

Przedsiębiorstwo Badawczo-Produkcyjne
OPTEL Sp. z o.o.
ul. Otwarta 10a
PL-50-212 Wrocław
tel.: +48 (071) 329 68 53
fax.: +48 (071) 329 68 52
e-mail: optel@optel.pl
<http://www.optel.pl>

Ultrasonic Testing Card OPCON-01/100

OPCON-01/100 is particularly well suited for ultrasonic measurements as well as other kinds of measurements which employ multiplexed channels (the card is capable of controlling such devices). Together with Pulser & Receiver OPGUD-01 or Pulser & Receiver OPGUD-Ex and ultrasonic probes it could be used as complete ultrasonic testing device. OPCON-01/100 is a short PCI card and can be installed in a standard PC.

Technical data

A/D converter:

- Resolution: 8 bits
- Sampling frequency: 12.5,25,50 or 100MHz¹

Analog parameters:

- Input amplifier gain: -10dB-50dB¹
- Input preamplifier 0dB or 10dB¹
- Sensitivity: 0.1mV – 1.6Vpp
- Bandwidth: 0,1 – 25MHz
- Input impedance: 50Ohm, 10pF

Delay time:

Post trigger 256us, measurement accuracy better than 1ns

Data buffer:

256, 512, 1K, 2K,4K,8K,16K or 32K¹

Output:

2 input of alarms

Data bus

PCI (short card)

1 - software selected

Software Package

The software (for Windows NT 4.0/2000/XP/2003) includes all necessary functions for ultrasonic measurements (demo version ready to use) and sample application Lab View and Visual C ++.

Signals on the external connectors

BNC:

Uin - Measured input signal;
TrgOutput - Trigger output

DB9:

Pin 1 - -12V
Pin 2 - 12V
Pin 3 - +5V
Pin 4 - **Ureg** - software controlled voltage (2-10V) used for pulse amplitude control of the OPGUD pulser&receiver units;
Pin 5-9 - Control lines

DB9 (additional connector for Alarms controlling)

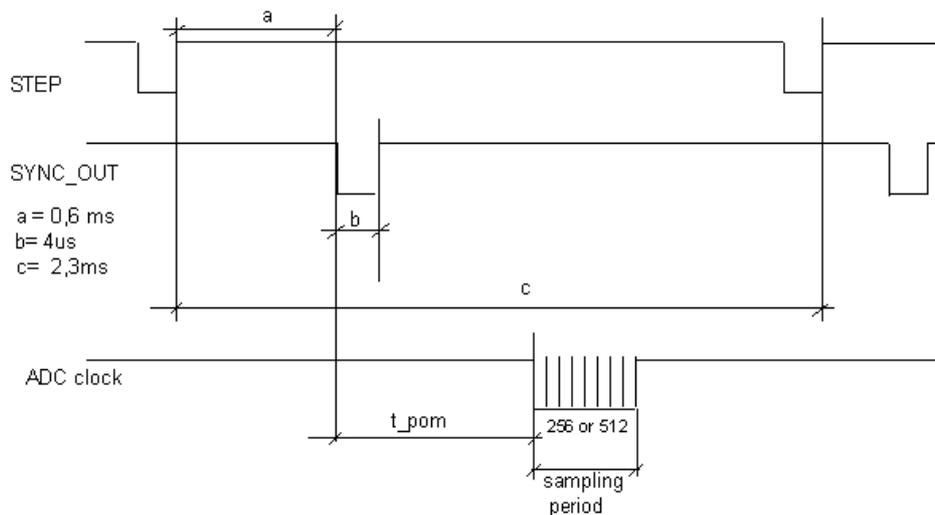
Pin 1 - 12V
Pin 2 - GND
Pin 3 - Controlled voltage signal: 0V or 5V
Pin 4 - Controlled voltage signal: 0V or 5V

Characteristic of the card:

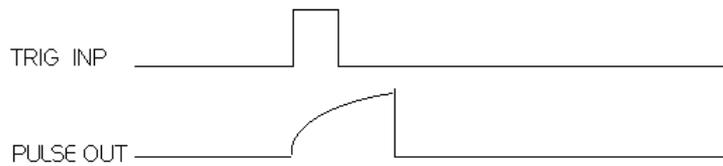
One of the most important features of the card is a precise synchronization between the transmitter trigger signal T_NAD and the moment when the sampling of the input signal starts.

This time (t_{pom}) is software programmable in the range of 0 – 255 μ s with a resolution of 1 μ s and has stability within the range of 1ns. It is particularly important in the case of scanning devices, since it allows to achieve a very small time skew between different positions (channels). 1ns would correspond to a clock frequency of 1GHz which is much higher than the actual frequency used.

Automatic measurement mode



Signal form



Additional notices

In the current version of the card, there is no analogue trigger input and this function is implemented in software.

The card is programmed through the memory operations. Thanks to the feature which allows the base address change, a number of our cards can work in a single computer.

All the control signals generated by the card can be modified according to specific customer requirements.

Multiplexer OPCOM-01/16

OPCOM-01/100 is particularly well suited for ultrasonic measurements as well as other kinds of measurements which needs many channels. Together with the card OPCOM-01 and ultrasonic probes it could be used as complete ultrasonic testing device. Each channel has separate pulser&receiver.

Technical data

Channels: 16¹
Pulse amplitude range: 50-360V¹
Max cable length between computer and the box: 30m

Connectors

BNC:

Transducers Trigger input
 Signal output

DB15:

Control signals/Power supply

1 - software selected

Features and Specifications

Pulser circuit waveforms: The rising edge of the Trig In signal initiates the transducer charging process which takes about 3 μ s. After this time a transistor switch which discharges the transducer is turned on (discharge time is about 20ns, but it can be longer, if the capacitance of the transducer is too large; the limit is reached, if the transducer is made from the standard ceramics, is 0.1 mm thick and has 8 mm diameter).

Because of the very low output impedance of the device (<1 Ohm) and short discharge time the pulse generated with this device could be concerned as a real pulse answer for the most transducers

Alternative (One channel data acquisition) OPGUD-01 Pulser & Receiver

The basic version of our pulser&receiver circuit have the size of a matchbox and integrated amplifier. It is powered directly from our oscilloscope card and enable the adjustment of the transmitter voltage directly from a PC. This device is intended for use with our OPKUD card or OPBOX.

Specifications:

Pulser: Step pulser, 50V -360V in 8 levels (the voltage depends on the used transducer), 20ns edge falling time, separate,

Amplifier: 40dB

Size: 40x85x25mm

Features:

Pulser circuit waveforms: The rising edge of the Trigger signal (described as Trig Inp) initiates the transducer charging process which takes about 3us. After this time a transistor switch which discharges the transducer is turned on (discharge time is about 20ns, but it can be longer, if the capacitance of the transducer is too large; the limit is reached, if the transducer is made from the standard ceramics, is 0.1 mm thick and has 8 mm diameter).

Comments:

Because of the very low output impedance of the device (<1 Ohm) and short discharge time the pulse generated with this device could be concerned as a real pulse answer for the most transducers. Transducers with a parallel matching inductance should not be used, since they do not allow the transducer to be pre-charged (the inductance causes a short-circuit).

How to control the card from the PC

OPCON Card - Software Install and Use Information

1. Please put the card into appropriate PCI slot.
2. Connecting Pulser/receiver or multiplexer trigger (T and S signal to):
BNC marked with green-yellow colour on upper connector - Signal;
BNC marked with black colour on middle connector - Trigger;
DB9 connector on its place on lower connector - All control voltages, address line etc..
3. After PC is switched on you can hear relay's clicking. It is regular behaviour of the card.
4. BIOS of the PC should watch the card with following data: Vendor ID 10E8, Device ID 4750 and Class "docking station".
5. The card is Plug-and-Play so after the card is found out by system, all necessary information consists "optel.inf" file.
6. After restart of system you are to install software in directory "c:\optel" running "setup.exe" from CDROM\scope directory.

OPCON Card is plug and play compatible with Windows higher than NT 4.0.

Standard packed software include:

Driver:

optel.inf
optel.sys

Libraries:

amcc.h
amcc.dll
amcc.lib

and source code amcc.c and amcc.h

User can use standard dll library or source amcc.c and amcc.h to the control OPCON Card

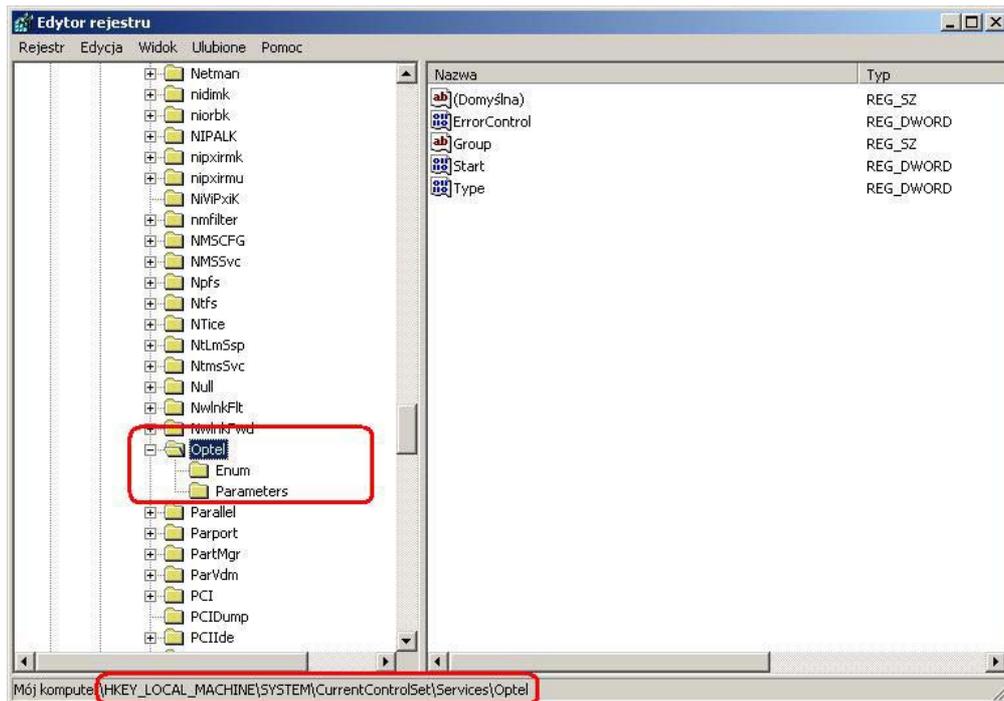
Install and register the OPTEL (optel.sys) drivers under WinNT or Win 2000 / XP

Please perform the following steps to install the driver:

- 1) If necessary, edit the OPTEL.INF file using any editor and change the DeviceID and VendorID fields to match your device. They are currently set at VID=0x10E8 and DID=0x4750. This vendor ID is registered to AMCC, but you are free to use it.
- 2) Using RegEdit, remove the registry entry for your PCI device.
- 3) Now reboot your machine with your PCI board installed. Win's hardware installation wizard should detect your PCI board as a new device.
- 4) When asked for a driver, select the "Have Disk" option, then browse your hard drive for the provided OPTEL.INF file.
- 5) If the process complains that it can't find the OPTEL.SYS, manually direct it into the windows/system32/drivers/subdirectory. This procedure should install the OPTEL.SYS as the driver for your PCI device. Each time you change the driver and want to use the new version, simply overwrite the old driver in the \WINDOWS\SYSTEM subdirectory, and reboot your machine.



Picture 1. Bios recognize our card.



Picture 2. OS register with OPTEL key.

DESCRIPTION OF CONTROL FUNCTION CARD

List of functions includes on **amcc.dll** and amcc.c amcc.h:

```

void reset_fifo (void);
int get_sample (void);
void on_preamp (void);
void off_preamp (void);
void set_gain (int gain);
void set_channel(int channel);
void set_vreg (int vreg);
void set_delay(int hdelay);
void set_depth_128w(void);
void set_depth_256w(void);
void set_depth_512w(void);
void set_depth_1024w (void);
void set_depth_2048w (void);
void set_depth_4096w (void);
void set_depth_8192w (void);
void set_depth_16kw(void);
void set_freq_100 (void);
void set_freq_50 (void);
void set_freq_25 (void);
void set_freq_12 (void);
void init_meas (void);
int back_data (int index);
void reset (void);
void checkfifo (void);
void set_alarm1_on (void);
void set_alarm1_off (void);
void set_alarm2_on (void);
void set_alarm2_off (void);

```

```
char open_driver(int);
void close_driver(int);
```

Basic control functions:

reset_fifo	This function reset line of data in FIFO register
get_sample	This function turns back depth of measurement (amount of samples); Amount of samples is parameters for follow functions: first_in_plus, first_in_minus, second_in_plus, second_in_minus, distance.
set_gain	This function sets gain in input amplifier in range from -10dB up to 50dB. Input parameter less than 8. Parameter this function is value (type 16 signed int) 8 between 8 and 255.
set_delay	This function set delay between pulse and time when starts measurement. Parameter of this function is value between 0 and 255 (in [us]).
set_depth_256w	This function set 512 bytes samples in depth of measurement
set_depth_512w	This function set 1024 bytes samples in depth of measurement
set_depth_1024w	This function set 2048 bytes samples in depth of measurement
set_depth_16kw	This function set 32768 bytes samples in depth of measurement
set_freq_100	This function set measurement frequency of 100MHz
set_freq_50	This function set measurement frequency of 50MHz
set_freq_25	This function set measurement frequency of 25MHz.
set_freq_12	This function set measurement frequency of 12.50MHz.
init_meas	This function start measurement.
On_preAmp	This function add 10dB pre - Amplifier
Off_preAmp	This function switch off pre - Amplifier
set_vreg	This function set amplitude of sender impulse
back_data	This function back items from array of data. Parameter of this function is index of array.
reset	This function reboots card
open_driver(1)	This function loading driver for card to the system. Parameter of this function must be number of card. Function turns back 0 if loading successfully.
close_driver(1)	This function unloading driver for card from the system.
First_in_plus	functions turns back index of data table pointing on the first peak in plus
First_in_minus	functions turns back index of data table pointing on the first peak in minus
Second_in_plus	functions turns back index of data table pointing on the second peak in plus
Second_in_minus	functions turns back index of data table pointing on the second peak in minus
Distance	functions turns back amount in samples between the first and the second signal. for example distance [mm] = amount of samples * sampling frequency * sound velocity in plastic / 2.

Definition basic control functions in amcc.c and amcc.h

Amcc.h header with definition of basic function and variables like:

```
DEVICE_ID    0x4750           // OPTEL Device ID
VENDOR_ID    0x10E8           // OPTEL Vendor ID

PCI Operation Register Offsets          */
FIFO Port
NV- Commands
Functions writes/read a kind value to the specified address or register.
```

The OPTEL driver for Win is a DeviceIOControl based programming interface which calls a dynamically loaded sys. The driver sys is installed as a Plug and Play driver and is loaded at boot time by the Configuration Manager and the PnP Manager. The driver is opened with a CreateFile call, which gives your program a handle to it. When done, your program releases the handle with a CloseHandle call. All intermediate calls to the driver are implemented through DeviceIOControl calls. The exact syntax for making these calls can be found in the sample program and under DeviceIOControl in the Microsoft Knowledge Base, but is demonstrated in the code fragment below.

```
#define IOCTL_OPTEL_MAP_USER_PHYSICAL_MEMORY
CTL_CODE(FILE_DEVICE_OPTEL, \
          OPTEL_IOCTL_INDEX, \
          METHOD_BUFFERED, \
          FILE_ANY_ACCESS)

#define IOCTL_OPTEL_UNMAP_USER_PHYSICAL_MEMORY
CTL_CODE(FILE_DEVICE_OPTEL, \
          OPTEL_IOCTL_INDEX+1, \
          METHOD_BUFFERED, \
          FILE_ANY_ACCESS)
```

Amcc.c

```
/* Include files
```

```
windows.h
winioclt.h
amcc.h
```

```
/* definitions:
FIFO register, MCSR – Master Control/Status Register
```

```
/* Function prototypes
```

Because the driver supports overlapped transfers, an overlapped structure must be passed in each DeviceIOControl call (even if the call is not overlapped). This overlapped structure should include an appropriate event structure.

To get a handle to the device, use the following code as a template:

```
BYTE open_driver(BYTE nr)
{
    char s_str[20];
    DWORD cbReturned;
```

```

switch(nr - 1)
{
    case 0 : strcpy(s_str, "\\.\.\OPTEL0");
              break;
    case 1 : strcpy(s_str, "\\.\.\OPTEL1");
              break;
    case 2 : strcpy(s_str, "\\.\.\OPTEL2");
              break;
    case 3 : strcpy(s_str, "\\.\.\OPTEL3");
              break;
    case 4 : strcpy(s_str, "\\.\.\OPTEL4");
              break;
}

if ((hDriver[nr - 1] = CreateFile(s_str,
                                GENERIC_READ | GENERIC_WRITE,
                                0,
                                NULL,
                                OPEN_EXISTING,
                                FILE_ATTRIBUTE_NORMAL,
                                NULL
                                )) == ((HANDLE)-1))
    return(1);

DeviceIoControl (hDriver[nr - 1],
                (DWORD) IOCTL_OPTEL_MAP_USER_PHYSICAL_MEMORY,
                NULL,
                0,
                &mmi[nr - 1],
                sizeof(MAPPED_MEMORY_INFO),
                &cbReturned,
                0);
return(0);
}

```

Under WinNT, each device has a unique WIN32 name. To open a device, you simply use that name and then are able to access the specific device you want.

Under Win95, each driver has a single name, regardless of the number of devices it supports. By making this call, you are telling the driver that the hDevice handle used in this call is to be associated with the given 'zero' based device number. If you only have one device supported by the driver, then this call should still be made, passing a zero as the device number. The sample application does this.

And close driver function looks like:

```

void close_driver(BYTE nr)
{
    DWORD  cbReturned;

    DeviceIoControl (hDriver[nr - 1],
                    (DWORD) IOCTL_OPTEL_UNMAP_USER_PHYSICAL_MEMORY,
                    &(PULONG)mmi[nr - 1].MappedAdr[0],
                    sizeof(PVOID),
                    NULL,
                    0,
                    &cbReturned,
                    0);
    CloseHandle(hDriver[nr - 1]);
}

```

Description of control functions:

Format of commands:

- Each commands begin with „@“ and terminates by „/“.
- Length of commands are 3 (without parameters) or 4 (with parameter) bytes.
- Each command is sent thought FIFO and FIFO1 is mask to 0xFF (where FIFO 0x20 and FIFO1 0x21).

For example : Command @r/ looks like:

```
amcc_write_byte(1,FIFO1,'@');
amcc_write_byte(1,FIFO,0xff);
amcc_write_byte(1,FIFO1,'r');
amcc_write_byte(1,FIFO,0xff);
amcc_write_byte(1,FIFO1,'/');
amcc_write_byte(1,FIFO,0xff);
```

reset_fifo	This function reset line of data in FIFO register. Command: @r/
get_sample	This function turns back depth of measurment (amount of samples); Amount of samples is parameters for follow functions: first_in_plus, first_in_minus, second_in_plus, second_in_minus, distance. Command: @p/ After Initialize Measurement read data following: <pre>STATUS=amcc_read_byte(1,MCSR); // read status FIFO flag while((STATUS & 0x20) == 0) // if fifo are full or not empty { dana2 = amcc_read_byte(1,FIFO1); // read data from older fifo dana1 = amcc_read_byte(1,FIFO); // read data from younger fifo STATUS = amcc_read_byte(1,MCSR); // read status FIFO flag Tab[i] =dana1; // put data to the array Tab[++i] =dana2; // put data to the array plus increment number of data i++; result++; }</pre>
set_gain	This function sets gain in input amplifier in range from -10dB up to 50dB. Input parameter less than 8. Parameter this function is value (type 16 signed int) 8 between 8 and 255. Command: @w <value> / where: <value> - 0 .. 255
set_delay	This function set delay between pulse and time when starts measurement. Parameter of this function is value between 0 and 255 (in [us]). Command: @c <value> / where : <value> - 1 .. 255
set_depth_256w	This function set 512 bytes samples in depth of measurement Command: @l <value> / Where : <value> - 1 .. 7 For 512 bytes <value> = 1
set_depth_512w	This function set 1024 bytes samples in depth of measurement

	Command: @l <value> / Where : <value> - 1 .. 7 For 512 bytes <value> = 2
set_depth_1024w	This function set 2048 bytes samples in depth of measurement Command: @l <value> / Where : <value> - 1 .. 7 For 512 bytes <value> = 3
set_depth_2048w	This function set 4098 bytes samples in depth of measurement Command: @l <value> / Where : <value> - 1 .. 7 For 512 bytes <value> = 4
set_depth_4096w	This function set 8192 bytes samples in depth of measurement Command: @l <value> / Where : <value> - 1 .. 7 For 512 bytes <value> = 5
set_depth_8192w	This function set 16k bytes samples in depth of measurement Command: @l <value> / Where : <value> - 1 .. 7 For 512 bytes <value> = 6
set_depth_16kw	This function set 32768 bytes samples in depth of measurement Command: @l <value> / Where : <value> - 1 .. 7 For 512 bytes <value> = 7
set_freq_100	This function set measurement frequency of 100MHz Command: @f <value> / Where: <value> - 0 .. 3 For 100MHz sampling frequency <value> = 3
set_freq_50	This function set measurement frequency of 50MHz Command: @f <value> / Where: <value> - 0 .. 3 For 100MHz sampling frequency <value> = 2
set_freq_25	This function set measurement frequency of 25MHz. Command: @f <value> / Where: <value> - 0 .. 3 For 100MHz sampling frequency <value> = 1
set_freq_12	This function set measurement frequency of 12.50MHz. Command: @f <value> / Where: <value> - 0 .. 3 For 100MHz sampling frequency <value> = 0
init_meas	This function start measurement. Command: @p/ (see Get_samples)
set_vreg	This function set amplitude of sender impuls Command: @n <value> / Where: <value> - 0 .. 7; (0 – 60V to 7 – 360V)
back_data	This function back items from array of data. Parameter of this function is index of array. int back_data (int index) { return(Tab[index]); // Array of data }

reset	This function reboots card Command: @r/
On_preAmp	This function add 10dB pre – Amplifier Command: @z1/
Off_preAmp	This function switch off pre – Amplifier Command: @z0/
open_driver(1)	This function loading driver for card to the system. Parameter of this function must be number of card. Function turns back 0 if loading successfull. <pre> BYTE open_driver(BYTE nr) { char s_str[20]; DWORD cbReturned; switch(nr - 1) { case 0 : strcpy(s_str, "\\.\OPTEL0"); break; case 1 : strcpy(s_str, "\\.\OPTEL1"); break; case 2 : strcpy(s_str, "\\.\OPTEL2"); break; case 3 : strcpy(s_str, "\\.\OPTEL3"); break; case 4 : strcpy(s_str, "\\.\OPTEL4"); break; } if ((hDriver[nr - 1] = CreateFile(s_str, GENERIC_READ GENERIC_WRITE, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL)) == ((HANDLE)-1)) return(1); DeviceIoControl (hDriver[nr - 1], (DWORD) IOCTL_OPTEL_MAP_USER_PHYSICAL_MEMORY, NULL, 0, &mmi[nr - 1], sizeof(MAPPED_MEMORY_INFO), &cbReturned, 0); return(0); } </pre>
close_driver(1)	This function unloading driver for card from the system. <pre> void close_driver(BYTE nr) { DWORD cbReturned; DeviceIoControl (hDriver[nr - 1], (DWORD) IOCTL_OPTEL_UNMAP_USER_PHYSICAL_MEMORY, &(PULONG)mmi[nr - 1].MappedAdr[0], sizeof(PVOID), NULL, 0, &cbReturned, 0); CloseHandle(hDriver[nr - 1]); } </pre>
First_in_plus	functions turns back index of data table pointing on the first peak in plus
First_in_minus	functions turns back index of data table pointing on the first

	peak in minus
Second_in_plus	functions turns back index of data table pointing on the second peak in minus
Second_in_minus	functions turns back index of data table pointing on the second peak in minus
Distance	functions turns back amount in samples between the first and the second signal. for example distance [mm] = amount of samples * sampling frequency * sound velocity in plastic / 2.

Sample of software

In this moment we can offer to you:

- Lab View vi sample
- Visual C ++ sample
- Ready to use demo software (Lab Windows CVI)

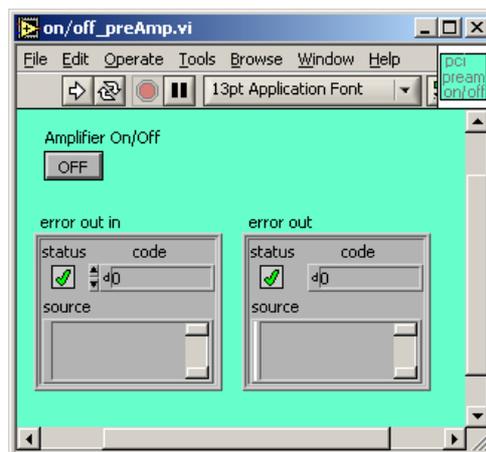
Lab View sample application

Lab View sample application is tested with Lab View version 6.02

- Optelcard_PCI.llb
- Amcc.dll

Optelcard_PCI.llb includes:

- Vi's for each function of control card
(Each function represent small green box with description and is ready to use. For example function on/off_preamp.vi)



on/off_preamp.vi

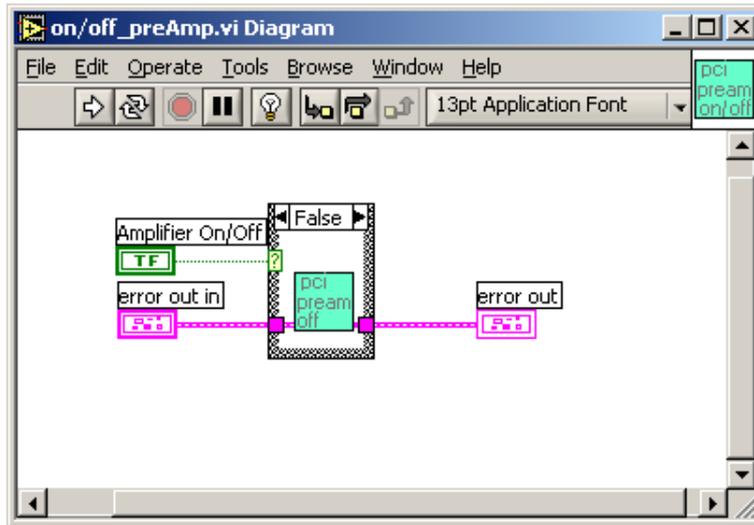
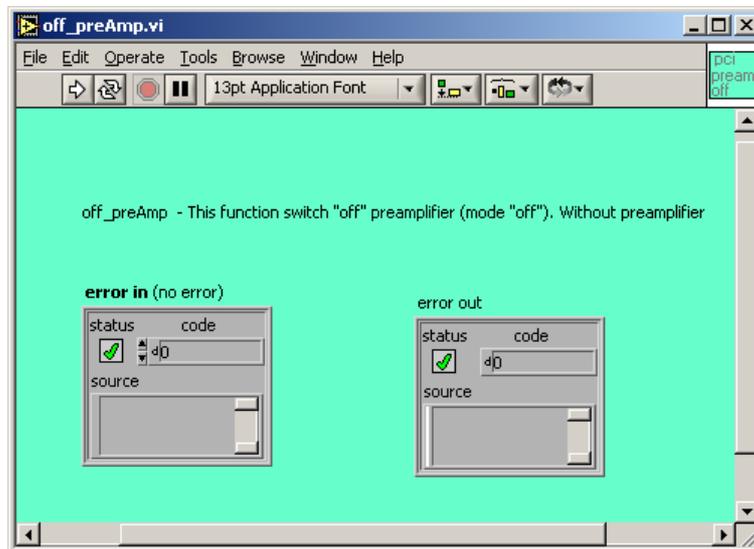


Diagram on/off_preamp.vi



Description of off_preamp function (off_preamp.vi)

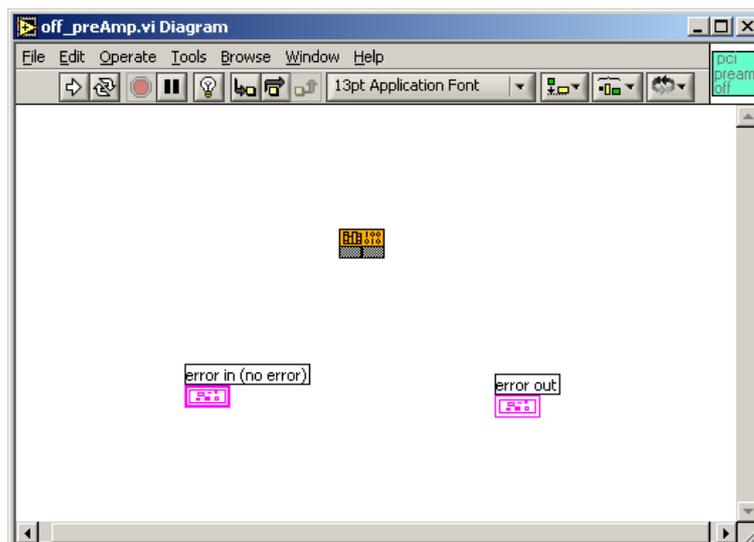
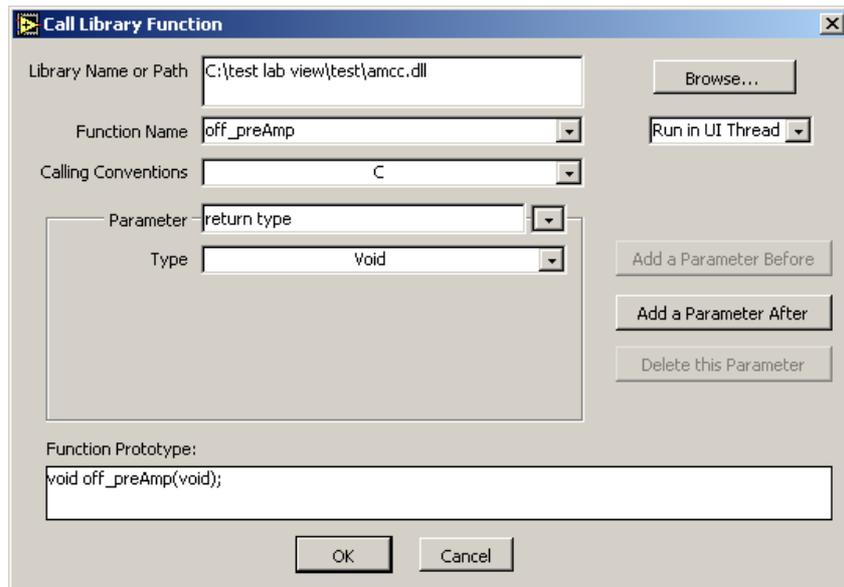
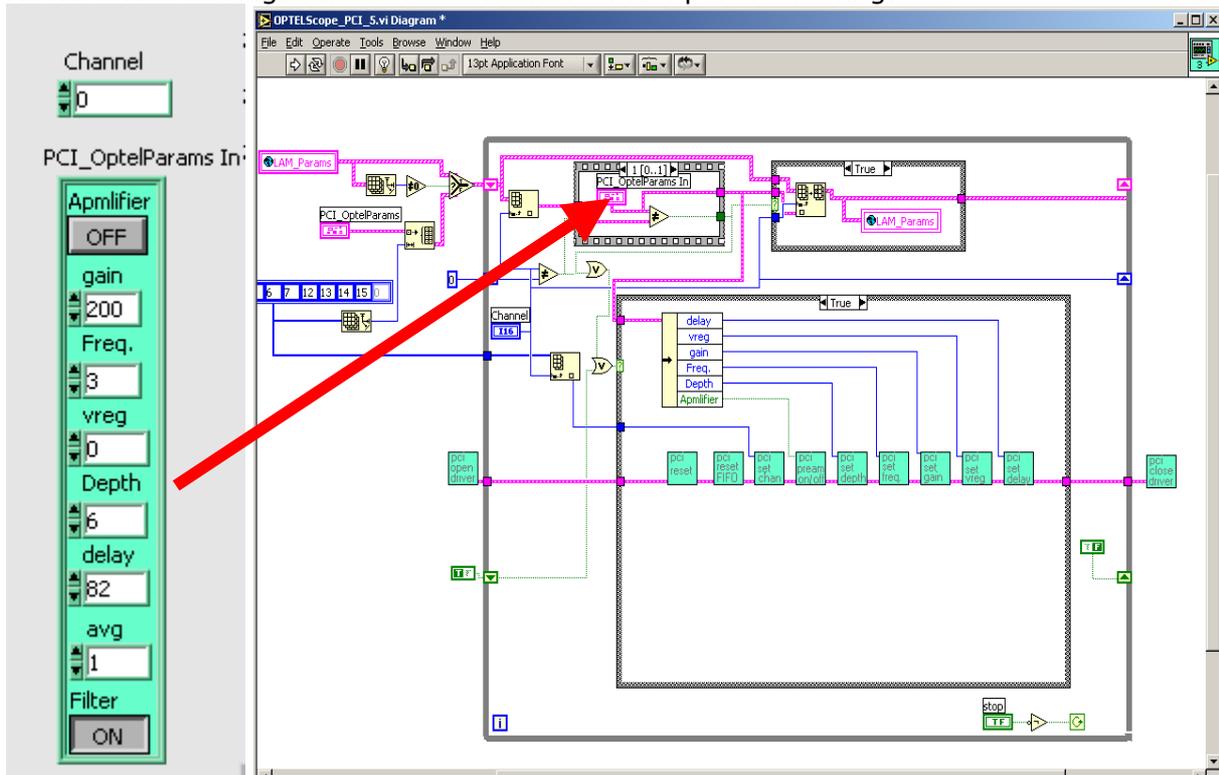


Diagram off_preamp function

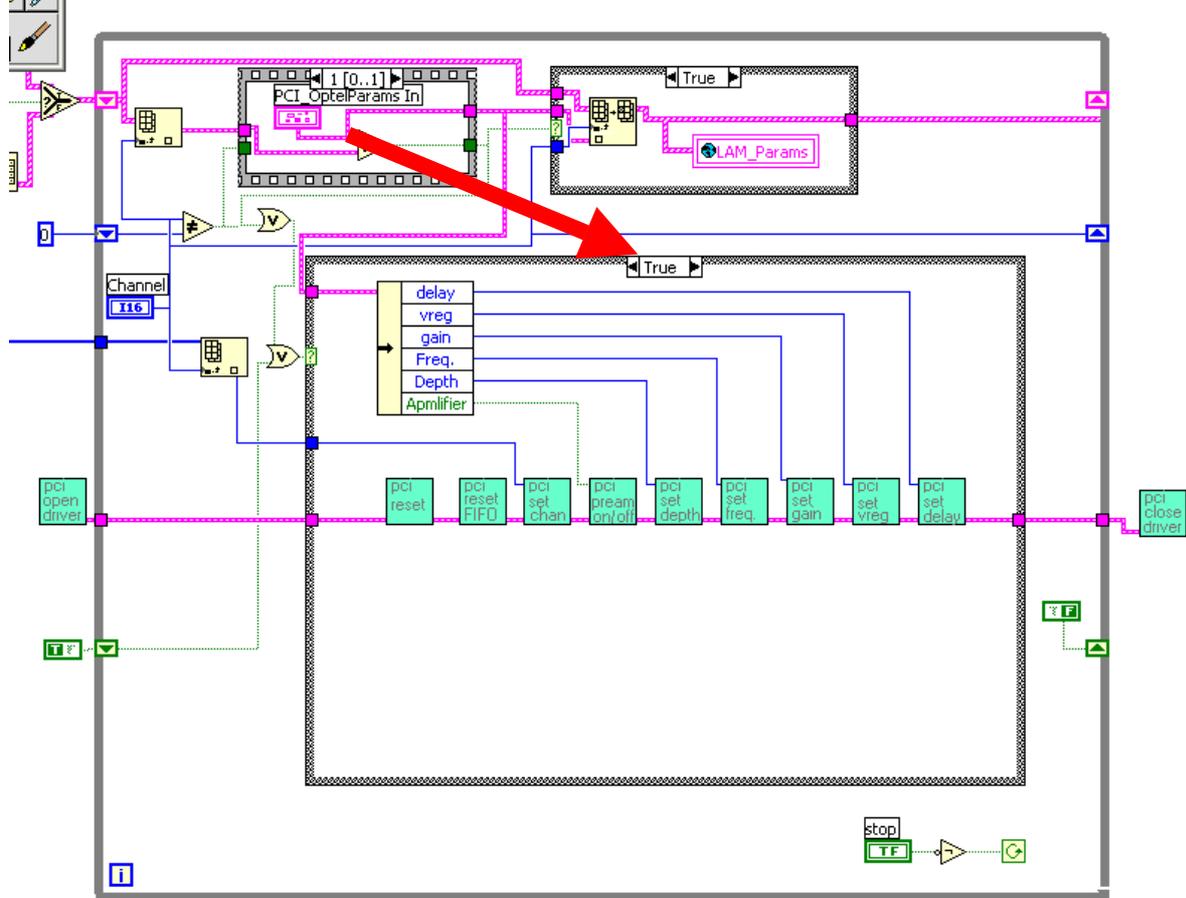


Call Library Function for off_preamp function.

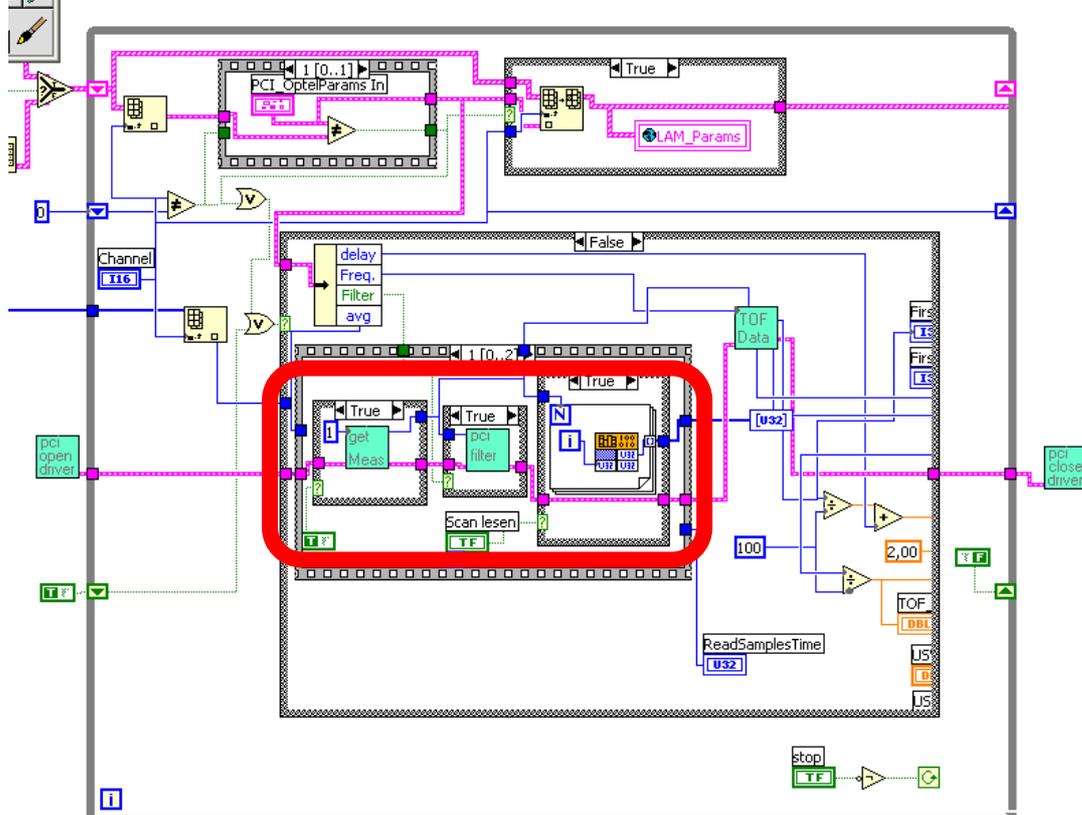
All hardware settings functions show on the main panel following:



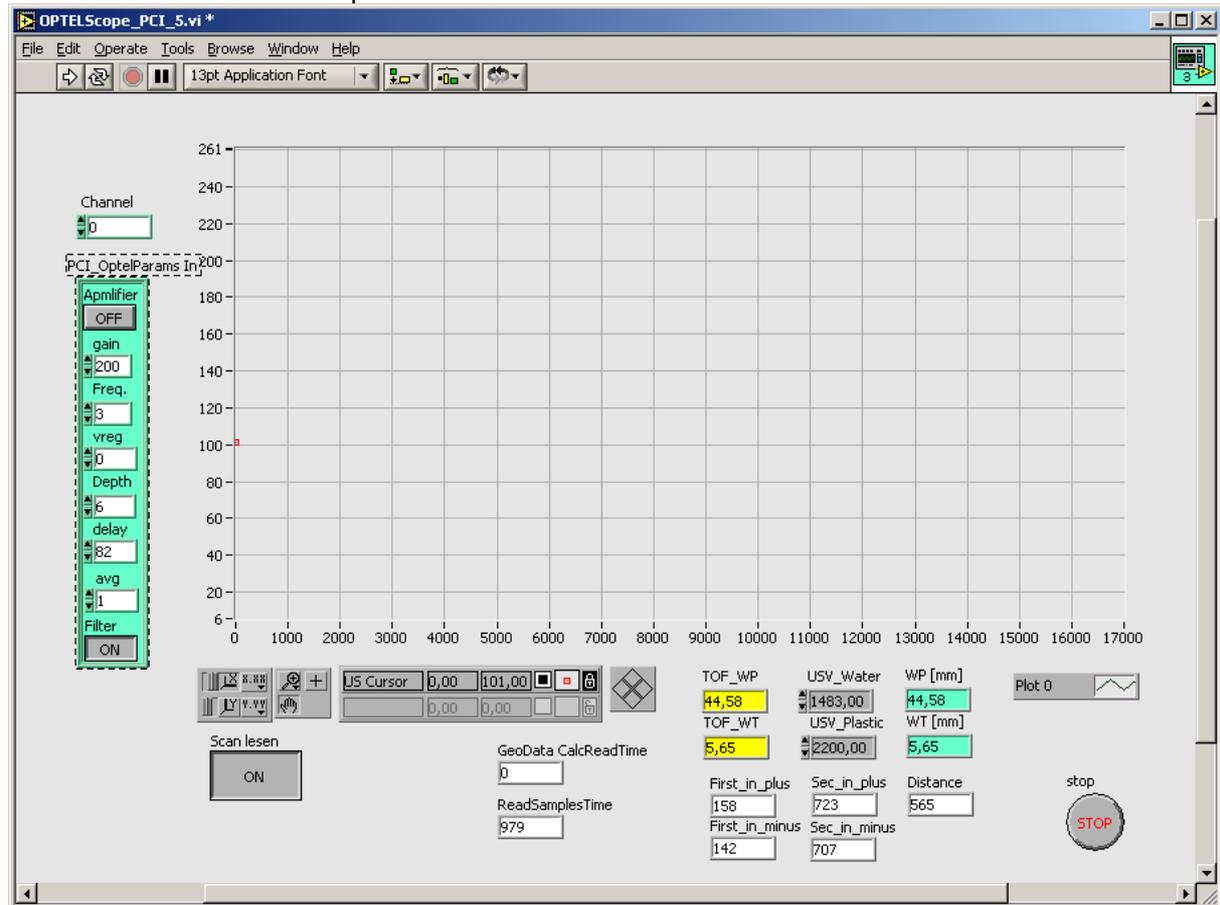
If any settings will change then main loop works:



If not main loop works following (get_meas, filtr data and show on the screen):



Main Panel Lab View sample:



Visual C ++ sample application

Visual C++ sample is win32 console (tested with visual studio 6.0 with service pack 6.0) and includes:

- Test.c
- Amcc.c
- Amcc.h
- Test.dsw

When amcc.c and amcc.h is source code for amcc.dll and is described above.

Test.c

```

/* Include files
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <windows.h>
#include <winioctl.h>
#include "amcc.h"

```

```

/* definition of variable, array and offset register
MAPPED_MEMORY_INFO mmi[3];
#define FIFO 0x20
#define FIFO1 0x21
#define MCSR 0x3c

```

```

#define MCSR3 0x3f

int result=0,i=0,ii=0,niewiem, x,y;
char STATUS;
int sp,sm,fp,fm,dis;
unsigned char  dana1,dana2,t[256];

int
__cdecl
main(
    IN int  argc,
    IN char *argv[])
{
    BYTE error;

    printf("Test program for OPTEL Driver \n\n");
    //
    // Try to open the device
    //
    error = open_driver(1);      // load driver for optel card
    printf("Error %d\n",error);

    reset();                    // reset card; set default settings for each hardware parameters.
    reset_fifo();              // reset fifo status and fifo stock
    on_preAmp();               // switch on preamplifier; add 10dB gain
    set_gain(125);             // set hardware gain (main amplifier) to 125 (middle of range)
    set_freq_12();             // set frequency of sampling to 12.5MHz
    set_vreg(0);               // set amplitude of sender pulse to 60V
    set_depth_128w();          // set length of measurement memory to 256 bytes
    set_delay(1);              // set postrigger to 1us.
    set_channel(1);            // set channel 1- if you use multiplexer 16 channel from optel

    for(i=0; i<13; i++)        // on the loop 13 times init measurement, acquisition
                                //data on the array
    {

        checkfifo();
        init_meas();
        checkfifo();

        y=get_sample();

        for(x=0; x<256; x++)
        {
            t[x]=back_data(x);
            printf("%d \t %d \t",x,t[x]);
        }
    }

    close_driver(1);           // close driver
    return 1;
}

```