Focus Microwaves Inc.
277 Lakeshore Road
Pointe-Claire, Quebec H9S-4L2, Canada
Tel 514-630-6067 Fax 514-630-7466

Product Note No 15

FOCUS.LIB
A C-Library for Tuner Control and GPIB Operation

This note describes the routines and operation of the FOCUS.LIB,
a C language library that permits to
- Initialize and control the position of the programmable tuners
- Tune to any interpolated Impedance using actual calibration files
- Communicate with programmable Instruments using the GPIB

Load Pull and Noise measurement programmes can easily be compiled by the Users themselves.

Introduction

Existing Load Pull and Noise measurement systems permit simple User programming in Basic or Pascal languages. This operation mostly consists into moving the tuners to certain mechanical positions and make GPIB instrument readings.
Tuning to desired impedances is not possible and tuner loss, the key issue of an automatic Load Pull or Noise setup, is not considered.

Focus Microwaves has assembled all routines of the CCMT (Computer Controlled Microwave Tuner) system required to control the tuners, tune to any impedance within the calibration range and exchange data with any programmable instrument via GPIB into a single C language library: FOCUS.LIB.

The average User, even with little programming experience in C, is able to write complete executable measurement programmes using the tuners, their calibration data and any GPIB programmable instrument.

The routines of FOCUS.LIB are callable from Pascal and Quick Basic programmes.

Two examples are presented: One for **Load Pull** and one for **Noise** measurement programmes compiled using exclusively routines of the FOCUS.LIB.

June 1994

Routines of FOCUS.LIB

1. GPIB Communication Routines

The GPIB section of FOCUS.LIB includes the following routines:

gpib_open (void)
gpib_close (void)
These two routines permit to initialize the GPIB interface and define the IEEE file handler in order to communicate with programmable instruments.

gpib_send (char *message, int adr)
gpib_read (char *message, int adr)
These two routines permit to send and receive character strings (message) to and from any GPIB instrument at address (adr).

gpib_timeout (int t)
This routine permits to set the Time Out of the GPIB card for operating with a particular instrument.

2. TUNER CONTROL Routines

The Tuner Control section of FOCUS.LIB includes the following routines:

tuner_to (int unit, long xpos, int ypos)
tuner_init (int unit, int axis)
tuner_speed (int sp)
The first routine moves the tuner to an absolute position XPOS (horizontal) and YPOS (vertical).
The second routine moves the particular tuner to the Zero position (XPOS=YPOS=0).
The third routine can be used to modify the speed (in steps/second) at which the tuners move.

3. TUNING Routines

The Tuning section of FOCUS.LIB includes the following routines:

allocate_cal(int points)
This routine is used to allocate dynamic memory for the tuner calibration data.

load_tuner_cal (char *path, int unit, int cal_no, float freq, char *title)
tune_to (int unit, int side, float *pol_target, float *load_gamma, float *loss)
These two routines are used to load a tuner calibration file into active memory and to

tune to a required Impedance. All tunable impedances within the step resolution of the tuners (not only calibrated points) can be synthesized using this routine. The tuner calibration files must have been created using the tuner calibration software of Focus Microwaves. Each time the **tune_to( )** routine is called the user has access to the interpolated S-parameters of the tuner network. He also has access to the tuner calibration data (S-parameters at the calibration points) themselves in order to be able to cascade with other twoports.

tunerloss (int unit, int loss_type, float *gpolar)
This routine permits to compute the available or the power loss of the actual tuner at the actual tuner position. It uses, like the **tune_to( )** routine interpolated values and is not limited to calibrated points only.

Both the **tune_to( )** and **tunerloss( )** routines operate with a non-zero reflection factor connected to the idle (non DUT facing) port of the tuner. This may be either the source or the load reflection factor.

invert_tuner (int unit)
This routine is used to inverse the S-parameters of a tuner calibration file in order to be able to connect port 2 of a tuner to the DUT.

4. Utility Routines

These routines are updated regularly for specific applications. Presently they include:

s_casc (float *spar1, float *spar2, float *casc_s)
This routine cascades the S-parameters of two twoports 1 and 2 and returns the result in the array casc_s [8].

gauss_elimination (float *nm, float *gm, float **gamma, int n, float *npar, float *gpar)
This routine will compute the four Noise and the four Gain parameters of a twoport if we pass to it the noise / gain measurement data in form of the arrays nm [n] and gm [n] together with the source impedances gamma [2][n] at n impedance points. The result will be returned in form of 4 noise (npar [4]) and 4 gain (gpar [4]) parameters.

Software Requirements

FOCUS.LIB is written in MicroSoft® C programming language using large model.
It can be linked with any C, Pascal or Quick Basic programme on IBM® PC's.

To generate executable programmes in either of the above languages a Compiler, Linker and Library Manager are required.

Only basic programming knowledge is required to generate executable programmes that permit to tune and read instruments in order to make Load Pull or Noise measurements; this is demonstrated by the following two examples.

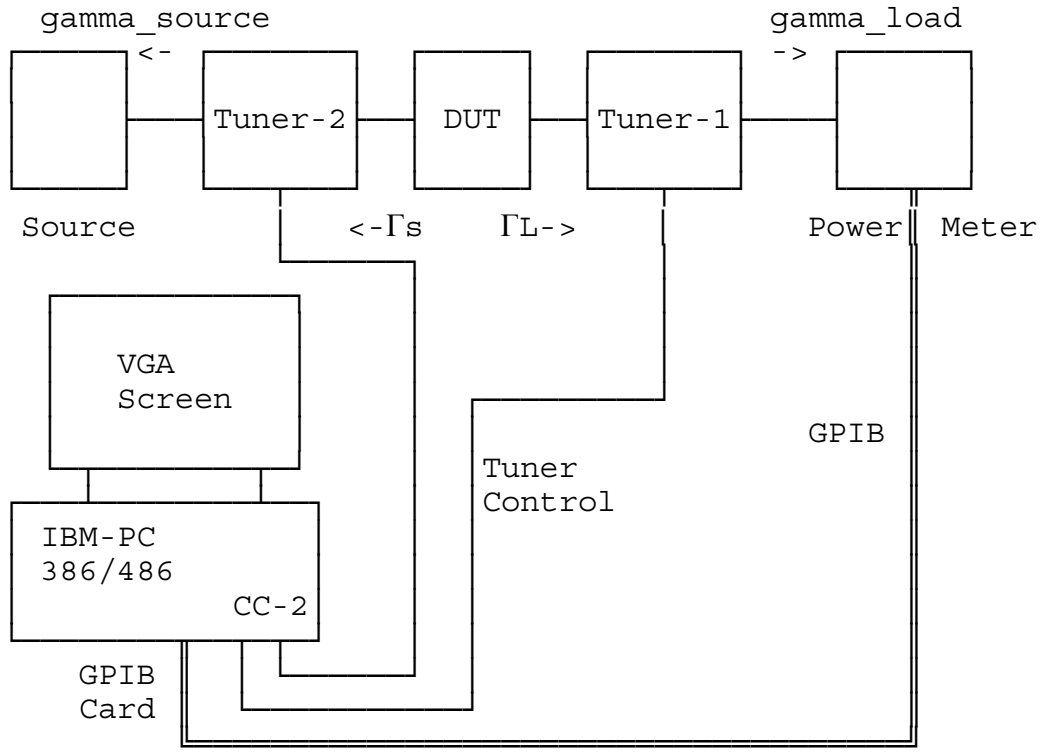
Programming Example 1: Load Pull (Power, Gain) Measurements

The following is an example of source code in C that can be compiled using MS C compiler and linked to the FOCUS.LIB in order to generate an executable programme
that measures the output power and Gain of a DUT.

```
/*   Comment:
     This programme loads calibration data from disk, inverts tuner 2 and tunes both tuners to an
impedance. It also initializes and reads the power from the Boonton® 4200  power meter
     connected to the output of the setup. The power read is multiplied by the
     power loss of the output tuner and the gain is computed for a given injected power
*/
#include <stdio.h>
#include "tuner.h"
#include <stdlib.h>
#include <conio.h>
#include <math.h>

 int unit,posv[2]={0,0},stph[2]={0,0},stpv[2]={0,0},speed=400,port=544,ypos[2];
 long posh[2]={0L,0L},xpos[2];
 int _WAIT,errnox,ieee,dir;
 float * * *tunerdat,R_FACTOR[2],lambda_zero,step_size[2];
 double tuner_s[8];
 unsigned caldat[2][10][5],limit[2][4];

main( )
{
int calno,adr,i,j,k;
float freq,out_loss,in_loss,power,pin,gamma[2],gamma_load[2],gamma_source[2],
      gain,relsp;
char str[80],comment[80];
FILE *f;

    /* Initialize the tuners */
    printf("\n   Initializing Tuner 1 (y/n) ");
    gets(str);
    if(*str=='y')
      tuner_init(1,0);
    printf("\n   Initializing Tuner 2 (y/n) ");
    gets(str);
    if(*str=='y')
       tuner_init(2,0);
```

```
    /* Load calibration for tuners */
    allocate_cal(385);
    printf("\n      Enter Frequency [GHz] = ");
    gets(str);
    freq = atof(str);
    printf("\n      Enter Cal Number for Tuner 1 = ");
    gets(str);
    calno = atoi(str);

    i = load_tuner_cal( "C:\\CCMT\\CAL" ,1 ,calno, freq, comment);

    /* Detect and Process Error messages */
    switch(i) {
          case -1:printf("\n   Wrong cal number \n\n"); exit( i );
          case -2:printf("\n   Freq Out of Range \n\n"); exit ( i );
          case -3:printf("\n   Wrong Unit \n\n"); exit ( i );
          case -4:printf("\n   Path Not Found ! Use double '\\'   \n\n"); exit ( i );
          case -5:printf("\n   Cal File Not Found \n\n"); exit ( i );
    }
    printf("\n      Tuner 1: Loaded %d Points ", i );
    printf("\n      Tuner 1 Comment = %s ", comment );  /* Displays Comment of Tuner Cal */

    printf("\n\n      Enter Cal Number for Tuner 2 = ");
    gets(str);
    calno = atoi(str);
    i = load_tuner_cal("C:\\CCMT\\CAL", 2, calno, freq, comment );

    invert_tuner( 2 );   /* Invert S-parameter of tuner 2, since it is connected to DUT with port 2 */

    /* Configure the Boonton 4200 Power Meter*/
    gpib_open( );
    printf("\n   Enter Address of Boonton 4200 Power Meter =  ");
    gets(str);
    adr = atoi(str);
    gpib_send ("CLEAR",adr);
    gpib_send ("MS",adr);
    gpib_send ("DB",adr);
    gpib_send ("RA",adr);
    gpib_send ("FA",adr);
    printf("\n     Power Meter Configured.  ");

/* Define the Target Reflection Factor */
    gamma[0] = 0.5;
    gamma[1] = 90.;            /* degrees */
    gamma_load[0] = 0.05;
    gamma_load[1] = 45.0;    /* degrees */
    gamma_source[0] = 0.1;
    gamma_source[1] = 90.;   /* degrees */
```

```
    printf("\n\n     Enter Input Power [dBm]  ");  /* Input Power entered by keyboard */
    gets(str);
    pin = atof(str);

    tune_to( 2, INPUT, gamma, gamma_source, &in_loss );     /* Tunes both tuners to target and */
    tune_to( 1, OUTPUT, gamma, gamma_load, &out_loss );   /* determines Tuner Loss   */
    printf("\n\n     Input Loss = %.2f dB , Output Loss = %.2f dB",10*log10(in_loss),
            10*log10(out_loss));;
    gpib_send ("1N",adr);   /* Configure channel 1 */
    gpib_read ( str, adr );
    sscanf(str,"%*3c%f,%d,%d",&power,&j,&k); /* Reads response, Power is in dBm */
                                        /* j,k are dummy variables  */
    power += 10*log10( out_loss );     /* Corrects for output power loss */
    gain = power- ( pin-10*log10( in_loss ) );
    printf("\n     Gain = %.3f dB ", gain );

    gpib_close ( ); /* releases IEEE  file handler */
    printf("\n     Terminated ... ");
}
```



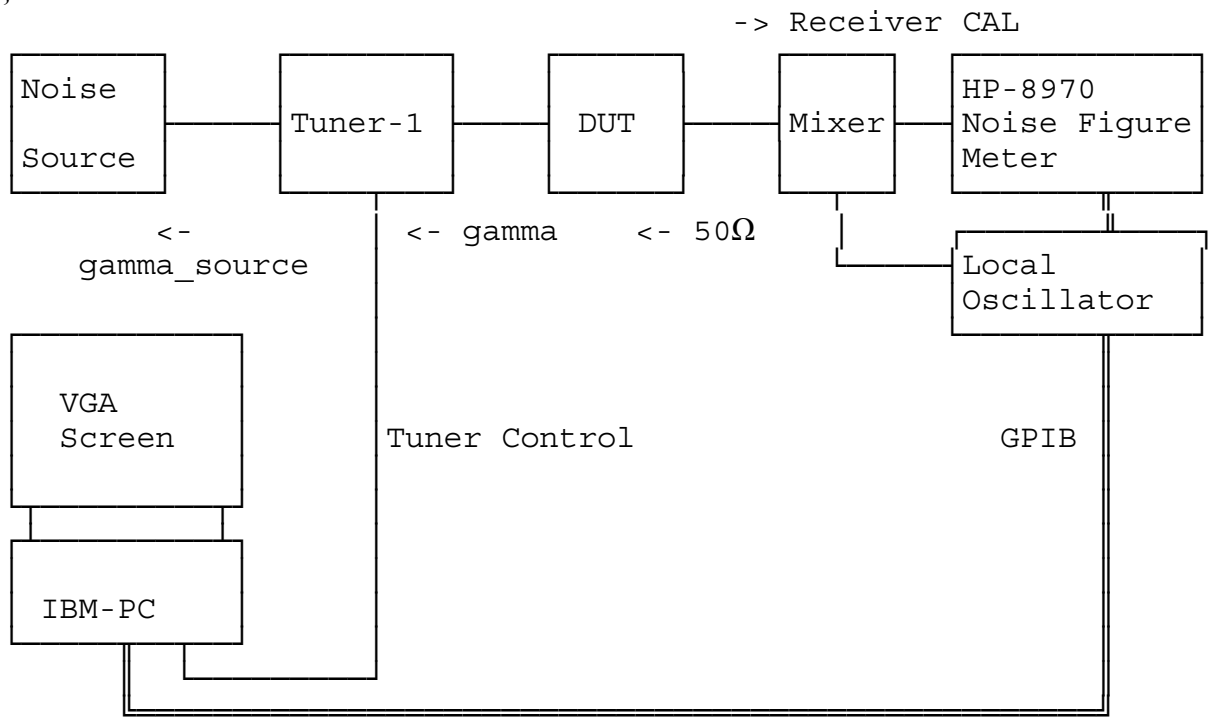
Setup for operating the above example and other Load Pull programmes.
CC-2 is the PC insertable tuner controller of Focus Microwaves.
DUT = Device Under Test.

Programming Example 2: Measure the 4 Noise Parameters

The following is an example of source code in C that can be compiled using MS C compiler and linked to the FOCUS.LIB in order to generate an executable programme
that measures the **4 Noise and 4 Gain Parameters of an Amplifier**

```
/*   Comment:
    This programme uses only one tuner. It loads calibration data from disk, initializes the tuner and the
Noise Figure Meter Hewlett Packard® 8970B. It then tunes thru a pattern of source impedances and
measures the Noise Figure. A (not shown) Gauss elimination algorithm permits to compute the four
Noise and Gain Parameters of the DUT.
*/
#include <stdio.h>
#include "tuner.h"
#include <stdlib.h>
#include <conio.h>
#include <math.h>

 int unit,posv[2]={0,0},stph[2]={0,0},stpv[2]={0,0},speed=400,port=544,ypos[2], _WAIT,errnox,ieee,dir;
 long posh[2]={0L,0L},xpos[2];
 float * * *tunerdat,R_FACTOR[2],lambda_zero,step_size[2];
 double tuner_s[8];
 unsigned caldat[2][10][5],limit[2][4];

main( )
{
int calno,i,adr=8;
float freq, in_loss, gain, nfig, gamma[2], gamma_source[2], noise_mes[20], gain_mes[20],
      npar[4], gpar[4], gamma_mes[2][20], x, y, z;
char str[80];

    /* Initialize tuner 1 */
     tuner_init (1,0);
     gamma_source[0] = 0.05;     /* Source Γ values to be measured independently */
     gamma_source[1] = 45.;
   /* Load calibration for tuner 1 */
     allocate_cal (361);
     printf("\n      Enter Frequency [GHz] = ");
     gets(str);
     freq = atof(str);
     printf("\n      Enter Cal Number for Tuner 1 = ");
     gets(str);
     calno = atoi(str);
     if( i=load_tuner_cal("C:\\CCMT\\CAL", 1, calno, freq, str ) < 0 ) exit(i);

     /* Configure the Hewlett Packard 8970 Noise Figure Meter (previously manually calibrated) */
     gpib_open( );
     gpib_send ("H1M2",adr);                /* corrected noise figure+gain measurement */
     sprintf(str,"FR%fMZT2",1000*freq);   /* generate string to set frequency and Trigger */
     gpib_send (str,adr);                    /* set measurement frequency, trigger */
     gpib_read (str,adr);                    /* read instrument. Response in char buffer STR */
     sscanf(str,"%f,%f,%f",&x,&y,&z);      /* read values for CAL verification, x,y,z are dummy variables */
     gpib_send ("E1",adr);                  /* DUT test configuration 1, fixed IF, external LO */
```

```
   /* Set Impedances using the Tuner : Tuned Reflection Factor is gamma[2] in Magnitude and Phase */
   i = 0 ;
   for (gamma[0]=0.4; gamma[0]<0.8; gamma[0]+=0.3)              /* Magnitude: makes 2 circles */
      for (gamma[1]=-45; gamma[1] < 91; gamma[1] += 45) {      /* Phase: with 4 points each  */
                                                               /*        from -45 to +90 degrees */
          tune_to (1, INPUT, gamma, gamma_source, &in_loss) ;
          gpib_send ("T2",adr) ;        /* Trigger corrected Noise Figure and Gain measurement*/
          gpib_read (str,adr);
          sscanf("%f,%f,%f",&freq,&gain,&nfig) ;     /* values returned in dB */

          nfig   -= 10*log10(in_loss);          /* correct for tuner available loss */
          gain += 10*log10(in_loss);
          noise_mes [i] = nfig;                /* save corrected data in array     */
          gain_mes [i] = gain;
          gamma_mes[0][i] = gamma[0];      /* save  measurement Γs in array */
          gamma_mes[1][i] = gamma[1];
          i++;
      }   /* end of measurement loop */

   gauss_elimination ( noise_mes, gain_mes, gamma_mes, i, npar, gpar );  /* up to 20 points */

  /*  The "gauss_elimination( )"  routine computes the  4 Noise and Gain Parameters.
      4 Noise Parameters = Minimum Noise Figure, Equiv. Noise Resistance, Optimum Γs for Noise
      4 Gain Parameters  = Maximum Gain (MAG), Equiv. Gain Resistance, Optimum Γs for Gain
   */
   printf("\n\n   Noise Parameters = %f,%f,%f,%f", npar[0],npar[1],npar[2],npar[3]);
   gpib_close( );       /* release IEEE file handler */
}
```



Setup for operating the Noise Measurement programme. For exact measurements the DUT should be either an output matched amplifier or use an output isolator.