

void lookup_varichar(char c)  // przekodowanie znaku (c) na „vericode“
{
  if(c > 127)    {              // jezeli znak spoza dopuszczalnego zakresu
    psk1.doing_char = 0;        // zerowanie sygnalizatora aby umozliwic przejscie do nastepnego znaku
    return;                     // zignorowanie znaku
  }
  else {                        // w dozwolonym zakresie – poszukiwanie w tabeli kodu
    psk1.idx = c;
    psk1.idx += c;              // podwojenie indeksu
    psk1.v_byte = varicode[psk1.idx];  // pobranie kodu „varicode“
    psk1.doing_char = 1;        // sygnalizacja, ze dane gotowe do nadania
    psk1.b_count = 0;           // inicjalizacja licznika bitow
    psk1.bit_send = 1;          // inicjalizacja bufora biezacego bitu
    psk1.last_bit = 1;          // inicjalizacja bufora ostatniego nadanego bitu
  }
}

void send_message()
```

```
{
  Serial.print(beacon_text);
  o1.phase_increment = phaseinc_from_fractional_frequency(psk1.fracfreq);  // PSK – czestotliwosc
31,.25 Hz * 256 = 8000

  //psk1.fracfreq += 1; // zwiekszenie czestotliwosci ulamkowej o 1 Hz w kazdym cyklu do celow
diagnostycznych

  char c = 32; // znak odstepu. Bez znaczenia gdy  > 0.
  int cnt = 0; // Wskaznik biezacej litery w tekscie

  for(cnt = 0; cnt < 40; cnt++) {      // nadanie zer na poczatku transmisji tekstu
    psk1.bit_send = 0;
    psk1.mbit_send = 0;
    psk1.bit_wait = 0;
    while(!psk1.bit_wait)  {
      //Serial.println("calling bitsend()");
      bitsend();
      //Serial.println("bitsend() done");
      //Serial.print("bit_wait = ");
      //Serial.println(psk1.bit_wait, DEC);
    }
  }

  cnt = 0;

  while (c > 0) // zakonczenie gdy tekst zakonczony zerem
  {
    c = beacon_text.charAt(cnt);
    //if((c > 31) && (c < 127)) { Serial.print(c);} // wyswietlenie znakow alfabetu w oknie monitora
    cnt++;
    lookup_varichar(c);
    while(psk1.doing_char) {              // petla do czasu zakonczenia znaku
      sendbit();
      while(!psk1.bit_wait)  {
        bitsend();                      // petla do czasu zakonczenia nadawania biezacego bitu
      }
    }
  }
}

uint16_t gps_CRC16_checksum (char *string)              // obliczanie sumy kontrolnej
{
  size_t i;
  uint16_t crc;
  uint8_t c;

  crc = 0xFFFF;

  // Obliczanie sumy kontrolnej z opuszczeniem znaku \n i dwoch znakow $
  for (i = 3; i < strlen(string); i++)
  {
    c = string[i];
    crc = _crc_xmodem_update (crc, c);
  }
```

```
  return crc;
}

void print_string_in_hex(char *string)
{
  size_t i;
  uint16_t crc;
  uint8_t c;

        Serial.print("hex: ");
  for (i = 0; i < (1 + strlen(string)); i++)
  {
    c = string[i];
    Serial.print(c, HEX);
              Serial.print(" ");
  }
        Serial.println();
}

//void disable_bod_and_sleep()
//{
  /* Wylaczanie wykrywania stanu „brown-out“ (BOD) w czasie uspienia
   * Nie funkcjonuje to jednak w stanie jalowym.
   * Szczegolowe informacje na ten temat w katalogu na str. 44
   *
   * Procedura sluzaca do wylaczenia BOD:
   *
   * 1. BODSE i BODS musza miec stan 1
   * 2. zerowanie BODSE
   * 3. BODS automatycznie wraca na zero po 4 cyklach
   *
   * Procesor *musi* przejsc w stan uspienia pomiedzy 2 i 3 cyklem
   * czyli zanim poziom BODS wroci na 0.
   */
//  unsigned char mcucr;

//  cli();
//  mcucr = MCUCR | (_BV(BODS) | _BV(BODSE));
//  MCUCR = mcucr;
//  MCUCR = mcucr & (~_BV(BODSE));
//  sei();
//  sleep_mode();    // Go to sleep
//}

void power_save()
{
  /* Wejscie w tryb oszczednosci energii. Tryb SLEEP_MODE_IDLE daje najmniejsza oszczednosc
   * ale jest jedynym, w ktorym pracuje UART.
   * Dodatkowo potrzebny jest licznik 0 (timer0) dla sledzenia uplywu czasu, licznik 1 (timer 1)
   * do sterowania brzeczyka i licznik 2 (timer2) aby utrzymac na wyjsciu pwm napiecie
   * spoczynkowe. TK: zmieniony na warunkowy tryb SLEEP_MODE_PWR_DOWN
   */
//   if (! modem_busy()) {  // W czasie nadawania niemozliwe przejscie w stan uspienia.
```

```
   if (newPositionStillUnknown == true)
   {

     // Zlacze szeregowe niezbedne dla wczytania
     // danych GPS NMEA
     set_sleep_mode(SLEEP_MODE_IDLE);
     sleep_enable();
     power_adc_disable();
     power_spi_disable();
     power_twi_disable();
     power_timer1_disable();
     digitalWrite(LED_PIN, LOW);
     sleep_mode();    // przejscie w stan uspienia
   }
   else
   {
     //Serial.end();
     // po wczytaniu wspolrzednych zlacze szeregowe
     // nie jest potrzebne
     // mozliwe jest przejscie do pelnego trybu SLEEP_MODE_PWR_DOWN.
     // Wyjscie z niego mozliwe tylko przez przerwanie ukladu nadzorczego (watchdog).
     set_sleep_mode(SLEEP_MODE_PWR_DOWN);

     // Wylaczenie ukladu GPS

     digitalWrite(GPS_POWER_PIN, HIGH);
     delay(500);   // Odczekanie na rozladowanie kondensatora blokujacego na plytce GPS
     gpsIsOn = false;
     sleep_enable();
     power_adc_disable();
     power_spi_disable();
     power_twi_disable();
     power_timer1_disable();

     digitalWrite(LED_PIN, LOW);
     //disable_bod_and_sleep();    // przejscie w stan uspienia
     sleep_mode();    // przejscie w stan uspienia
   }

   digitalWrite(LED_PIN, HIGH);

   sleep_disable();  // wznowienie pracy po obudzeniu
   power_all_enable();
//  }
}

//**************************************************************
// 0=16 ms, 1=32 ms,2=64 ms,3=128 ms,4=250 ms,5=500 ms
// 6=1 sec,7=2 sec, 8=4 sec, 9= 8 sec
void setup_watchdog(int ii)
{
  byte bb;
  int ww;
  if (ii > 9 ) ii=9;
  bb=ii & 7;
```

```
  if (ii > 7) bb|= (1<<5);
  bb|= (1<<WDCE);
  ww=bb;
//  Serial.println(ww);
  MCUSR &= ~(1<<WDRF);
  // start timed sequence
  WDTCSR |= (1<<WDCE) | (1<<WDE);
  // set new watchdog timeout value
  WDTCSR = bb;
  WDTCSR |= _BV(WDIE);
}

//*******************************************************************
// Program przerwania ukladu nadzorczego (watchdog) wywolywany po uplywie jego czasu
ISR(WDT_vect)
{
  wd_counter++; // zlicza sekundy czasu uspienia
}

float meters_to_feet(float m)                  // przeliczanie metrow na stopy
{
  // 10000 ft = 3048 m
  return m / 0.3048;
}

void blink3() // diagnoza
{
  digitalWrite(LED_PIN, LOW);
  delay(100);
  digitalWrite(LED_PIN, HIGH);
  delay(100);
  digitalWrite(LED_PIN, LOW);
  delay(100);
  digitalWrite(LED_PIN, HIGH);
  delay(100);
  digitalWrite(LED_PIN, LOW);
  delay(100);
  digitalWrite(LED_PIN, HIGH);
  delay(100);
}

// Podprogram inicjalizacji
void setup() {
   Serial.begin(BAUD);

#ifdef DEBUG_CALC
   Serial.println("reset");
#endif

   digitalWrite(FSYNCpin, HIGH);    // stany poczatkowe na zlaczu syntezera
   digitalWrite(SCLKpin,  LOW);
   digitalWrite(FSELpin,  LOW);
   digitalWrite(PSEL0pin, LOW);
   digitalWrite(PSEL1pin, LOW);
   pinMode(FSYNCpin, OUTPUT);    // programowanie wyjsc
```

```
  pinMode(SCLKpin,  OUTPUT);
  pinMode(FSELpin,  OUTPUT);
  pinMode(PSEL0pin, OUTPUT);
  pinMode(PSEL1pin, OUTPUT);
  pinMode(LED_PIN, OUTPUT);

  SPI.setDataMode(SPI_MODE1);     // inicjalizacja zlacza SPI
  SPI.setBitOrder(MSBFIRST);
  SPI.setClockDivider(SPI_CLOCK_DIV2);
  SPI.begin();

  // Zerowanie i przejscie do stanu wyjsciowego
  SPIwrite(0xF8, 0x00);
  delay(1);

  // wlaczenie napiecia
  SPIwrite(0xC0, 0x00);

  // Set sync and FSEL source (external pins)
  SPIwrite(0x80, 0x00);

  psk1.send_ok = 0;        // zawartosc struktury psk1
  psk1.doing_char = 0;
  psk1.bit_wait = 0;
  psk1.bit_send = 0;
  psk1.mbit_send = 0;
  psk1.b_count = 0;
  psk1.last_bit = 0;
  psk1.OUT_BIT = 0;
  psk1.fracfreq = 8000UL;

  initializeTimer();
  gps_setup();
  gpsIsOn = true;
  newPositionStillUnknown = true;
  setup_watchdog(9); // czas dla ukladu nadzorczego (watchdog) okolo 8 sekund
  wd_counter = (int)(TRANSMIT_PERIOD_SECONDS / 8); // Czas transmisji po wlaczeniu.

  // inicjalizacja rejestrow czestotliwosci dla przelaczania
  // miedzy 5 MHz  i 14,071 MHz za pomoca sygnalu FSEL
  writeFTW(calcFTW32(    1000, 50000000), 0);
  writeFTW(calcFTW32(14070500, 50000000), 1);

  // inicjalizacja rejestrow fazy dla QPSK za pomoca sygnalow PSEL1, PSEL0
  writePTW(calcPTW12d(0),   0);
  writePTW(calcPTW12d(90),  1);
  writePTW(calcPTW12d(180), 2);
  writePTW(calcPTW12d(270), 3);

  o1.amplitude = 255*256; // pelna amplituda

  blink3();
}

// Glowna petla
```

```
void loop()
{
  int c;
  char temp[120];
  if (wd_counter >= (int)(TRANSMIT_PERIOD_SECONDS / 8))
  {
    beacon_text = "\n$$KT5TK,";
    beacon_text = beacon_text + sentence_id + ",";

    if (newPositionStillUnknown)
    {
      beacon_text = beacon_text + "000000,0.0,0.0,0,No GPS lease... ";
    }
    else
    {
      // wspolrzedne GPS
      beacon_text = beacon_text + gps_time + ",";

      dtostrf(gps_lat,5,4,temp);
      beacon_text = beacon_text + temp + ",";

      dtostrf(gps_lon,5,4,temp);
      beacon_text = beacon_text + temp + ",";

      dtostrf(gps_altitude,1,0,temp);
      beacon_text = beacon_text + temp + ",";

//      sprintf(temp, "%ld", (long)((gps_altitude) + 0.5));
//      beacon_text = beacon_text + temp;
    }

    if (sentence_id % 3 == 0)
    {
      beacon_text = beacon_text + "high altitude balloon";
    }

    if (sentence_id % 3 == 1)
    {
      beacon_text = beacon_text + "http://kt5tk.tkrahn.com";
    }

    if (sentence_id % 3 == 2)
    {
      beacon_text = beacon_text + "solar powered";
    }

    // dodanie sumy kontrolnej CRC na koncu
    beacon_text.toCharArray( temp, ( 1 + beacon_text.length() ) );
    print_string_in_hex( temp );

    sprintf(temp,"*%04X\n", gps_CRC16_checksum( temp ));
    beacon_text = beacon_text + temp;

    digitalWrite(FSELpin,  HIGH); // Ustawienie czestotliwosci nadawania (w.cz.)
    send_message();
```

```
   digitalWrite(FSELpin,  LOW);  // ustawienie czestotliwosci 1 kHz gdy nadajnik wylaczony
   wd_counter = 0;
   newPositionStillUnknown = true;
   sentence_id++;
  }

  if (Serial.available() && newPositionStillUnknown)
  {
   c = Serial.read();
   if (gps_decode(c))
   {
    // odbior i zdekodowanie polozenia
    newPositionStillUnknown = false;

    blink3();  // sygnalizacja prawidlowego komunikatu GPS
   }
  }
  else
  {
   power_save();
  }
}

// Puls syntezy PSK31.
SIGNAL(PWM_INTERRUPT)
{
  if(o1.phase_counter >= 1000)
  {
   psk1.send_ok = 1;
   o1.phase_counter = 0;
  }
  o1.phase_counter++;
  //o1.phase += (uint32_t)o1.phase_increment;
}
```

Plik config.h

```
////////////////////////////////////////////////////////////////
// AD9835_PSK31
//
// Prosty program radiolatarni PSK31 na 20 m nadajacy wspolrzedne GPS
// pracuje na ATMega328 i AD9835 DDS
//
// autor Thomas Krahn KT5TK / DL4MDW (2012)
//
// Program jest bezplatnie dostepny na zasadach licencji GNU
//

#ifndef __CONFIG_H__
#define __CONFIG_H__

 #define BAUD 4800                  // szybkosc transmisji na zlaczu GPS
 #define TRANSMIT_PERIOD_SECONDS 20     // Czestotliwosc transmisji?

 // Programowanie wejsc i wyjsc
```

```
#define LED_PIN 3
#define GPS_POWER_PIN 4
#define GPS_RESET_PIN 5

#define SCLKpin  13   // SCK
// 12 = MISO
#define SDATApin 11   // MOSI
// 10 = SS
#define FSYNCpin 9    // PORTB.1
#define FSELpin  8    // PORTB.0
#define PSEL1pin 7    // PORTD.7
#define PSEL0pin 6    // PORTD.6
```

Biblioteka GPS
Plik gps.cpp

```
//////////////////////////////////////////////////////////////
// AD9835_PSK31
//
// Prosty program radiolatarni PSK31 na 20 m nadajacy wspolrzedne GPS
// pracuje na ATMega328 i AD9835 DDS
//
// autor Thomas Krahn KT5TK / DL4MDW (2012)
//
// Program jest bezplatnie dostepny na zasadach licencji GNU
//

/*
 * Kod pochodzi z trackuino - copyright (C) 2010  EA5HAV Javi
 * Zmodyfikowany w 2012 dla wlaczenia trybu  uBlox przez VK5QI
 * Zmodyfikowany w 2012 przez KT5TK do uzytku w radiolatarni PSK31
*/

#define DEBUG_GPS

#include "config.h"
#include <Arduino.h>
#include <stdlib.h>
#include <string.h>
#include "gps.h"

// Prototypy funkcji
static void parse_sentence_type(const char * token);
static void parse_time(const char *token);
static void parse_status(const char *token);
static void parse_lat(const char *token);
static void parse_lat_hemi(const char *token);
static void parse_lon(const char *token);
static void parse_lon_hemi(const char *token);
static void parse_speed(const char *token);
static void parse_course(const char *token);
static void parse_altitude(const char *token);
static void parse_satellites(const char *token);
```

```
// definicje typow
typedef void (*t_nmea_parser)(const char *token);

enum t_sentence_type {
  SENTENCE_UNK,
  SENTENCE_GGA,
  SENTENCE_RMC
};

// Definicje stalych
static const t_nmea_parser unk_parsers[] = {
  parse_sentence_type,    // $GPxxx
};

static const t_nmea_parser gga_parsers[] = {
  NULL,            // $GPGGA
  parse_time,       // Czas
  NULL,            // szerokosc geograficzna
  NULL,            // N/S
  NULL,            // dlugosc geograficzna
  NULL,            // E/W
  NULL,            // jakosc danych
  parse_satellites, // liczba satelitow
  NULL,            // pozioma niedokladnosc pozycji
  parse_altitude,   // wysokosc
  NULL,            // "M" (oznacza poziom morza)
  NULL,            // wysokosc GEOID (MSL) ponad elipsa WGS84
  NULL,            // "M" (oznacza poziom morza)
  NULL,            // Czas od ostatniego odczytu DGPS
  NULL             // numer identyfikacyjny stacji DGPS
};

static const t_nmea_parser rmc_parsers[] = {
  NULL,            // $GPRMC
  parse_time,       // czas
  parse_status,     // A=czynny, V=nieczynny
  parse_lat,       // szerokosc geograficzne,
  parse_lat_hemi,   // N/S
  parse_lon,       // dlugosc geograficzna
  parse_lon_hemi,   // E/W
  parse_speed,      // szybkosc ruchu wzgledem ziemi w wezlach
  parse_course,     // kurs w stopniach
  NULL,            // data (DDMMYY)
  NULL,            // Zmiennosc magnetyczna
  NULL             // E/W
};

static const int NUM_OF_UNK_PARSERS = (sizeof(unk_parsers) / sizeof(t_nmea_parser));
static const int NUM_OF_GGA_PARSERS = (sizeof(gga_parsers) / sizeof(t_nmea_parser));
static const int NUM_OF_RMC_PARSERS = (sizeof(rmc_parsers) / sizeof(t_nmea_parser));

// Deklaracje zmiennych
static t_sentence_type sentence_type = SENTENCE_UNK;
static bool at_checksum = false;
static unsigned char our_checksum = '$';
```

```
static unsigned char their_checksum = 0;
static char token[16];
static int num_tokens = 0;
static unsigned int offset = 0;
static bool fix_ok = false;
static char gga_time[7], rmc_time[7];
static char new_time[7];
static float new_lat;
static float new_lon;
static char new_aprs_lat[9];
static char new_aprs_lon[10];
static float new_course;
static float new_speed;
static float new_altitude;
static char new_satellites = 0;

// deklaracje zmiennych publicznych dostepnych z innych modulow
char gps_time[7];       // GGMMSS
float gps_lat = 0;
float gps_lon = 0;
char gps_aprs_lat[9];
char gps_aprs_lon[10];
float gps_course = 0;
float gps_speed = 0;
float gps_altitude = 0;
char gps_fix = false;
char gps_satellites = 0;

// Funkcje
unsigned char from_hex(char a)
{
  if (a >= 'A' && a <= 'F')
    return a - 'A' + 10;
  else if (a >= 'a' && a <= 'f')
    return a - 'a' + 10;
  else if (a >= '0' && a <= '9')
    return a - '0';
  else
    return 0;
}

void parse_sentence_type(const char *token)
{
  if (strcmp(token, "$GPGGA") == 0) {
    sentence_type = SENTENCE_GGA;
  } else if (strcmp(token, "$GPRMC") == 0) {
    sentence_type = SENTENCE_RMC;
  } else {
    sentence_type = SENTENCE_UNK;
  }
}

void parse_time(const char *token)
{
```

```
  // Czas moze zawierac ulamki sekund ale wykorzystywane tylko godziny, minuty i pelne sekundy
GGMMSS
  strncpy(new_time, token, 6);
}

void parse_status(const char *token)
{
  // "A" = czynny, "V" = nieczynny. Dane oznaczone jako „void” powinny byc ignorowane
  if (strcmp(token, "A") == 0)
    fix_ok = true;
  else
    fix_ok = false;
}

void parse_lat(const char *token)
{
  // formatowanie szerokosci geograficznej "SS" + "MM" (+ ".M{...}M")
  char degs[3];
  if (strlen(token) >= 4) {
    degs[0] = token[0];
    degs[1] = token[1];
    degs[2] = '\0';
    new_lat = atof(degs) + atof(token + 2) / 60;
  }
  // szerokosc w formacie dla APRS
  strncpy(new_aprs_lat, token, 7);
}

void parse_lat_hemi(const char *token)
{
  if (token[0] == 'S')
    new_lat = -new_lat;
  new_aprs_lat[7] = token[0];
  new_aprs_lon[8] = '\0';
}

void parse_lon(const char *token)
{
  // formatowanie dlugosci geograficznej "SSS" + "MM" (+ ".M{...}M")
  char degs[4];
  if (strlen(token) >= 5) {
    degs[0] = token[0];
    degs[1] = token[1];
    degs[2] = token[2];
    degs[3] = '\0';
    new_lon = atof(degs) + atof(token + 3) / 60;
  }
  // dlugosc w formacie dla APRS
  strncpy(new_aprs_lon, token, 8);
}

void parse_lon_hemi(const char *token)
{
  if (token[0] == 'W')
    new_lon = -new_lon;
```

```
  new_aprs_lon[8] = token[0];
  new_aprs_lon[9] = '\0';
}

void parse_speed(const char *token)
{  // zamiana wartosci na typ float i zapamietanie
  new_speed = atof(token);
}

void parse_course(const char *token)
{  // zamian wartosci  na typ float i zapamietanie
  new_course = atof(token);
}

void parse_altitude(const char *token)
{  // zamiana wartosci na typ float i zapamietanie
  new_altitude = atof(token);
}

void parse_satellites(const char *token)
{
  /* zamiana wartosci na integer i zapamietanie */
  int temp = atoi(token);

  /* liczba pomiedzy 0-32 satelitami */
  if(temp < 0)
    temp = 0;

  if(temp > 32)
    temp = 32;

  /* zapamietanie wartosci */
  new_satellites = (char)temp;
}

//
// funkcje eksportowane
//
void gps_setup()
{
  strcpy(gps_time, "000000");
  strcpy(gps_aprs_lat, "0000.00N");
  strcpy(gps_aprs_lon, "00000.00E");

  pinMode(GPS_RESET_PIN, OUTPUT);
  digitalWrite(GPS_RESET_PIN, HIGH);
  pinMode(GPS_POWER_PIN, OUTPUT);
  digitalWrite(GPS_POWER_PIN, HIGH); // wylaczenie modulu Venus na sekunde
  delay(1000);
  digitalWrite(GPS_POWER_PIN, LOW);  // wlaczenie modulu Venus.
  delay(1000);
  digitalWrite(GPS_RESET_PIN, LOW);  // zerowanie modulu Venus GPS aby zapewnic w 100%
prawidlowy start
  delay(100);
  digitalWrite(GPS_RESET_PIN, HIGH);
```

```
  pinMode(GPS_RESET_PIN, INPUT);

/*
  // wybor trybu nawigacji (Airborne, 1G)
  uint8_t setNav[] = {0xB5, 0x62, 0x06, 0x24, 0x24, 0x00, 0xFF, 0xFF, 0x06, 0x03, 0x00, 0x00, 0x00,
0x00, 0x10, 0x27, 0x00, 0x00, 0x05, 0x00, 0xFA, 0x00, 0xFA, 0x00, 0x64, 0x00, 0x2C, 0x01, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x16, 0xDC};
  sendUBX(setNav, sizeof(setNav)/sizeof(uint8_t));
  //getUBX_ACK(setNav);

*/
}

bool gps_decode(char c)
{
  int ret = false;

  switch(c) {
    case '\r':
    case '\n':
      // koniec komunikatu

     if (num_tokens && our_checksum == their_checksum) {
#ifdef DEBUG_GPS
       Serial.print(" (OK!)");
#endif
       // Wydanie pozycji tylko gdy odebrane dwa komunikaty rmc i gga
       // z tego samego czasu.
       switch (sentence_type) {
         case SENTENCE_UNK:
           break;    // typ nieokreslony
         case SENTENCE_GGA:
           strcpy(gga_time, new_time);
           break;
         case SENTENCE_RMC:
           strcpy(rmc_time, new_time);
           break;
       }

       // Sytuacja prawidlowego odbioru pozycji:
       //
       // 1. Czasy podane w dwoch poprzednich komunikatach GGA/RMC musza sie zgadzac.
       //
       // 2. Poprzednio odebrane i wykorzystane wazne komunikaty (GGA/RMC). Zakladajac stan
       //    przeciwny: po uruchomieniu tego modulu, gga_time i rmc_time
       //    sa sobie rowne (oba zainicjalizowane jako ""), sygnalizacja (1)
       //    blednie spowodowalaby uznanie pozycji za prawidlowa.
       //
       // 3. Komunikat  GPS wazny. Z niewiadomych powodow Venus 634FLPX
       //    podaje wspolrzedne 24 deg N, 121 deg E (srodek Tajwanu) do czasu odbioru
       //    waznego komunikatu:
       //
       //    $GPGGA,120003.000,2400.0000,N,12100.0000,E,0,00,0.0,0.0,M,0.0,M,,0000**69 (OK!)
       //    $GPGSA,A,1,,,,,,,,,,,,,0.0,0.0,0.0**30 (OK!)
       //    $GPRMC,120003.000,V,2400.0000,N,12100.0000,E,000.0,000.0,280606,,,N**78 (OK!)
```

```
      //    $GPVTG,000.0,T,,M,000.0,N,000.0,K,N**02 (OK!)

      if (sentence_type != SENTENCE_UNK &&      // sprawdzenie czy znany typ komunikatu ?
          strcmp(gga_time, rmc_time) == 0 &&    // sprawdzenie czy czasy RMC/GGA sa zgodne?
          fix_ok) {                             // sprawdzenie czy komunikaty prawidlowe?
        // Zlozenie danych z obu komunikatow
        strcpy(gps_time, new_time);
        gps_lat = new_lat;
        gps_lon = new_lon;
        strcpy(gps_aprs_lat, new_aprs_lat);
        strcpy(gps_aprs_lon, new_aprs_lon);
        gps_course = new_course;
        gps_speed = new_speed;
        gps_altitude = new_altitude;
        gps_satellites = new_satellites;
        ret = true;
      }

      /* aktualizacja sygnalizacji waznego komunikatu GPS */
      gps_fix = (fix_ok) ? 1 : 0;
    }
#ifdef DEBUG_GPS
    if (num_tokens)
      Serial.println();
#endif
    at_checksum = false;        // znak CR/LF na zakonczenie sumy kontrolnej
    our_checksum = '$';         // zerowanie sumy kontrolnej
    their_checksum = 0;
    offset = 0;                 // Przygotowanie do odbioru nastepnego komunikatu GPS
    num_tokens = 0;
    sentence_type = SENTENCE_UNK;
    break;

  case '*':
    // Poczatek sumy kontrolnej i sprawdzania (odbior bez przerwy)
    at_checksum = true;
    our_checksum ^= c;
#ifdef DEBUG_GPS
    Serial.print(c);
#endif

  case ',':
    // sprawdzanie
    token[offset] = '\0';
    our_checksum ^= c;  // suma kontrolna ',', skasowanie '*'

    // analiza danych
    switch (sentence_type) {
      case SENTENCE_UNK:
        if (num_tokens < NUM_OF_UNK_PARSERS && unk_parsers[num_tokens])
          unk_parsers[num_tokens](token);
        break;
      case SENTENCE_GGA:
        if (num_tokens < NUM_OF_GGA_PARSERS && gga_parsers[num_tokens])
          gga_parsers[num_tokens](token);
```

```
        break;
       case SENTENCE_RMC:
        if (num_tokens < NUM_OF_RMC_PARSERS && rmc_parsers[num_tokens])
          rmc_parsers[num_tokens](token);
        break;
      }

      // przygotowanie do odbioru nastepnych danych
      num_tokens++;
      offset = 0;
#ifdef DEBUG_GPS
      Serial.print(c);
#endif
      break;

    default:
      // jakikolwiek inny znak
      if (at_checksum) {
        // Checksum value
        their_checksum = their_checksum * 16 + from_hex(c);
      } else {
        // Prawidlowe dane NMEA
        if (offset < 15) {  // unikanie przepelnienia bufora (dlugosc nie moze przekraczac 15 znakow)
          token[offset] = c;
          offset++;
          our_checksum ^= c;
        }
      }
#ifdef DEBUG_GPS
      Serial.print(c);
#endif
  }
  return ret;
}

// Nadanie tabeli danych protokolu UBX do GPS
void sendUBX(unsigned char *MSG, uint8_t len)
{
       for(int i=0; i<len; i++)
              Serial.write(MSG[i]);
//    Serial.print(MSG[i], BYTE);
}

// obliczanie oczekiwanego pakietu potwierdzajacego UBX (ACK) i zlozenie odpowiedzi UBX z GPS
int getUBX_ACK(unsigned char *MSG)
{
  unsigned char b;
  unsigned char ackByteID = 0;
  unsigned char ackPacket[10];
  int startTime = millis();

  // Konstrukcja oczekiwanego pakietu potwierdzajacego
  ackPacket[0] = 0xB5;  // naglowek
  ackPacket[1] = 0x62;  // naglowek
  ackPacket[2] = 0x05;  // klasa
```

```
  ackPacket[3] = 0x01;  // identyfikator
  ackPacket[4] = 0x02;  // dlugosc
  ackPacket[5] = 0x00;
  ackPacket[6] = MSG[2];        // klasa ACK
  ackPacket[7] = MSG[3];        // identyfikator ACK
  ackPacket[8] = 0;             // CK_A
  ackPacket[9] = 0;             // CK_B

  // obliczanie sumy kontrolnej
  for (unsigned char i=2; i<8; i++)
  {
    ackPacket[8] = ackPacket[8] + ackPacket[i];
    ackPacket[9] = ackPacket[9] + ackPacket[8];
  }

  while (1)
  {
    // sprawdzenie czy sukces
    if (ackByteID > 9)
      return true;

    // Czas oczekiwania na wazna odpowiedz mija po trzech sekundach
    if (millis() - startTime > 3000)
      return false;

    // Upewnienie sie, że dane gotowe do odczytu
    if (Serial.available())
    {
      b = Serial.read();

      // Sprawdzenie czy dane docieraja w kolejnosci zgodnej z oczekiwaniami dla pakietu ACK
      if (b == ackPacket[ackByteID])
      {
        ackByteID++;
      }
      else
      {
        ackByteID = 0;  // Kolejnosc niewlasciwa, zerowanie i dalsze oczekiwanie.
      }
    }
  }
}
```

Plik varicode.h

```
//////////////////////////////////////////////////////////////
// AD9835_PSK31
//
// Prosty program radiolatarni PSK31 na 20 m nadajacy wspolrzedne GPS
// pracuje na ATMega328 i AD9835 DDS
//
// autor Thomas Krahn KT5TK / DL4MDW (2012)
//
// Program jest bezplatnie dostepny na zasadach licencji GNU
//

/* Tabela zawiera kod „varicode“ dla PSK3 – 128 kodow odpowiadajacych znakom 0-127 kodu ASCII,
Kody maja dlugosc dwoch bajtow. Sa one pobierane poczawszy od bitow najstarszych (MSB) przez
przesuniecie w lewo obu bajtow. Bajt starszy jest nadawany jako pierwszy.
Zera na poczatku pierwszegio bajtu sa ignorowane co oznacza koniecznioscprzesuniecia bitow
Az do otrzymania pierwszej jedynki. W kodzie moga wystepowac tylko pojedyncze zera,
ich wieksza liczba oznacza koniec znaku. Do oddzielenia znakow od siebie sluzy sekwencja dwoch zer.

Zero logiczne powoduje zmiane fazy sygnalu nadawanego a jedynka – stan bez zmiany.

Plik powstal w oparciu o artykul "PSK31 Fundamentals"
autorstwa Petera Martineza, G3PLX i Clinta Turnera, KA7OEI
*/

//const unsigned char varicode[256] PROGMEM = {
char varicode[256] = {
0b10101010,
0b11000000, // 0 NUL
0b10110110,
0b11000000, // 1 SOH
0b10111011,
0b01000000, // 2 STX
0b11011101,
0b11000000, // 3 ETX
0b10111010,
0b11000000, // 4 EOT
0b11010111,
0b11000000, // 5 ENQ
0b10111011,
0b11000000, // 6 ACK
0b10111111,
0b01000000, // 7 BEL
0b10111111,
0b11000000, // 8 BS
0b11101111,
0b00000000, // 9 HT
0b11101000,
0b00000000, // 10 LF
0b11011011,
0b11000000, // 11 VT
```

```
0b10110111,
0b01000000, // 12 FF
0b11111000,
0b00000000, // 13 CR
0b11011101,
0b01000000, // 14 SO
0b11101010,
0b11000000, // 15 SI
0b10111101,
0b11000000, // 16 DLE
0b10111101,
0b01000000, // 17 DC1
0b11101011,
0b01000000, // 18 DC2
0b11101011,
0b11000000, // 19 DC3
0b11010110,
0b11000000, // 20 DC4
0b11011010,
0b11000000, // 21 NAK
0b11011011,
0b01000000, // 22 SYN
0b11010101,
0b11000000, // 23 ETB
0b11011110,
0b11000000, // 24 CAN
0b11011111,
0b01000000, // 25 EM
0b11101101,
0b11000000, // 26 SUB
0b11010101,
0b01000000, // 27 ESC
0b11010111,
0b01000000, // 28 FS
0b11101110,
0b11000000, // 29 GS
0b10111110,
0b11000000, // 30 RS
0b11011111,
0b11000000, // 31 US
0b10000000,
0b00000000, // 32 SP
0b11111111,
0b10000000, // 33 !
0b10101111,
0b10000000, // 34 "
0b11111010,
0b10000000, // 35 #
0b11101101,
0b10000000, // 36 $
0b10110101,
0b01000000, // 37 %
```

```
0b10101110,
0b11000000, // 38 &
0b10111111,
0b10000000, // 39 pojedynczy cudzyslow
0b11111011,
0b00000000, // 40 (
0b11110111,
0b00000000, // 41 )
0b10110111,
0b10000000, // 42 *
0b11101111,
0b10000000, // 43 +
0b11101010,
0b00000000, // 44 ,
0b11010100,
0b00000000, // 45 -
0b10101110,
0b00000000, // 46 .
0b11010111,
0b10000000, // 47 /
0b10110111,
0b00000000, // 48 0
0b10111101,
0b00000000, // 49 1
0b11101101,
0b00000000, // 50 2
0b11111111,
0b00000000, // 51 3
0b10111011,
0b10000000, // 52 4
0b10101101,
0b10000000, // 53 5
0b10110101,
0b10000000, // 54 6
0b11010110,
0b10000000, // 55 7
0b11010101,
0b10000000, // 56 8
0b11011011,
0b10000000, // 57 9
0b11110101,
0b00000000, // 58 :
0b11011110,
0b10000000, // 59 ;
0b11110110,
0b10000000, // 60 ,
0b10101010,
0b00000000, // 61 =
0b11101011,
0b10000000, // 62 >
0b10101011,
0b11000000, // 63 ?
```

```
0b10101111,
0b01000000, // 64 @
0b11111010,
0b00000000, // 65 A
0b11101011,
0b00000000, // 66 B
0b10101101,
0b00000000, // 67 C
0b10110101,
0b00000000, // 68 D
0b11101110,
0b00000000, // 69 E
0b11011011,
0b00000000, // 70 F
0b11111101,
0b00000000, // 71 G
0b10101010,
0b10000000, // 72 H
0b11111110,
0b00000000, // 73 I
0b11111110,
0b10000000, // 74 J
0b10111110,
0b10000000, // 75 K
0b11010111,
0b00000000, // 76 L
0b10111011,
0b00000000, // 77 M
0b11011101,
0b00000000, // 78 N
0b10101011,
0b00000000, // 79 O
0b11010101,
0b00000000, // 80 P
0b11101110,
0b10000000, // 81 Q
0b10101111,
0b00000000, // 82 R
0b11011110,
0b00000000, // 83 S
0b11011010,
0b00000000, // 84 T
0b10101011,
0b10000000, // 85 U
0b11011010,
0b10000000, // 86 V
0b10101110,
0b10000000, // 87 W
0b10111010,
0b10000000, // 88 X
0b10111101,
0b10000000, // 89 Y
```

```
0b10101011,
0b01000000, // 90 Z
0b11111011,
0b10000000, // 91 [
0b11110111,
0b10000000, // 92 \ (ukosnik wsteczny)
0b11111101,
0b10000000, // 93 ]
0b10101111,
0b11000000, // 94 ^
0b10110110,
0b10000000, // 95 _ (podkreslnik)
0b10110111,
0b11000000, // 96 `
0b10110000,
0b00000000, // 97 a
0b10111110,
0b00000000, // 98 b
0b10111100,
0b00000000, // 99 c
0b10110100,
0b00000000, // 100 d
0b11000000,
0b00000000, // 101 e
0b11110100,
0b00000000, // 102 f
0b10110110,
0b00000000, // 103 g
0b10101100,
0b00000000, // 104 h
0b11010000,
0b00000000, // 105 i
0b11110101,
0b10000000, // 106 j
0b10111111,
0b00000000, // 107 k
0b11011000,
0b00000000, // 108 l
0b11101100,
0b00000000, // 109 m
0b11110000,
0b00000000, // 110 n
0b11100000,
0b00000000, // 111 o
0b11111100,
0b00000000, // 112 p
0b11011111,
0b10000000, // 113 q
0b10101000,
0b00000000, // 114 r
0b10111000,
0b00000000, // 115 s
```

```
0b10100000,
0b00000000, // 116 t
0b11011100,
0b00000000, // 117 u
0b11110110,
0b00000000, // 118 v
0b11010110,
0b00000000, // 119 w
0b11011111,
0b00000000, // 120 x
0b10111010,
0b00000000, // 121 y
0b11101010,
0b10000000, // 122 z
0b10101101,
0b11000000, // 123 {
0b11011101,
0b10000000, // 124 |
0b10101101,
0b01000000, // 125 }
0b10110101,
0b11000000, // 126 ~
0b11101101,
0b01000000 // 127 (kasowanie)
};
```

Biblioteka GPS
Plik gps.h

```
//////////////////////////////////////////////////////////////
// AD9835_PSK31
//
// Prosty program radiolatarni PSK31 na 20 m nadajacy wspolrzedne GPS
// pracuje na ATMega328 i AD9835 DDS
//
// autor Thomas Krahn KT5TK / DL4MDW (2012)
//
// Program jest bezplatnie dostepny na zasadach licencji GNU
//

/*
 * Kod pochodzi z trackuino copyright (C) 2010  EA5HAV Javi
 * Zmodyfikowany w 2012 dla wlaczenia trybu uBlox przez VK5QI
 * Zmodyfikowany w 2012 przez KT5TK do uzytku w radiolatarni PSK31
 *
 */

#ifndef __GPS_H__
#define __GPS_H__

extern char gps_time[7];       // HHMMSS
extern char gps_date[7];       // DDMMYY
extern float gps_lat;
```

```
extern float gps_lon;
extern char gps_aprs_lat[9];    //DDMM.MMH0
extern char gps_aprs_lon[10];   //DDDMM.MMH0
extern float gps_course;
extern float gps_speed;
extern float gps_altitude;
extern char gps_fix;
extern char gps_satellites;

void gps_setup();
bool gps_decode(char c);
void sendUBX(unsigned char *MSG, unsigned char len);
int getUBX_ACK(unsigned char *MSG);

#endif
```

Odczyt czasu w protokóle NTP

Znajomość dokładnego czasu jest sprawą bardzo istotną dla krótkofalowców (i nie tylko). Dotyczy to zarówno protokółów zawodów, wystawianych kart QSL, łączności przez odbicia od śladów meteorytów (MS) albo od księżyca (EME) jak i w coraz większym stopniu pracy niektórymi emisjami cyfrowymi. Emisje takie jak WSPR, JT65, FSK441, JT6M, ISCAT, JT2,JT4 itp. wymagają rozpoczynania transmisji z dużą dokładnością we właściwym momencie cyklu (na początku odpowiedniej minuty lub odcinka 30-sekundowego). Wymagają one nastawienia zegara komputera z dokładnością co do sekundy. Zegar można oczywiście nastawiać ręcznie opierając się na sygnałach czasu z radia bądź telewizji albo odczytu sterowanego radiowo zegara ale można też wyposażyć komputer w odbiornik wzrocowych sygnałów czasu albo zainstalować na nim program pobierający dokładny czas z serwerów internetowych w protokóle NTP.
Programy takie mogą pracować także na minikomputerach w rodzaju Arduino lub Raspberry Pi. Dwa z poniższych programów są przykładami programów odbierających dokładny czas z internetu i korzystających z modułu ethernetowego lub WiFi dla Arduino. Nastepne dwa są przykładami praktycznego ich zastosowania w zegarach internetowych wyposażonych w wyświetlacze ciekłokrystaliczne różnych typów.

Ethernetowy klient NTP

```
/*
Ethernetowy klient Udp NTP

Pobiera czas z serwera internetowego w protokole “Network Time Protocol” (NTP)
Ilustruje sposob uzycia funkcji UDP sendPacket() i ReceivePacket()
Wiecej informacji na temat serwerow NTP i sposobu komunikacji z nimi
Podano pod adresem http://en.wikipedia.org/wiki/Network_Time_Protocol

Uwaga: serwery NTP pracuja niestabilnie i czesto zmieniaja adresy IP.
Zaleca sie sprawdzanie ich dostepnosci.

http://tf.nist.gov/tf-cgi/servers.cgi

Program z 4 wrzesnia 2010
autor Michael Margolis
zmodyfikowany 9 kwietnia 2012
przez Toma Igoe

Program jest dostepny bezplatnie na zasadach licencji GNU.
*/

#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>

// Wprowadzic adres MAC modulu.
// Nowsze moduly maja adres MAC naklejony na plytce
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

unsigned int localPort = 8888; // lokalny kanal odbiorczy UDP

IPAddress timeServer(132, 163, 4, 101); // serwer NTP time-a.timefreq.bldrdoc.gov
// IPAddress timeServer(132, 163, 4, 102); // serwer NTP time-b.timefreq.bldrdoc.gov
// IPAddress timeServer(132, 163, 4, 103); //serwer NTP time-c.timefreq.bldrdoc.gov
```

```
const int NTP_PACKET_SIZE= 48; // czas NTP w pierwszych 48 bajtach komunikatu

byte packetBuffer[ NTP_PACKET_SIZE]; // bufor dla pakietow nadawanych i obieranych

// Obiekt UDP
EthernetUDP Udp;

void setup()
{
  // Inicjalizacja zlacza szeregowego i oczekiwanie na otwarcie:
  Serial.begin(9600);
  while (!Serial) {
    ; // oczekiwanie konieczne tylko dla Arduino Leonardo
  }

  // Nawiazanie polaczenia ethernetowego i UDP
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // w przypadku niepowodzenia zaprzestanie dzialanosci:
    for(;;)
      ;
  }
  Udp.begin(localPort);
}

void loop()
{
  sendNTPpacket(timeServer); // Nadanie pakietu zapytania NTP do serwera

  // oczekiwanie na odpowiedz
  delay(1000);
  if ( Udp.parsePacket() ) {
    // po odebraniu pakietu odczyt danych
    Udp.read(packetBuffer,NTP_PACKET_SIZE); // wczytanie pakietu do bufora

    // informacja o czasie rozpoczyna sie od 40 bajtu i ma dlugosc 4 bajtow
    // lub dwoch slow. Najpierw konieczne wydzielenie tych dwoch slow:

    unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
    unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
    // zlozenie czterech bajtow w zmienna typu long integer
    // oznaczajaca liczbe sekund od 1 stycznia 1900 (czas NTP):
    unsigned long secsSince1900 = highWord << 16 | lowWord;
    Serial.print("Seconds since Jan 1 1900 = " );
    Serial.println(secsSince1900);

    // zamiana na standardowy format czasu:
    Serial.print("Unix time = ");
    // Unixowa rachuba czasu rozpoczyna sie od 1 stycznia 1970, co odpowiada liczbie sekund
2208988800:
    const unsigned long seventyYears = 2208988800UL;
    // odjecie 70 lat:
    unsigned long epoch = secsSince1900 - seventyYears;
    // wyswietlenie czasu Unixowego:
    Serial.println(epoch);
```

```
    // wyswietlenie godzin, minut i sekund:
    Serial.print("The UTC time is "); // czas UTC czyli Greenwich Meridian (GMT)
    Serial.print((epoch % 86400L) / 3600); // wyswietlenie godziny (86400 sekund dziennie)
    Serial.print(':');
    if ( ((epoch % 3600) / 60) < 10 ) {
      // Dla pierwszych 10 minut godziny zero na poczatku - '0'
      Serial.print('0');
    }
    Serial.print((epoch % 3600) / 60); // wyswietlenie minuty (3600 sekund na godzine)
    Serial.print(':');
    if ( (epoch % 60) < 10 ) {
      // Dla pierwszych 10 sekund godziny zero na poczatku – '0'
      Serial.print('0');
    }
    Serial.println(epoch %60); // wyswietlenie sekund
  }
  // oczekiwanie 10 sekund przed nadaniem nastepnego zapytania
  delay(10000);
}

// nadanie zapytania NTP pod podany adres serwera
unsigned long sendNTPpacket(IPAddress& address)
{
  // ustawienie wszystkich bajtow w buforze na 0
  memset(packetBuffer, 0, NTP_PACKET_SIZE);
  // inicjalizacja danych niezbednych do zapytania NTP
  // (szczegolowy format pakietu opisany pod podanym wyzej adresem)
  packetBuffer[0] = 0b11100011; // LI, wersja, tryb
  packetBuffer[1] = 0; // “Stratum”, czyli typ zegara
  packetBuffer[2] = 6; // Odstep czasu miedzy zapytaniami
  packetBuffer[3] = 0xEC; // Dokladnosc zegara
  // 8 bajtow o wartosci 0 dla parametrow “Root Delay” i “Root Dispersion”
  packetBuffer[12] = 49;
  packetBuffer[13] = 0x4E;
  packetBuffer[14] = 49;
  packetBuffer[15] = 52;

  // wszystkie pola NTP wypelnione
  // nadanie zapytania o czas:
  Udp.beginPacket(address, 123); // Zapytania kierowane do kanalu logicznego 123
  Udp.write(packetBuffer,NTP_PACKET_SIZE);
  Udp.endPacket();
}
```

Bezprzewodowy klient NTP

```
/*
Ethernetowy klient Udp NTP

Pobiera czas z serwera internetowego w protokole “Network Time Protocol” (NTP)
Ilustruje sposob uzycia funkcji UDP sendPacket() i ReceivePacket()
Wiecej informacji na temat serwerow NTP i sposobu komunikacji z nimi
Podano pod adresem http://en.wikipedia.org/wiki/Network_Time_Protocol
```

```
Program z 4 wrzesnia 2010
autor Michael Margolis
zmodyfikowany 9 kwietnia 2012
przez Toma Igoe

Kod jest dostepny bezplatnie.
*/

#include <SPI.h>
#include <WiFi.h>
#include <WiFiUdp.h>

int status = WL_IDLE_STATUS;
char ssid[] = "mynetwork"; // nazwa (SSID) sieci bezprzewodowej
char pass[] = "mypassword"; // haslo dostepu do sieci
int keyIndex = 0; // Indeks do hasla (konieczny tylko dla WEP)

unsigned int localPort = 2390; // lokalny kanal logiczny dla odbioru pakietow UDP

IPAddress timeServer(129, 6, 15, 28); // serwer NTP time.nist.gov

const int NTP_PACKET_SIZE = 48; // czas NTP w pierwszych 48 bajtach pakietu

byte packetBuffer[ NTP_PACKET_SIZE]; // bufor dla pakietow nadawanych i odbieranych

// Obiekt UDP dla odbioru i transmisji danych w protokole UDP
WiFiUDP Udp;

void setup()
{
  // Inicjalizacja zlacza szeregowego i oczekiwanie na jego otwarcie:
  Serial.begin(9600);
  while (!Serial) {
    ; // oczekiwanie niezbedne tylko dla Arduino Leonardo
  }

  // sprawdzenie czy modul podlaczony:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // zaprzestanie dzialanosci:
    while(true);
  }

  // Proba nawiazania polaczenia z siecia bezprzewodowa:
  while ( status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
  // Polaczenie z sieciami WPA/WPA2. Nalezy dopasowac wywolanie dla sieci otwartych lub
WEP:
    status = WiFi.begin(ssid, pass);
```

```
    // oczekiwanie 10 sekund na nawiazanie polaczenia:
    delay(10000);
  }

  Serial.println("Connected to wifi");              // meldunek dla uzytkownika
  printWifiStatus();

  Serial.println("\nStarting connection to server...");
  Udp.begin(localPort);
}

void loop()
{
  sendNTPpacket(timeServer); // nadanie zapytania NTP do serwera
  // oczekiwanie na odpowiedz
  delay(1000);
  Serial.println( Udp.parsePacket() );
  if ( Udp.parsePacket() ) {
    Serial.println("packet received");
    // Po odebraniupakietu odczyt danych
    Udp.read(packetBuffer,NTP_PACKET_SIZE); // wczytanie pakietu do bufora

    // informacja o czasie zaczyna sie od 40 bajtu i ma dlugosc 4 bajtow
    // lub dwoch slow. Na poczatek wydzielenie tych slow:

    unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
    unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
    // zlozenie ich w zmienna typu long integer
    // Jest to czas NTP (liczba sekund od 1 stycznia 1900):
    unsigned long secsSince1900 = highWord << 16 | lowWord;
    Serial.print("Seconds since Jan 1 1900 = " );
    Serial.println(secsSince1900);

    // Zamiana formatu na stanardowy format czasu:
    Serial.print("Unix time = ");
    // Unixowa rachuba czasu rozpoczyna sie od 1 stycznia 1970, co odpowiada 2208988800
sekundom:
    const unsigned long seventyYears = 2208988800UL;
    // odjecie siedemdziesieciu lat:
    unsigned long epoch = secsSince1900 - seventyYears;
    // Wyswietlenie czasu Unixowego:
    Serial.println(epoch);

    // wyswietlenie godzin, minut i sekund:
    Serial.print("The UTC time is "); // czas UTC czyli Greenwich Meridian (GMT)
    Serial.print((epoch % 86400L) / 3600); // wyswietlenie godzin (86400 sekund dziennie)
    Serial.print(':');
    if ( ((epoch % 3600) / 60) < 10 ) {
      // W ciagu pierwszych 10 minut godziny konieczne zero na poczatku - '0'
      Serial.print('0');
    }
```

```
    Serial.print((epoch % 3600) / 60); // wyswietlenie minut (3600 sekund na godzine)
    Serial.print(':');
    if ( (epoch % 60) < 10 ) {
    // W czasie pierwszych 10 sekund minuty konieczne zero na poczatku – '0'
    Serial.print('0');
   }
   Serial.println(epoch %60); // wyswietlenie sekund
   }
   // oczekiwanie 10 sekund przed nadaniem nastepnego zapytania
   delay(10000);
}

// Nadanie zapytania NTP do serwera o podanym adresie
unsigned long sendNTPpacket(IPAddress& address)
{
  //Serial.println("1");
  // ustawienie wszystkich bajtow w buforze na 0
  memset(packetBuffer, 0, NTP_PACKET_SIZE);
  // Inicjalizacja wartosci niezbednych dla zapytania NTP
  // (szczegolowy opis formatu pod podanym powyzej adresem)
  //Serial.println("2");
  packetBuffer[0] = 0b11100011; // LI, wersja, tryb
  packetBuffer[1] = 0; // “Stratum” czyli typ zegara
  packetBuffer[2] = 6; // Odstep czasu miedzy zapytaniami
  packetBuffer[3] = 0xEC; // Dokladnosc zegara
  // 8 bajtow o wartosci 0 dla parametrow “Root Delay” i “Root Dispersion”
  packetBuffer[12] = 49;
  packetBuffer[13] = 0x4E;
  packetBuffer[14] = 49;
  packetBuffer[15] = 52;

  //Serial.println("3");

  // Wszystkie pola NTP wypelnione
  // nadanie pakietu zapytania:
  Udp.beginPacket(address, 123); // Zapytania NTP kierowasne do kanalu logicznego 123
  //Serial.println("4");
  Udp.write(packetBuffer,NTP_PACKET_SIZE);
  //Serial.println("5");
  Udp.endPacket();
  //Serial.println("6");
}

void printWifiStatus() {
  // wyswietlenie nazwy (SSID sieci:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // Wyswietlenie adresu IP modulu:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
```

```
  Serial.println(ip);

  // wyswietlenie sily odbieranego sygnalu:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

Zegar internetowy z wyświetlaczem ciekłokrystalicznym 2 x 16 znaków

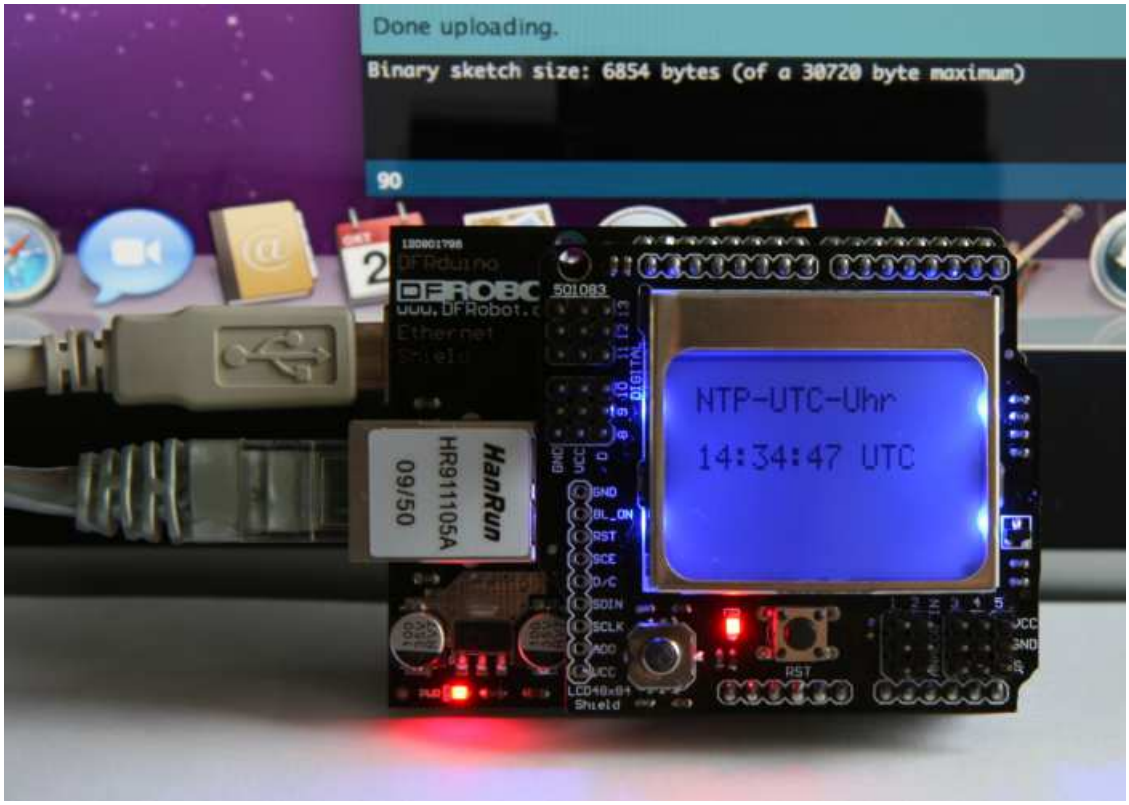
Fot. 7.1. Wygląd zegara internetowego z wyświetlaczem

```
/*
Klient Udp NTP z wyswietlaczem cieklokrystalicznym LCD 16x2

Pobiera informacje o czasie z serwerow internetowych za posrednictwem protokolu Network Time
Protocol (NTP)
Ilustruje wykorzystanie funkcji UDP sendPacket() i ReceivePacket()
Wiecej informacji na temat protokolu NTP, serwerow i komunikacji z nimi znajduje sie,
pod adresem http://en.wikipedia.org/wiki/Network_Time_Protocol

Program z 4 wrzesnia 2010
autor Michael Margolis
zmodyfikowany 17 wrzesnia 2010
przez Toma Igoe
zmodyfikowany 23 pazdziernika 2010
```

```
przez Juergena Mayera, DL8MA, Grossheppach

Program dostepny bezplatnie.
*/

#include <LCD4Bit_mod.h>
LCD4Bit_mod lcd = LCD4Bit_mod(2);

#include <SPI.h>
#include <Ethernet.h>
#include <Udp.h>

// Wprowadzenie adresow MAC i IP modulu.
// Adres IP zalezny od lokalnej sieci:
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192,168,178,15 };

char string[ 17 ] = { "" };

int stunde, minute, sekunde;

unsigned int localPort = 8888; // lokalny kanal odbiorczy UDP

byte timeServer[] = { 192, 43, 244, 18}; // serwer NTP time.nist.gov

const int NTP_PACKET_SIZE= 48; // Czas NTP zawarty w pierwszych 48 bajtach komunikatu

byte packetBuffer[ NTP_PACKET_SIZE]; // bufor dla pakietow nadawnych i odbieranych

void setup()
{
  // Inicjalizacja polaczenia internetowego i UD
  Ethernet.begin(mac,ip);
  Udp.begin(localPort);
  // inicjalizacja wyswietlacza
  lcd.init();
  lcd.clear();
  lcd.printIn("NTP-Uhr");
}

void loop()
{
  sendNTPpacket(timeServer); // nadanie zapytania NTP do serwera

  // oczekiwanie na odpowiedz
  delay(1000);
  if ( Udp.available() ) {
    Udp.readPacket(packetBuffer,NTP_PACKET_SIZE); // wczytanie pakietu do bufora

    //informacja o czasie rozpoczyna sie od 40 bajtu i ma dlugosc 4 bajtow
    // czyli dwoch slow. Na poczatek koniedzne jest wydzielenie tych dwoch slow:

    unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
    unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
    // zlozenie ich w zmienna typu long integer
```

```
    // Jest to czas NTP (liczba sekund od 1 stycznia 1900):
    unsigned long secsSince1900 = highWord << 16 | lowWord;

    // zamiana na standardowy format czasu:
    // Unixowa rachuba czasu rozpoczyna sie od 1 stycznia 1970, co odpowiada 2208988800 sekundom:
    const unsigned long seventyYears = 2208988800UL;
    // odjecie siedemdziesieciu lat:
    unsigned long epoch = secsSince1900 - seventyYears;

    stunde = (epoch % 86400L) / 3600;
    minute = (epoch % 3600) / 60;
    sekunde = (epoch % 60);

    sprintf( string, "%02d:%02d:%02d UTC", stunde, minute, sekunde );

    lcd.cursorTo(2, 0);                          // linia=2, x=0
    lcd.printIn( string );
   }
   // odczekanie 10 sekund przed nastepnym zapytaniem
   delay( 333 );
}

// nadanie zapytania NTP do serwera o podanym adresie
unsigned long sendNTPpacket(byte *address)
{
   // ustawienie bajtow w buforze na 0
   memset(packetBuffer, 0, NTP_PACKET_SIZE);
   // Wprowadzenie wartosci niezbednych do utworzenia zapytania NTP
   // (szczegoly pod podanym powyzej adresem)
   packetBuffer[0] = 0b11100011; // LI, wersja, tryb
   packetBuffer[1] = 0; // „Stratum” czyli typ zegara
   packetBuffer[2] = 6; // Odstep czasu miedzy zapytaniami
   packetBuffer[3] = 0xEC; // dokladnosc zegara
   // 8 bajtow o wartosci 0 dla parametrow „Root Delay” i „Root Dispersion”
   packetBuffer[12] = 49;
   packetBuffer[13] = 0x4E;
   packetBuffer[14] = 49;
   packetBuffer[15] = 52;

   // wszystkie pola NTP wypelnione
   // nadanie zapytania do serwera:
   Udp.sendPacket( packetBuffer,NTP_PACKET_SIZE, address, 123); // zapytania NTP kierowane do
kanalu logicznego 123
}
```

Zegar internetowy z wyświetlaczem LCD4884



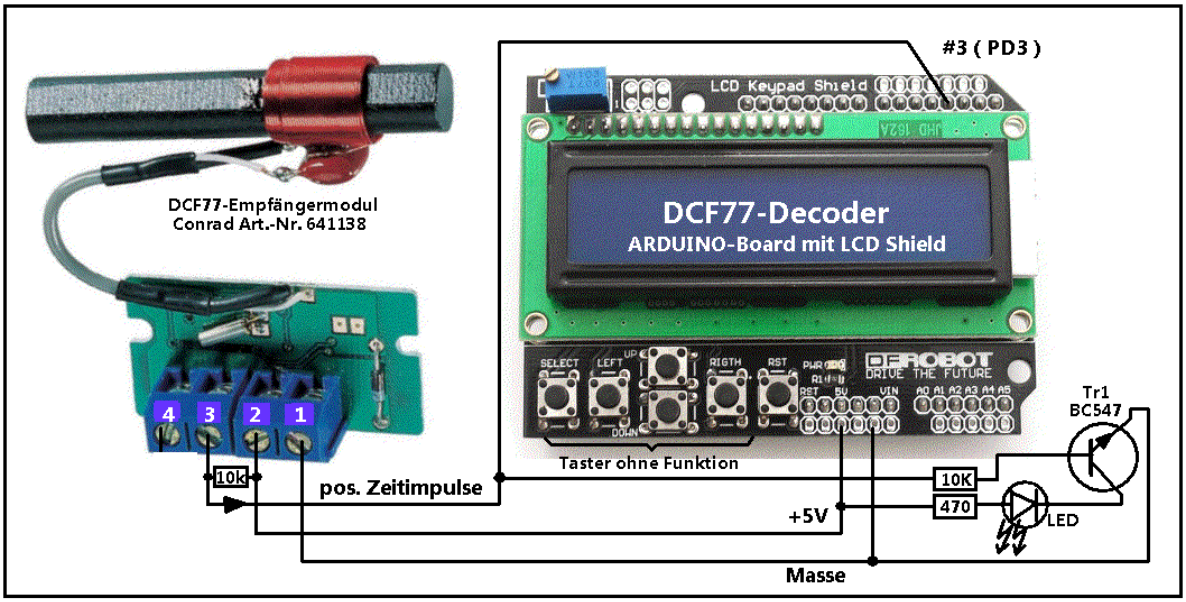
Fot 7.2. Wygląd zegara z wyświetlaczem 4884

```
/*
Klient Udp NTP z wyswietlaczem LCD4884 (DFROBOT)

Pobiera informacje o czasie z serwera w protokole Network Time Protocol (NTP)
Ilustruje sposob wykorzystania funkcji UDP sendPacket() i ReceivePacket()
Wiwecej informacji na temat protokolu, serwerow i sposobow komunikacji z nimi
pod adresem http://en.wikipedia.org/wiki/Network_Time_Protocol

Program z 4 wrzesnia 2010
autor Michael Margolis
zmodyfikowany 17 wrzesnia 2010
przez Toma Igoe
zmodyfikowany 23 pazdziernika 2010
przez Juergena Mayera, Grossheppach

Program dostepny bezpłatnie.
*/

#include <LCD4884.h>
#include <SPI.h>
#include <Ethernet.h>
#include <Udp.h>

// Wprowadzenie adresow MAC i IP modulu.
// Adres IP zalezny od sieci lokalnej:
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192,168,178,15 };
```

```
char string[ 17 ] = { "" };

int stunde, minute, sekunde;

unsigned int localPort = 8888; // lokalny kanal odbiorczy dla pakietow UDP

byte timeServer[] = { 192, 43, 244, 18}; // serwer NTP time.nist.gov

const int NTP_PACKET_SIZE= 48; // Informacja o czasie NTP w pierwszych 48 bajtach komunikatu

byte packetBuffer[ NTP_PACKET_SIZE]; // bufor dla odbieranych i nadawanych pakietow

void setup()
{
  // inicjalizacja polaczen ethernetowego i UDP
  Ethernet.begin(mac,ip);
  Udp.begin(localPort);
  // inicjalizacja wyswietlacza
  lcd.LCD_init();
  lcd.LCD_clear();
  lcd.LCD_write_string( 0, 0, "NTP-UTC-Uhr", 0 );
}

void loop()
{
  sendNTPpacket(timeServer); // nadanie zapytania NTP do serwera

  // oczekiwanie na odpowiedz
  delay(1000);
  if ( Udp.available() ) {
    Udp.readPacket(packetBuffer,NTP_PACKET_SIZE); // wczytanie pakietu do bufora

    //informacja o czasie rozpoczyna sie od 40 bajtu i ma dlugosc czterech bajtow
    // czyli dwoch slow. Na poczatek ich wydzielenie:

    unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
    unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
    // i zlozenie w zmienna typu long integer
   // Jest to czas NTP (liczba sekund od 1 stycznia 1900):
    unsigned long secsSince1900 = highWord << 16 | lowWord;

    // zamiana na standardowy format czasu:
    // Unixowa rachuba czasu rozpoczyna sie od 1 stycznia 1970, co odpowiada 2208988800 sekundom:
    const unsigned long seventyYears = 2208988800UL;
    // odjecie siedemdziesieciu lat:
    unsigned long epoch = secsSince1900 - seventyYears;

    stunde = (epoch % 86400L) / 3600;
    minute = (epoch % 3600) / 60;
    sekunde = (epoch % 60);

    sprintf( string, "%02d:%02d:%02d UTC", stunde, minute, sekunde );

    lcd.LCD_write_string( 0, 2, string, 0 );
```

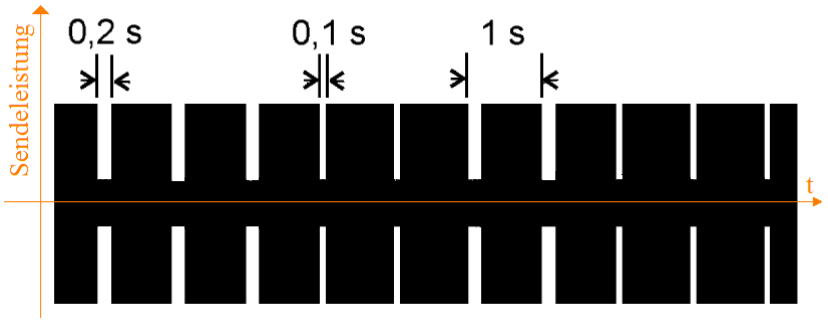
```
  }
  // odczekanie 10 sekund przed nastepnym zapytaniem
  delay( 333 );
}

// nadanie zapytania NTP do serwera o podanym adresie
unsigned long sendNTPpacket(byte *address)
{
  // ustawienie bajtow w buforze na zero
  memset(packetBuffer, 0, NTP_PACKET_SIZE);
  // Wprowadzenie wartosci niezbednych do utworzenia zapytania NTP
  // (szczegoly pod podanym powyzej adresem)
  packetBuffer[0] = 0b11100011; // LI, wersja, tryb
  packetBuffer[1] = 0; // „Stratum” czyli typ zegara
  packetBuffer[2] = 6; // Odstep czasu miedzy zapytaniami
  packetBuffer[3] = 0xEC; // dkladnosc zegara
  // 8 o wartosci 0 dla parametrow „Root Delay” i „Root Dispersion”
  packetBuffer[12] = 49;
  packetBuffer[13] = 0x4E;
  packetBuffer[14] = 49;
  packetBuffer[15] = 52;

  // wszystkie pola NTP wypelnione
  // nadanie zapytania:
  Udp.sendPacket( packetBuffer,NTP_PACKET_SIZE, address, 123); // zapytania NTP kierowane do
kanalu logicznego 123
}
```

Dekoder sygnałów czasu stacji DCF77



Rys. 8.1. Połączenia modułów dekodera: odbiornika DCF77, wyświetlacza i Arduino

W układzie zegara DCF77 wykorzystano ten sam typ wyświetlacza ciekłokrystalicznego co w opisanym powyżej układzie syntezera cyfrowego. Tranzystor Tr1 steruje diodą elektroluminescencyjną służącą jako wskaźnik odbieranych impuldsów. Sygnalizacja ta nie jest wprawdzie konieczna ale ułatwia znalezienie korzystnego miejsca dla odbiornika. Jako odbiornik wykorzystano moduł nr 611138 dostępny w firmie „Conrad". Moduł posiada dwa wyjścia impulsów – o polaryzacji dodatniej i ujemnej. W układzie zegara wykorzystano impulsy o polaryzacji dodatniej. Oba wyjścia są typu „otwarty kolektor" i dlatego też konieczne jest włączenie w obwód kolektora opornika polaryzującego – ma on wartość ok. 10 kΩ.
Program dekodujący jest napisany w języku Bascom i wymaga po skompilowaniu go zapisania w pamięci procesora poprzez złącze ICSP. Archiwum programu jest dostępne w internecie pod adresem *http://www.kh-gps.de/dcf77.zip*.
Sygnały czasu stacji DCF77 sa odbieralne w znacznej części Europy. Oficjalnie podawany jest zasięg ok. 2000 km. Sposób kodowania informacji podano w pierwszym tomie skryptu.



Rys. 8.2. Impulsy stacji DCF77. Na osi Y moc nadajnika

Kod źródłowy

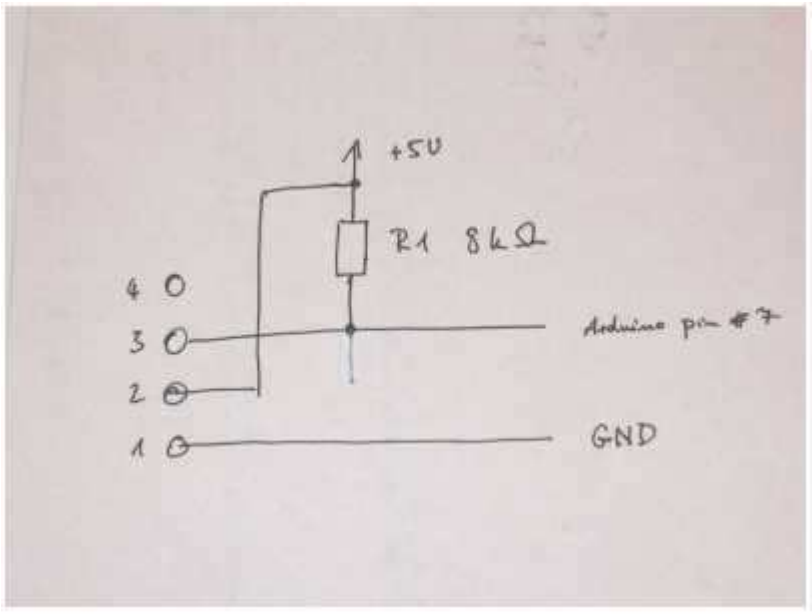
```
' Program BASCOM      dla ARDUINO
' autor Stefan Hoffmann 2009
' zegar sterowany przez stacje DCF77
' sygnal DCF pobierany z odbiornika
'
```

```
' Wejscie:  PortD.3 dla sygnalu z odbiornika DCF77 (polaryzacja dodatnia; zacisk odbiornika: 3)
' Wyjscie: PortD4 do PortD7; PortB.0 do PortB.1: wyswietlacz LCD
' Wyjscie: PortB.5 dla diody swiecacej LED sygnalizujacej odbior DCF
$regfile = "m328def.dat"
$crystal = 16000000
$hwstack = 40
$swstack = 32
$framesize = 60

Config Lcdpin = Pin, Db7 = Portd.7, Db6 = Portd.6, Db5 = Portd.5, Db4 = Portd.4, Rs = Portb.0, E =
Portb.1
Config Lcd = 16 * 2
Deflcdchar 0 , 32 , 14 , 17 , 32 , 14 , 17 , 32 , 4         ' Odbior prawidlowy
Cls
Cursor Off

Config Dcf77 = Pind.3, Inverted = 0, Timer = 1, Debug = 0, Check = 1, Gosub = Sectic
'DCF77 uzywa licznika 1 (Timer1)
Config Date = Dmy , Separator = .
Enable Interrupts

Portd.3 = 1                          'Oporniki podtrzymujace. Niektore odbiorniki nie wymagaja tego.

Config Portb.5 = Output                         'Dla sygnalizacji - LED
Signal_kontroll_led Alias Portb.5

Dim Neue_sekunde As Bit                          'sygnalizator
Dim Synchron As Bit                             '1: synchronizacja  0: zly odbior
Dim Tag_im_jahr As Byte                          'dzien roku 1..366
Dim Wochentag As Byte                            'dzien tygodnia 0..6
Dim Differenz_zu_utc As Byte                       '1: czas zimowy MEZ  2: czas letni MESZ

Lcd "    DCF77-Uhr"
Wait 2
'-----Glowny program-----
Do
Signal_kontroll_led = Dcf_status.0                 'Diagnoza: miga przy odbiorze

If Neue_sekunde = 1 Then
  Neue_sekunde = 0
  Gosub Anzeige
End If

'...pozostale rozkazy.

Loop
End
'-------------------------------------
Sectic:
Neue_sekunde = 1
Return

Anzeige:
If Dcf_status.7 = 0 Then                        'Przed nastawieniem zegara przez DCF
  Locate 1 , 1                                  '
```

```
  Lcd "Zeit empfangen"
  Locate 2 , 1
  Lcd Time$                                   '..zliczanie sekund.
Else                                          'Gdy zegar nastawiony.
  Gosub Zeit_anzeigen                         '..wyswietlanie czasu
  If _sec = 59 Then Synchron = Dcf_status.2   'sprawdzenie czy wszystko w komplecie?
End If
Return

Zeit_anzeigen:
  If _sec = 15 Or _sec = 30 Or _sec = 45 Or _
     _sec = 16 Or _sec = 31 Or _sec = 46 Then
    Cls
'   Dzien roku:
    Tag_im_jahr = Dayofyear()                 ' 0 = 1 stycznia
    Incr Tag_im_jahr                          ' 1 = 1 stycznia
    Lcd "Tag " ; Tag_im_jahr
'   MEZ/MESZ
    Differenz_zu_utc = Dcf77timezone()        ' 1: czas zimowy MEZ  2: czas letni MESZ
    Locate 2 , 1
    If Differenz_zu_utc = 1 Then
      Lcd "MEZ "                              'Czas zimowy
    Else
      Lcd "MESZ"                              'Czas letni
    End If
  Else
'   Dzien tygodnia:
    Wochentag = Dayofweek()                   '0=Po 1=Wt ... 6=Nd
    Locate 1 , 1
    Lcd " " ; Lookupstr(wochentag , Wochentage)
'   Datum:
    Lcd _day ; "." ; Lookupstr(_month , Monate) ; " 20" ; _year ; " "
'   Lcd Date$                                 'mozliwe takze w formacie dd.mm.rr

'   Czas:
    Locate 2 , 1 : Lcd "    " ; Time$

'   Odbior o.k. Symbol:
    Locate 2 , 16
    If Synchron = 1 Then
      Lcd Chr(0)                              'DCF w porzadku
    Else
      Lcd " "                                 'DCF nie w porzadku
    End If
  End If
Return

Wochentage:
Data "Po " , "Wt " , "Sr " , "Cz " , "Pt " , "So " , "Nd "

Monate:
Data "" , "Sty" , "Lut" , "Mar" , "Kw " , "Maj" , "Cze" , "Lip" , "sie" , "Wrz" , "Paz" , "Lis" , "Gru"
```

Biblioteka odbiorcza DCF77

Najważniejsze funkcje związane z odbiorem i dekodowaniem sygnałów czasu stacji DCF77 można przenieść do oddzielnej biblioteki w standardowym formacie Arduino, co zapewnia więcej wygody w ich wykorzystaniu. Sposób kompilacji i zintegrowania biblioteki ze środowiskiem programistycznym Arduino opisano w dodatkach w tomie pierwszym.



Rys. 9.1. Sposób podłączenia odbiornika DCF77 „Conrada”

Plik DCF77.h

```
/* Biblioteka Arduino DCF77 v0.1
 * Copyright (C) 2006 Mathias Dalheimer (md@gonium.net)
 *
 * Program dostepny na warunkach licencji GNU
. * moze być dowolnie rozpowszechniany i modyfikowany
*/

#ifndef DCF77_h
#define DCF77_h

/**
 * Wlaczanie lub wylaczanie diagnozy
 */
//#define DCF_DEBUG 1
/**
 * Liczba milisekund stanowiaca granice miedzy impulsem o dlugosci odpowiadajacej „0“ i „1“,
 * dla impulsow krotszych od niej (zbocze opadajace odebrane wczesniej) – wartosc „0“.
 */
#define DCF_split_millis 140
/**
 * Brak impulsu w 59 sekundzie, sygnalizuje poczatek minuty
*/
#define DCF_sync_millis 1200
/**
 * Struktura dla danych DCF
 */
```

```
struct DCF77Buffer {
        unsigned long long prefix:21  ;
        unsigned long long Min        :7   ;    // 7 bitow dla minut
        unsigned long long P1         :1   ;    // bit parzystosci minut
        unsigned long long Hour       :6   ;    // 6 bitow dla godzin
        unsigned long long P2         :1   ;    // bit parzystosci godzin
        unsigned long long Day        :6   ;    // 6 bitow dla dnia
        unsigned long long Weekday    :3   ;    // 3 bity dla dnia tygodnia
        unsigned long long Month      :5   ;    // 3 bity dla miesiaca
        unsigned long long Year       :8   ;    // 8 bitow dla roku ** 5 dla roku 2005**
        unsigned long long P3         :1   ;    // bit parzystosci P3
};

class DCF77 {
    private:
        unsigned char DCF77Pin;
        unsigned char blinkPin;
        unsigned char previousSignalState;
        int previousFlankTime;
        int bufferPosition;
        unsigned long long dcf_rx_buffer;
        void appendSignal(unsigned char);
        void addSecond();
        void finalizeBuffer(void);
    public:
        unsigned char ss;
        unsigned char mm;
        unsigned char hh;
        unsigned char day;
        unsigned char mon;
        unsigned int year;
     /**
      * Inicjalizacja biblioteki DCF77. Podanie numeru linii uzywanej jako wejscie
      * sygnalu DCF.
      */
        DCF77(unsigned char dcfPin);
        /**
         * Probkowanie sygnalu DCF powinno sie odbywac co 15 ms (wywolania funkcji co 15 ms).
         * Wynik: stan wejscia, 1: wysoki (HIGH), 0 niski (LOW).
         */
        int scanSignal(void);
        /**
         * wydawanie czasu szeregowo.
         */
        void serialDumpTime();
};

#endif
```

Plik DCF77.cpp

```
/* Biblioteka Arduino DCF77 v0.1
 * Copyright (C) 2006 Mathias Dalheimer (md@gonium.net)
 *
 * Program dostepny na warunkach licencji GNU
```

```
. * moze być dowolnie rozpowszechniany i modyfikowany
*/

#undef abs
#include "WConstants.h"
#include "HardwareSerial.h"
// wlaczenie naglowka biblioteki
#include "DCF77.h"

DCF77::DCF77(unsigned char dcfPin){                    // inicjalizacja biblioteki DCF77
      DCF77Pin = dcfPin;
      previousSignalState=0;
      previousFlankTime=0;
      bufferPosition=0;
      dcf_rx_buffer=0;
      ss=mm=hh=day=mon=year=0;
      Serial.begin(9600);
#ifdef DCF_DEBUG
      Serial.println("Initializing DCF77 lib");
      Serial.print("Using DCF77 pin #");
      Serial.println(DCF77Pin);
#endif
      pinMode(DCF77Pin, INPUT);
}

void DCF77::appendSignal(unsigned char signal) {          // zbieranie danych w buforze
#ifdef DCF_DEBUG
      Serial.print(", appending value ");
      Serial.print(signal);
      Serial.print(" at position ");
      Serial.println(bufferPosition);
#endif
      dcf_rx_buffer = dcf_rx_buffer | ((unsigned long long) signal << bufferPosition);
  bufferPosition++;
      if (bufferPosition > 59) {
            finalizeBuffer();
      }
}

void DCF77::finalizeBuffer(void) {     // zakonczenie odbioru danych z calej minuty i konwersja
#ifdef DCF_DEBUG
      Serial.println("Finalizing Buffer");
#endif
  if (bufferPosition == 59) {
      struct DCF77Buffer *rx_buffer;
            rx_buffer = (struct DCF77Buffer *)(unsigned long long)&dcf_rx_buffer;
            //przetworzenie odebranych danych z kodu BCD
            mm = rx_buffer->Min-((rx_buffer->Min/16)*6);
            hh = rx_buffer->Hour-((rx_buffer->Hour/16)*6);
            day= rx_buffer->Day-((rx_buffer->Day/16)*6);
            mon= rx_buffer->Month-((rx_buffer->Month/16)*6);
            year= 2000 + rx_buffer->Year-((rx_buffer->Year/16)*6);
      }
      // zerowanie
      ss = 0;
```

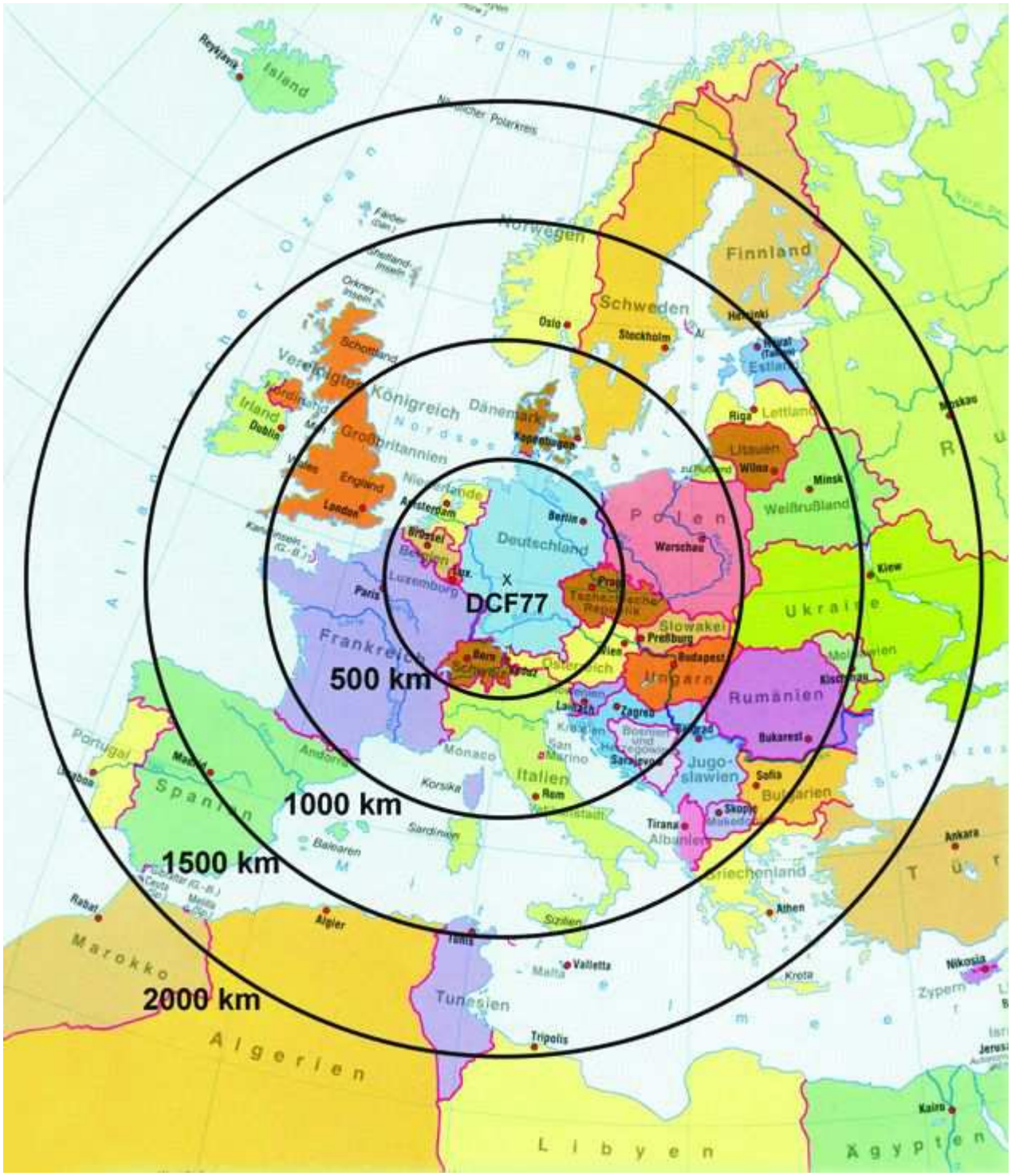
```
	bufferPosition = 0;
	dcf_rx_buffer=0;
}

void DCF77::serialDumpTime(void){
  Serial.print("Time: ");
  Serial.print(hh, DEC);
  Serial.print(":");
  Serial.print(mm, DEC);
  Serial.print(":");
  Serial.print(ss, DEC);
  Serial.print(" Date: ");
	Serial.print(day, DEC);
	Serial.print(".");
	Serial.print(mon, DEC);
	Serial.print(".");
	Serial.println(year, DEC);
}

int DCF77::scanSignal(void){
	unsigned char DCFsignal = digitalRead(DCF77Pin);
	if (DCFsignal != previousSignalState) {
	  if (DCFsignal == 1) {
			/* Wykryte zbocze narastajace, dodanie sekundy */
			addSecond();
#ifdef DCF_DEBUG
			serialDumpTime();
#endif
			int thisFlankTime=millis();
			if (thisFlankTime - previousFlankTime > DCF_sync_millis) {
#ifdef DCF_DEBUG
				Serial.println("####");
				Serial.println("#### Begin of new Minute!!!");
				Serial.println("####");
#endif
				finalizeBuffer();
			}
    previousFlankTime=thisFlankTime;
#ifdef DCF_DEBUG
    Serial.print(previousFlankTime);
        Serial.print(": DCF77 Signal detected, ");
#endif
	  } else {
			/* lub zbocze opadajace */
	   int difference=millis() - previousFlankTime;
#ifdef DCF_DEBUG
    Serial.print("duration: ");
        Serial.print(difference);
#endif
			if (difference < DCF_split_millis) {
				appendSignal(0);
			} else {
				appendSignal(1);
			}
   }
```

```
    previousSignalState = DCFsignal;
  }
      return DCFsignal;
}

/**
 * ############################################################
 * ###              Funkcje pomocnicze                      ###
 * ############################################################
 */
/**
 * Dodanie sekundy do zmiennych zegara „hh:mm:ss“.
 */
void DCF77::addSecond() {
      ss++;
      if (ss==60) {
            ss=0;
            mm++;
            if (mm==60) {
                  mm=0;
                  hh++;
                  if (hh==24)
                        hh=0;
            }
      }
}
```

Przykład wykorzystania biblioteki

```
#include <DCF77.h>

#define redPin 13

int DCF77Pin=7;
int blinkPin=13;
int seconds=0;
int previousSecond =0;
int minutes=0;
int hours=0;
DCF77 myDCF=DCF77(DCF77Pin);

void setup(void) {
  pinMode(blinkPin, OUTPUT);
}

void loop(void) {
  int DCFsignal = myDCF.scanSignal();
  if (DCFsignal) {
    digitalWrite(blinkPin, HIGH);
  } else {
    digitalWrite(blinkPin, LOW);
  }
  hours=myDCF.hh;
  minutes=myDCF.mm;
  seconds=myDCF.ss;
```

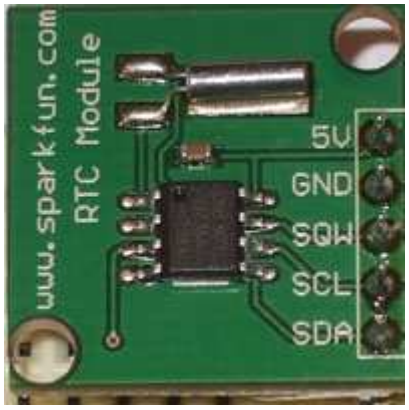
```
  if (seconds != previousSecond)
    myDCF.serialDumpTime();
  delay(20);
  previousSecond = seconds;
}
```



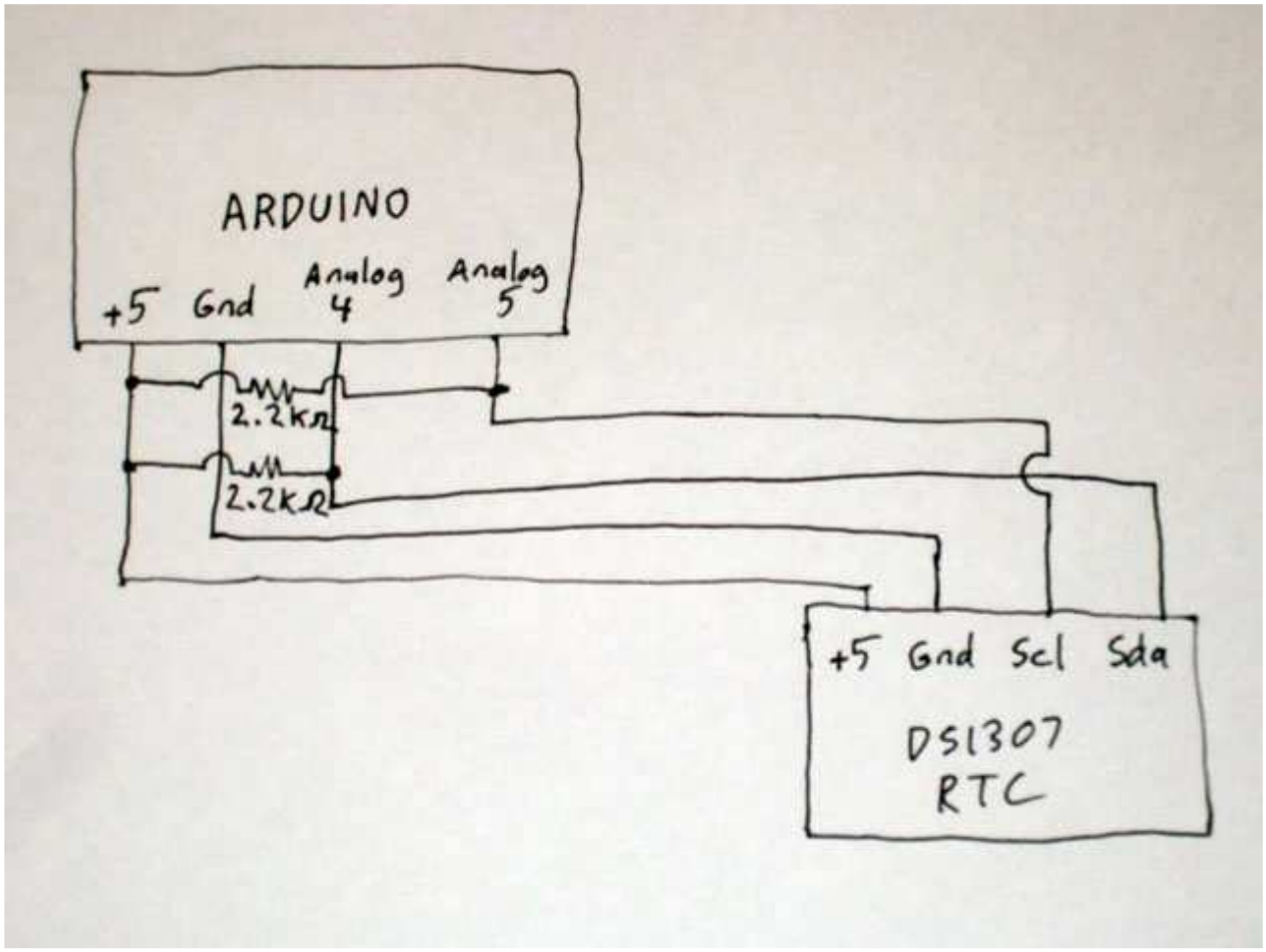
Rys. 9.2. Zasięg stacji DCF77

Podłączenie modułu zegarowego przez magistralę I2C

Moduł zegarowy typu DS1307 komunikuje się z Arduino za pomocą magistrali szeregowej I2C. Pozwala ona na podłączenie większej liczby urządzeń jednocześnie i pomimo, że szybkość transmisji jest niewysoka wystarcza ona do wielu zastosowań. Do komunikacji poprzez magistralę I2C konieczne jest użycie biblioteki „Wire". Wejście analogowe A4 jest w tym przypadku wykorzystywane do transmisji danych (SDA) a wejście A5 – do transmisji sygnału zegarowego (SCL). Oba przewody wymagają podłączenia oporników podtrzymujących do napięcia zasilania 5 V (rys. 10.2).



Fot. 10.1. Wygląd modułu DS1307



Rys. 10.2. Schemat podłączenia modułu zegarowego DS1307 do Arduino (magistrali I2C)

Czas jest zapisywany i odczytywany w formacie BCD.

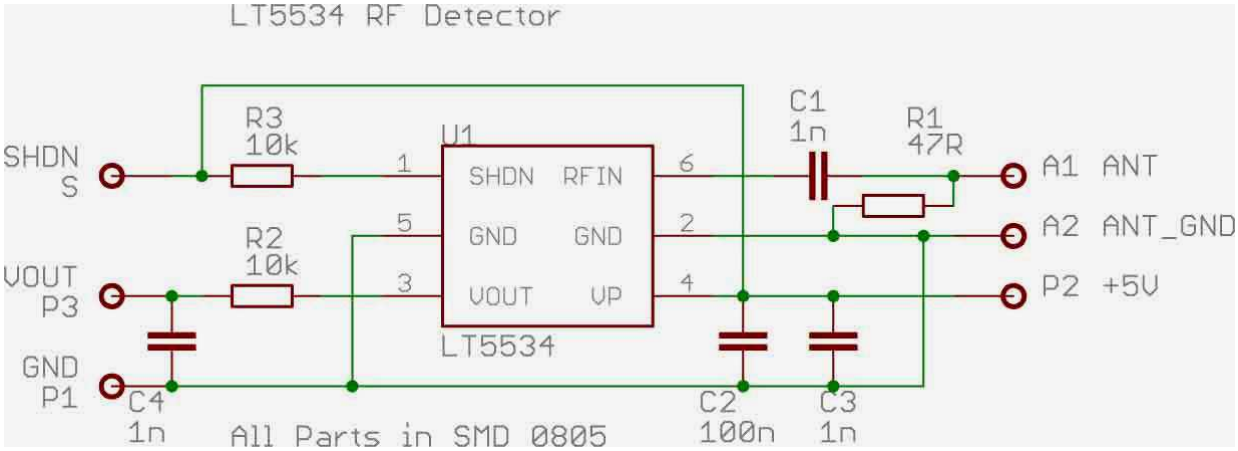
```
//
// Maurice Ribble
// 4-17-2008
// http://www.glacialwanderer.com/hobbyrobotics

// Probny program do wspolpracy zegara DS1307 z Arduino.
// DS1307 moze dostarczac danych dwojkowych lub BCD.
// Pelny opis kodu BCD i karta katalogowa DS1307 sa dostepne w Internecie

#include "Wire.h"
#define DS1307_I2C_ADDRESS 0x68

// Zamiana liczb dziesietnych na BCD
byte decToBcd(byte val)
{
  return ( (val/10*16) + (val%10) );
}

// Zamiana kodu BCD na liczby dziesietne
byte bcdToDec(byte val)
{
  return ( (val/16*10) + (val%16) );
}

// Zatrzymuje DS1307, ale licznik sekund jest zerowany
// Funkcja moze miec glownie znaczenie diagnostyczne
/*void stopDs1307()
{
  Wire.beginTransmission(DS1307_I2C_ADDRESS);
  Wire.send(0);
  Wire.send(0x80);
  Wire.endTransmission();
}*/

// 1) Nastawienie daty i czasu w ds1307
// 2) Wlaczenie zegara
// 3) Wybor trybu 24-godzinnego
// Pod warunkiem podania prawidlowych danych
void setDateDs1307(byte second,        // 0-59
                   byte minute,        // 0-59
                   byte hour,          // 1-23
                   byte dayOfWeek,     // 1-7
                   byte dayOfMonth,    // 1-28/29/30/31
                   byte month,         // 1-12
                   byte year)          // 0-99
{
   Wire.beginTransmission(DS1307_I2C_ADDRESS);
   Wire.send(0);
   Wire.send(decToBcd(second));    // 0 w bicie siodmym uruchamia zegar
   Wire.send(decToBcd(minute));
   Wire.send(decToBcd(hour));      // Dla trybu 12-godzinnego + „am/pm“ ustawic bit 6
                                   // (konieczna tez zmiana w funkcji readDateDs1307())
   Wire.send(decToBcd(dayOfWeek));
```

```
   Wire.send(decToBcd(dayOfMonth));
   Wire.send(decToBcd(month));
   Wire.send(decToBcd(year));
   Wire.endTransmission();
}

// Odczyt daty i czasu z ds1307
void getDateDs1307(byte *second,
          byte *minute,
          byte *hour,
          byte *dayOfWeek,
          byte *dayOfMonth,
          byte *month,
          byte *year)
{
  // Zerowanie wskaznika rejestrow
  Wire.beginTransmission(DS1307_I2C_ADDRESS);
  Wire.send(0);
  Wire.endTransmission();

  Wire.requestFrom(DS1307_I2C_ADDRESS, 7);

  // Niektore z bitow musza byc ignorowane poniewaz sluza jako sterujace
  *second     = bcdToDec(Wire.receive() & 0x7f);
  *minute     = bcdToDec(Wire.receive());
  *hour       = bcdToDec(Wire.receive() & 0x3f);  // Konieczna zmiana dla trybu 12-godzinnego +
„am/pm“
  *dayOfWeek  = bcdToDec(Wire.receive());
  *dayOfMonth = bcdToDec(Wire.receive());
  *month      = bcdToDec(Wire.receive());
  *year       = bcdToDec(Wire.receive());
}

void setup()
{
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
  Wire.begin();
  Serial.begin(9600);

  // Podac wlasciwy czas do nastawienia zegara.
  // Funkcja potrzebna zasadniczo tylko raz.
  // potem mozna usunac z programu wywolanie setDateDs1307().
  second = 45;
  minute = 3;
  hour = 7;
  dayOfWeek = 5;
  dayOfMonth = 17;
  month = 4;
  year = 8;
  setDateDs1307(second, minute, hour, dayOfWeek, dayOfMonth, month, year);
}

void loop()
{
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
```

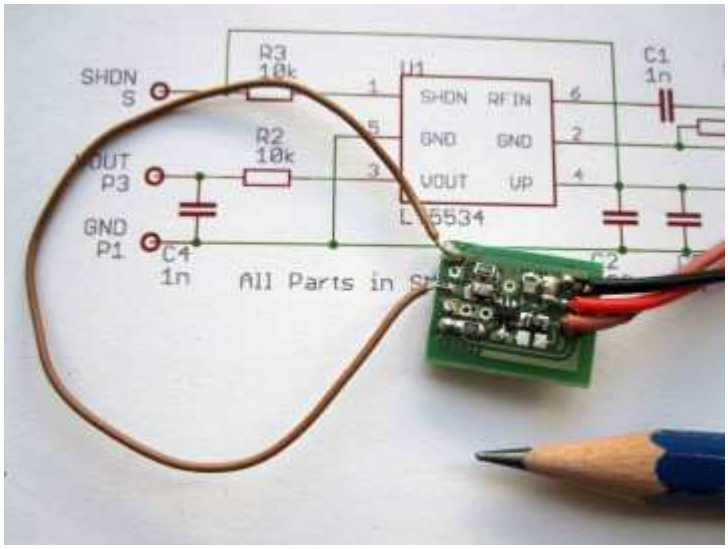
```
  // odczyt czasu i jego wydanie przez zlacze szeregowe
  // do wyswietlenia w oknie monitora szeregowego srodowiska Arduino
  // lub do innych celow
  getDateDs1307(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);
  Serial.print(hour, DEC);
  Serial.print(":");
  Serial.print(minute, DEC);
  Serial.print(":");
  Serial.print(second, DEC);
  Serial.print("  ");
  Serial.print(month, DEC);
  Serial.print("/");
  Serial.print(dayOfMonth, DEC);
  Serial.print("/");
  Serial.print(year, DEC);
  Serial.print("  Day_of_week:");
  Serial.println(dayOfWeek, DEC);

  delay(1000);
}
```

Czujnik natężenia polaw.cz.

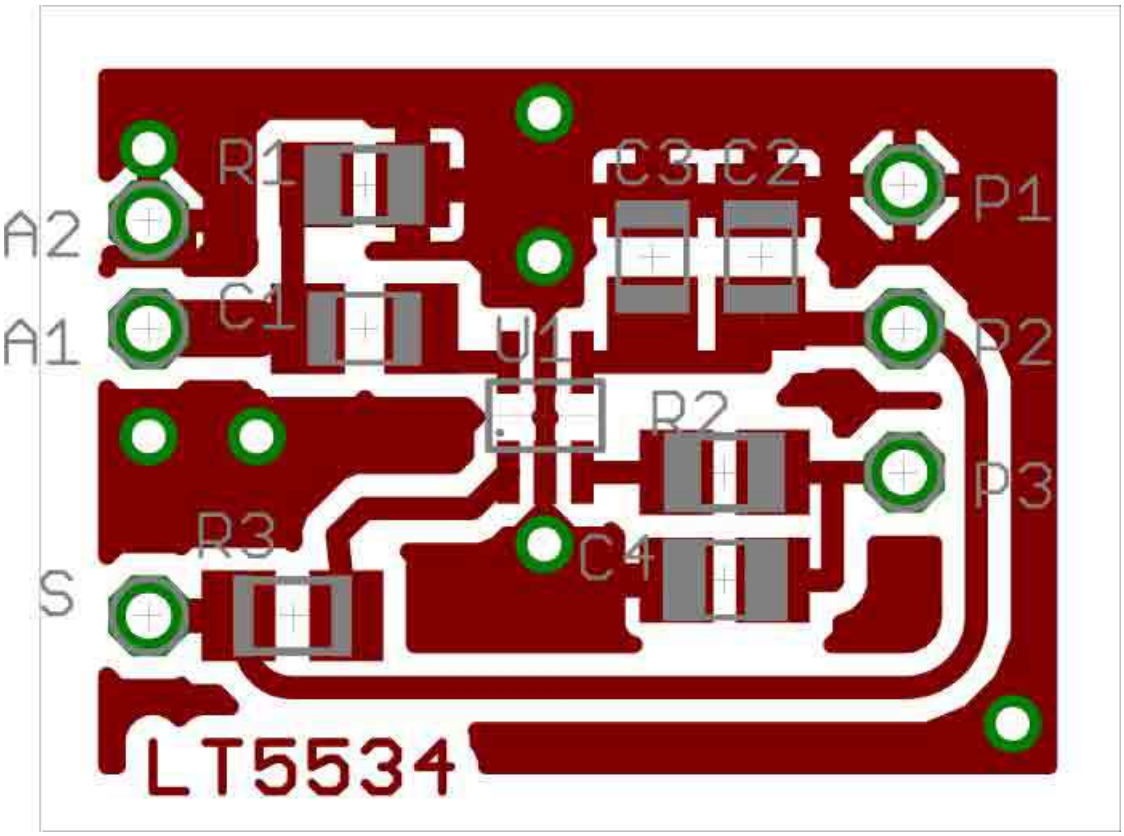
Czujnik jest przeznaczony do wykrywania elektromagnetycznych pól wielkiej częstotliwości w zakresie od 50 MHz do 3 GHz. Do wejścia scalonego detektora w.cz. podłączona jest prosta antena pętlowa. Rozwiązanie to pozwala na wykrywanie urządzeń Bluetooth, sieci WLAN, telewizji, telefonów komórkowych itd. Wyjście detektora jest podłączone do wejścia analogowego Arduino. Napięcie zasilania czujnika pochodzi również z płytki Arduino. Możliwe jet też podłączenie słuchawek lub głośnika do wyjścia detektora (ewentualnie przez wzmacniacz m.cz.) i rozpoznawanie na słuch sygnałów (w czujniku następuje detekcja amplitudy).
Dobre wyniki uzyskuje się dla następujących długości pętli:
12,5 cm – zakres 2,4 GHz, WLAN, Bluetooth.
16 cm – zakres 1800 MHz, telefonia komórkowa GSM,
33 cm – zakres 900 MHz, telefonia komórkowa,
60 cm – 500 MHz, telewizja,pasmo IV,
150 cm – 100 MHz, radiofonia UKF.



Rys. 11.1. Schemat szerokopasmowego detektora w.cz. na obwodzie scalonym LT5534



Rys. 11.2. Konstrukcja czujnika natężenia pola wcz.



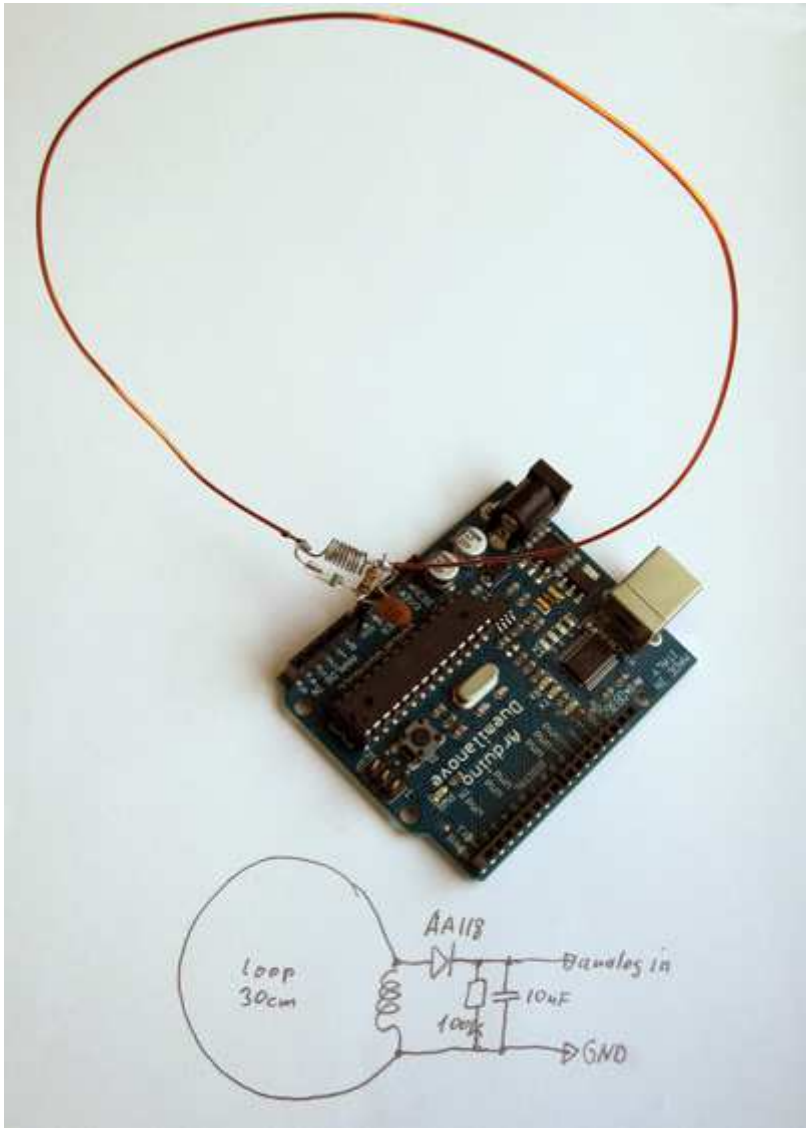
Rys. 11.3. Płytka drukowana detektora

Układ detektora można oprzeć także na obwodzie AD8307.
Do wykrywania bliskich i raczej silniejszych źródeł sygnałów (np. telefonów GSM) wystarcza zwykły detektor diodowy (rys. 11.4). Ze względu wartość napięcia progowego zalecane jest użycie w nim małosygnałowych diod germanowych np. AA118 lub innego dowolnego typu. Pętla dla wykrywania telefonów komórkowych ma długość ok. 33 cm.
Wyjście detektora jest podłączone do jednego z wejść analogowych Arduino. W przeciwieństwie do poprzedniego rozwiązania nie wymaga on zasilania.

Kod źródłowy

```
/* Probny program dla czujnika natezenia pola LT5534 RSSI
 * Lab3 2010
 * Kunsthochschule fuer Medien Koeln
 * Academy of Media Arts Cologne
 * http://interface.khm.de
 */

int analogIn = 0;
int analogValue = 0;
int pinLed = 13;
int cnt;

void setup(){
  Serial.begin(57600);                    // inicjalizacja zlacza szeregowego
  pinMode(pinLed,OUTPUT);                 // programowanie wyjscia

}
```

```
void loop(){

  analogValue=0;
  for (cnt=0;cnt< 100;cnt++) {
    digitalWrite(pinLed,1);
    analogIn=analogRead(1);
    digitalWrite(pinLed,0);
    if (analogIn > analogValue) analogValue=analogIn;
    delayMicroseconds(100);

  }
  Serial.println(analogValue);          // wyswietlenie wyniku w oknie monitora szeregowego Arduino
  delay(100);
}
```

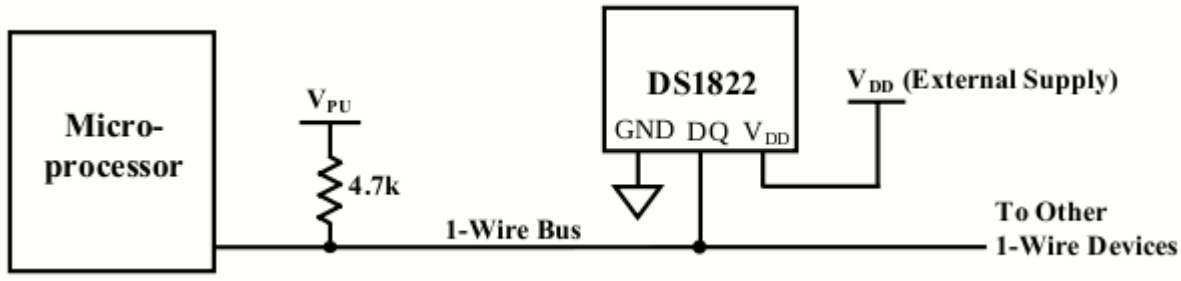


Rys. 11.4. Prosty detektor diodowy

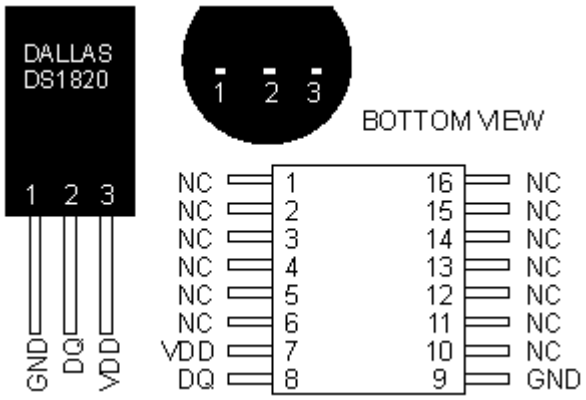
Pomiar temperatury z DS1820/1822

Fot. 12.1. Wygląd termometru z Arduino MEGA i wyświetlaczem ciekłokrystalicznym

Przykładowy program termometru pracuje na Arduino Duemilanove, UNO, MEGA i innych modelach Arduino (konieczne jest użycie biblioteki „OneWire"). Zastosowano w nim wyświetlacz ciekłokrystaliczny 2 x 16 znaków i czujnik temperatury DS1820 lub DS1822 firmy Dallas. Czujnik jest połączony z Arduino za pomocą magistrali"1-Wire".
Zakres mierzonych temperatur wynosi dla dla DS18S20 i DS18B20 -55 do +125°C z dokładnością +/- 0,5°C i rozdzielczością 0,5 °C a dla DS1822 – -55 do +125°C z dokładnością +/-2°C i rozdzielczością 0,06°C. Adres na magistrali wynosi dla DS1820 – 0x10 a dla DS1822 – 0x22.



Rys. 12.2. Schemat podłączenia czujnika przez magistralę „1-Wire". Jest on podłączony do kontaktu 10 Arduino.



Rys. 12.3. Wyprowadzenia czujnika DS1820 dla różnych typów obudów

Kod źródłowy wersji podstawowej

```
/*
Termometr z czujnikami Dallas 18xx i modulem wyswietlacza LCD

03.09.2011 Jürgen Mayer, DL8MA, Grossheppach

Program dostępny bezpłatnie.
*/

#include <OneWire.h>
#include <LCD4Bit_mod.h>

LCD4Bit_mod lcd = LCD4Bit_mod(2);

OneWire ds(10); // Podlaczenie czujnika temperatury Dallas DS18xx do przewodu 10
                // utworzenie i inicjalizacja obiektu ds
char buffer[20] = ""; // Bufor dla konwersji danych

void setup(void) {

  byte addr[8];
  Serial.begin(9600);         // Inicjalizacja zlacza szeregowego
  lcd.init();                 // Inicjalizacja wyswietlacza
l  cd.clear();

  if ( !ds.search(addr)) {    // Sprawdzenie czy t ermometr podlaczony
    ds.reset_search();
    lcd.printIn( "Sensor nicht" ); // Jesli nie – meldunek i wyjscie z programu
    lcd.cursorTo(2, 0);
    lcd.printIn( "gefunden :-(" );
    return;
  }

  lcd.printIn("Thermometer"); // Wyswietlenie typu czujnika

  if ( addr[0] == 0x10 or addr[0] == 0x28) // Rozroznienie w oparciu o adres z magistrali „1-Wire“
    lcd.printIn(" 1820" );
  if ( addr[0] == 0x22)
    lcd.printIn(" 1822" );
}
```

```
void loop(void) {
  byte i;
  byte present = 0;
  byte data[12];
  byte addr[8];
  int HighByte, LowByte, TReading, SignBit, Tc_100, Whole, Fract;

  if ( !ds.search(addr)) {
    ds.reset_search();
    return;
  }

  ds.reset();
  ds.select(addr);
  ds.write(0x44,1); // Wlaczenie czujnika i konwersji

  delay( 750 ); // Oczekiwanie na wynik

  present = ds.reset();
  ds.select(addr);
  ds.write(0xBE); // Rozkaz wydania wyniku

  for ( i = 0; i < 9; i++) { // Wczytanie 9 bajtow
    data[i] = ds.read();
  }

  LowByte = data[0];
  HighByte = data[1];
  TReading = (HighByte << 8) + LowByte;
  SignBit = TReading & 0x8000;
  if (SignBit)
  {
    TReading = (TReading ^ 0xffff) + 1;
  }

  if ( addr[0] == 0x10 or addr[0] == 0x28) // Obliczenia zalezne od typu czujnika
    Tc_100 = (TReading*100/2);
  if ( addr[0] == 0x22)
    Tc_100 = (6 * TReading) + TReading / 4;

  Whole = Tc_100 / 100; // oddzielenie od siebie czesci calkowitej i ulamkowej
  Fract = Tc_100 % 100;

  /* Wydanie przez zlacze szeregowe */

  if (SignBit) // sprawdzanie wartosci ujemnych
  {
    Serial.print("-");
  }
  Serial.print(Whole);
  Serial.print(".");
  if (Fract < 10)
  {
    Serial.print("0");
```

```
  }
  Serial.print(Fract);
  Serial.println();

/* Wyswietlenie na wyswietlaczu */

  lcd.cursorTo(2, 0);
  if (SignBit) // sprawdzanie wartosci ujemnych
  {
    lcd.printIn("-");
  }
  lcd.printIn( itoa( Whole, buffer, 10 ) );
  lcd.printIn(".");
  if (Fract < 10)
  {
    lcd.printIn("0");
  }
  lcd.printIn( itoa( Fract, buffer, 10 ) );
  lcd.printIn( " Grad" );
}
```

Kod źródłowy z użyciem biblioteki „DallasTemperature”

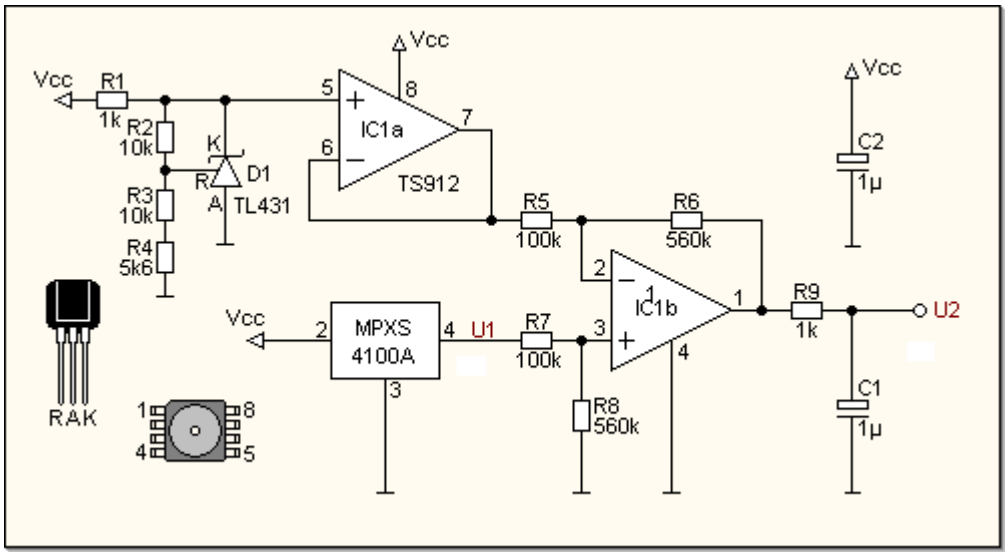
Biblioteka „DallasTemperature” do obsługi czujników temperatury jest dostępna w Internecie pod adresem *http://milesburton.com/Dallas_Temperature_Control_Library.*

```
/*
Termometr z czujnikiem Dallas 18xx i modulem LCD / Wersja z uzyciem biblioteki „Dallas
Temperatur Control Library“

04.09.2011 Jürgen Mayer, DL8MA, Grossheppach

Program udostepniony bezplatnie.
*/

#include <OneWire.h>
#include <DallasTemperature.h>
#include <LCD4Bit_mod.h>

#define ONE_WIRE_BUS 10

char buffer[20] = ""; // Bufor dla konwersji danych

LCD4Bit_mod lcd = LCD4Bit_mod(2);

OneWire oneWire(ONE_WIRE_BUS); // Czujnik DS18xx podlaczony przez magistrale do linii 10
                               // utworzenie i nicjalizacja obiektu
DallasTemperature sensors(&oneWire);

void setup(void) {
  byte addr[8];
  Serial.begin(9600);    // Inicjalizacja zlacza szeregowego
  lcd.init();            // inicjalizacja wyswietlacza
  lcd.clear();
```

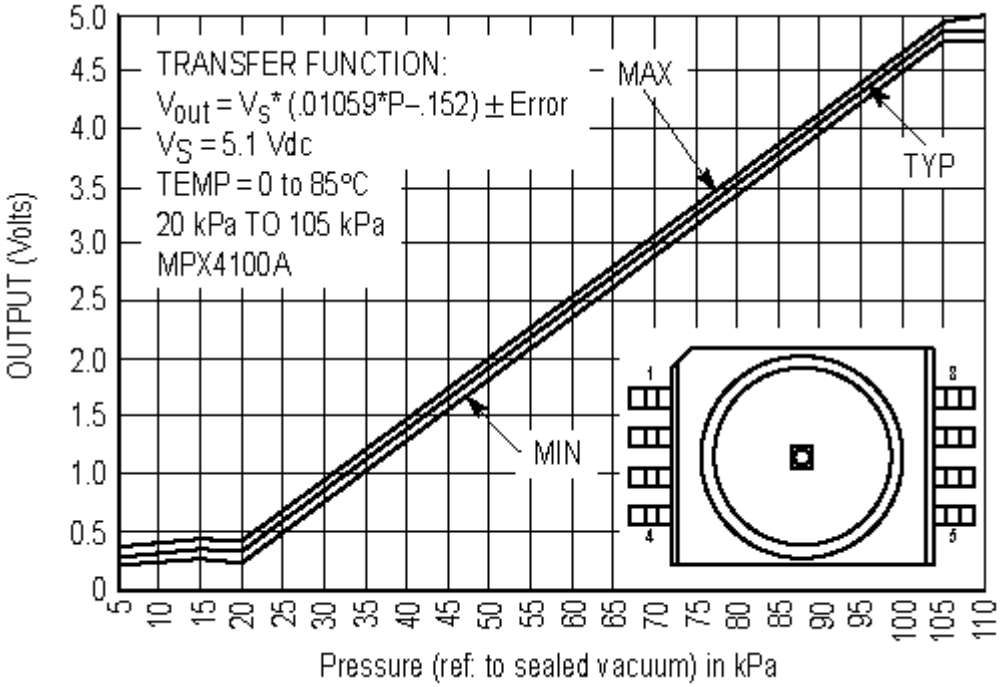
```
  sensors.begin();          // Inicjalizacja czujnika
  sensors.setResolution( 12 );// Wybor rozdzielczosci

  lcd.printIn("Thermometer"); // wyswietlenie informacji na wyswietlaczu
}

void loop(void) {
  float temperatur = 0;
  char stringbuffer[ 17];

  sensors.requestTemperatures(); // odpytanie czujnika i wyswietlenie wyniku pomiaru
  Serial.println(sensors.getTempCByIndex(0));
  lcd.cursorTo(2, 0);
  temperatur = sensors.getTempCByIndex(0);
  lcd.printIn( dtostrf( temperatur, 5, 2, stringbuffer ) );
  lcd.printIn( " Grad" );
}
```

Komunikaty telemetryczne DPRS

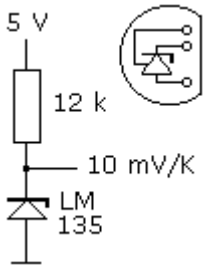
Program generuje komunikaty telemetryczne DPRS w formacie GPS-A zawierajace wyniki pomiaru ciśnienia atmosferycznego, temperatury i napięcia (np. napięcia zasilania). Do pomiaru ciśnienia atmosferycznego wykorzystano czujnik MPXS4100A a do pomiaru temperatury – czujnik LM135. Oba czujniki dostarczają napięć analogowych zależnych liniowo od wartości mierzonej. Charakterystyka wyjściowa czujnika MPXS4100A i wzór będący podstawą do przeliczenia napięcia na ciśnienie są przedstawione na rys. 13.2.
Utworzony w programie komunikat zawierający na początku sumę kontrolną CRC jest przekazywany przez złącze szeregowe do radiostacji D-STAR. W obecnej wersji programu przewidziano współpracę z radiostacjami typów IC92D, ID31/ID51 i IC2820. Z punktu widzenia programu główną różnicę między nimi stanowi szybkość transmisji w złączu szeregowym dlatego też praktycznie może on współpracować ze wszystkimi dostępnymi obecnie radiostacjami D-Starowymi. Należy tylko wybrać pasującą szybkość transmisji za pomocą polecenia *#define* i wyłączyć pozostałe za pomocą poleceń *#undef* oraz odpowiednio zmodyfikować trasę pakietów tak, aby występowało w niej standardowe oznaczenie typu używanej radiostacji. Znormalizowane poziomy napięć na złączu RS232 zapewnia układ scalony MAX232 podłączony do logicznych kontaktów 0 i 1 płytki Arduino.
Ze względu na rozrzut parametrów czujników należy przeprowadzić najpierw co najmniej dwa pomiary (różnych wartości – a więc w różne dni lub w różnych porach dnia o ile wystąpiła znacząca zmiana ciśnienia) ciśnienia, porównać z podawanym przez IMGW i w razie potrzeby odpowiednio skorygować współczynniki we wzorach (a i b we wzorze na ciśnienie). W obliczeniach ciśnienia atmosferycznego należy także uwzględnić własną wysokość n.p.m. Niezależnie od tego należy przeprowadzić kalibrację czujnika temperatury (rys.13.4).
Wyniki pomiarów są uśredniane w programie przy użyciu bieżącej średniej o długości 4 punktów. Jej długość podana jest w deklaracjach i można ją łatwo dopasować do potrzeb.
Czujnik ciśnienia jest podłączony do wejścia analogowego A0 w układzie podanym na rys. 13.1, temperatury – do wejścia analogowego A1 w układzie podanym na rys. 13.4b, a wejście A2 służy do pomiaru napięcia. Dla napięć wyższych niż 5 V należy włączyć odpowiedni dzielnik oporowy i uwzględnić stosunek podziału w programie. Na wszelki wypadek dobrze jest też zabezpieczyć wejście przed przepięciami za pomocą diody Zenera 5,1 V lub zwykłych diod połączonych z przewodami zasilania 5 V i masy.



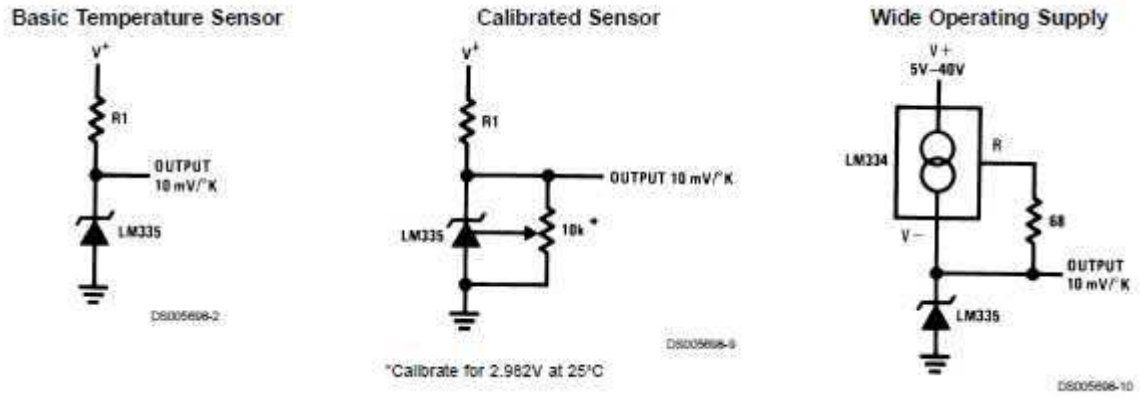
Rys. 13.1. Sposób podłączenia czujnika ciśnienia. Dla zwiększenia rozdzielczości od napięcia wyjściowego czujnika odejmowane jest napięcie ok. 4 V a mierzona jest jedynie różnica w stosunku do niego. Schemat pochodzi z witryny internetowej:
http://www.umnicom.de/Elektronik/Projekte/Wetterstation/Sensoren/Luftdruck/Luftdruck.htm



Rys. 13.2. Charakterystyka wyjściowa czujnika MPSX4100A. Źródło: karta katalogowa czujnika.



Rys. 13.3. Standardowy układ czujnika temperatury



Rys. 13.4. Możliwe sposoby podłączenia czujnika LM135: a) standardowy bez kalibracji, b) z możliwością kalibracji potencjometrycznej, c) zasilanego przez źródło prądowe z napięcia 5 – 40 V. Kalibrację należy przeprowadzić tak aby w temperaturze 25 °C otrzymać napięcie wyjściowe 2,982 V. Nachylenie charakterystyki czujnika wynosi 10 mV/°K. Źródło: karta katalogowa czujnika.

Kod źródłowy

```
//----------------------------------------
// Radiolatarnia telemetryczna DPRS
// Krzysztof Dabrowski OE1KDA
// 01.02.2013
//----------------------------------------
#include <avr/pgmspace.h>
// wybrac tylko jeden z typow radiostacji
#define IC92D
#undef  IC2820
#undef  ID31
// stale programowe
#define CISN 1
#define TEMP 2
#define NAP 3
#define LICZNIK 60

// do celow diagnostycznych
#define TEST
//#undefine TEST

// prototypy funkcji
void obliczSrednia(int, int);
void zlozDPRS(void);
int sumaCCITTCRC(int, int);
double obliczCisnienie(void);
double obliczTemperature(void);
double obliczNapiecie(void);
void zlozTelemetrie(void);

const int dlugoscSrednej = 4;     // liczba punktow biezacej sredniej
int WeTypTRX = 7;                 // Decyduje o szybkosci transmisji w zlaczu szeregowym
int cisnienie = 0;                // Usrednione wartosci z przetwornika a/c
int temperatura = 0;
int napiecie = 0;
int bufor;                        // Zmierzona wartosc z przetwornika a/c
int radiostacja = 0;
int CRC;
int licznik_cykli;
String CRC_blok;
String telemetria;                // Komunikat telemetryczny
String dprs = String();           // Calkowity komunikat
                                  // Dane dla stacji OE1KDA. Zastapic wlasnymi
/*PROGMEM*/ String dlugosc = "01621.11E";
/*PROGMEM*/ String szerokosc = "4809.21N";
/*PROGMEM*/ String tabela1 = String('/');         // tabele symboli APRS
/*PROGMEM*/ String tabela2 = String('\\');
/*PROGMEM*/ String stacja_UKF = String('-');
/*PROGMEM*/ String trasa92 = "API92,DSTAR*";
/*PROGMEM*/ String trasa2820 = "API282,DSTAR*";
/*PROGMEM*/ String trasa31 = "API31,DSTAR*";
/*PROGMEM*/ String znak = "OE1KDA";
// dane z witryny Umni
// stale bez korekty wysokosci
```

```
// const double a = 16.963972;
// const double b = 919.0
// korekta wysokosci dla a
// ah = a * exp(h/7960) h – m, wzor uwzgledniajcy wplyw wysokosci nad poziomem morza na cisnienie
// stale z korekta wysokosci dla 59 m
//const double a = 17.078308;
//const double b = 925.244860;
// stale obliczone dla wysokosci 180 m
// z uwzglednieniem stanu w konstrukcji OE1KDA
const double a = 17.351950;
const double b = 977.761309;
const double uzas = 5.00;
String trasa;

void setup()
{
  pinMode(WeTypTRX, INPUT);

  // wybor parametrow w zaleznosci od typu radiostacji
# ifdef IC92D
  Serial.begin(38400);
  trasa = trasa92;
# endif
# ifdef IC2820
  Serial.begin(9600);
  trasa = trasa2820;
# endif
# ifdef ID31
  Serial.begin(9600);
  trasa = trasa31;
# endif

# ifdef TEST
  Serial.println(znak+'>'+trasa);
# endif

  // Inicjalizacja sredniej. Wprowadzane pierwsze odczytane wartosci dla kazdego z kanalow.
 // przyspiesza osiagniecie stanu rzeczywistego
  delay(1000);
  bufor = analogRead(A0);
  cisnienie = dlugoscSrednej *bufor;
  delay(1000);
  bufor = analogRead(A1);
  temperatura = dlugoscSrednej * bufor;
  delay(1000);
  bufor = analogRead(A2);
  napiecie = dlugoscSrednej * bufor;
  licznik_cykli = LICZNIK;
}
```

```
//---------------------------------------
void loop()
{
 licznik_cykli--;
 bufor = analogRead(A0);                    // odczyt wartosci pomiarowych z kazdego z kanalow
# ifdef TEST
 Serial.println(bufor);
# endif
 obliczSrednia(CISN, bufor);                // i uwzglednienie ich w sredniej
# ifdef TEST
 Serial.println(cisnienie);
# endif
 delay(5000);
 bufor = analogRead(A1);
# ifdef TEST
 Serial.println(bufor);
# endif
 obliczSrednia(TEMP, bufor);
# ifdef TEST
 Serial.println(temperatura);
# endif
 delay(5000);
 bufor = analogRead(A2);
# ifdef TEST
 Serial.println(bufor);
# endif
 obliczSrednia(NAP, bufor);
# ifdef TEST
 Serial.println(napiecie);
# endif
 delay(5000);
// nadawanie komunikatu w kazdym cyklu
# ifdef TEST
 zlozTelemetrie();                          // zlozenie czesci telemetrycznej komunikatu
 zlozDPRS();                                // zlozenie calego komunikatu DPRS
 Serial.print(CRC_blok);                    // wyswietlenie sumy kontrolnej CRC dla jej sprawdzenia
 Serial.println(dprs);
# endif
// nadawanie komunikatow co LICZNIK/3 minut
 if (licznik_cykli == 0)
  {
   zlozTelemetrie();                        // zlozenie czesci telemetrycznej komunikatu
   zlozDPRS();                              // zlozenie calego komunikatu DPRS
   Serial.print(CRC_blok);                  // wyswietlenie sumy kontrolnej CRC dla jej sprawdzenia
   Serial.println(dprs);
   licznik_cykli = LICZNIK;
  }
 delay(5000);
}
//---------------------------------------
void obliczSrednia(int kanal, int nowa)     // obliczanie sredniej dla podanego kanalu
{
 switch(kanal)
 {
  case CISN:
```

```
      cisnienie = cisnienie - cisnienie / dlugoscSrednej + nowa;
      break;
    case TEMP:
      temperatura = temperatura - temperatura / dlugoscSrednej + nowa;
      break;
    case NAP:
      napiecie = napiecie - napiecie / dlugoscSrednej + nowa;
      break;
   }
}
//----------------------------------------
void zlozDPRS()                                 // zlozenie calego komunikatu DPRS
{
  dprs = String('=');
  dprs += String(znak);
  dprs += '>';
  dprs += trasa;
  dprs += ':';
  dprs += szerokosc;
  dprs += tabela1;
  dprs += dlugosc;
  dprs += stacja_UKF;
  dprs += telemetria;
  CRC = sumaCCITTCRC(0, (int)dprs.length());                  // obliczenie i dodanie na poczatku
  CRC_blok = String("$$CRC") + String(CRC, HEX) + String(','); // sumy kontrolnej CRC
}
//----------------------------------------
//int sumaCCITTCRC(char *buffer, int startpos, int dlugosc)
int sumaCCITTCRC(int startpos, int dlugosc)     // obliczenie sumy kontrolnej dla danych w buforze
{
  unsigned int icomcrc = 0xffff;
  for (int j = startpos; j < startpos+dlugosc; j++)
  {
    unsigned int ch = dprs.charAt(j) & 0xff;// buffer[j] & 0xff;
    for (int i = 0; i < 8; i++)
    {
      boolean xorflag = (((icomcrc ^ ch) & 0x01) == 0x01);
      icomcrc >>= 1;
      if (xorflag)
        icomcrc ^= 0x8408;
      ch >>= 1;
    }
  }
  return (~icomcrc) & 0xffff;
}
//----------------------------------------
double obliczCisnienie()                        // obliczanie cisnienia atmosferycznego
// Wzor na obliczanie z katalogu
// Vwy = Vs (0,01059 P - 0,152) [+/- poprawka]
// P = U * a + b
// Dla ukladu z odejmowaniem 4,096 V i wzmocnieniem 5,6 x
// a = 17,077308  b = 925,224860
{
  double P;
  P = ((double)cisnienie * uzas);
```

```
  P = P/1023.0;
  P = P/(double)dlugoscSrednej;
  P = P * a + b;
# ifdef TEST
  Serial.println(P);
# endif
  return P;
}
//----------------------------------------
double obliczTemperature()                          // obliczanie temperatury
{
  double T;
  T = (double)temperatura * uzas;
  T = T/1.023;
  T = T/(double)dlugoscSrednej;
  T = T/10 - 273;
  return T;
}
//----------------------------------------
double obliczNapiecie()                             // przeliczanie odczytanej wartosci na napiecie
{
  double U;
  U = (double)napiecie * uzas;
  U = U/1023.0;
  U = U/double(dlugoscSrednej);
  return U;

}
//----------------------------------------
void zlozTelemetrie()                               // zlozenie czesci telemetrycznej komunikatu
{ double a;
  int b;
  telemetria = String("P = ");
  a = obliczCisnienie();
  b = int(a * 10 + 0.5);
  telemetria += String(b/10) + String(',') + String(b%10);
  telemetria += String(" T = ");
  a = obliczTemperature();
  b= int(a + 0.5);
  telemetria += String(b);
  telemetria += String(" U = ");
  a = obliczNapiecie();
  b = int(a * 10 + 0.5);
  telemetria += String(b/10) + String(',') + String(b%10);
}
```

W serii „Biblioteka polskiego krótkofalowca” dotychczas ukazały się:

Nr 1 – „Poradnik D-STAR”
Nr 2 – „Instrukcja do programu D-RATS”
Nr 3 – „Technika słabych sygnałów” Tom 1
Nr 4 – „Technika słabych sygnałów” Tom 2
Nr 5 – „Łączności cyfrowe na falach krótkich” Tom 1
Nr 6 – „Łączności cyfrowe na falach krótkich” Tom 2
Nr 7 – „Packet radio”
Nr 8 – „APRS i D-PRS”
Nr 9 – „Poczta elektroniczna na falach krótkich” Tom 1
Nr 10 – „Poczta elektroniczna na falach krótkich” Tom 2
Nr 11 – „Słownik niemiecko-polski i angielsko-polski” Tom 1
Nr 12 – „Radiostacje i odbiorniki z cyfrową obróbką sygnałów” Tom 1
Nr 13 – „Radiostacje i odbiorniki z cyfrową obróbką sygnałów” Tom 2
Nr 14 – „Amatorska radioastronomia”
Nr 15 – „Transmisja danych w systemie D-STAR”
Nr 16 – „Amatorska radiometeorologia”
Nr 17 – „Radiolatarnie małej mocy”
Nr 18 – „Łączności na falach długich”
Nr 19 – „Poradnik Echolinku”
Nr 20 – „Arduino w krótkofalarstwie” Tom 1
Nr 21 – „Arduino w krótkofalarstwie” Tom 2