GUISDAP Documentation

M. S. Lehtinen* and A. Huuskonen**

* Sodankylä Geophysical Observatory, Sodankylä, Finland
** Finnish Meteorological Institute, Helsinki, Finland

GUISDAP experiment definition

The GUISDAP experiment definition is a set of variables which define the transmitter envelopes, receiver impulse responses and lag profiles for an incoherent scatter experiment. The GUISDAP data analysis, documentation and experiment evaluation tools can be applied to any Incoherent scatter experiment provided that these parameters are correctly specified. These variables are listed below for the CP1H experiment.

Parameters defined for each virtual channel. The experiment has 12 virtual channels and 38 separate sampling intervals:

`vc_ch`	1 by 12	Real channel for the virtual channel
`vc_env`	352 by 12	Envelope of the transmitted waveform
`vc_envo`	1 by 12	Start time of each envelope [`p_dtau`]
`vc_p`	120 by 12	Receiver impulse response
`vc_adcint`	1 by 12	Analog–to–digital conversion intervals [`p_dtau`]
`vc_ba`	3 by 12	Buffer memory start addresses, preliminary specification
`vc_sampling`	38 by 4	Sampling intervals, preliminary specification [`p_dtau`]

Parameters specified for each lag profile, the present experiment has 318 lag profiles

`lp_vc`	1 by 318	Virtual channel of the lag profile
`lp_t1`	1 by 318	Sampling time of first factor in the first product [`p_dtau`]
`lp_t2`	1 by 318	Sampling time of second factor in the first product [`p_dtau`]
`lp_dt`	1 by 318	sample increment [`p_dtau`]
`lp_fir`	40 by 318	Lag profile filtering coefficients
`lp_nfir`	1 by 318	Length of the lag profile filter
`lp_dec`	1 by 318	Decimation interval after filtering
`lp_nt`	1 by 318	Number of points in the profile after decimation
`lp_ra`	1 by 318	Result memory address of the first product
`lp_ri`	1 by 318	Result memory address increment
`lp_T`	1 by 318	Calibration temperature [K], 0 for signal and background
`lp_h`	1 by 318	Difference of sampling start time and transmission time [`p_dtau`]
`lp_bcs`	1 by 318	Type of profile, b=background, c=calibration, s=signal, o=offset, x=multipulse zero lag
`lp_code`	1 by 318	Label to group lag profiles

Other parameters

`ch_fradar`	1 by 8	The transmitter frequency for each channel [Hz]
`ch_gain`	1 by 8	Antenna gain for each channel
`p_dtau`	1 by 1	Basic time unit [μs]
`p_XMITloc`	1 by 3	transmitter location [latitude (deg N), longitude (deg E), altitude (km)]
`p_RECloc`	1 by 3	receiver location [latitude (deg N), longitude (deg E), altitude (km)]
`p_rep`	1 by 1	Experiment cycle time [`p_dtau`]
`p_ND`	1 by 1	Repetition factor for lag profiles

The present document describes how the GUISDAP variables are produced for EISCAT experiments

Initializing EISCAT experiments

The following steps are necessary in order to initialize and EISCAT experiment for GUISDAP:

1 Run the TLAN2PS program to combine the ELAN and TLAN files

2 Define the virtual channels

3 Define the lag profiles

4 Run the `init_EISCAT` program to produce the GUISDAP varibles and to store those to a file

The TLAN2PS program

The TLAN2PS program combines the ELAN and TLAN files into a '`pat_PS.m`' file. The '`pat`' in the name is a short form of 'pulse pattern'. The main task of the file is to define the pulse patterns and reception times for all channels, as well as the times when the calibration signal is injected. It also contains the filter information and sampling intervals for the channels. The '`PS`' in the name means that the file can be interpreted as Postscript commands. This feature is used in the documentation system when plotting the pulse patterns and reception arrangements. And finally, the '`.m`' denotes that the file is also a Matlab program, which imports the above mentioned information into the analysis package. The existense of the TLANTOPS program means that one need not transfer the pulse form, reception and filter information to the analysis program manually. At present, the most up–to–date version of the program, distributed with GUISDAP, has been written by Bertrand Cabrit at Swedish Institute of Space Physics at Uppsala.

The following example file contains part of the `pat_PS.m` file for the CP1H experiment. Lines 2–25 are read from the ELAN file. The first eight of those connect the receiver channels with the transmitter frequencies. The next eight lines contain the filter information and the next eight the sampling intervals. The rest is read from the TLAN file, starting with the calibration information. The first `PS_PHA` command (line 58) shows that on the transmitter frequency corresponding to the receiver channel 5 a positive phase is transmitter between 750 and 764 microseconds. Later on (line 71) one can see that the reception (`PS_ON`) occurs between 1211 and 2432.

```
/geo/gmt/askoh/guisdap/exps/CP1H/CP1HTpat_PS.m(part)
1       initialize
2       [1 1] ;PS_SETFREQ
3       [2 3] ;PS_SETFREQ
4       [3 0] ;PS_SETFREQ
5       [4 2] ;PS_SETFREQ
6       [5 4] ;PS_SETFREQ
7       [6 5] ;PS_SETFREQ
8       [7 6] ;PS_SETFREQ
9       [8 9] ;PS_SETFREQ
10      [1 25.0] ;PS_BUFILT
11      [2 25.0] ;PS_BUFILT
12      [3 25.0] ;PS_BUFILT
13      [4 25.0] ;PS_BUFILT
14      [5 35.4] ;PS_LIFILT
15      [6 35.4] ;PS_LIFILT
16      [7 35.4] ;PS_LIFILT
17      [8 35.4] ;PS_LIFILT
18      [1 10.0] ;PS_SETADCINT
19      [2 10.0] ;PS_SETADCINT
20      [3 10.0] ;PS_SETADCINT
21      [4 10.0] ;PS_SETADCINT
22      [5  6.0] ;PS_SETADCINT
23      [6  6.0] ;PS_SETADCINT
24      [7  6.0] ;PS_SETADCINT
25      [8  6.0] ;PS_SETADCINT
26      [0 0 450] ;PS_EMPTY
27      [0 450 645] ;PS_CALON
28      [0 645 5850] ;PS_EMPTY
29      [0 5850 6545] ;PS_CALON
30      [0 6545 12300] ;PS_EMPTY
31      [0 12300 12907] ;PS_CALON
32      [0 12907 13000] ;PS_EMPTY
33      [1 0 5] ;PS_EMPTY
34      [2 0 5] ;PS_EMPTY
35      [5 0 183] ;PS_EMPTY
36      [6 0 183] ;PS_EMPTY
37      [7 0 183] ;PS_EMPTY
38      [8 0 183] ;PS_EMPTY
39      [1 5 450] ;PS_ON
40      [2 5 450] ;PS_ON
```

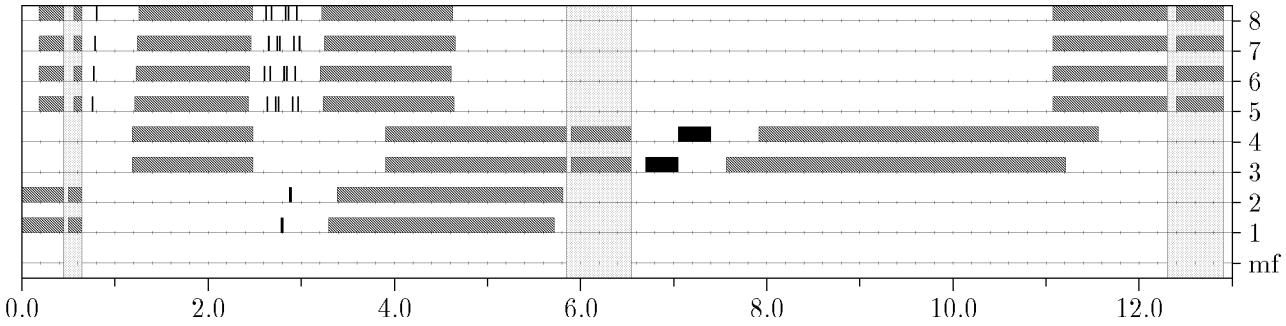
```
    [5 183 450] ;PS_ON
    [6 183 450] ;PS_ON
    [7 183 450] ;PS_ON
    [8 183 450] ;PS_ON
    [1 450 500] ;PS_EMPTY
    [2 450 500] ;PS_EMPTY
    [5 450 558] ;PS_EMPTY
    [6 450 558] ;PS_EMPTY
    [7 450 558] ;PS_EMPTY
    [8 450 558] ;PS_EMPTY
    [1 500 645] ;PS_ON
    [2 500 645] ;PS_ON
    [5 558 645] ;PS_ON
    [6 558 645] ;PS_ON
    [7 558 645] ;PS_ON
    [8 558 645] ;PS_ON
    [5 645 750] ;PS_EMPTY
    [5 750 764] ;PS_PHA
    [6 645 765] ;PS_EMPTY
    [6 765 779] ;PS_PHA
    [7 645 780] ;PS_EMPTY
    [7 780 794] ;PS_PHA
    [8 645 795] ;PS_EMPTY
    [8 795 809] ;PS_PHA
    [3 0 1185] ;PS_EMPTY
    [4 0 1185] ;PS_EMPTY
    [5 764 1211] ;PS_EMPTY
    [6 779 1226] ;PS_EMPTY
    [7 794 1241] ;PS_EMPTY
    [8 809 1256] ;PS_EMPTY
    [5 1211 2432] ;PS_ON
    [6 1226 2447] ;PS_ON
    [7 1241 2462] ;PS_ON
    [8 1256 2477] ;PS_ON
    [3 1185 2480] ;PS_ON
    [4 1185 2480] ;PS_ON

    (lines removed)

    [3 11210 13000] ;PS_EMPTY
    [4 11561 13000] ;PS_EMPTY
    [5 12907 13000] ;PS_EMPTY
    [6 12907 13000] ;PS_EMPTY
    [7 12907 13000] ;PS_EMPTY
    [8 12907 13000] ;PS_EMPTY
    filetrailer
```

When the `pat_PS.m` file is ready, it is possible to use the documentation features of the package and to produce plots of transmitting and receiving arrangements. The following figure shows those for the CP1H experiment. In the figure, the solid blocks show the transmission and the dotted columns the times when the noise injection is on. The remaining blocks then denote the reception times. In a Barker–coded experiment the lowest line would give the times when the matched filter is in operation. A similar figure is obtained by issuing the Matlab command `plot_PS`.

EISCAT CP-1-H coding scheme



Defining virtual channels

The figure above is a good starting point when defining the virtual channels. The concept of a virtual channel is needed, because the real channels are often used for transmitting multiple modulations during a pulse repetition period. This is the case for channels 5, 6, 7 and 8 in CP1H (figure above). They are used for transmitting a power profile pulse and a five pulse code, and they have to be divided into two virtual channels, the first containing the power profile pulse with the reception connected to it, and the second containing the five pulse code with the reception connected to it. If this were not done, the initialization calculations would treat the transmission as an immensely long six pulse code, which is not what is wanted. Instead we treat the transmission as if it had happened at completely different frequencies and receiver channels. It also means that the clutter coming from the other transmission in the same channel is neglected. This, however, is not a serious fault in the analysis of a well designed experiment. If during one pulse repetition period a channel is used only once for transmission, the virtual and real channels are the same. This is the case for channels 1,2,3 and 4 in CP1H.

The file defining virtual channels for CP1H is shown below. The variable `vc_ch` gives the real channel number for each virtual channel. The variables `vc_t1` and `vc_t2` give the time limits of the virtual channels in microseconds. In a Barker–coded experiment, a variable `vc_mf` is used to show that data coming from a certain virtual channel passes through the Barker decoder.

/geo/gmt/askoh/guisdap/exps/CP1H/CP1HTvcinit.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   GUP ver. 1.1       Sodankyla  17 Aug 1990                                  %
%   copyright Markku Lehtinen, Asko Huuskonen, Matti Vallinkoski    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% cp1vcinit.m
vc_ch=[1 2  3  4   5   6   7   8   5   6   7   8];
vc_t1=[0 0  1  1 2.5 2.5 2.5 2.5   0   0   0   0]*1000;
vc_t2=[6 6 12 12  13  13  13  13 2.5 2.5 2.5 2.5]*1000;
```

Defining lag profiles

The most difficult part in the initialization is to describe how the correlator was used during the experiment. This has to be done manually, as the GUISDAP package does not contain a CORLAN interpreter. The aim of this part of initialization is to define the lag profiles that describe the experiment. A lag profile is a set of crossed products, who all have the same ambiguity functions, but successive points in the profile correspond to increasing altitudes so that the separation between the altitudes does not vary. The most simple example of a lag profile is a set of measurements of the received power, i.e. the power profile. Fortunately one does not have to specify all the lag profiles separately. The GUISDAP package contains a set of function calls, which correspond to the various types of correlator algorithms in use and produce the lag profiles from a small set of input parameters. Thus, in order to define the lag profiles, it is only necessary to write a program which calls these functions.

The program for CP1H is shown below. The meanings of the input parameters are explained in the respective functions. Most of the input parameters are found from the experiment documentation. The virtual channel numbers, defined earlier, need now to be known. The last parameter in each of the calls is a flag to the modulation, here it is chosen to be 1 for the power profile, 2 for the multipulse including the zero lag, 3 for the 29 μs power profile and 4 for the long pulse.

/geo/gmt/askoh/guisdap/exps/CP1H/CP1HT_LP.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   GUP ver. 1.1       Sodankyla  17 Aug 1990                                  %
%   copyright Markku Lehtinen, Asko Huuskonen, Matti Vallinkoski    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
COR_init

for vc=9:12,
  COR_pp( 0,1,vc,'b',3,15,0,1)
  COR_pp(15,1,vc,'c',3, 5,0,1)
  COR_pp(20,1,vc,'s',3,40,0,1)
end

for vc=5:8,
  COR_mp( 60,12,vc,'b',3,25,0:30:330,0,2)
  COR_mp(360,12,vc,'c',3, 5,0:30:330,0,2)
end

for vc=9:12,
```

```
  COR_pp(420,12,vc,'s',3,60,24,2)
end

for vc=5:8,
  COR_mp(421,12,vc,'s',3,60,30:30:270,0,2)
  COR_mp(430,12,vc,'o',3,60,   300   ,0,2)
  COR_mp(431,12,vc,'s',3,60,   330   ,0,2)
end

for vc=1:2
  COR_pp(1140,1,vc,'b',3,15,0,3)
  COR_pp(1155,1,vc,'c',3, 5,0,3)
  COR_pp(1160,1,vc,'s',3,81,0,3)
end

for vc=3:4,
  COR_lp(1241,26,vc,'b',15,50, 5,0:10:250,4)
  COR_lp(1371,26,vc,'c',15, 0, 1,0:10:250,4)
  COR_lp(1397,26,vc,'s',15, 0,21,0:10:250,4)
end

COR_end
```

The basic time unit

The last user defined file contains the radar frequency (Hz) and the basic time unit (`p_dtau`), which is given in μs. It should be chosen so that all lag resolutions are exact multiples of it. The `init_EISCAT` program will stop with an error message if that is not the case. The following example is again from CP1H. It can be copied as such for other experiment by changing `p_dtau`, if necessary. If the file does not exist, standard values are used for the frequency (931.5 MHz for UHF and 224 MHz for VHF) and for the basic time units (1 μs). The radar frequency should actually be defined separately for each channel and ought to be read from the ELAN file. This weakness will be corrected sooner or later.

/geo/gmt/askoh/guisdap/exps/CP1H/CP1HT_init.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   GUP ver. 1.1       Sodankyla  17 Aug 1990                          %
%   copyright Markku Lehtinen, Asko Huuskonen, Matti Vallinkoski       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CP1HT_init.m
% Radar frequency and basic time unit

ch_fradar=933.5e6; % Hz
p_dtau=1.0;        % us
```

The init_EISCAT program

After the previous steps have been taken, the initialization can be started by running the `init_EISCAT` program. The program makes the following operations (the numbers on the left refer to the line numbers in the program listing).

33–40 Specify the standard frequency and basic time unit values

44–52 Transmitter and receiver locations

67–75 Execute the `_init` file (if existing) or use the standard values

77–83 Check whether the old `p_dtau` definition has been used

92–96 The pulse pattern, filter and sampling information are read from a file or are obtained by executing the `pat_PS.m` file

98–112 Execute the `_vc` program to define the virtal channels. If the file does not exist, the virtual channels and the real channels are assumed equivalent. This feature is useful at the remotes

116 `envcalc`: The transmitter envelope and the receiver impulse response is calculated for each virtual channel

119 `find_sampling` detects the sampling intervals

124–126 The lag profiles are obtained here

129 As the last operation, `init_EISCAT` stores a set of variables to a file. These variables, listed at the beginning of this document, form the GUISDAP experiment specification.

/geo/gmt/askoh/guisdap/m152/init_EISCAT.m

```
1   % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2   %
3   % The main program to produce the GUISDAP variables for EISCAT experiments
4   % The user files are found in the exps/CP1K directory
5   %     CP1KT_init.m   : definition of basic time unit etc
6   %     CP1KTpat_PS.m  : transmission, reception and filter definitions
7   %     CP1KT_vcinit.m : virtual channel definitions
8   %     CP1KT_LP.m     : lag profile definitions
9   %
10  % Other files called
11  %  td_arrange      % various helpful things performed
12  %  envcalc         % forms the transmitter envelopes
13  %  find_sampling   % find when virtual channels sample
14  %  vc_arrange      % cleaning of the virtual channel variables
15  %
16  % init_EISCAT and the routines called by it do not, strictly speaking,
17  % belong to GUISDAP. . They form a method of producing GUISDAP variables
18  % for EISCAT experiments. Therefore these routines are allowed to contain
19  % explicit reference to EISCAT radar parameters and the site letters
20  %  'T', 'V', 'K', 'S' and 'R' have predefined meanings
21  %
22  % See also: design glob_EISCAT td_arrange envcalc find_sampling vc_arrange
23  %
24  clg, hold off
25  glob_EISCAT
26
27  % Chdir to the experiment directory, store the present path as a return address
28  original_path=pwd;
29  fprintf('\n\nChdir to the experiment directory\n')
30  cdir(canon(path_expr))
31
32  % These values will be used if not redefined later
33  p_dtau=1;
34  if name_site=='T' | name_site=='K' | name_site=='S' | name_site=='R'
35    ch_gain=10^4.81*ones(1,8);
36    ch_fradar=931.5e6*ones(1,8);
37  elseif name_site=='V'
38    ch_gain=10^4.31*ones(1,8);
39    ch_fradar=224e6*ones(1,8);
40  end
41  % For compatibility with the experiment design package define
42  p_ND=1;
43
44  % Put the transmitter/receiver locations to GUP variables
45  p_XMITloc=[69.583, 19.21, .030];  % Latitude, longitude, altitude
46  if name_site=='T' | name_site=='V'
47    p_RECloc=[69.583, 19.21, .030];
```

```
48    elseif name_site=='K'
49      p_RECloc=[67.863, 20.44, .412];
50    elseif name_site=='S' | name_site=='R'
51      p_RECloc=[67.367, 26.65, .180];
52    end
53
54    % The calibration temperatures, which may be redefined during analysis
55    if name_site=='T' | name_site=='V'
56      p_calTemp=210;
57    else
58      p_calTemp=30;
59    end
60
61    if exist('N_rcprog')~=1, N_rcprog=1; end
62    for d_rcprog=1:N_rcprog
63      Stime=clock;
64
65      if N_rcprog>1, apustr=['_',int2str(d_rcprog)]; else apustr=[]; end
66
67      file=canon([name_expr name_site '_init'],0);
68      for i=1:3,fprintf('\n'), end
69      if exist(file)==2
70        eval(file)
71        fprintf(['# ',file, ' executed\n'])
72      else
73        fprintf(['# ',file, ' not available .......... Standard values will be used\n'])
74      end
75      for i=1:3,fprintf('#\n'), end
76
77      if p_dtau<0.01;  % Due to change in p_dtau definition
78         p_dtau=p_dtau*1e6;
79         for i=1:10,fprintf('*********\n');end
80         fprintf(' Starting from GUP 1.50 p_dtau is given in microseconds\n')
81         fprintf(' change p_dtau to get rid of this message\n')
82         for i=1:10,fprintf('*********\n');end
83      end
84      if length(ch_fradar)==1,
85        ch_fradar=ch_fradar*ones(1,8);
86      end
87      fprintf('#\n# Basic time unit and transmitter frequencies have been defined:\n')
88      fprintf('#\n#\n#\n# Basic time unit is %.1f us\n# Frequencies are\n', p_dtau)
89      fprintf('# %.1f MHz %.1f MHz %.1f MHz %.1f MHz\n', ch_fradar/1E6), fprintf('\n\n')
90
91      global td_am td_ch td_t1 td_t2 c_f
92      if name_site=='T' | name_site=='V'
93        load_PS(d_rcprog,N_rcprog)
94      else
95        load_PSrem
96      end
97
98      fprintf('\n\n# Defining virtual channels:\n')
99      file=canon([name_expr name_site 'vcinit'],0);
100     for i=1:3,fprintf('#\n'), end
101     if exist(file)==2
102       eval(file)
103       fprintf(['# ',file, ' executed\n'])
104     else
105       fprintf(['# ',file, ' not available \n'])
106       fprintf('# Assuming virtual channels extending from 0 to REP (%.0f)\n',p_rep)
107       vc_ch=diff_val(td_ch(td_ch~=0));
108       vc_t1=zeros(size(vc_ch));
109       vc_t2=p_rep*ones(size(vc_ch));
110       vc_mf=zeros(size(vc_ch));
111     end
112     fprintf('#\n#\n# Virtual channels defined \n')
113
114     td_arrange
115     fprintf(['\n\nCalculating transmission envelopes and receiver impulse responses ...\n\n'])
116     envcalc,
117     fprintf(['\n\n .....Calculated \n\n'])
118
119     find_sampling % find when virtual channels sample
120
121     clear td_am td_t1 td_t2 td_ch
122     vc_arrange
123
124     fprintf(['\nDefining lag profiles by executing ',name_expr name_site '_LP:\n\n'])
125     eval(canon([name_expr name_site '_LP'],0))
126     fprintf(['\n' name_expr name_site '_LP passed\n\n\n\n'])
127
128     fprintf('  Time used in initialization:%8.2f min\n',etime(clock,Stime)/60)
129     save_GUPvar
130     fprintf('Radar controller program %g ready\n',d_rcprog)
131   end
132   fprintf(['\nChdir back to the original directory ', original_path,'\n'])
```

```
133     cdir(original_path)
134     fprintf('\n*******************************************************\n')
135     fprintf('*\n*\n* Execute plot_td to see the timing diagram\n')
136     fprintf('*\n*\n*******************************************************\n')
137     clear original_path i file
```

The script `fullinit.m` initializes a set of experiments. It also produces the interpolation table of plasma dispersion function values.

/geo/gmt/askoh/guisdap/m152/fullinit.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % script for initializing some EISCAT CP-experiments
4       t=clock;f=flops;
5       eval(canon(['save ' ,path_tmp,'timing.mat t f']))
6       eval(canon(['delete ' ,path_tmp,'fullinit.diary']))
7       eval(canon(['diary '  ,path_tmp,'fullinit.diary']))
8
9       start_GUP,name_expr='CP1H'; name_site='T';N_rcprog=1;init_EISCAT;init_GUP
10      start_GUP,name_expr='CP1H'; name_site='R';N_rcprog=1;init_EISCAT;init_GUP
11      %start_GUP,name_expr='CP1K'; name_site='T';N_rcprog=1;init_EISCAT;init_GUP
12      %start_GUP,name_expr='CP1K'; name_site='R';N_rcprog=1;init_EISCAT;init_GUP
13      %start_GUP,name_expr='CP3F'; name_site='T';N_rcprog=6;init_EISCAT;init_GUP
14      %start_GUP,name_expr='CP3F'; name_site='R';N_rcprog=6;init_EISCAT;init_GUP
15      %start_GUP,name_expr='CP7E'; name_site='V';N_rcprog=1;init_EISCAT;init_GUP
16
17      diary off
18      eval(canon(['load ' ,path_tmp,'timing.mat']))
19      eval(canon(['delete ' ,path_tmp,'timing.mat']))
20      fprintf('Time used %4.1f hours\n',etime(clock,t)/3600)
21      fprintf('Total number of float point operations was %4.1e \n',flops-f)
22      fprintf('Speed was %4.1f klops\n',(flops-f)*1e-3/etime(clock,t))
```

Routines for defining the lag profiles

The following routines have been developed to make the specification of commonly used EISCAT correlator algorithms easy. Each of them specifies the set of lag profiles calculated by an EISCAT correlator subroutine.

`COR_alter.m` specifies lag profiles as are calculated for alternating codes. This routine is limitid to the area, where estimates of all lags, even the longest, are obtained. We suppose that the lags are multiples of baud separation. (This is not a necessary condition for using alternating codes, and it would be useful to generalize this routine in the future).

/geo/gmt/askoh/guisdap/m152/COR_alter.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % COR_alter defines lag profiles for decoded alternating code experiments. It specifies
4       % the gates where all lags, up to the longest one, are available. It is further restricted
5       % to the case, where the lag values are exact multiples of the bit separation.
6       %
7       % User specified input parameters:
8       % ra      : Result memory addresses for the first gate, one address for each lag.
9       % ri      : result memory address increment, assumed equal for all lags
10      % vc      : virtual channel number
11      % type    : lag profile type, which MUST be signal ('s')
12      % gating  : number of successive crossed products added to the same memory location
13      %           This is ofter used with oversampling
14      % N_gates : number of gates to be spedified
15      % Nbits   : Number of bits in the alternating code
16      % bitsep  : distance of leading edges of code bauds (optional).
17      %           The value equals the bit length often, but not e.g. for interlaced alternating codes
18      % lags    : Lag  values in us
19      % N_skipped : number of samples skipped at the beginning of the sample vector (often =0)
20      % code     : user specified code number
21      % Output is stored in the lp_xxx parameters (global)
22      %
23      % See also: CORR_alter, COR_check, COR_caltemp, COR_status, pulse_times, sample_times
24      %
25      %function COR_alter(ra,ri,vc,type,gating,N_gates,Nbits,bitsep,lags,N_skipped,code)
26       function COR_alter(ra,ri,vc,type,gating,N_gates,Nbits,bitsep,lags,N_skipped,code)
27
28      global vc_adcint vc_sampling vc_samplingend vc_ba bm_samples bm_next vc_mf
29      global lp_t1 lp_t2 lp_h lp_ra lp_nfir lp_fir lp_dec lp_T lp_dt p_dtau
30      global lp_nt lp_vc lp_ri lp_bcs lp_code lp_ind lp_indold ra_next ra_prev
31
32      if (type~='s'),  error(' Unknown data type in COR_alter'), end
33
34      ra_prev=ra;
35
```

```
36      adcint=vc_adcint(vc); lags=lags/p_dtau; bitsep=bitsep/p_dtau;
37      COR_check(lags,adcint,bitsep)
38      if rem(bitsep,adcint)
39        error(' Bit separation must be an exact multiple of adc interval')
40      end
41
42      N_lags=length(lags);
43      N_lp=N_lags*gating;
44      index=(lp_ind+1):(lp_ind+N_lp);
45      lp_T(index)=COR_caltemp(type)*ones(1,N_lp);
46
47      % find first when pulses are transmitted and samples are taken
48      pulsetimes=pulse_times(vc);
49      am=env(vc,pulsetimes(1)+[0:(Nbits-1)]*bitsep+1)'; % 1 added to get +-1 as the result
50
51      fs_time=sample_times(vc,type2ind(type));
52      t1=(fs_time(1)+N_skipped*adcint)+kron(ones(1,N_lags),(0:gating-1)*adcint);
53      lp_t1(index)=t1;
54      lp_h(index)=t1-pulsetimes(1);
55
56      lp_t2(index)=t1+kron(lags,ones(1,gating));
57      lp_dt(index)=adcint*ones(1,N_lp);
58
59      SB=bitsep/adcint; % Samples per bit in the alternating code
60      apu(index)=kron(Nbits-round(lags/bitsep),ones(1,gating));
61      lp_nfir(index)=(apu(index)-1)*SB+1;
62      for ind=index
63        Ntaps=apu(ind);
64        signs=am(1:Ntaps).*am(Nbits-Ntaps+1:Nbits);
65        if SB==1,
66          lp_fir(1:lp_nfir(ind),ind)=signs;
67        else
68          lp_fir(1:lp_nfir(ind),ind)=[kron(signs(1:Ntaps-1),[1;zeros(SB-1,1)]);signs(Ntaps)];
69        end
70      end
71      lp_dec(index)=gating*ones(1,N_lp);
72
73      lp_T(index)=zeros(1,N_lp);
74      lp_nt(index)=N_gates*ones(1,N_lp);
75      lp_vc(index)=vc*ones(1,N_lp);
76      lp_ra(index)=kron(ra,ones(1,gating));
77      lp_ri(index)=ri*ones(1,N_lp);
78      lp_bcs(index)=type*ones(1,N_lp);
79      lp_code(index)=code*ones(1,N_lp);
80      lp_ind=lp_ind+N_lp;
81
82      % first result memory location not used
83      ra_next=max(lp_ra(index)+(lp_nt(index)-1).*lp_ri(index))+1;
84
85      COR_status('COR_alter',vc,type,index)
```

Lag profiles generated by the GEN-system long pulse algorithm (Turunen: The Gen-system for the EISCAT incoherent scatter radars, EISCAT Technical Note 85/44).

/geo/gmt/askoh/guisdap/m152/COR_lp.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % COR_lp defines lag profiles for GEN-system long pulses
4       % input parameters
5       % ra      : First result memory address to use
6       % ri      : result memory address increment
7       % vc      : virtual channel number
8       % type    : background ('b'), calibration ('c') or signal ('s')
9       % gating : number of crossed products added together for the lag zero
10      % overlap: number of products skipped between gates (positive skipping)
11      %         : if negative, then same crossed products are used for successive gates
12      % N_gates : number of gates calculated
13      % lags    : Lags for which output needed in us, e.g. 0:10:250;
14      % code    : user specified code number
15      %
16      % See also: CORR_lp, COR_check, COR_caltemp, COR_status, pulse_times, sample_times
17      %
18      %function COR_lp(ra,ri,vc,type,gating,overlap,N_gates,lags,code)
19       function COR_lp(ra,ri,vc,type,gating,overlap,N_gates,lags,code)
20
21      global vc_adcint vc_sampling vc_samplingend vc_ba bm_samples bm_next
22      global lp_t1 lp_t2 lp_h lp_ra lp_nfir lp_fir lp_dec lp_T lp_dt p_dtau
23      global lp_nt lp_vc lp_ri lp_bcs lp_code lp_ind lp_indold ra_next ra_prev
24
25      if (type~='b' & type~='c' & type~='s'),
26        error(' Unknown data type in COR_lp')
27      end
28
29      ra_prev=ra;
```

```
30
31      adcint=vc_adcint(vc); lags=lags/p_dtau;
32      COR_check(lags,adcint)
33
34      N_lags=length(lags);
35      index=(lp_ind+1):(lp_ind+N_lags);
36
37      lp_T(index)=COR_caltemp(type)*ones(1,N_lags);
38
39      % find first when pulses are transmitted and samples are taken
40      pulsetimes=pulse_times(vc);
41      t1=sample_times(vc,type2ind(type));
42      t1=t1(1)+max(lags)-lags;
43      lp_t1(index)=t1;
44      lp_h(index)=t1-pulsetimes(1);
45
46      % finally parameters common to background, calibration and signal
47      lp_t2(index)=t1+lags;
48      lp_dt(index)=adcint*ones(1,N_lags);
49      lp_nfir(index)=gating+lags/adcint;
50      for i=index
51        lp_fir(1:lp_nfir(i),i)=ones(lp_nfir(i),1); end
52      lp_dec(index)=(gating+overlap)*ones(1,N_lags);
53      lp_nt(index)=N_gates*ones(1,N_lags);
54      lp_vc(index)=vc*ones(1,N_lags);
55      lp_ra(index)=ra+(0:N_lags-1);
56      lp_ri(index)=ri*ones(1,N_lags);
57      lp_bcs(index)=type*ones(1,N_lags);
58      lp_code(index)=code*ones(1,N_lags);
59      lp_ind=lp_ind+N_lags;
60
61      % first result memory location not used
62      ra_next=max(lp_ra(index)+(lp_nt(index)-1).*lp_ri(index))+1;
63
64      COR_status('   COR_lp',vc,type,index)
```

Lag profiles as defined for the old-type multipulse systems, where the lags measured are arranged to form heightwise ACF:s.

/geo/gmt/askoh/guisdap/m152/COR_mp.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % Defines lag profiles for decoded multipulse experiments
4       %
5       % User specified input parameters:
6       % ra       : First result memory address to use
7       % ri       : result memory address increment
8       % vc       : virtual channel number
9       % type     : lag profile type, 's','b','c' and 'o' allowed
10      % gating   : number of successive crossed products added to the same memory location
11      % N_gates  : number of gates calculated
12      % lags     : Lag  values in us
13      % N_skipped : number of samples skipped at the beginning of the sample vector (often =0)
14      % code      : user specified code number
15      % Output is stored in the lp_xxx parameters (global)
16      %
17      % See also: CORR_mp, COR_check, COR_caltemp, COR_status, pulse_times, sample_times
18      %
19      %function COR_mp(ra,ri,vc,type,gating,N_gates,lags,N_skipped,code)
20       function COR_mp(ra,ri,vc,type,gating,N_gates,lags,N_skipped,code)
21
22      global p_dtau vc_adcint vc_mf
23      global lp_t1 lp_t2 lp_h lp_ra lp_nfir lp_fir lp_dec lp_T lp_dt p_dtau
24      global lp_nt lp_vc lp_ri lp_bcs lp_code lp_ind lp_indold ra_next ra_prev
25
26      if (type~='b' & type~='c' & type~='s' & type~='o'),
27        error(' Unknown data type in COR_mp')
28      end
29
30      ra_prev=ra;
31
32      adcint=vc_adcint(vc); lags=lags/p_dtau;
33      COR_check(lags,adcint)
34
35      N_lags=length(lags);
36      index=(lp_ind+1):(lp_ind+N_lags);
37      lp_T(index)=COR_caltemp(type)*ones(1,N_lags);
38
39      % find first when pulses are transmitted and samples are taken
40      pulsetimes=pulse_times(vc)';
41      fs_ti=sample_times(vc,type2ind(type));fs_ti=fs_ti(1);
42      if vc_mf(vc)>0, % Compensate for the delay caused by matched filter
43        t1=t1+(vc_mf(vc)-1)*adcint;
44      end
```

```
45
46      apu=pulsetimes(1)-pulsetimes;
47      codelags=[0];codeskips=[0];len=length(pulsetimes);
48      for i=1:len-1;
49        codelags=[ codelags,pulsetimes(i+1:len)-pulsetimes(i)];
50        codeskips=[codeskips,apu(i)*ones(1,len-i)];
51      end
52      [codelags,ind]=sort(codelags);codeskips=codeskips(ind);
53      [codelags',codeskips'];
54      t1=[];
55      for lag=lags,
56        ind=find(codelags==lag);
57        if length(ind)==1,
58          tt=fs_ti+N_skipped*adcint-codeskips(ind);
59        else % for missing lags in the code
60          tt=fs_ti+N_skipped*adcint;
61        end
62        t1=[t1,tt];
63      end
64
65      lp_t1(index)=t1;
66      lp_h(index)=t1-pulsetimes(1);
67
68      % finally parameters common to all types
69      lp_t2(index)=t1+lags;
70      lp_dt(index)=adcint*ones(1,N_lags);
71      lp_nfir(index)=gating*ones(1,N_lags);
72      lp_fir(1:gating,index)=ones(gating,N_lags);
73      lp_dec(index)=gating*ones(1,N_lags);
74      lp_nt(index)=N_gates*ones(1,N_lags);
75      lp_vc(index)=vc*ones(1,N_lags);
76      lp_ra(index)=ra+(0:N_lags-1);
77      lp_ri(index)=ri*ones(1,N_lags);
78      lp_bcs(index)=type*ones(1,N_lags);
79      lp_code(index)=code*ones(1,N_lags);
80      lp_ind=lp_ind+N_lags;
81
82      % first result memory location not used
83      ra_next=max(lp_ra(index)+(lp_nt(index)-1).*lp_ri(index))+1;
84
85      COR_status('   COR_mp',vc,type,index)
```

Simple power profiles:

/geo/gmt/askoh/guisdap/m152/COR_pp.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % calculates the lag profile parameters for power profiles. The call is identical to
4       % calling COR_uprog with the same input parameters and with lags=0;
5       %
6       % User specified input parameters:
7       % ra      : First result memory address to use
8       % ri      : result memory address increment
9       % vc      : virtual channel number
10      % type    : lag profile type, all allowed 's','b','c','o','x'
11      % gating  : number of successive crossed products added to the same memory location
12      % N_gates : number of gates for zero lag, the number will be smaller for longer lags
13      % N_skipped : number of samples skipped at the beginning of the sample vector (often =0)
14      % code     : user specified code number
15      % Output is stored in the lp_xxx parameters (global)
16      %
17      % See also: COR_uprog
18      %
19      % The function calls the uniprog routine.
20      % function COR_pp(ra,ri,vc,type,gating,N_gates,N_skipped,code)
21        function COR_pp(ra,ri,vc,type,gating,N_gates,N_skipped,code)
22
23       lags=0;
24       COR_uprog(ra,ri,vc,type,gating,N_gates,lags,N_skipped,code)
```

The old-type long-pulse algorithm used before GEN-system. (The range ambiguity function width of the longest lags is much smaller than that of the shortest lags). At present the algorithms is only used at the remotes.

/geo/gmt/askoh/guisdap/m152/COR_trilp.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % COR_trilp defines lag profiles for the long pulse algorithm used before the
4       % GEN-system algorithm. The range ambiguity function for the longer lags
5       % is much narrower than for the shorter lags.
6       % input parameters
7       % ra      : First result memory address to use
8       % ri      : result memory address increment
```

```
9       % vc      : virtual channel number
10      % type    : background ('b'), calibration ('c') or signal ('s')
11      % gating : number of crossed products added together for the lag zero
12      % overlap: number of products skipped between gates (positive skipping)
13      %         : if negative, then same crossed products are used for successive gates
14      % N_gates : number of gates calculated
15      % lags    : Lags for which output needed in us, e.g. 0:10:250;
16      % code    : user specified code number
17      %
18      % See also: CORR_trilp, COR_check, COR_caltemp, COR_status, pulse_times, sample_times
19      %function COR_trilp(ra,ri,vc,type,gating,overlap,N_gates,lags,code)
20       function COR_trilp(ra,ri,vc,type,gating,overlap,N_gates,lags,code)
21
22      global vc_adcint vc_sampling vc_samplingend vc_ba bm_samples bm_next
23      global lp_t1 lp_t2 lp_h lp_ra lp_nfir lp_fir  lp_dec lp_T lp_dt p_dtau
24      global lp_nt lp_vc lp_ri lp_bcs lp_code lp_ind lp_indold ra_next ra_prev
25
26      if (type~='b' & type~='c' & type~='s'),
27        error(' Unknown data type in COR_trilp')
28      end
29
30      ra_prev=ra;
31
32      adcint=vc_adcint(vc); lags=lags/p_dtau;
33      COR_check(lags,adcint)
34
35      N_maxlag=(gating-1);
36      if any(lags>N_maxlag*adcint);
37        disp(lags(lags>N_maxlag*adcint));
38        error('Lags longer than possible with this algorithm ')
39      end
40
41      N_lags=length(lags);
42      index=(lp_ind+1):(lp_ind+N_lags);
43
44      lp_T(index)=COR_caltemp(type)*ones(1,N_lags);
45
46      % find first when pulses are transmitted and samples are taken
47      pulsetimes=pulse_times(vc);
48      t1=sample_times(vc,type2ind(type));t1=t1(1);
49      lp_t1(index)=t1*ones(1,N_lags);
50      lp_h(index)=lp_t1(index)-pulsetimes(1);
51
52      % finally parameters common to background, calibration and signal
53      lp_t2(index)=lp_t1(index)+lags;
54      lp_dt(index)=adcint*ones(1,N_lags);
55      lp_nfir(index)=gating-lags/adcint;
56      for i=index
57        lp_fir(1:lp_nfir(i),i)=ones(lp_nfir(i),1); end
58      lp_dec(index)=(gating+overlap)*ones(1,N_lags);
59      lp_nt(index)=N_gates*ones(1,N_lags);
60      lp_vc(index)=vc*ones(1,N_lags);
61      lp_ra(index)=ra+(0:N_lags-1);
62      lp_ri(index)=ri*ones(1,N_lags);
63      lp_bcs(index)=type*ones(1,N_lags);
64      lp_code(index)=code*ones(1,N_lags);
65      lp_ind=lp_ind+N_lags;
66
67      % first result memory location not used
68      ra_next=max(lp_ra(index)+(lp_nt(index)-1).*lp_ri(index))+1;
69
70      COR_status('COR_trilp',vc,type,index)
```

In the GEN-system remote programs, the background and calibration ACF's have an equal number of crossed products for all lags. These calculation are handled by

/geo/gmt/askoh/guisdap/m152/COR_box.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % COR_box defines lag profiles for GEN-remote background and calibration
4       % input parameters
5       % ra      : First result memory address to use
6       % ri      : result memory address increment
7       % vc      : virtual channel number
8       % type    : background ('b'), calibration ('c') or signal ('s')
9       % gating : number of crossed products added together
10      % overlap: number of products skipped between gates (positive skipping)
11      %         : if negative, then same crossed products are used for successive gates
12      % N_gates : number of gates calculated
13      % lags    : Lags for which output needed in us, e.g. 0:10:250;
14      % code    : user specified code number
15      %
16      % See also: CORR_box, COR_check, COR_caltemp, COR_status, pulse_times
17      %function COR_lp(ra,ri,vc,type,gating,overlap,N_gates,lags,code)
```

```
18      function COR_lp(ra,ri,vc,type,gating,overlap,N_gates,lags,code)
19
20      global vc_adcint vc_sampling vc_samplingend vc_ba bm_samples bm_next
21      global lp_t1 lp_t2 lp_h lp_ra lp_nfir lp_fir lp_dec lp_T lp_dt p_dtau
22      global lp_nt lp_vc lp_ri lp_bcs lp_code lp_ind lp_indold ra_next ra_prev
23
24      if (type~='b' & type~='c'),
25        error(' Unknown data type in COR_box')
26      end
27
28      ra_prev=ra;
29
30      adcint=vc_adcint(vc); lags=lags/p_dtau;
31      COR_check(lags,adcint)
32
33      N_lags=length(lags);
34      index=(lp_ind+1):(lp_ind+N_lags);
35
36      lp_T(index)=COR_caltemp(type)*ones(1,N_lags);
37
38      % find first when pulses are transmitted and samples are taken
39      pulsetimes=pulse_times(vc);
40      t1=0;% The remote timing is still not OK
41      %t1=sample_times(vc,type2ind(type));
42      lp_t1(index)=t1(1)*ones(1,N_lags);
43      lp_h(index)=lp_t1(index)-pulsetimes(1);
44
45      % finally parameters common to background, calibration and signal
46      lp_t2(index)=lp_t1(index)+lags;
47      lp_dt(index)=adcint*ones(1,N_lags);
48      lp_nfir(index)=gating*ones(1,N_lags);
49      lp_fir(1:gating,index)=ones(gating,length(index));
50      lp_dec(index)=(gating+overlap)*ones(1,N_lags);
51      lp_nt(index)=N_gates*ones(1,N_lags);
52      lp_vc(index)=vc*ones(1,N_lags);
53      lp_ra(index)=ra+(0:N_lags-1);
54      lp_ri(index)=ri*ones(1,N_lags);
55      lp_bcs(index)=type*ones(1,N_lags);
56      lp_code(index)=code*ones(1,N_lags);
57      lp_ind=lp_ind+N_lags;
58
59
60      % first result memory location not used
61      ra_next=max(lp_ra(index)+(lp_nt(index)-1).*lp_ri(index))+1;
62
63      COR_status('  COR_box',vc,type,index)
```

Lag profiles as calculated by the UNIPROG system (Ho T., Turunen, T., Silen, J. and Lehtinen, M.: The lag profile routine and the universal program for the EISCAT digital correlators, EISCAT Technical Note 83/37). The data appears in lag profile order instead of heightwise ACF:s.

/geo/gmt/askoh/guisdap/m152/COR_uprog.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % COR_uprog defines the Uniprog lag profiles.
4       %
5       % User specified input parameters:
6       % ra      : First result memory address to use
7       % ri      : result memory address increment
8       % vc      : virtual channel number
9       % type    : lag profile type, all allowed 's','b','c','o','x'
10      % gating  : number of successive crossed products added to the same memory location
11      % N_gates : number of gates for zero lag, the number will be smaller for longer lags
12      % lags    : Lag  values in us
13      % N_skipped : number of samples skipped at the beginning of the sample vector (often =0)
14      % code     : user specified code number
15      % Output is stored in the lp_xxx parameters (global)
16      %
17      % See also: CORR_uprog, COR_pp, COR_check, COR_caltemp, COR_status, pulse_times, sample_times
18      %
19      %function COR_uprog(ra,ri,vc,type,gating,N_gates,lags,N_skipped,code)
20       function COR_uprog(ra,ri,vc,type,gating,N_gates,lags,N_skipped,code)
21
22      global p_dtau vc_adcint vc_mf
23      global lp_t1 lp_t2  lp_dt lp_h lp_nfir lp_fir lp_dec lp_T lp_nt
24      global lp_vc lp_ra lp_ri lp_bcs lp_code lp_ind lp_indold ra_next ra_prev
25
26      if (type~='b' & type~='c' & type~='s' & type~='o' & type~='x'),
27        error([' Unknown data type in COR_uprog: ' type])
28      end
29
30      ra_prev=ra;
31
```

```
32      adcint=vc_adcint(vc); lags=lags/p_dtau;
33      COR_check(lags,adcint)
34
35      N_lags=length(lags);
36      index=(lp_ind+1):(lp_ind+N_lags);
37
38      lp_T(index)=COR_caltemp(type)*ones(1,N_lags);
39
40      % find first when pulses are transmitted and samples are taken
41      pulsetimes=pulse_times(vc);
42      t1=sample_times(vc,type2ind(type));
43      if vc_mf(vc)>0, % Compensate for the delay caused by matched filter
44        t1=t1+(vc_mf(vc)-1)*adcint;
45      end
46
47      t1=(t1(1)+N_skipped*adcint)*ones(1,N_lags);
48      lp_t1(index)=t1;
49      lp_h(index)=t1-pulsetimes(1);
50
51      % finally parameters common to all types
52      lp_t2(index)=t1+lags;
53      lp_dt(index)=adcint*ones(1,N_lags);
54      lp_nfir(index)=gating*ones(1,N_lags);
55      lp_fir(1:gating,index)=ones(gating,N_lags);
56      lp_dec(index)=gating*ones(1,N_lags);
57      lp_nt(index)=N_gates-lags/adcint;
58      lp_vc(index)=vc*ones(1,N_lags);
59      lp_ra(index)=ra+ri*cumsum([0,lp_nt(index(1:N_lags-1))]);
60      lp_ri(index)=ri*ones(1,N_lags);
61      lp_bcs(index)=type*ones(1,N_lags);
62      lp_code(index)=code*ones(1,N_lags);
63      lp_ind=lp_ind+N_lags;
64
65      % first result memory location not used
66      ra_next=max(lp_ra(index)+(lp_nt(index)-1).*lp_ri(index))+1;
67
68      COR_status('COR_uprog',vc,type,index)
```

The last call in all the previous `COR_` routines is `COR_status` which displays the number of lag profiles defines, as well as how much of the result memory has been used. Certain features are used only with the experiment design package.

/geo/gmt/askoh/guisdap/m152/COR_status.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % This is an internal routine called by COR_XX routines. It displays
4       % - the number of lag profiles defined by the present call
5       % - total number of lag profiles specified up to this call
6       % - Number of samples required and total number of samples taken (design package only)
7       % - The last result memory address used
8       % In the design use, the routine calculates the sampling end time
9       % Input parameters:
10      % routine: Name of calling routine as string for output
11      % vc     : virtual channel number
12      % type   : lag profile type
13      % index  : lag profile numbers
14
15      % function COR_status(routine,vc,type,index)
16        function COR_status(routine,vc,type,index)
17
18      global vc_sampling vc_adcint vc_ba vc_ch bm_next ra_next p_dtau
19      global lp_t2 lp_nfir lp_nt lp_dec lp_ind
20
21      fprintf(['  ',routine,': '])
22      fprintf(' %3.0f lps, total %5.0f; ',length(index), lp_ind)
23
24      typeind=type2ind(type);
25      adcint=vc_adcint(vc);
26      [st_sampl,end_sampl,ind]=sample_times(vc,typeind);
27
28      % Calculate the sampling interval, if not already there
29      % i.e. calculate when used by experiment design package
30      if end_sampl==0 & length(ind)==1, % This is executed in experiment design
31        end_sampl=max(lp_t2(index)+((lp_nfir(index)+(lp_nt(index)-1).*lp_dec(index))-1)*adcint);
32        vc_sampling(ind,4)=end_sampl;
33        % Number of samples required for calculations
34        vc_ba(ind,vc)=bm_next;
35        bm_samples=1+sum(end_sampl-st_sampl)/adcint;
36        bm_next=bm_next+bm_samples;
37        fprintf(' %3.0f samples, total %5.0f;',bm_samples, bm_next-1)
38
39        % Check now that no other channel is transmitting at the time of reception
40        index=(st_sampl:end_sampl)';
```

```
41        [chs,RECch,CALon]=active_ch(index);
42        if length(chs)
43          fprintf('\n\n  ERROR\nClash in the design:\n')
44          fprintf('Channel %.0f was asked to receive  ',vc_ch(vc))
45          fprintf('from %.1f to %.1f, but\n', st_sampl*p_dtau, end_sampl*p_dtau)
46          fprintf('channel %.0f is transmitting during that period\n',chs)
47          error(' ')
48        else
49          if type~='c' & CALon
50            fprintf('\nWARNING:\nThe required reception period\n')
51            fprintf('from %.1f to %.1f\n', st_sampl*p_dtau, end_sampl*p_dtau)
52            fprintf('will be contaminated by the calibration injection\n')
53          end
54        end
55      elseif end_sampl==0 & length(ind)>1, % One never should end up here
56        error([' Error in ',routine,' Contact a GUIZARD'])
57      end
58      fprintf(' ResMem used to %5.0f\n', ra_next-1)
```

For efficiency, the `COR_init` routine initializes the `lp_`-parameters to a length greater than the number of lag profiles in most experiments. You must call it before starting to define the correlator program. The routine `COR_end` removes the unused elements from the end after the lag profiles have been completely specified. You must call this routine last.

/geo/gmt/askoh/guisdap/m152/COR_init.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % Function to initialize the lag profile variables to a sufficient size
4       % Input parameters:
5       % size: length of the matrices (optional, 5000 assumed)
6       % nfir: maximum filter coefficient length (optional, 1 assumed)
7       %
8       % See also: design, COR_end
9       %
10      %function COR_init(size,nfir)
11      function COR_init(size,nfir)
12
13      global lp_t1 lp_t2 lp_h lp_ra lp_nfir lp_fir lp_dec lp_T lp_dt
14      global lp_nt lp_vc lp_ri lp_bcs lp_code lp_ind
15      global ra_next ra_prev bm_next vc_next
16
17      lp_ind=0;
18      ra_next=0;
19      ra_prev=0;
20      bm_next=1;
21      vc_next=1;
22
23      if nargin==0, size=5000; nfir=1; end
24      if nargin==1, nfir=1; end
25      lp_t1=zeros(1,size);
26      lp_t2=zeros(1,size);
27      lp_h=zeros(1,size);
28      lp_ra=zeros(1,size);
29      lp_nfir=zeros(1,size);
30      lp_fir=zeros(nfir,size);
31      lp_dec=zeros(1,size);
32      lp_T=zeros(1,size);
33      lp_dt=zeros(1,size);
34      lp_nt=zeros(1,size);
35      lp_vc=zeros(1,size);
36      lp_ri=zeros(1,size);
37      lp_bcs=zeros(1,size);
38      lp_code=zeros(1,size);
```

/geo/gmt/askoh/guisdap/m152/COR_end.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % Function to cut the unused part of the lp_XXX parameters.
4       % Called automatically by design
5       %
6       % See also: design, COR_init
7       %
8       % function COR_end
9       function COR_end
10
11      global lp_t1 lp_t2 lp_h lp_ra lp_nfir lp_fir lp_dec lp_T lp_dt
12      global lp_nt lp_vc lp_ri lp_bcs lp_code lp_ind
13      global ra_next ra_prev bm_next vc_next
14
15      len=length(lp_t1);
16      if lp_ind<=len,
17        lp_t1(lp_ind+1:len)=[];
```

```
18        lp_t2(lp_ind+1:len)=[];
19        lp_h(lp_ind+1:len)=[];
20        lp_ra(lp_ind+1:len)=[];
21        lp_nfir(lp_ind+1:len)=[];
22        lp_dec(lp_ind+1:len)=[];
23        lp_fir(:,lp_ind+1:len)=[];
24        lp_T(lp_ind+1:len)=[];
25        lp_dt(lp_ind+1:len)=[];
26        lp_nt(lp_ind+1:len)=[];
27        lp_vc(lp_ind+1:len)=[];
28        lp_ri(lp_ind+1:len)=[];
29        lp_bcs(lp_ind+1:len)=[];
30        lp_code(lp_ind+1:len)=[];
31      end
32      [M,N]=size(lp_fir);
33      MA=max(lp_nfir);
34      if MA<M,
35        lp_fir(MA+1:M,:)=[];
36      end
37      clear lp_ind ind
```

The following routine is used to check internally the following: 1) The lag values specified by the user should be multiples `p_dtau`. 2) The lag values should also be multiples of sampling interval. 3) The lags should be multiples of baud separation in case of `COR_alter` routine.

/geo/gmt/askoh/guisdap/m152/COR_check.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % COR_check.m
4       % This function checks that the lag values are exact multiples of p_dtau and ADC interval
5       % Execution is stopped if a mismatch is found
6       % Input parameters:
7       % p_dtau: basic time unit (global)
8       % lags:   Lag values in p_dtau units
9       % adcint: sampling interval
10      % bitsep: bit separation in alternating code experiments (optional)
11      % function COR_check(lags,adcint,bitsep);
12      function COR_check(lags,adcint,bitsep)
13
14      global p_dtau
15
16      % Check if lag values are exact multiples of p_dtau
17      ind=find(abs(lags-round(lags))>1000*eps);
18      if length(ind)>0,
19         fprintf('Error: All lag values are not exact multiples of p_dtau of %5.2f us\n',p_dtau)
20         for lag=lags;
21         fprintf('Lag %4.0f divided by p_dtau is %5.1f\n',lag*p_dtau,lag);end
22         err=1;
23      end
24
25      % Check if lag values are exact multiples of adcint
26      ind=find(abs(lags-round(lags/adcint)*adcint)>1000*eps);
27      if length(ind)>0,
28         fprintf('Error: All lag values are not exact multiples of adcint of %5.1f us\n',adcint)
29         for lag=lags;fprintf('Lag %4.0f divided by adcint is %5.1f\n',lag,lag/adcint);end
30         err=1;
31      end
32
33      if nargin==3,
34        % Check if lag values are exact multiples of bitsep
35        ind=find(abs(lags-round(lags/bitsep)*bitsep)>1000*eps);
36        if length(ind)>0,
37          fprintf('Error: All lag values are not exact multiples of bitsep of %5.2f us\n',bitsep)
38          for lag=lags;fprintf('Lag %4.0f divided by bitsep is %5.1f\n',lag,lag/bitsep);end
39          err=1;
40        end
41      end
42
43      if err,
44        fprintf(' Check parameters, stopping\n')
45        error(' Error found by COR_check'),
46      end
```

The following routineare used internally to find the starting times of all pulses sent in a virtual channel and the find the sampling start times.

/geo/gmt/askoh/guisdap/m152/pulse_times.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % Find the times of the pulse leading edges. Phase transitions are not counted
4       %
5       % Parameters
```

```
6       % vc:         virtual channel number
7       % pulsetimes: times in p_dtau units
8       %
9       % See also: sample_times
10      %
11      % function pulsetimes=pulse_times(vc)
12        function pulsetimes=pulse_times(vc)
13
14      global vc_env vc_envo
15
16      % find lines that belong to the virtual channel and contain transmission
17      %ind=find(td_ch==vc_ch(vc) & td_t1>=vc_t1(vc) & td_t2<=vc_t2(vc) & abs(td_am)==1);
18      %pulsetimes=td_t1(ind);
19
20      % Calculation based on stored envelopes. Note that the result is not the same
21      % for coded pulses. This version gives only the start time of the whole pulse,
22      % whereas the one above gives also all the phasechanges.
23      ind=find([0; vc_env(:,vc)]==0 & [vc_env(:,vc); 0]~=0);
24      pulsetimes=vc_envo(vc)+ind+1-abs(vc_env(ind,vc));
```

/geo/gmt/askoh/guisdap/m152/sample_times.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % returns the sampling times for a virtual channel.
4       % Input parameters:
5       % vc_sampling (global): matrix of sampling start and end times
6       %                       Each row is [vc, type, starttime, endtime];
7       % vc    : virtual channel number
8       % stype : reception type in coded form
9       % Output parameters:
10      % start_sampling: Sampling start times for the virtual channel and reception type
11      %  stop_sampling: Sampling   end times for the virtual channel and reception type
12      % ind : Lines in the vc_sampling matrix, which contain the data
13      %
14      % See also: pulse_times, type2ind
15      % function [start_sampling, stop_sampling,ind]=sample_times(vc,stype)
16        function [start_sampling, stop_sampling,ind]=sample_times(vc,stype)
17
18      global vc_sampling p_rep vc_t1 vc_ch
19
20      % For most cases, this line is what is sufficient
21      ind=find(vc_sampling(:,1)==vc & vc_sampling(:,2)==stype);
22
23      % Now certain virtual channels do not have background and calibration
24      % because they are measured at time defined for some other virtual channel
25      if length(ind)==0 & stype==type2ind('s');
26        fprintf(' \nSerious case, signal sampling missing for virtual channel %.0f\n',vc)
27        dbstop error, error(' Entering debug mode')
28      elseif  length(ind)==0 & (stype==type2ind('b') | stype==type2ind('c'));
29        ind=[]; vct=[];
30        for vcs=find(vc_ch==vc_ch(vc) & 1:length(vc_ch)~=vc); % these vc on the same channel
31          ind1=find(vc_sampling(:,1)==vcs & vc_sampling(:,2)==stype)'; % Background/calibration
32          ind=[ind ind1];
33          vct=[vct vcs*ones(size(ind1))];
34        end
35        if length(ind)>0; % Background/calibration found
36          starts=[vc_sampling(ind,3) vc_sampling(ind,3)+p_rep]; % To handle the last vc on channel
37          ind=[ind ind];
38          vct=[vct vct];
39          ind1=find(starts>vc_t1(vc)); % find sampling times following the vc
40          ind=ind(ind1(1)); % take the first one as the best guess    This may be wrong
41      %    fprintf(' \nTaking back/cal sampling times to virtual channel %.0f from channel %.0f\n',vc, vct(ind1))
42        elseif length(ind)==0; % No badkground/calibration found
43          fprintf(' \nSerious case, backgr/cal sampling missing for real channel %.0f\n',vc_ch(vc))
44          dbstop error, error(' Entering debug mode')
45        end
46      end
47
48      start_sampling=vc_sampling(ind,3);
49      stop_sampling=vc_sampling(ind,4);
50
```

The following routine sets the calibration temperature

/geo/gmt/askoh/guisdap/m152/COR_caltemp.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % COR_caltemp.m
4       %
5       % sets the calibration temperature
6       % function Temp=COR_caltemp(type)
7
8       function Temp=COR_caltemp(type)
9
```

```
10      global p_calTemp
11
12      if type=='c' | type=='C'  % Calibration measurement
13        Temp=p_calTemp;
14      else
15        Temp=0;
16      end
```

Other routines called by EISCAT initialization

The `pat_PS.m` if load with the `load_PS` call. It executes the file, puts the contents in the various `td_`-parameters and stores the contents in a `pat_PS.mat` file. The latter operation is done for monostatic experiments, for which the file is large and the execution take quite a long time. During subsequent calls, the parameters can be rapidly loaded from the `pat_PS.mat` file. For remotes, the loading is done with the `load_PSrem.m` routine, which combines the information in the monostatic and remote files together.

`/geo/gmt/askoh/guisdap/m152/load_PS.m`

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % loads in the timing diagram variables for radar controller program number
4       % 'rcprog'. If there are multiple programs for the experiment, 'Nrcprog'
5       % must specify the total number
6       %
7       % See also: path_expr init_EISCAT
8       % function load_PS(rcprog, Nrcprog)
9         function load_PS(rcprog, Nrcprog)
10
11      global path_exps name_expr name_site
12      global td_ch td_t1 td_t2 td_am p_rep ch_f ch_filter ch_adcint
13
14      if nargin==0, rcprog=1; Nrcprog=1;
15      elseif nargin==1, Nrcprog=1; end
16      if Nrcprog>1, apustr=['_',int2str(rcprog)]; else apustr=[]; end
17
18      create=1;
19      file=canon([ path_expr name_expr name_site apustr 'pat_PS.mat']);
20      if exist(file)==2,
21        fprintf(' Loading existing pat_PS.mat from file\n')
22        load(file),
23        if exist('ch_f')==1 & any(td_ch==0),
24          return
25        elseif all(td_ch~=0)   % No calibration data in the file
26          fprintf(' No calibration times in the file: Creating new one\n')
27        else
28          fprintf(' Existing pat_PS.mat file is not up-to-date: Creating new one\n')
29        end
30      end
31
32      if create
33        cdir(path_expr)
34        td_len=10000;
35        td_ind=0;
36        td_ch=zeros(1,td_len);
37        td_t1=zeros(1,td_len);
38        td_t2=zeros(1,td_len);
39        td_am=zeros(1,td_len);
40        ch_f=-1*ones(1,8);
41
42        eval(canon([name_expr name_site apustr 'pat_PS'],0))
43        disp([name_expr name_site apustr 'pat_PS passed'])
44
45        if td_ind<td_len;
46          td_ch(td_ind+1:td_len)=[];
47          td_t1(td_ind+1:td_len)=[];
48          td_t2(td_ind+1:td_len)=[];
49          td_am(td_ind+1:td_len)=[];
50        end
51        if all(ch_f==-1) | all(td_ch~=0)
52          row(ch_f), row(td_ch)
53          fprintf(' The pat_PS.m file is not up-to-date\n')
54          fprintf(' Create again with new version of TLAN2PS before proceeding\n')
55          fprintf(' Contact a GUISPERT or GUIZARD if needed\n'),error(' ')
56        end
57        clear r
58
59        eval(canon(['save ' file ' td_ch td_t1 td_t2 td_am p_rep ch_f ch_filter ch_adcint']));
60      end
```

`/geo/gmt/askoh/guisdap/m152/load_PSrem.m`

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
```

```
%
% script for loading td_variable for EISCAT remotes and for connecting the transmitter
% frequencies with the receiver channels.
%
% See also: load_PS
load_PS(d_rcprog,N_rcprog)
% For remotes we must find transmission from Tromso files
R_ch=td_ch; R_f=ch_f; % Store the necessary remote variables
name_site='T';
load_PS(d_rcprog,N_rcprog)
ind=[]; % ind will contain the lines containing transmission to remotes
Rch=[]; % will tell in which channel they are received
for ch=diff_val(R_ch(R_ch>0)); %These channels in use
  chT=find(ch_f==R_f(ch)); % This is the Tromso channel for the transmission
  index=find(td_ch==chT & abs(td_am)==1);
  ind=[ind, index];
  Rch=[Rch, ch*ones(size(index))];
end
T_t1=td_t1(ind);
T_t2=td_t2(ind);
T_am=td_am(ind);
name_site='R';
load_PS(d_rcprog,N_rcprog) % load back the remote variables
td_ch=[Rch, td_ch];
td_t1=[T_t1, td_t1];
td_t2=[T_t2, td_t2];
td_am=[T_am, td_am];
```

The following routine is used to interpret timing data in `td_`-parameters and to created the GUISDAP parameter `vc_env` containing transmission waveforms for each virtual channel. It also checks, if matched filters are used and correspondingly modifies the impulse responces in `vc_p`.

/geo/gmt/askoh/guisdap/m152/envcalc.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% Script to calculate vc_env and vc_p variables from td_variables
%
% See also: impulse_resp Barker
%
Nvc=length(vc_ch);
maxlen=600/p_dtau;
vc_env=zeros(maxlen,Nvc);
vc_envo=zeros(1,Nvc);
vc_p=zeros(maxlen,Nvc);
lenmax=zeros(1,2); % order vc_env, vc_p

impuls_resp; % create impulse responses for receiver channels

fprintf('Virtual channel: ')
for vc=find(vc_ch>0)
  fprintf(' %.0f',vc)
  envel=0;
  ind=find(td_ch==vc_ch(vc) & abs(td_am)==1 ...
           & td_t1>=vc_t1(vc) & td_t2<=vc_t2(vc) );
  start_env=floor(min(td_t1(ind)))-2;
  for i=ind
    clear enve
    alku=td_t1(i);
    loppu=td_t2(i);
    alku_c=ceil(alku);
    loppu_f=floor(loppu);
    enve(alku_c-1-start_env,1)=alku_c-alku;
    if loppu_f>alku_c
      enve((alku_c:(loppu_f-1)) -start_env,1)=ones(loppu_f-alku_c,1);end
    enve(loppu_f-start_env,1)=loppu-loppu_f;
    ii=(alku_c-1:loppu_f)-start_env;
    maxi=ii(length(ii));
    if length(envel)<maxi, envel(maxi,1)=0;end
    envel(ii)=envel(ii)+td_am(i)*enve(ii);
  end
  len=length(envel);
  if envel(len)~=0; envel(len+1)=0; len=len+1; end
  vc_env(1:len,vc)=envel;
  vc_envo(vc)=start_env;
  lenmax(1)=max(lenmax(1),len);
%  plot((0:len-1)*p_dtau,envel)  ,keyboard

  ind=1:max(find(ch_p(:,vc_ch(vc))~=0));
  impresp=ch_p(ind,vc_ch(vc));
  if vc_mf(vc)>0,
    B=Barker(vc_mf(vc));
    adcint=vc_adcint(vc); temp=[];
    len=length(impresp);
```

```
51          for i=1:length(B);
52            ind=(i-1)*adcint+1:(i-1)*adcint+len;
53            temp(max(ind),1)=0;
54            temp(ind)=temp(ind)+B(i)*impresp;
55          end
56          impresp=temp;
57        end
58        len=length(impresp);
59        if impresp(len)~=0; impresp(len+1)=0; len=len+1; end
60        vc_p(1:len,vc)=impresp;
61        lenmax(2)=max(lenmax(2),len);
62      %  plot((0:len-1)*p_dtau,impresp),keyboard
63      end
64
65      vc_env(lenmax(1)+1:maxlen,:)=[];
66      vc_p(lenmax(2)+1:maxlen,:)=[];
67
68      plot((0:lenmax(2)-1)*p_dtau,vc_p),title('Filter impulse responses')
69
70      clear Nvc maxlen vc ind start_env i alku loppu alku_c loppu_f env ii
71      clear len impresp adcint temp enve envel lenmax maxi vc
```

The sampling information contained in the `td_` parameters is interpreted by the `find_sampling` routine. It stores the information in a matrix, where the first column gives the virtual channel, the next the type of sampling (signal, background or calibration), the third the start of sampling and the last one the sampling end time. The information is retrieved by the `sample_times` routine when necessary.

/geo/gmt/askoh/guisdap/m152/find_sampling.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % finds sampling times for all virtual channel and stores results
4       % to a global variable vc_sampling
5       % NOTE: This is a EISCAT specific function
6       %
7       % See also: init_EISCAT
8       % function find_sampling
9         function find_sampling
10
11      global vc_sampling vc_ch vc_t1 vc_t2 vc_envo vc_env
12      global td_ch td_t1 td_t2 td_am p_rep
13
14      s_ind=1;
15      vc_sampling=zeros(1000,4);
16
17      ind1=find(td_ch==0);  % indicates calibration times
18      cal_start=[td_t1(ind1)]; % AH 94-04-20 Changed here due to change in td_arrange
19      cal_stop=[td_t2(ind1)];
20
21      for vc=find(vc_ch>0) % These channels in use
22        % find lines that belong to the virtual channel
23        ind=find(td_ch==vc_ch(vc) & td_t1>=vc_t1(vc) & td_t2<=vc_t2(vc) & td_am==2);
24        if length(ind)==0,
25          fprintf(' Error in virtual channel specification?\n')
26          fprintf(' Virtual channel %.0f contains no sampling intervals\n',vc)
27          error(' ')
28        end
29        t1=td_t1(ind);
30        t2=td_t2(ind);
31        am=td_am(ind);
32
33        % Look first for the signal sampling, it should be there. The logic is
34        % that the first sampling period after the transmission is the signal sampling
35        ind1=find(t1>vc_envo(vc));
36        [dummy,ind]=min(t1(ind1));
37        ind1=ind1(ind);
38        vc_sampling(s_ind,:)=[vc,type2ind('s'),t1(ind1),t2(ind1)];
39        s_ind=s_ind+1;
40
41        for i=1:length(t1); % Look if calibration sampling done
42          if any(cal_start<=t1(i) & t2(i)<=cal_stop),
43            vc_sampling(s_ind,:)=[vc,type2ind('c'),t1(i),t2(i)];
44            s_ind=s_ind+1;
45            ind1=[ind1, i];
46          end
47        end
48        t1(ind1)=[];
49        t2(ind1)=[];
50        am(ind1)=[];
51
52        len=length(t1); % The ones remaining must be background
53        vc_sampling(s_ind+(0:len-1),:)=[vc*ones(len,1),type2ind('b')*ones(len,1),t1',t2'];
54        s_ind=s_ind+len;
55      end
```

```
56
57      len=length(vc_sampling(:,1));
58      vc_sampling(s_ind:len,:)=[];
```

This routine arranges timing specifications in cases, where the virtual channel time interval runs from one repetition period to the next one. (It adds `p_rep` to specified timings, where necessary).

/geo/gmt/askoh/guisdap/m152/td_arrange.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % This script checks the user defined variables and the td_-variables and
4       % corrects them in order to make programming of other functions/scripts easier
5       %
6       % Warning: This is NOT a user callable routine
7       %
8       % See also: vc_arrange
9
10      % Produce variable vc_mf if it was not defined by the user
11      if ~exist('vc_mf'), vc_mf=zeros(1,length(vc_t1)); end
12      if length(vc_mf)==0, vc_mf=zeros(1,length(vc_t1)); end
13
14      % Scale all time variables by the basic time unit
15      td_t1=td_t1/p_dtau; td_t2=td_t2/p_dtau;
16      vc_t1=vc_t1/p_dtau; vc_t2=vc_t2/p_dtau;
17      p_rep=p_rep/p_dtau;
18
19      % Produce another set of td-variables by adding p_rep to all
20      % This increases flexibility in the virtual channel definition AH 94-04-20
21      td_ch=[td_ch,td_ch];
22      td_am=[td_am,td_am];
23      td_t1=[td_t1,td_t1+p_rep];
24      td_t2=[td_t2,td_t2+p_rep];
25
26
27      % Produce ADC intervals for all virtual channels in units of p_dtau
28      vc_adcint=zeros(1,length(vc_ch));
29      for vc=find(vc_ch>0)   % These virtual channels are in use
30        vc_adcint(vc)=ch_adcint(vc_ch(vc))/p_dtau;
31      end
32
33      ind=find(vc_t1<0);
34      if length(ind)>0,vc_t1(ind)=vc_t1(ind)+p_rep;end
35      for vc=find(vc_ch>0)   % These virtual channels are in use
36        % Find the lines belonging to the virtual channel
37        if vc_t1(vc)<vc_t2(vc),
38          ind=find(td_ch==vc_ch(vc) & td_t1>=vc_t1(vc) & td_t2<=vc_t2(vc) );
39        else   % This is needed when the virtual channel extends to the next rep
40          ind=find(td_ch==vc_ch(vc) & (td_t1>=vc_t1(vc) | td_t2<=vc_t2(vc)) );
41          % Add p_rep to those td_t1 and td_t2 and vc_t2, which are smaller than vc_t1.
42          ind1=find(td_t1(ind)<vc_t1(vc));
43          ind2=find(td_t2(ind)<vc_t1(vc));
44          if all(ind1==ind2),
45            td_t1(ind(ind1))=td_t1(ind(ind1))+p_rep;
46            td_t2(ind(ind1))=td_t2(ind(ind1))+p_rep;
47            vc_t2(vc)=vc_t2(vc)+p_rep;
48          else
49            error('Error in td_arrange, something funny with td_t1 and td_t2')
50          end
51        end
52
53      end
54      clear vc ind ind1
```

The next routine defines the global variables

/geo/gmt/askoh/guisdap/m152/glob_EISCAT.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % All the global variables needed to run init_EISCAT
4       %
5       % See also: init_EISCAT, globals, glob_GUP, start_GUP
6
7       global ch_f ch_p ch_adcint ch_filter ch_fradar ch_gain
8
9       global lp_t1 lp_t2 lp_dt lp_nt lp_vc lp_ra lp_ri lp_T lp_code lp_bcs lp_h lp_ind
10      global lp_nfir lp_fir lp_dec
11
12      global p_dtau  p_rep p_ND ra_prev ra_next
13      global p_XMITloc p_RECloc p_calTemp
14
15      global td_ch td_t1 td_t2 td_am p_rep
16
17      global vc_ch vc_t1 vc_t2 vc_adcint vc_p vc_env vc_envo vc_sampling vc_mf
```

```
18    global vc_ba bm_next % included to  achieve compatibility with the experiment design system
```

The following three routines give the EISCAT filter impulse responses. The routine **impulse_resp** is the main routine and it is called by **envcalc**. It makes calls to **but_filt** and **lin_filt** in order to produce impulse responses of proper width and type. The responses are stored to variable **ch_p**

/geo/gmt/askoh/guisdap/m152/impuls_resp.m

```
1     % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2     %
3     % Calculates the filter impulse responses, based on the data saved
4     % in the variable ch_filter by the PS_LIFILT and PS_BUFILT routines
5     %
6     % See also: lin_filt but_filt
7
8     ch_p=[];
9     channels=find(ch_filter(1,:)==1 | ch_filter(1,:)==2); % these channels in use
10    while length(channels)>0,
11      filtertype=ch_filter(1,channels(1));
12      BW=ch_filter(2,channels(1));
13      ind=find(ch_filter(1,channels)==filtertype & ch_filter(2,channels)==BW);
14      if filtertype==1,
15        but_filt(BW,channels(ind));
16      elseif  filtertype==2,
17        lin_filt(BW,channels(ind));
18      end
19      channels(ind)=[];
20    end
21    clear filtertype BW ind channels
```

/geo/gmt/askoh/guisdap/m152/but_filt.m

```
1     % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2     %
3     % Interpolates the butterworth filter impulse response function
4     % from the values measured by Jussi Markkanen
5     %
6     % input parameters:
7     % BW     : desired bandwidth in kHz (EISCAT definition)
8     % ch     : specifies the channel numbers
9     % p_dtau : global time unit in us (global parameter)
10    % output parameters:
11    % impresp: the impulse response
12    % ch_p   : impulse resposes for channels given by ch (global)
13    %
14    % See also: lin_filt, REC_impresp
15    %
16    % function impresp=but_filt(BW,ch);
17      function impresp=but_filt(BW,ch);
18
19      global ch_p p_dtau
20
21    %
22    % measured impulse response of a 25 kHz Butterworth filter.
23    % values given for each us.
24      p_m=[ -0.002 -0.003 -0.002 -0.001  0.001  0.006  0.013  0.023  0.036 ;
25             0.053  0.072  0.095  0.120  0.148  0.176  0.204  0.233  0.260 ;
26             0.285  0.307  0.327  0.342  0.353  0.360  0.362  0.360  0.353 ;
27             0.342  0.327  0.309  0.287  0.263  0.236  0.208  0.179  0.150 ;
28             0.121  0.093  0.065  0.040  0.016 -0.006 -0.025 -0.041 -0.055 ;
29            -0.067 -0.075 -0.081 -0.084 -0.085 -0.084 -0.081 -0.077 -0.070 ;
30            -0.063 -0.055 -0.047 -0.038 -0.029 -0.020 -0.012 -0.004  0.004 ;
31             0.010  0.016  0.021  0.025  0.028  0.030  0.031  0.032  0.031 ;
32             0.030  0.029  0.026  0.024  0.021  0.018  0.014  0.011  0.008 ;
33             0.004  0.001 -0.002 -0.004 -0.006 -0.008 -0.010 -0.011 -0.012 ;
34            -0.012 -0.013 -0.012 -0.012 -0.011 -0.010 -0.009 -0.008 -0.006 ;
35            -0.005 -0.003 -0.002  0.000  0.001  0.002  0.004  0.005  0.005 ;
36             0.006  0.007  0.007  0.007  0.007  0.007  0.007  0.006  0.006 ;
37             0.005  0.004  0.004  0      0      0      0      0      0];
38      p_m=col(p_m');
39      len = length(p_m);
40
41      dt  = 25/BW;  % time step for the desired bandwidth
42
43    % interpolation
44      p_mx= ((0:(len-1))*dt)';
45      px  = (0:p_dtau:(len-1)*dt)';
46      p = spline(p_mx,p_m,px);
47
48    % scale to get unit area and
49    % assign to all channels specified at input
50
51      impresp=p/sum(p);
52      if nargin==2,
53       ch_p(1:length(px),ch) = impresp*ones(1,length(ch));
```

```
54        end
```

/geo/gmt/askoh/guisdap/m152/lin_filt.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % Interpolates the linear phase filter impulse response function
4       % from the values measured by Jussi Markkanen
5       %
6       % input parameters:
7       % BW     : desired bandwidth in kHz (EISCAT definition)
8       % ch     : specifies the channel numbers (optional)
9       % p_dtau : global time unit in us (global parameter)
10      % output parameters:
11      % impresp: the impulse response
12      % ch_p   : impulse resposes for channels given by ch (global)
13      %
14      % See also: but_filt, REC_impresp
15      %
16      % function impresp=lin_filt(BW,ch);
17        function impresp=lin_filt(BW,ch);
18
19        global ch_p p_dtau
20      %
21      % measured impulse response of a 35.4 kHz linear phase filter.
22      % values given for each us.
23        p_m =[ 0.001 0.005 0.017  0.045  0.090  0.150  0.220  0.291 0.355 0.403 ...
24               0.432 0.438 0.422  0.387  0.339  0.282  0.223  0.166 0.115 0.073 ...
25               0.040 0.017 0.003 -0.005 -0.007 -0.006 -0.004 -0.001 0.002 0.003]';
26        len = length(p_m);
27
28        dt  = 35.4/BW; % time step for the desired bandwidth
29
30      % interpolation
31        p_mx= ((0:(len-1))*dt)';
32        px  = (0:p_dtau:(len-1)*dt)';
33        p = spline(p_mx,p_m,px);
34
35      % scale to get unit area and
36      % assign to all channels specified at input
37
38        impresp=p/sum(p);
39        if nargin==2,
40         ch_p(1:length(px),ch) = impresp*ones(1,length(ch));
41        end
```

The following routines are used to build `td_`-varibles on the basis of the `pat_PS.m`-files, made by TLANtoPS from EISCAT TLAN and ELAN files. The routines also read in the filter type and width information as well as the analog–to–digital conversion intervals.

/geo/gmt/askoh/guisdap/m152/PS_BUFILT.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % [ch BW] ;PS_BUFILT
4       r=ans;
5       ch_filter(1:2,r(1))=[1;r(2)];
```

/geo/gmt/askoh/guisdap/m152/PS_LIFILT.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % [ch BW] ;PS_LIFILT
4       r=ans;
5       ch_filter(1:2,r(1))=[2;r(2)];
```

/geo/gmt/askoh/guisdap/m152/PS_CALON.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % [ch t1 t2] ;PS_CALON
4       r=ans;
5       td_ind=td_ind+1;
6       td_ch(td_ind)=0;
7       td_t1(td_ind)=r(2);
8       td_t2(td_ind)=r(3);
9       td_am(td_ind)=1;
```

/geo/gmt/askoh/guisdap/m152/PS_EMPTY.m

```
1       % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2       %
3       % [ch t1 t2] ;PS_EMPTY
4       r=ans;
5       p_rep=r(3);
```

/geo/gmt/askoh/guisdap/m152/PS_MATCHFIL.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% [t1 t2] ;PS_MATCHFIL
```

/geo/gmt/askoh/guisdap/m152/PS_ON.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% [ch t1 t2] ;PS_ON
r=ans;
td_ind=td_ind+1;
td_ch(td_ind)=r(1);
td_t1(td_ind)=r(2);
td_t2(td_ind)=r(3);
td_am(td_ind)=2;
```

/geo/gmt/askoh/guisdap/m152/PS_PHA.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% [ch t1 t2] ;PS_PHA
r=ans;
td_ind=td_ind+1;
td_ch(td_ind)=r(1);
td_t1(td_ind)=r(2);
td_t2(td_ind)=r(3);
td_am(td_ind)=1;
```

/geo/gmt/askoh/guisdap/m152/PS_PHAN.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% [ch t1 t2] ;PS_PHAN
r=ans;
td_ind=td_ind+1;
td_ch(td_ind)=r(1);
td_t1(td_ind)=r(2);
td_t2(td_ind)=r(3);
td_am(td_ind)=-1;
```

/geo/gmt/askoh/guisdap/m152/PS_REP.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% [t] ;PS_REP
```

/geo/gmt/askoh/guisdap/m152/PS_SETADCINT.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% [ch dt] ;PS_SETADCINT
r=ans;
ch_adcint(r(1))=r(2);
```

/geo/gmt/askoh/guisdap/m152/PS_SETFREQ.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% [ch f] ;PS_SETFREQ
r=ans;
ch_f(r(1))=r(2);
```

The GUISDAP parameters are stored into a file by

/geo/gmt/askoh/guisdap/m152/save_GUPvar.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% Script to save GUP variables to a file
%
% See also: design path_expr

GUP_iniver=GUP_ver;

lp_firsto=sparse(diff([zeros(size(lp_vc));lp_fir]));

str='GUP_iniver ch_fradar ch_gain p_dtau p_rep p_ND p_XMITloc p_RECloc';
str=[str ' vc_ch vc_env vc_envo vc_p vc_adcint vc_sampling lp_dec lp_firsto lp_nfir'];
str=[str ' lp_T lp_bcs lp_code lp_dt lp_h lp_nt lp_ra lp_ri lp_t1 lp_t2 lp_vc'];
if ~exist('apustr'), apustr=''; end
eval([canon(['save ' path_expr name_expr name_site apustr 'GUPvar.mat ']) str]);
```