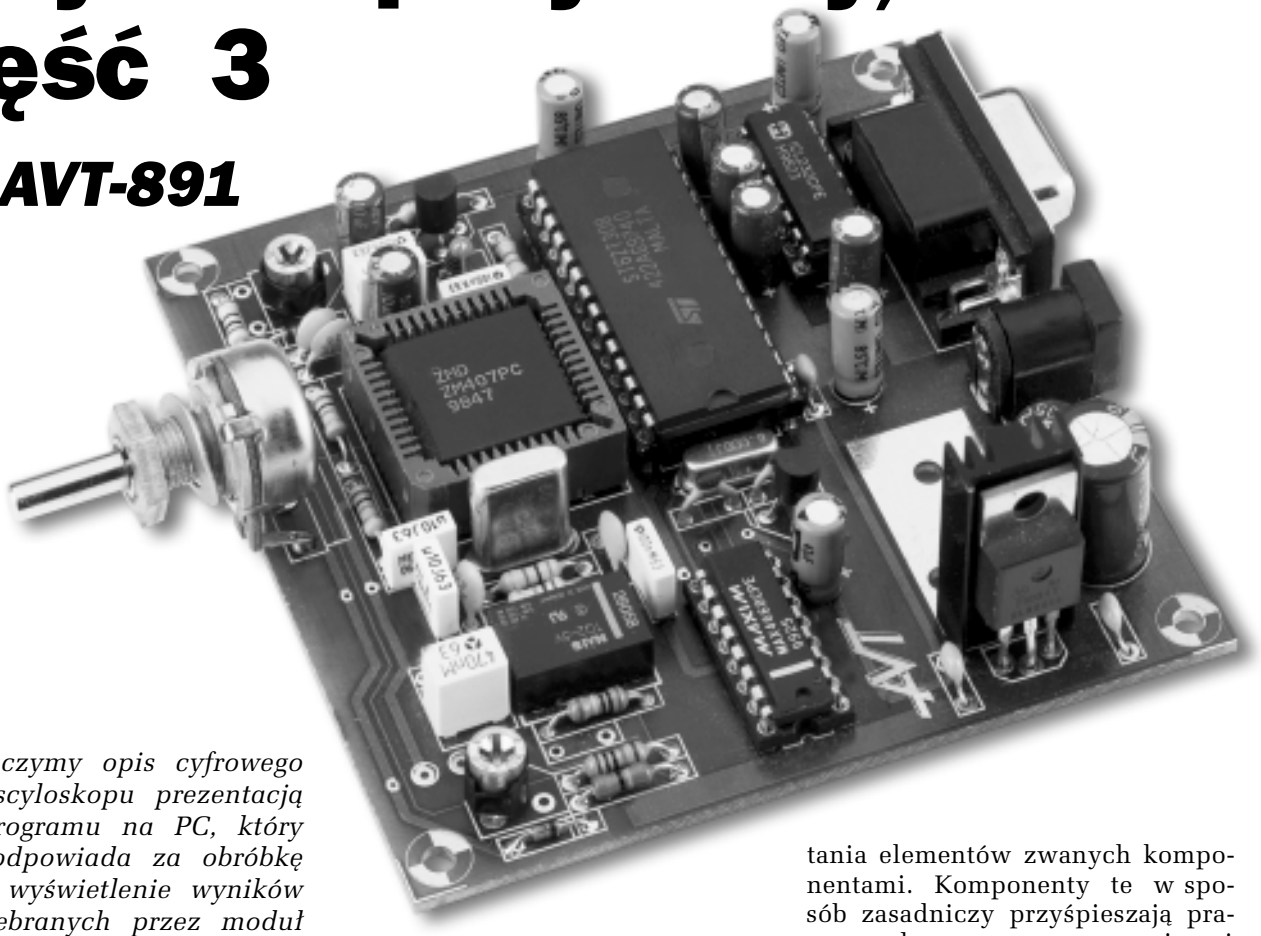
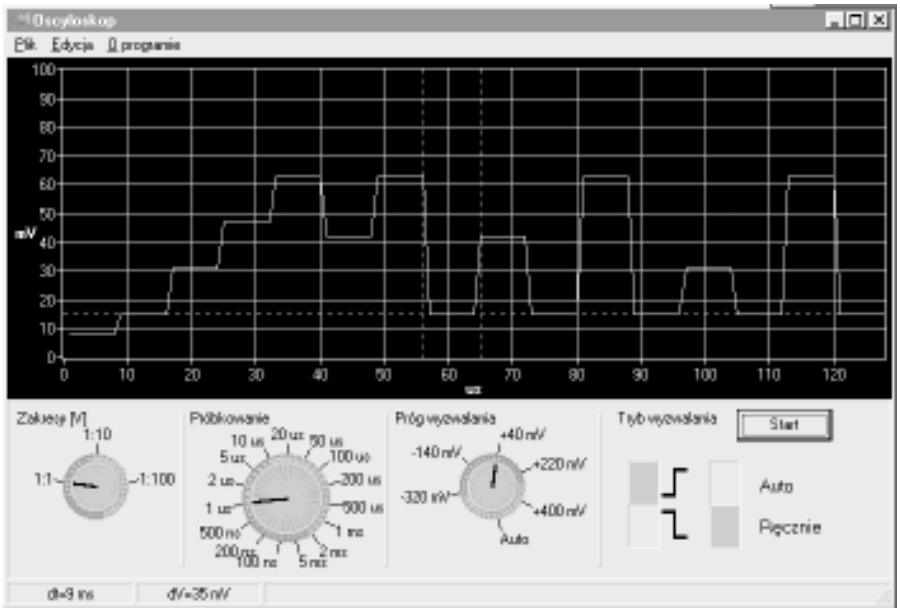
Oscyloskop cyfrowy, część 3

kit AVT-891

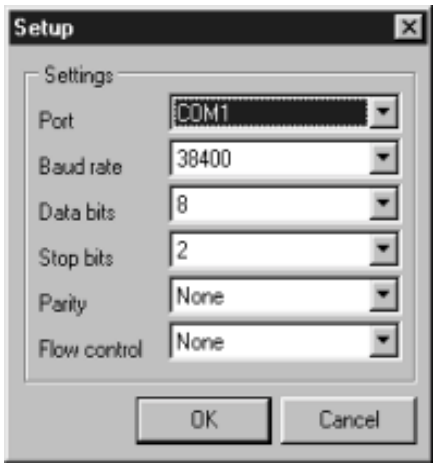


Kończymy opis cyfrowego oscyloskopu prezentacją programu na PC, który odpowiada za obróbkę i wyświetlenie wyników zebranych przez moduł sprzętowy.

Program sterujący oscyloskopem cyfrowym AVT-891 napisano w Delphi. Narzędzie to charakteryzuje się wyjątkowo łatwym w obsłudze interfejsem i bogatą biblioteką gotowych do wykorzystania elementów zwanych komponentami. Komponenty te w sposób zasadniczy przyśpieszają pracę nad programem, ponieważ przypomina ona bardziej składanie programu z gotowych klocków niż typową pracę programisty. W tym projekcie zostały wykorzystane cztery znalezione w Internecie elementy:



Rys. 8. Widok ekranu podczas pracy programu.



Rys. 9. Okno konfiguracji portu szeregowego.

- komponent *circlehandle* (pokrętła oscyloskopu),
- komponent *spsgraph* (ekran oscyloskopu),
- komponent *TComPort* (komunikacja z portem szeregowym),
- biblioteka *fourier* (procedury FFT).

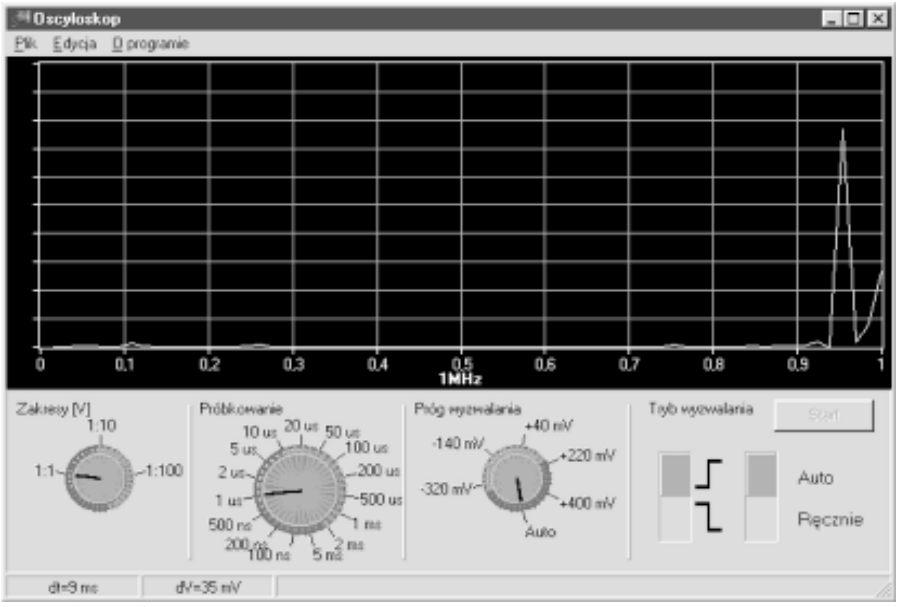
Program ten przeznaczony jest do pracy w środowisku Windows 95 lub nowszym. Co istotne, nie ma przy tym wygórowanych wymagań sprzętowych, powinien uruchomić się na każdym komputerze ze sprawnie działającym systemem operacyjnym. Jedyny problem może stworzyć znalezienie wolnego i prawidłowo podłączonego portu szeregowego.

Wszystkie czynności jakie wykonuje program obsłużone są w jednym wątku. Ze względu na specyfikę urządzenia, wystarczające jest pobieranie danych co 10ms. Obsługa tej czynności jest wykonywana w zdarzeniu *OnTimer* zwykłego komponentu *TTimer*, stanowiącego element pakietu Delphi. Zmiana dowolnego ustawienia za pomocą widocznych na ekranie elementów (**rys. 8**), powoduje wysłanie do modułu kodów sterujących opisanych w poprzednich częściach tego artykułu. Sterowanie do programu jest zwrócone po otrzymaniu przez program kodów potwierdzających odebranie polecenia.

W trakcie uruchamiania programu urządzenie z włączonym zasilaczem powinno być podłączone do komputera. Program

List. 1.

```
(*======================================================================
fourier.pas  -  Don Cross <dcross@intersrv.com>
This is a Turbo Pascal Unit for calculating the Fast Fourier Transform
(FFT) and the Inverse Fast Fourier Transform (IFFT).
Visit the following URL for the latest version of this code.
This page also has a C/C++ version, and a brief discussion of the theory
behind the FFT algorithm.
http://www.intersrv.com/~dcross/fft.html#pascal
======================================================================*)

{$N+,E+} (* Allows code to use type 'double' and run on any iX86 machine *)
{$R-} (* Turn off range checking...we violate array bounds rules *)

unit Fourier;

interface

(*--------------------------------------
  procedure fft
Calculates the Fast Fourier Transform of the array of complex numbers
represented by 'RealIn' and 'ImagIn' to produce the output complex
numbers in 'RealOut' and 'ImagOut'.
--------------------------------------*)
procedure fft (
    NumSamples:   word;   { must be a positive integer power of 2 }
    var  RealIn:   array of double;
    var  ImagIn:   array of double;
    var  RealOut:  array of double;
    var  ImagOut:  array of double );

(*--------------------------------------
  procedure ifft
Calculates the Inverse Fast Fourier Transform of the array of complex
numbers represented by 'RealIn' and 'ImagIn' to produce the output
complex numbers in 'RealOut' and 'ImagOut'.
--------------------------------------*)
procedure ifft (
    NumSamples:   word;   { must be a positive integer power of 2 }
    var  RealIn:   array of double;
    var  ImagIn:   array of double;
    var  RealOut:  array of double;
    var  ImagOut:  array of double );

(*--------------------------------------
  procedure fft_integer
Same as procedure fft, but uses integer input arrays instead of double.
Make sure you call fft_integer_cleanup after the last time you call
fft_integer to free up memory it allocates.
--------------------------------------*)
procedure fft_integer (
    NumSamples:   word;
    var  RealIn:   array of integer;
    var  ImagIn:   array of integer;
    var  RealOut:  array of double;
    var  ImagOut:  array of double );

(*--------------------------------------
   procedure fft_integer_cleanup
If you call the procedure 'fft_integer', you must call
'fft_integer_cleanup' after the last time you call 'fft_integer' in
order to free up dynamic memory.
--------------------------------------*)
procedure fft_integer_cleanup;

(*--------------------------------------
   procedure CalcFrequency
This procedure calculates the complex frequency sample at a given index
directly.  Use this instead of 'fft' when you only need one or two
frequency samples, not the whole spectrum.
It is also useful for calculating the Discrete Fourier Transform (DFT)
of a number of data which is not an integer power of 2.  For example,
you could calculate the DFT of 100 points instead of rounding up to 128
and padding the extra 28 array slots with zeroes.
--------------------------------------*)
procedure CalcFrequency (
    NumSamples: word;       { can be any positive integer }
    FrequencyIndex: word;   { must be in the range 0 .. NumSamples-1 }
    var  RealIn:  array of double;
    var  ImagIn:  array of double;
    var  RealOut: double;
    var  ImagOut: double );

implementation

function IsPowerOfTwo ( x: word ): boolean;
var   i, y:  word;
begin
    y := 2;
    for i := 1 to 15 do begin
        if x = y then begin
            IsPowerOfTwo := TRUE;
            exit;
        end;
        y := y SHL 1;
    end;

    IsPowerOfTwo := FALSE;
end;

function NumberOfBitsNeeded ( PowerOfTwo:
word ): word;
var     i: word;
begin
    for i := 0 to 16 do begin
       if (PowerOfTwo AND (1 SHL i)) <> 0 then
       begin
            NumberOfBitsNeeded := i;
            exit;
       end;
    end;
end;

function ReverseBits ( index, NumBits: word ): word;
var     i, rev: word;
begin
    rev := 0;
    for i := 0 to NumBits-1 do begin
        rev := (rev SHL 1) OR (index AND 1);
        index := index SHR 1;
    end;
    ReverseBits := rev;
end;

procedure FourierTransform (
    AngleNumerator:  double;
    NumSamples:   word;
    var  RealIn:   array of double;
    var  ImagIn:   array of double;
    var  RealOut:  array of double;
    var  ImagOut:  array of double );
var
 NumBits, i, j, k, n, BlockSize, BlockEnd: word;
 delta_angle, delta_ar: double;
 alpha, beta: double;
 tr, ti, ar, ai: double;
begin
    if not IsPowerOfTwo(NumSamples) or
    (NumSamples<2) then begin
        write ('Error in procedure Fourier:  NumSamples=', NumSamples);
        writeln ( ' is not a positive integer power of 2.' );
        halt;
    end;

    NumBits := NumberOfBitsNeeded (NumSamples);
    for i := 0 to NumSamples-1 do begin
        j := ReverseBits ( i, NumBits );
        RealOut[j] := RealIn[i];
        ImagOut[j] := ImagIn[i];
    end;
```



Rys. 10. Wyświetlenie wyniku FFT.

w trakcie uruchamiania zapyta o numer portu do którego jest podłączony moduł (**rys. 9**). Jest to w zasadzie jedyne ustawienie jakiego należy dokonać aby skonfigurować program do pracy z oscyloskopem. W przypadku wystąpienia przerw w transmisji (np. w wyniku zaniku zasilania modułu) wystąpi konieczność zrestartowania programu.

Aplikacja umożliwia pomiar charakterystycznych parametrów sygnału poprzez ustawienie za pomocą myszy linii odniesienia. Można w ten sposób zmierzyć np. okres lub amplitudę sygnału. Drugą użyteczną w codziennej pracy opcją może być kopiowanie aktualnego obrazu sygnału do schowka systemowego. Dzięki temu mo-

List. 1 (cd).

```
    BlockEnd := 1;
    BlockSize := 2;
    while BlockSize <= NumSamples do begin
      delta_angle := AngleNumerator/BlockSize;
       alpha := sin ( 0.5 * delta_angle );
       alpha := 2.0 * alpha * alpha;
       beta := sin ( delta_angle );
       i := 0;
       while i < NumSamples do begin
            ar := 1.0;     (* cos(0) *)
            ai := 0.0;     (* sin(0) *)
            j := i;
            for n := 0 to BlockEnd-1 do begin
                k := j + BlockEnd;
                tr := ar*RealOut[k]-ai*ImagOut[k];
                ti := ar*ImagOut[k] + ai*RealOut[k];
                RealOut[k] := RealOut[j] - tr;
                ImagOut[k] := ImagOut[j] - ti;
                RealOut[j] := RealOut[j] + tr;
                ImagOut[j] := ImagOut[j] + ti;
                delta_ar := alpha*ar + beta*ai;
                ai := ai - (alpha*ai - beta*ar);
                ar := ar - delta_ar;
                INC(j);
            end;
            i := i + BlockSize;
        end;
        BlockEnd := BlockSize;
        BlockSize := BlockSize SHL 1;
    end;
end;

procedure fft (
    NumSamples:   word;
    var  RealIn:   array of double;
    var  ImagIn:   array of double;
    var  RealOut:  array of double;
    var  ImagOut:  array of double );
begin
  FourierTransform (2*PI, NumSamples, RealIn, ImagIn, RealOut, ImagOut);
end;

procedure ifft (
    NumSamples:   word;
    var  RealIn:   array of double;
    var  ImagIn:   array of double;
    var  RealOut:  array of double;
    var  ImagOut:  array of double );
var
    i: word;
begin
 FourierTransform (-2*PI, NumSamples, RealIn, ImagIn, RealOut, ImagOut);

    (* Normalize the resulting time samples... *)
    for i := 0 to NumSamples-1 do begin
        RealOut[i] := RealOut[i] / NumSamples;
        ImagOut[i] := ImagOut[i] / NumSamples;
    end;
end;

type
    doubleArray = array [0..0] of double;
var
    RealTemp, ImagTemp: ^doubleArray;
    TempArraySize:  word;

procedure fft_integer (
    NumSamples:   word;
    var  RealIn:   array of integer;
    var  ImagIn:   array of integer;
    var  RealOut:  array of double;
    var  ImagOut:  array of double );
var
    i: word;
begin
    if NumSamples > TempArraySize then begin
     fft_integer_cleanup;  {free up memory in case we already have some}
     GetMem ( RealTemp, NumSamples * sizeof(double) );
     GetMem ( ImagTemp, NumSamples * sizeof(double) );
     TempArraySize := NumSamples;
    end;
    for i := 0 to NumSamples-1 do begin
        RealTemp^[i] := RealIn[i];
        ImagTemp^[i] := ImagIn[i];
    end;

  FourierTransform (2*PI,NumSamples,RealTemp^,ImagTemp^,RealOut,ImagOut);
end;

procedure fft_integer_cleanup;
begin
    if TempArraySize > 0 then begin
        if RealTemp <> NIL then begin
            FreeMem ( RealTemp, TempArraySize * sizeof(double) );
            RealTemp := NIL;
        end;

        if ImagTemp <> NIL then begin
            FreeMem ( ImagTemp, TempArraySize * sizeof(double) );
            ImagTemp := NIL;
        end;

        TempArraySize := 0;
    end;
end;

procedure CalcFrequency (
    NumSamples: word;       { must be integer power of 2 }
    FrequencyIndex: word;   { must be in the range 0 .. NumSamples-1 }
    var  RealIn:  array of double;
    var  ImagIn:  array of double;
    var  RealOut: double;
    var  ImagOut: double );
var
    k: word;
    cos1, cos2, cos3, theta, beta: double;
    sin1, sin2, sin3: double;
begin
    RealOut := 0.0;
    ImagOut := 0.0;
    theta := 2*PI * FrequencyIndex / NumSamples;
    sin1 := sin ( -2 * theta );
    sin2 := sin ( -theta );
    cos1 := cos ( -2 * theta );
    cos2 := cos ( -theta );
    beta := 2 * cos2;
    for k := 0 to NumSamples-1 do begin
        { Update trig values }
        sin3 := beta*sin2 - sin1;
        sin1 := sin2;
        sin2 := sin3;

        cos3 := beta*cos2 - cos1;
        cos1 := cos2;
        cos2 := cos3;

        RealOut := RealOut + RealIn[k]*cos3 - ImagIn[k]*sin3;
        ImagOut := ImagOut + ImagIn[k]*cos3 + RealIn[k]*sin3;
    end;
end;

begin  { Unit initialization code }
    TempArraySize := 0; {flag that buffers RealTemp, RealImag not
allocated}
    RealTemp := NIL;
    ImagTemp := NIL;
end.


(*- end of file fourier.pas -*)
```

że on być użyty w innych programach.

Chcielibyśmy zwrócić waszą uwagę na fakt, że otrzymując do ręki kompletne urządzenie wraz z opisem protokołu przesyłania danych, dostajecie też szansę na stworzenie własnego, zupełnie nowego przyrządu, poprzez napisanie własnego programu sterującego. Mógłby to być np. rejestrator wolnozmiennych przebiegów lub program sterujący równocześnie dwoma modułami AVT-891 podłączonymi do dwu portów szeregowych.

Co fajnego można zrobić z kompletem 128 próbek? Jeśli ktoś lubi obróbkę danych ma szansę się wyżyć. Do wyboru ma różne rodzaje regresji, spliny lub coś ekstra: szybką transformatę Fouriera (ang. FFT - fast Fourier transformation).

Co to jest FFT?

Szybka transformata Fouriera jest, mówiąc w uproszczeniu, narzędziem matematycznym pozwalającym przetworzyć szereg próbek obrazujących zmianę sygnału w dziedzinie czasu na sygnał w dziedzinie częstotliwości (**rys. 10**). Pozwala więc zanalizować widmo sygnału. O taki analizator widma pokusiliśmy się w tym programie. Wektor danych wejściowych do algorytmu FFT musi mieć 2^m składowych (w naszym przypadku m=7). W rezultacie otrzymamy inny wektor, składający się ze $1+2^{(m-1)}$ współczynników zespolonych oznaczających wartości w dziedzinie częstotliwości. Elementy wektora wynikowego z procedury FFT spełniają następujące równanie:

$$c_j = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} v_k e^{2\Pi i(j/n)k}$$

W powyższym wzorze n oznacza liczbę elementów *v* ($n=2^m$), zaś i jest jednostką urojoną. Elementy wektora obliczonego przez FFT odpowiadają różnym częstotliwościom. W celu uzyskania faktycznej częstotliwości konieczna jest znajomość częstotliwości próbkowania sygnału wyjściowego. Jeżeli v jest n-elementowym wektorem przekazywanym do procedury FFT, a częstotliwość próbkowania wynosi f_s, faktyczna częstotliwość odpowiadająca elementowi c_k wynosi:

$$f_k = \frac{k}{n} \cdot f_s$$

Warto zwrócić uwagę na fakt, że niemożliwe jest wykrycie częstotliwości wyższych niż częstotliwość próbkowania. Nie jest to ograniczenie możliwości procedur numerycznych, ale bazy matematycznej na której są one oparte. W celu poprawnego odczytania sygnału za pomocą jego próbki poddanej transformacie Fouriera należy przeprowadzić próbkowanie z szerokością co najmniej dwukrotnie większą niż szerokość pasma.

Elementy wektora wynikowego c maja dwie składowe: rzeczywistą i urojoną. Nas będzie interesował moduł wyniku czyli:

$$|c_j| = \sqrt{\left(\mathrm{Re}(c_j)^2 + \mathrm{Im}(c_j)^2\right)}$$

Warto zauważyć, że sposób w jaki w naszym przypadku wykorzystujemy *fft* należy do najprostszych. W przypadku ogólnym wektor wejściowy może składać się z szeregu mającego składowe zarówno rzeczywiste jak i urojone. W naszym przypadku na wejściu podajemy szereg 128 próbek pobranych z modułu jako wektor części rzeczywistych i wektor zer, jako wektor części urojonych.

Na **list. 1** przedstawiamy napisane w Pascalu procedury: *fft*, niewykorzystywaną przez nas procedurę *ifft* (inverse fft - odwrotna transformata Fouriera) i dodatkowe procedury „narzędziowe". Biblioteka ta została znaleziona w przepastnych zasobach Internetu, a zamieszczamy ją ponieważ w odróżnieniu od innych jakie można znaleźć w sieci jest napisana przejrzyście i została starannie sprawdzona. Stanowi klasyczną implementację algorytmu FFT bez zbędnych udziwnień. Jest to niezły materiał wyjściowy do napisania własnego programu.

Adam Dębowski, AVT

Wzory płytek drukowanych w formacie PDF są dostępne w Internecie pod adresem: ***http://www.ep.com.pl/pcb.html*** *oraz na płycie CD-EP11/2000 w katalogu* ***PCB****.*