13

LabVIEW Remote Lab

Aurel Gontean and Roland Szabó
"Politehnica" University of Timişoara
România

1. Introduction

LabVIEW is a quite interesting programming language, and despite its odd first impression is very powerful. We remember when we started to use it for the first time it was somehow strange to me, we thought what this, a graphical programming language? For the first contact whit this language, LabVIEW somehow hides its power, this could be due to fact that for the first contact everyone tries the simulation part. A usual first application would be to generate a sine wave to a graph. Unfortunately, this won't really impress a user.
All LabVIEW field engineers nightmare is to know that the users think that LabVIEW is a simulation environment. Their main job is to erase that from the users mind. Of course it has a very strong simulation part, but the main target is the real hardware controlling.
If we think put of the box a little, we will realize that if LabVIEW hides its power, that means is very simple to use, but if we think about the big amount of software and hardware National instruments make, we will see that it's quite an extraordinary programming language.
To make a short count, we can see that only LabVIEW 2009 with Modules, Toolkits and Device Drivers are 3 DVDs in compressed kit version. If we consider that in other hand we have NI Multisim circuit design suite, LabWindows/CVI C programming language for instruments and Measurement Studio LabVIEW style buttons add-on for Microsoft Visual Studio, then we can imagine that engineers National Instruments create a lot of software.
In hardware area we can just say that they have a lot of stuff, and if they don't have am equipment, then another company has it and LabVIEW has the driver for that equipment. Sometimes the diver is made by the National Instruments engineers, sometimes is made by third party vendors, but my only concern is to work, and they mostly do. A useful section in the www.ni.com webpage is the IDNET (Instrument Driver Network), where you can find the LabVIEW and LabWindows/CVI drivers for almost any equipment. You just have to search here: www.ni.com/devzone/idnet. A good instruments vendor would be Agilent Technologies; we can say that it's very hard to find equipments from Agilent without LabVIEW or LabWindows/CVI driver.
This way we can easily say that LabVIEW it's much more than MATLAB Simulink in color. LabVIEW has a strong programming part, despite its simple appearance, if we go deeply in some specific area, like NI Vision or CompactRIO, then things will start to get quite complex and complicated. But knowing that this is complicated in LabVIEW, if we think to do the same thing without LabVIEW, in another programming language, it could be almost impossible, for a reduced number of engineers, in a short amount of time.
In LabVIEW it's possible, almost everything is possible.

We would like to correct the sentence above; some things are possible only indirectly. Some extra programming in needed, but after all an engineer can do what he planned.

2. Tips and tricks in LabVIEW

As we said not everything can be done directly, some of them need some extra programming. We are not really sure why they did not include it in the LabVIEW package; maybe they had a good reason. On forums is really hard to find a solution for these problems, this way we will present a few common and useful tips and trick that we was forced to develop, because we needed them during my projects.

2.1 Very long HEX String to ASCII Conversion

We shall show a method in LabVIEW how to convert a very long HEX string to ASCII.
This could be useful if we want to analyze memory dumps or binary files, which are mostly in HEX, but to be interpretable, we need to convert it in some form, which we can understand, and mostly we convert it in ASCII.
As we can see in Fig. 1. the program is working.

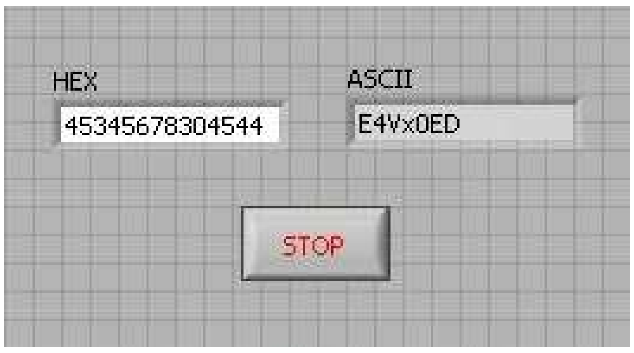


Fig. 1. HEX to ASCII Front Panel.

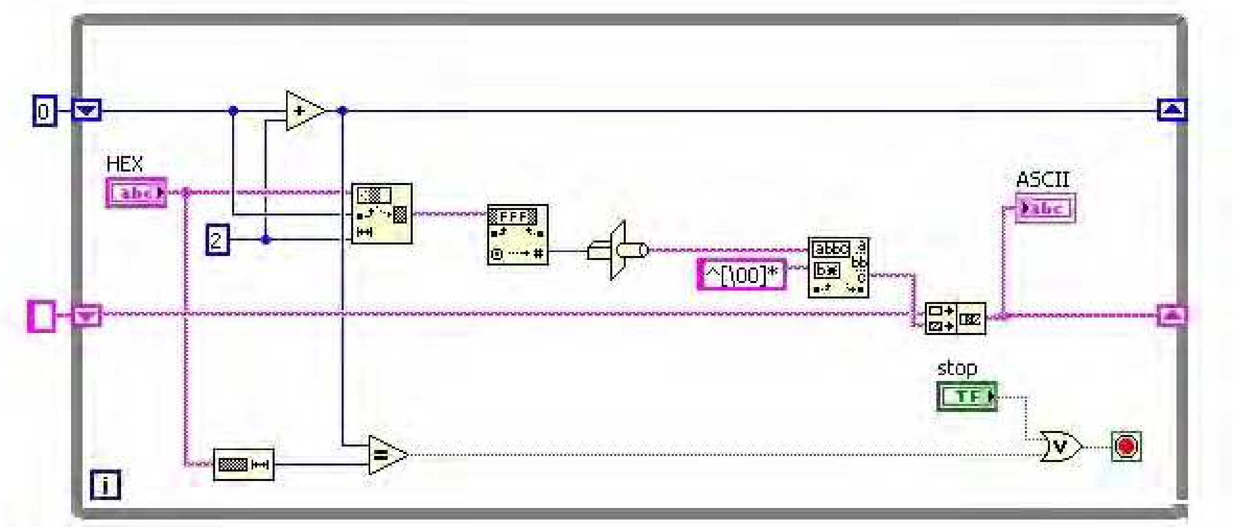


Fig. 2. HEX to ASCII Block Diagram.

In Fig. 2. we have the solution for the problem. First of all we have to enter a very long HEX string. With the String Subset VI we have to fragment the string into groups with 2 HEX numbers (8 bits). The first Shift Register with 0 initialization will always increase the fragmentation index with 2, this way we will reach at the end minus 2 (n-2) of the HEX word. There is a condition too, if we reach at the end of the HEX string to exist the While loop. From String Subset we convert the HEX string to number, the number is converted to ASCII with the Type Cast VI. For some strange reason the Type Cast VI puts an ASCII space before the converted ASCII character, this space is a \00 type space, so this will have to be deleted, it's deleted with the Match Pattern VI using the ^[\00]* command and wired to the after substring output (after the \00), and we will obtain the good converted ASCII value. This value will be concatenated with the following strings using the second Shift Register with the empty string initialization.

2.2 Very long ASCII to HEX string conversion

If we made the HEX to ASCII conversion, than we have to do the reverse operation too. In LabVIEW ASCII to HEX the conversion is not that simple. Unfortunately there is no VI which will do the job, so the user has to make his own VI. On possible method will be presented next.
In Fig. 3. we can see the working program.

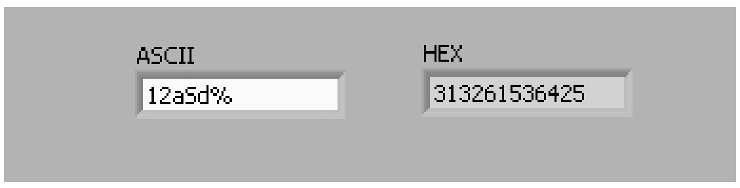


Fig. 3. ASCII to HEX Front Panel.

In Fig. 4. We have the ASCII input, which will be converted in a number, with byte data type. Byte is more than enough, because we have a total of 128 ASCII characters, in the other hand in LabVIEW there is no ASCII string to number conversion in with other data type, so this is our only solution. The numbers are converted into HEX with the length of 2 for each HEX number. The string to number conversion makes an array, this way the HEX values will be an array of HEX string with the length of 2. We want to make a long HEX string, not an array of strings. We made a For loop with the number of iteration equal to the length of the HEX array. We indexed the array and with the use of Shift Registers we concatenated it into a long HEX string.

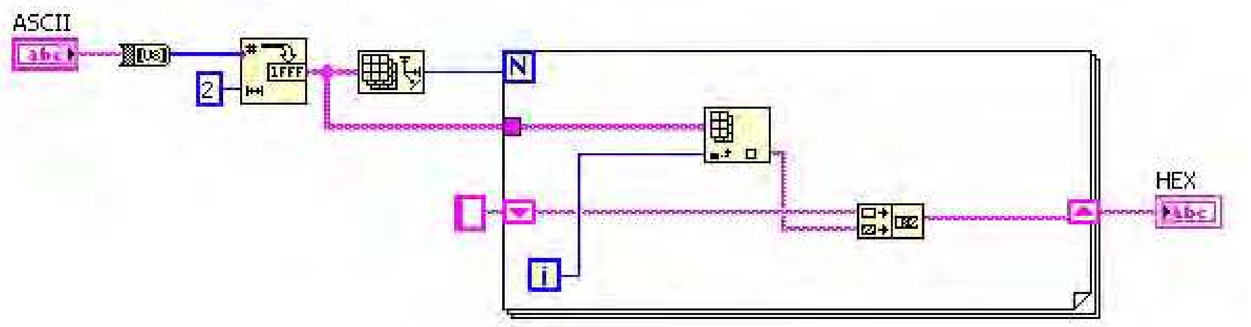


Fig. 4. ASCII to HEX Block Diagram.

2.3 Creating a toggle button from a push button

This application would be useful when we want to have an LED light to be ON, for very short impulses, like the sound of claps of falling coins, otherwise we would see only a LED blink and we would have to be very attentive. The idea would be to keep the signal on "1" logic for a longer period with shift registers. There could be many variants, but the hardest variant will be presented, the one where after the first "1" logic the light is ON and after the second "1" logic the light is OFF. For the Button the "Latch When Released" Mechanical Action was used, so it's a push-button.
As we can see the program's name is Button Latch (Fig. 5.), we need a latch to keep the LED ON after only one impulse and to turn the LED OFF after the second.

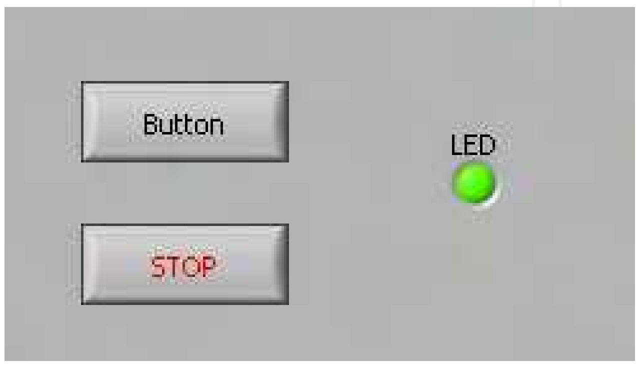


Fig. 5. Button Latch Front Panel.

In Fig. 6. we can see that Shift Registers were used. As we can see the initialization value for the Shift Registers is FALSE. The Button is connected to the Implies VI, which computes an OR logic between the negated x (first) input and y (second) input. In case of Button press, we will have FALSE in x input and FALSE in y input, so FALSE output, this means that from the Select VI the FALSE output is activated, which has an input of inverted FALSE (which is TRUE), so the LED will be ON.
If the Button is pressed again, then we will have FALSE in x input and FALSE at y input of the Implies VI, so at the Implies function will have FALSE output. The input of the second Shift Register will be TRUE, because before this the LED in ON. With the FALSE Implies VI we activate the inverted TRUE (which is FALSE), so the LED will be OFF.

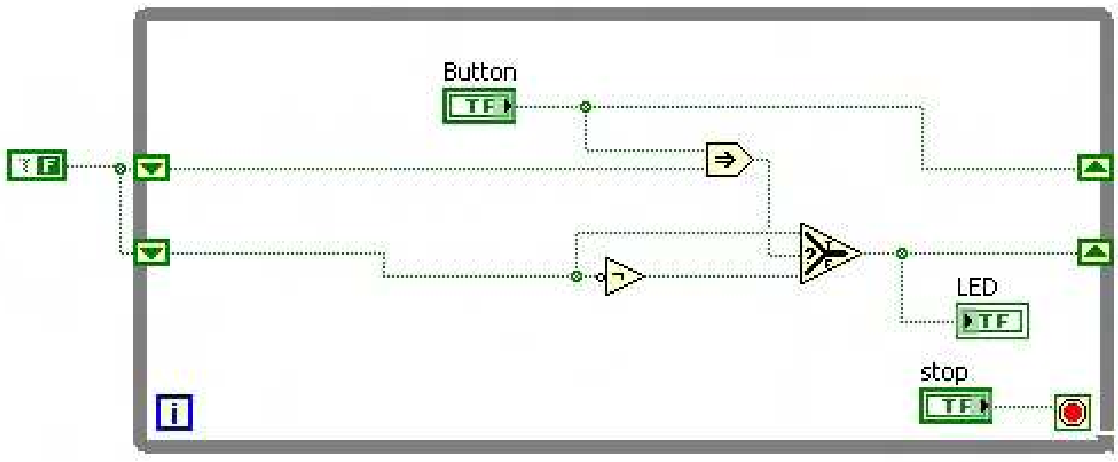
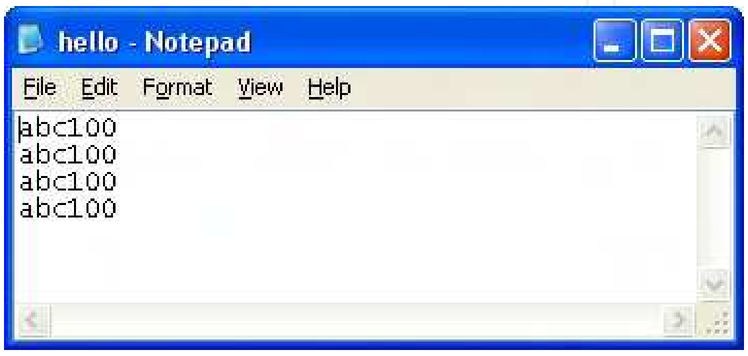


Fig. 6. Button Latch Block Diagram.

2.4 Append file (Write to file line by line)

Surprisingly in LabVIEW we can't find a simple method to append a file. If a program writes to a file for the second time, it will overwrite the previous data. This is what we want to avoid, when we want to make complex logs or even when we want to save the parameters of a program during execution. This program is useful when we make some moves, like mouse, joystick moves, and we want to save the coordinates into a file and maybe the file will be closed an opened in the program more times. We can create programs to move robotic arms with mouse, or to save the specific parameters of measuring equipments.
Fig. 7. shows that after multiple runs of the program, the previous text is not erased.



```
abc100
abc100
abc100
abc100
```

Fig. 7. Text File with Appended Text.

In Fig. 8. we have the standard dataflow of the programming with the settings and the writing to the file, between the opening/creating and closing the file. At the end it's good to put a Simple Error Handler. The file opening/creating VI is set to open or create and a certain path is given to it. The file setting VI is set to have an offset with the length of 0, and is set to write at the end of the file. The writing VI has an input with a text which is concatenated with a new line constant, this way after any running of the program we will have the appended text in a new line, keeping the old information.

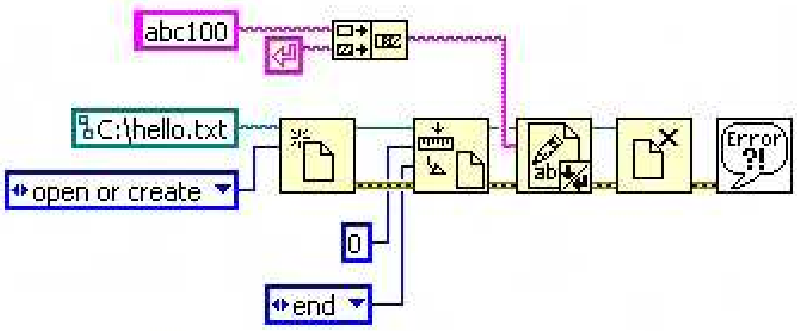


Fig. 8. Text Append Block Diagram.

2.5 Create an arbitrary signal

As we know LabVIEW has a lot of built in signals, but it's impossible to have any arbitrary signal. It has the possibility to create arbitrary signals, but we have found it not really flexible in some situations. It has an arbitrary signal creator with Express VIs, but we always avoid using them, because you can have more control over traditional VIs. The chosen

signal was the trapezoidal signal, which is mostly used when controlling motors. Nobody wants a square pattern for the motors acceleration. In cars, in lifts the motor usually accelerates in a trapezoidal pattern.
As we can see in Fig. 9., the program is working. The hardest part is configuring the parameters. The parameters are configured using equation (1).

$$\begin{cases} N_p \cdot U_{minr} = A - U_0 \\ N_p \cdot U_{minf} = A - U_0 \end{cases} \tag{1}$$

Where N_p is the number of points; U_{minr},U_{minf} are the minimal voltages for rise and fall; A is the amplitude of the signal; U_0 is the start voltage. In our case it will be: 0,0003 * 10000 = 5 – 2, this way we created the trapezoidal signal.

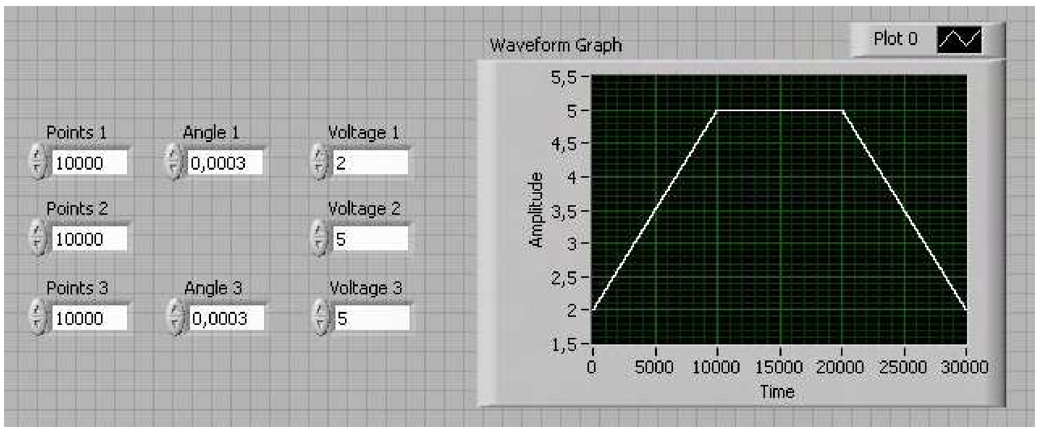

Points 1: 10000
Angle 1: 0,0003
Voltage 1: 2
Points 2: 10000
Voltage 2: 5
Points 3: 10000
Angle 3: 0,0003
Voltage 3: 5

Fig. 9. Trapezoidal Signal Front Panel.

The Block Diagram (Fig. 10.) it is not so complicated, the main thing is the idea to build arrays of rising edge, continuous part and falling edge. The building of the arrays is made with For loops, shift registers and the exit from the loops is made with enable indexing, which is by default at For loops. Maybe the hardest part is the building of the array, which builds three 1D arrays intro one 1D array. Normally LabVIEW makes the built array in multidimensional array, but we need a 1D array, because we want one single signal. This can be done by changing the enable indexing to disable indexing and then change it back to enable indexing again at the exit from the For loop of the horizontal signal.

3. Virtual instruments for the PXI chassis

Our LabVIEW remote lab has 6 PXI experiments working and controllable trough a web browser.

3.1 Transfer characteristic of a NAND gate

The first experiment makes the transfer characteristic of a gate; we made it for a NAND gate. In Fig. 11. we can see the block schematics of the transfer characteristics of the NAND gate.
We generate a rising ramp signal at the NAND gates one input and supply a constant voltage to the other input and measure the voltage at the output, this way we have the transfer characteristics of the gate.

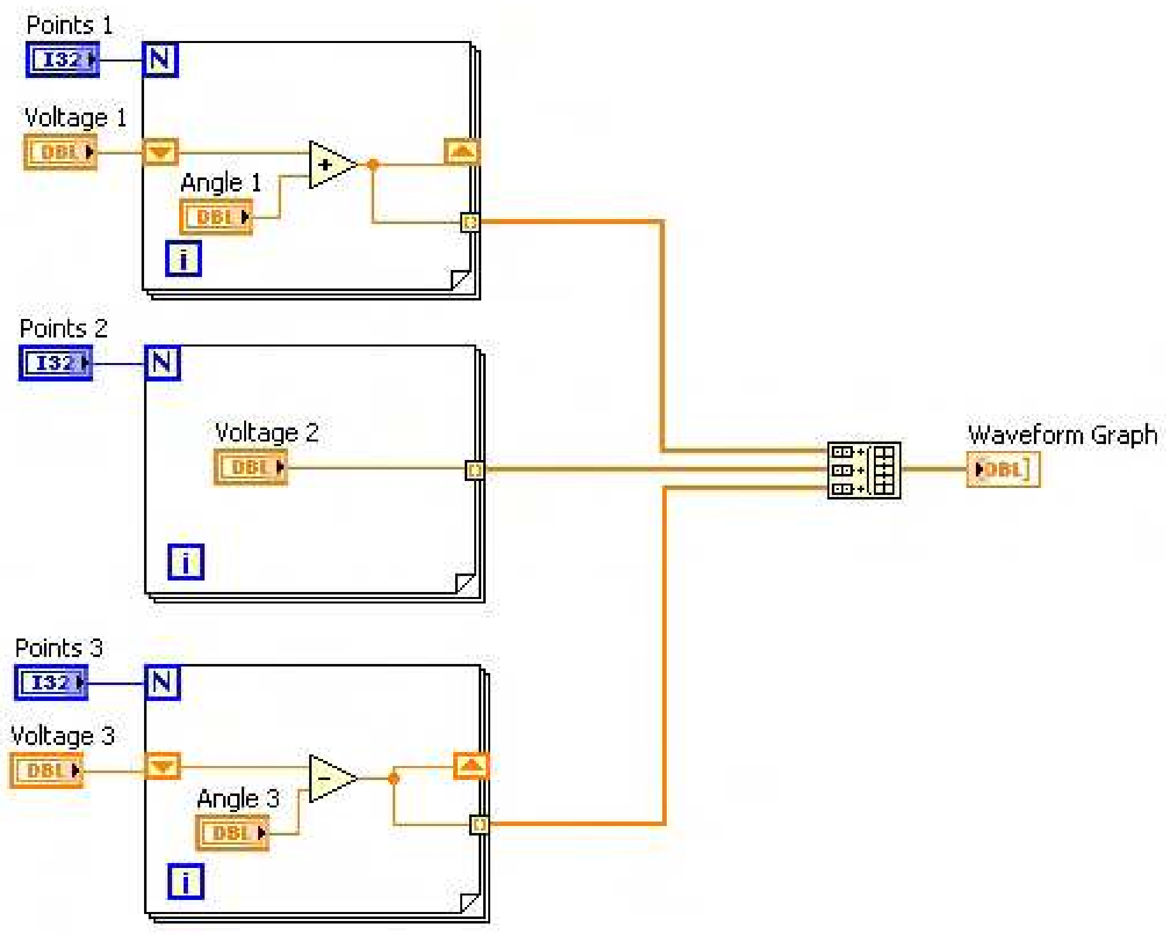


Fig. 10. Trapezoidal Signal Block Diagram.

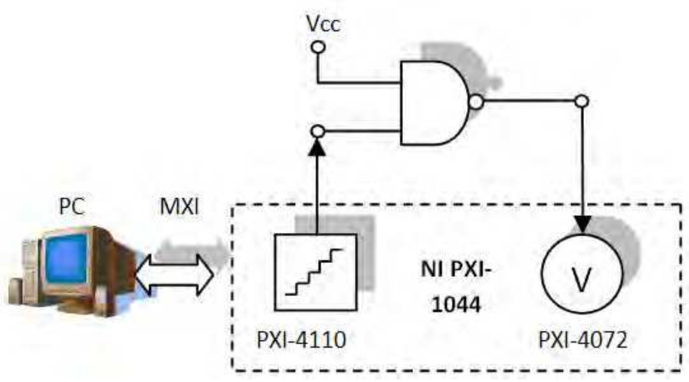


Fig. 11. Block Schematics of the Experimental Setup.

The transfer characteristic is the input voltage as a function of the output voltage like shown in equation (2).

$$U_{out} = f(U_{in}) \tag{2}$$

In Fig. 12. We can see the experimental setup of the NAND gate on a PCB with protection and connectors for accessing. The used NAND gate is integrated in a 74ACT00 IC.
The used NI equipment for this experiment was the NI PXI-4110 power supply and the NI PXI-4072 digital multimeter.

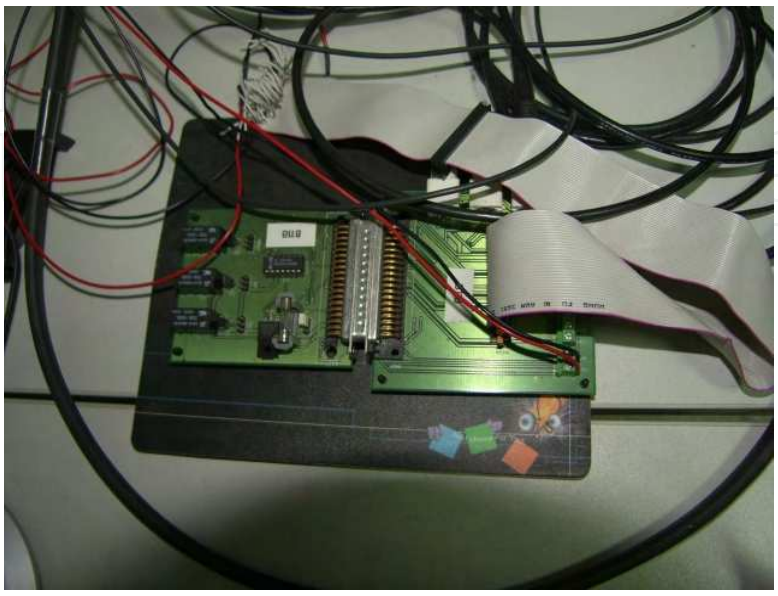
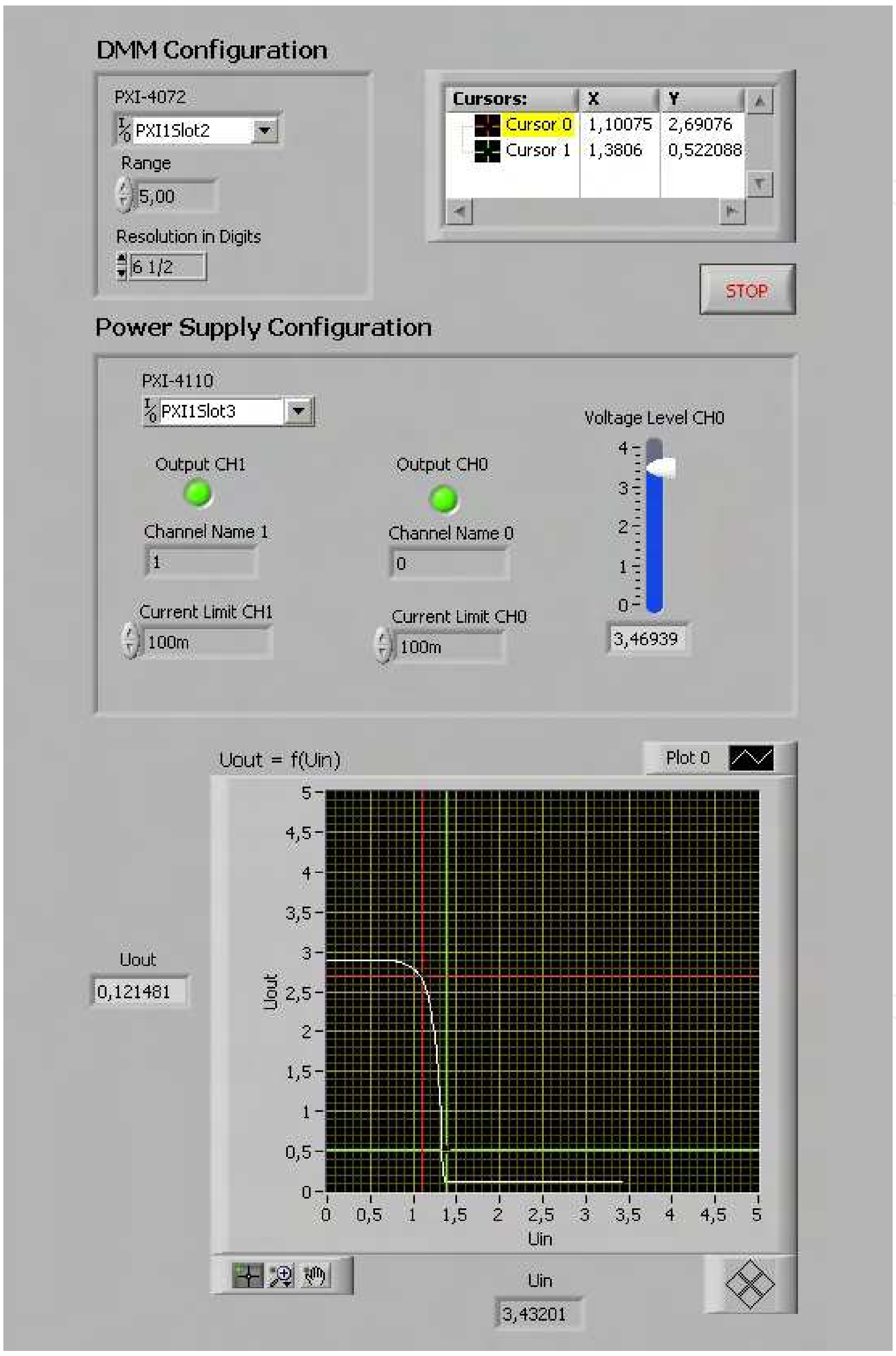


Fig. 12. NAND Gate with PCB.

In Fig. 13. we can see the Front Panel of the experiment.
As we can see we have the digital multimeter (DMM) configuration with reference ID, a range at 5 V and resolution at 6½ digits. We have two channels (0 and 1) of the power supply activated with the current limit set to 100 mA and a slider to set the voltage level from 0 – 4 V. The most important part is the XY Graph where we see the actual transfer characteristic of the gate, we have also indicators of the temporary value on both axes of the graph. We have also cursors to measure certain values.
As we can see is not so complicated to program, the hardest part is maybe that we use traditional NI acquisition cards, not the newer DAQ cards.
In Fig. 14. We have the Block Diagram of the program.
All the instruments are programmed in this pattern, first we have to create the channel and after it close it. Between these two VIs we have the configuration or acquisition and generation. Mostly we have the creation, configuration and closing outside the loop and the acquisition or generation part in the loop.
In out experiment we first create the channel for the DMM and than configure its range ad digits. After it we will create the power supply's channel we configure the voltage and current for the first channel and enable the output, after we will do the same or the other channel. For channel 1 we will measure the output voltage. We will measure the voltage with the multimeter at the output of the NAND gate. Finally we will close an reset the two equipments.
We can remark that LabVIEW uses the read expression for acquisitioning and the write expression for generation, this way the used icons for the VI are mostly a pair of glasses for reading and a pencil for writing.



Cursors:	X	Y
Cursor 0	1,10075	2,69076
Cursor 1	1,3806	0,522088

Fig. 13. Front Panel of the Transfer Characteristic Program.

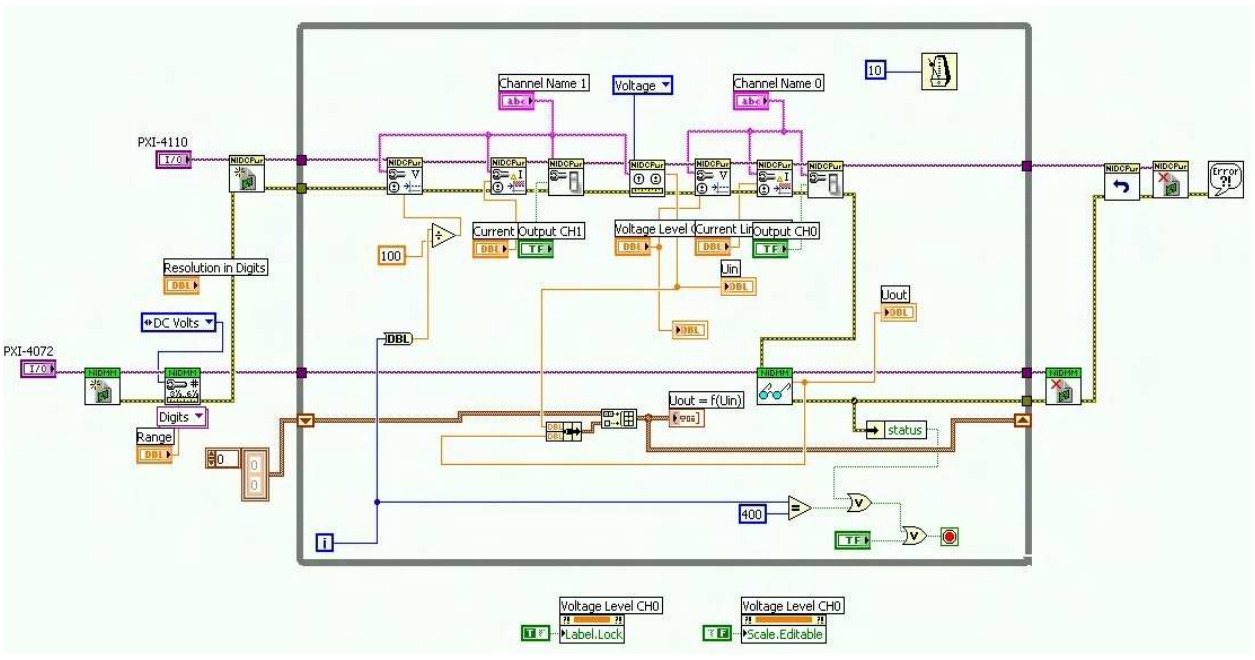


Fig. 14. Block Diagram of the Transfer Characteristic Program.

3.2 Propagation time measuring for a NAND Gate

This experiment uses the same 74ACT00 NAND gate, but measures the gate's propagation time.
In Fig. 15. we can see the block schematics of the experimental setup.

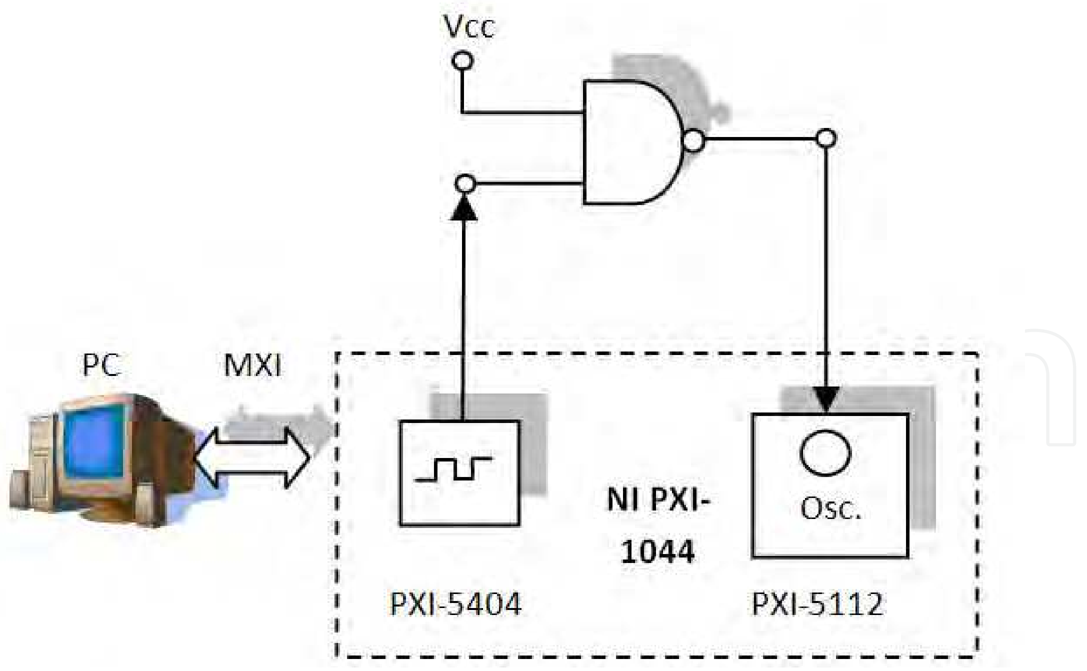
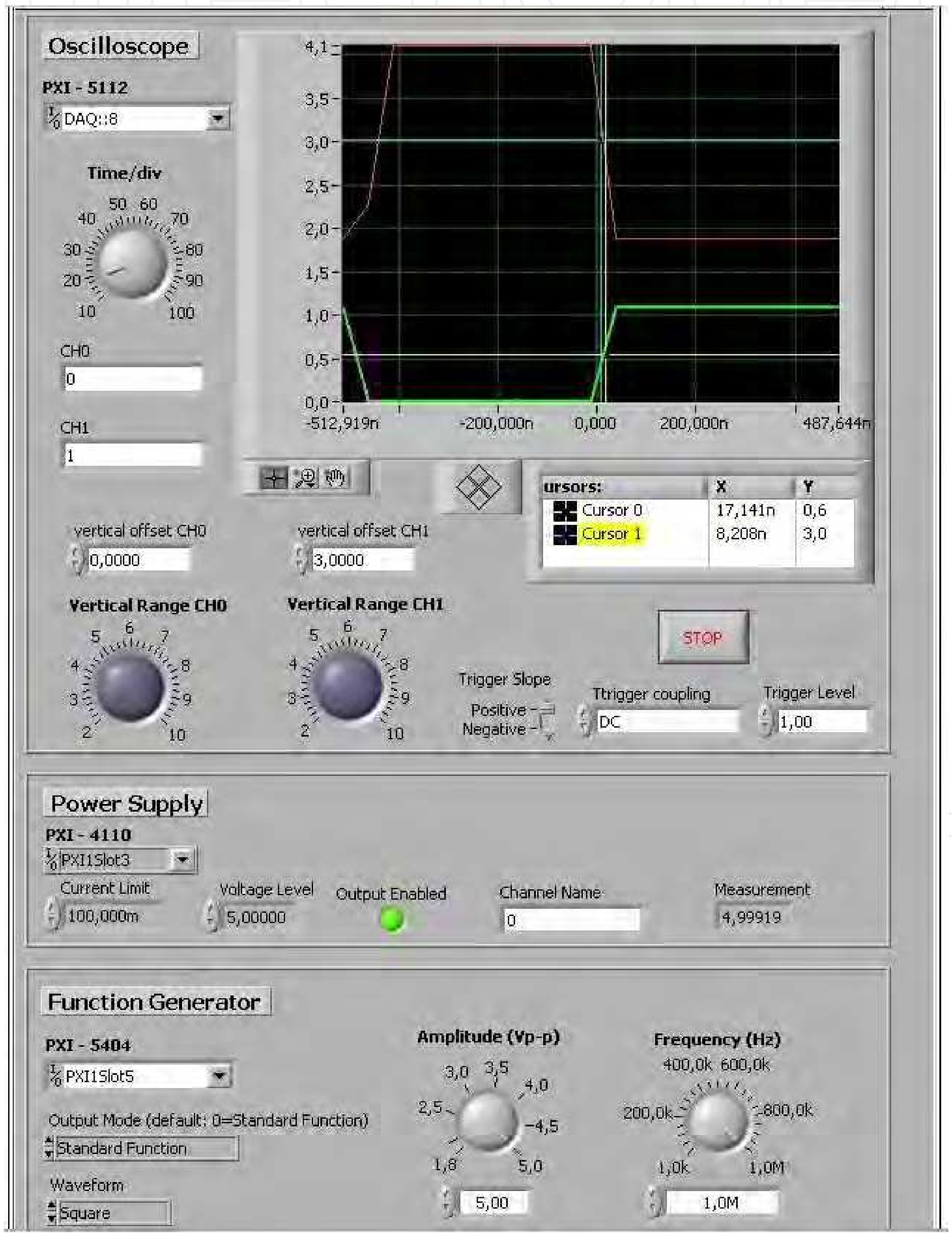


Fig. 15. Block schematics for the propagation time measuring.

We generate a square waveform to the gate's one input with a signal generator and measure its output with an oscilloscope.
The propagation time is given by the formula from equation (3).

$$t_p = \frac{t_{pHL} + t_{pLH}}{2} \tag{3}$$

In the standard TTL gate's case t_{pLH} = 12 ns and t_{pHL} = 8 ns, so tp = 10 ns.
The experimental setup can be seen on Fig. 12. The setup is on the same PCB as for the transfer characteristic.
The used NI equipments are NI PXI-4110 power supply, the NI PXI-5112 oscilloscope and the NI PXI-5412 signal generator.
In Fig. 16. we can see the Front Panel of the experiment.



Oscilloscope
PXI - 5112
DAQ::8
CH0
0
CH1
1
vertical offset CH0
0,0000
vertical offset CH1
3,0000

ursors:	X	Y
Cursor 0	17,141n	0,6
Cursor 1	8,208n	3,0

Trigger Slope
Positive
Negative
Ttrigger coupling
DC
Trigger Level
1,00

Power Supply
PXI - 4110
PXI1Slot3
Current Limit
100,000m
Voltage Level
5,00000
Output Enabled
Channel Name
0
Measurement
4,99919

Function Generator
PXI - 5404
PXI1Slot5
Output Mode (default: 0=Standard Function)
Standard Function
Waveform
Square
5,00
1,0M

Fig. 16. Front Panel of the Transfer Characteristic Program.

As we can see we have the configuration of the oscilloscope, the power supply and the function generator.
The oscilloscope has a configured resource ID and the Time/div setting. It has also both channels activated with 0 V offset for channel 0 and 3 V offset for channel 1 and Volts/div (Vertical Range) dial. It also has some triggering configurations like slope on positive, DC coupling and 1 V level. We have the both graphs (input and output of the NAND gate) on the waveform graph and cursors to measure the propagation time.
The power supply has a basic configuration just for supplying current to the IC. It has resource ID, current limit to 100 mA, 5 V voltage level, output enabled indicator, an indicator showing 0 as the selected channel, and a measurement of the output of the activated channel.
The function generator has resource indicator, the settings for output mode, which is on standard, square waveform type, and the setting for amplitude, which is at 5 V and the setting for frequency, which is at 1 MHz.
In Fig. 17. we can see the Block Diagram of the program.
As we can see we have the initialization for the function generator, the setting of the output mode, the amplitude and frequency setting, the enabling of the output and the starting of the generation.
The oscilloscope is initialized and has standard initialization.
Finally the power supply is initialized.
We enter in the While loop. We configure the voltage and the current of the power supply; we enable its output and measure the output voltage.

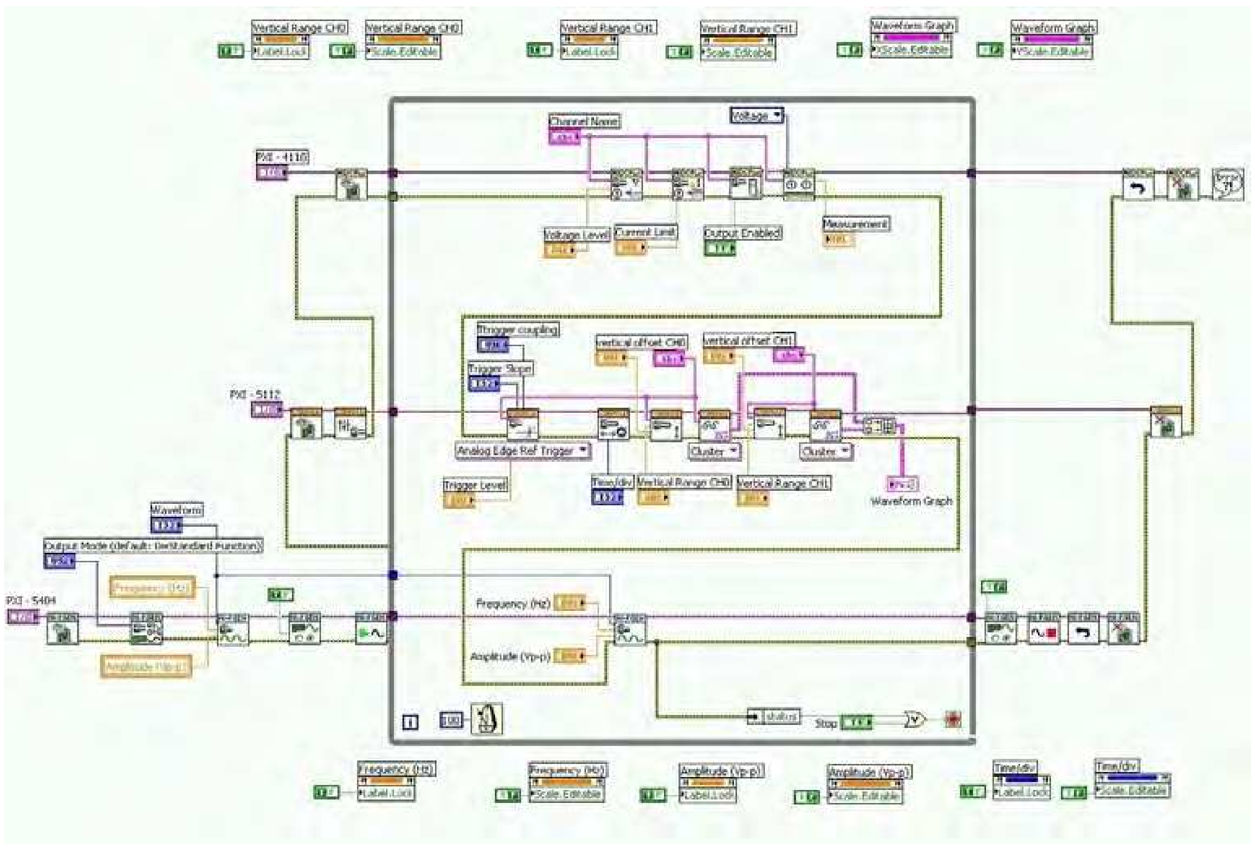


Fig. 17. Block Diagram of the Transfer Characteristic Program.

We continue to the oscilloscope, we configure the trigger settings, the timing settings. After we make the settings for the vertical range and start the readings of the values for representing on the graph both of the channels (0 and 1). We unite the two channels with a Build Array and output it on a Waveform Graph.
For the function generator we can control the frequency and amplitude during the program execution, this way we have it inside the loop too.
When we exit the loop we disabled the output, we stopped the generation we resettled the device. Finally we close the function generator, the oscilloscope, we reset and close the power supply and end the whole execution with a simple error handler to have the error messages in case if something goes wrong.

3.3 Duty cycle analyzer

The duty cycle analyzer experiment has a more complicated setup. We have the 33250A signal generator from Agilent and the NI PXI-6541 logic analyzer from National Instruments.
In Fig. 18. We have the block schematics. We have the Agilent signal generator, which generates a square signal to the Duty Cycle Analyzer. The signal is generated in the logic control and in the PLL. The signal that exists the PLL will be entering in a counter and after I in a display module. The signal is gathered with a logic analyzer from National Instruments and shown on a graph.

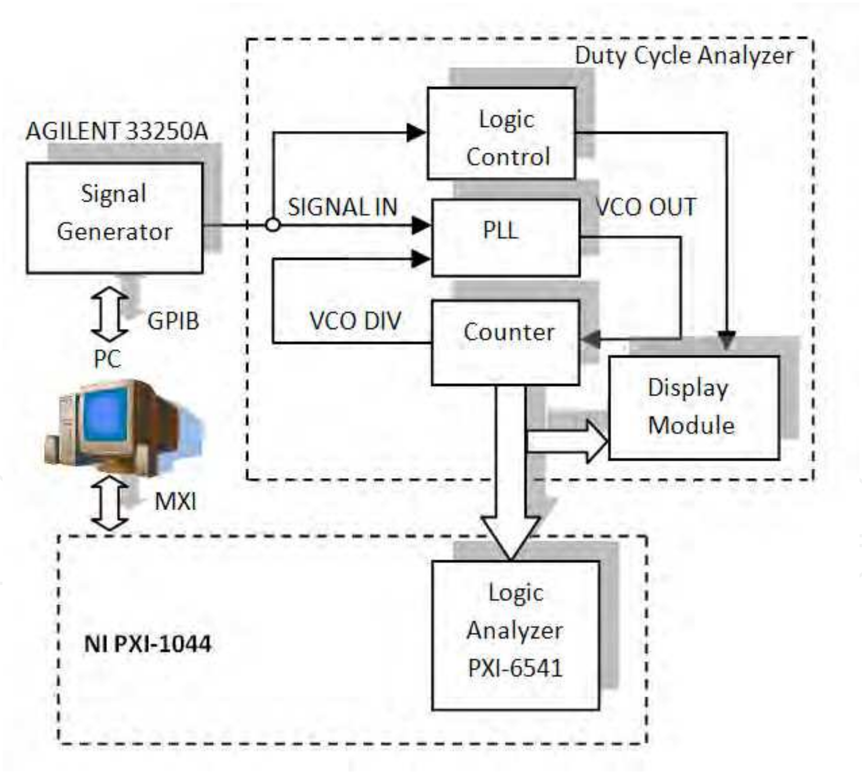


Fig. 18. Block Schematics of the Logic Analyzer Experiment.

We have also an external circuit which can be seen on Fig. 19. This external circuit is a duty cycle analyzer on which 14 different signals are analyzed with the experiment. The circuit has more modules like a logic control, a PLL and a counter and the duty cycle of the square signal is shown on the dual digit seven segment displays.

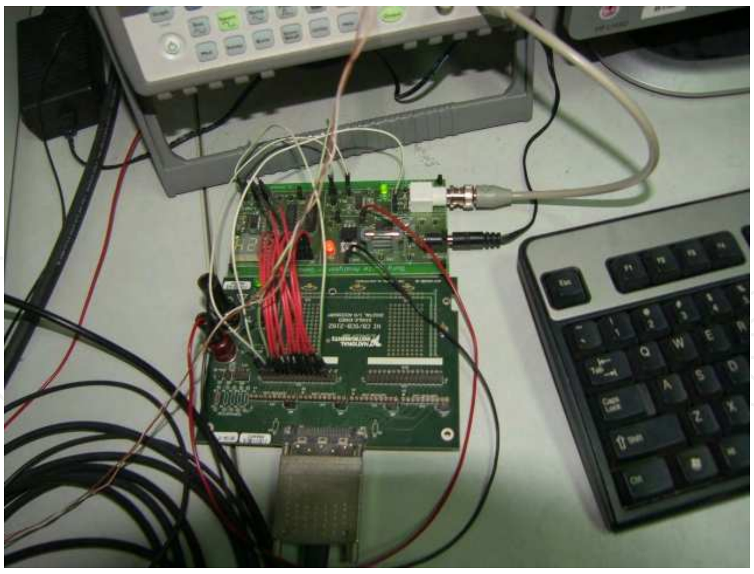
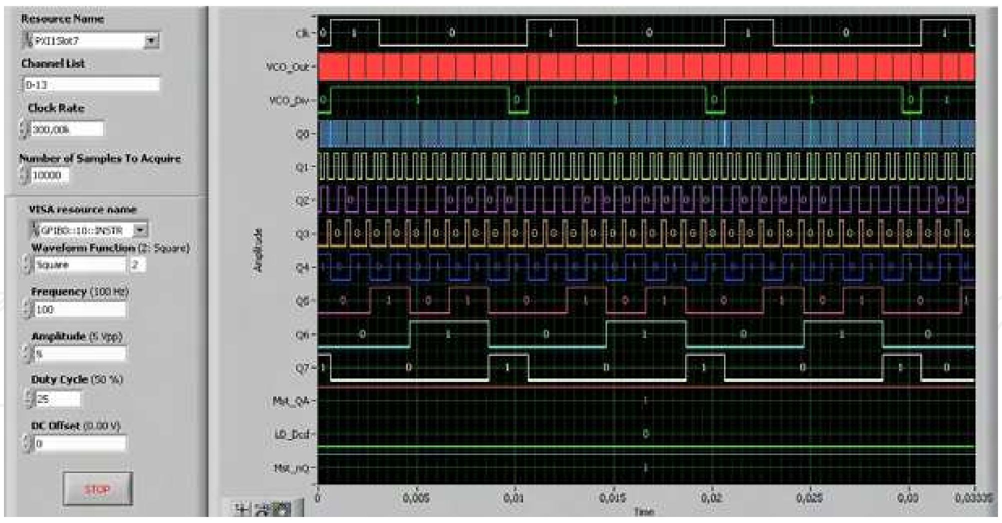


Fig. 19. Experimental Setup of the Logic Analyzer Experiment.

In Fig. 20. we can see the front panel of the Duty Cycle Analyzer program.
For the logic analyzer we have a resource ID, 14 activated channels, and the clock rate at 300 kHz and 10000 acquired samples.
For the Agilent signal generator we have resource ID, square waveform type, 100 Hz for frequency, 5 V amplitude, 25 % duty cycle, 0 V offset. The first signal (Clk) represents the duty cycle. The second signal (VCO_Out) is the representation of the PLL. The third signal (VCO_Div) is one input for the PLL. Q0 – Q7 is the counter part. Mst_QA, LD_Dcd, Mst_nQ represent logic levels for testing purposes.



Resource Name: PXI1Slot7
Channel List: 0-13
Clock Rate: 300,00k
Number of Samples To Acquire: 10000
VISA resource name: GPIB0::10::INSTR
Waveform Function (2: Square): Square 2
Frequency (100 Hz): 100
Amplitude (5 Vpp): 5
Duty Cycle (50 %): 25
DC Offset (0.00 V): 0

Fig. 20. Front Panel of the Logic Analyzer Experiment.

In Fig. 21. we have the block diagram of the experiment.
We have the initialization of the logic analyzer. In a while loop we have the configuring of the channels, settings of the clock, setting of the buffer (Number of Samples To Acquire) and finally we represent the data.
The Agilent signal generator is initialized, the amplitude, frequency, offset and waveform type is configured. We continue with the duty cycle is configuration, the output is enabling

and at the end we close the signal generator. Finally the logic analyzer is closed too, and the program is ended with a simple error handler.

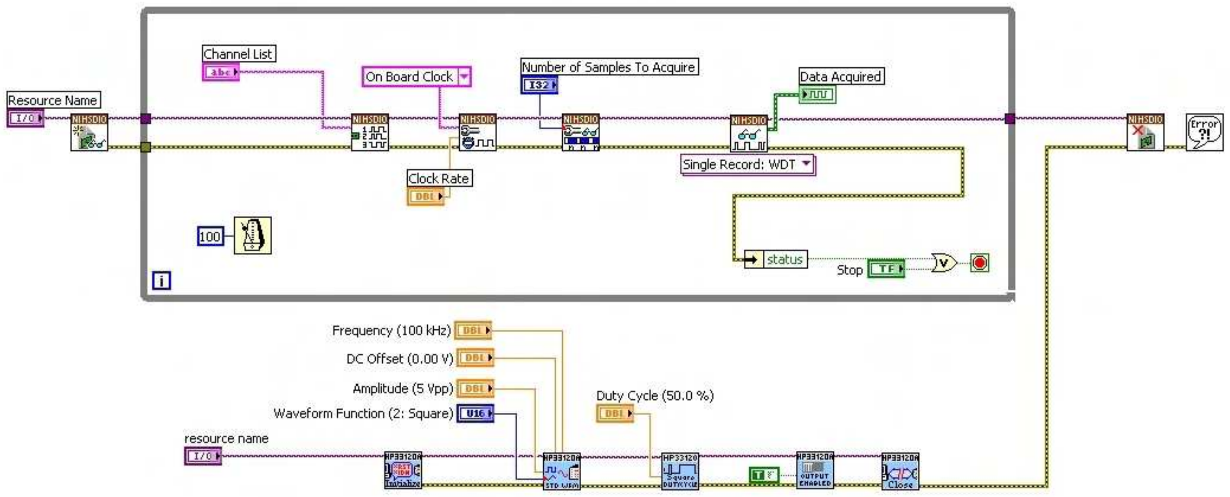


Fig. 21. Block Diagram of the Logic Analyzer Experiment.

3.4 Simple motor control

In Fig. 22. we can see the Block Diagram of the experiment.

This experiment uses only PXI instruments and thee PICDEM Mechatronics board from Microchip for amplifying and PID control. We have the NI PXI-6115 as the signal generator which generates a trapezoidal signal using the method presented in paragraph 2.5. The signal is amplified for the motors with the PICDEM Mechatroncis board; this board has also an implemented PID algorithm. The motor has a disk with slots, which rotates between an optocoupler pair, similar to and old mouse with ball. The signal from the optocoupler is sent in a signal; amplifying and conditioning circuit. The output signal is sent to the NI PXI-6608 counter and with a simple formula the RPM is calculated. The signal that the optcoupler reads should be similar to the generated signal, in our case both trapezoids should look the same.

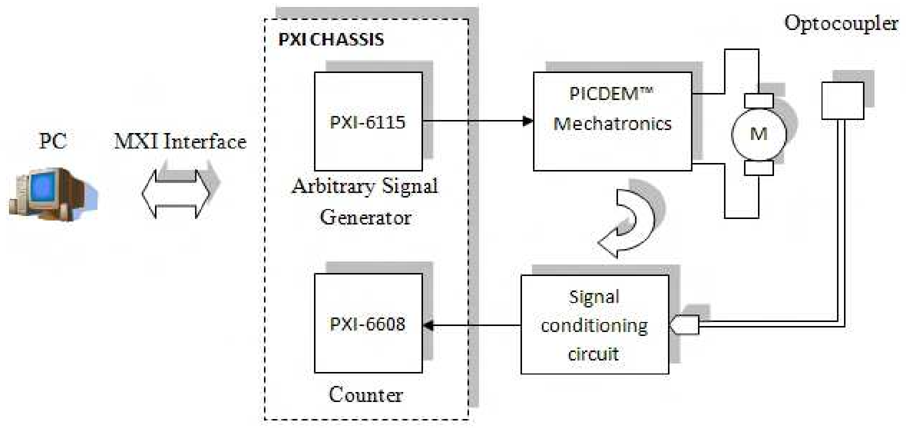


Fig. 22. Block Schematics of the Simple Motor Control Program.

In Fig. 23. we can se the PICDEM Mechatronics board and the motor with the disk with two slots.

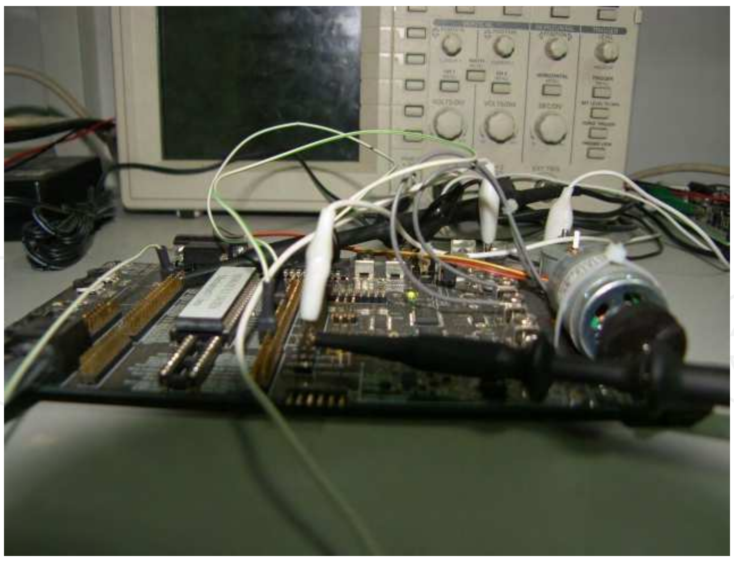
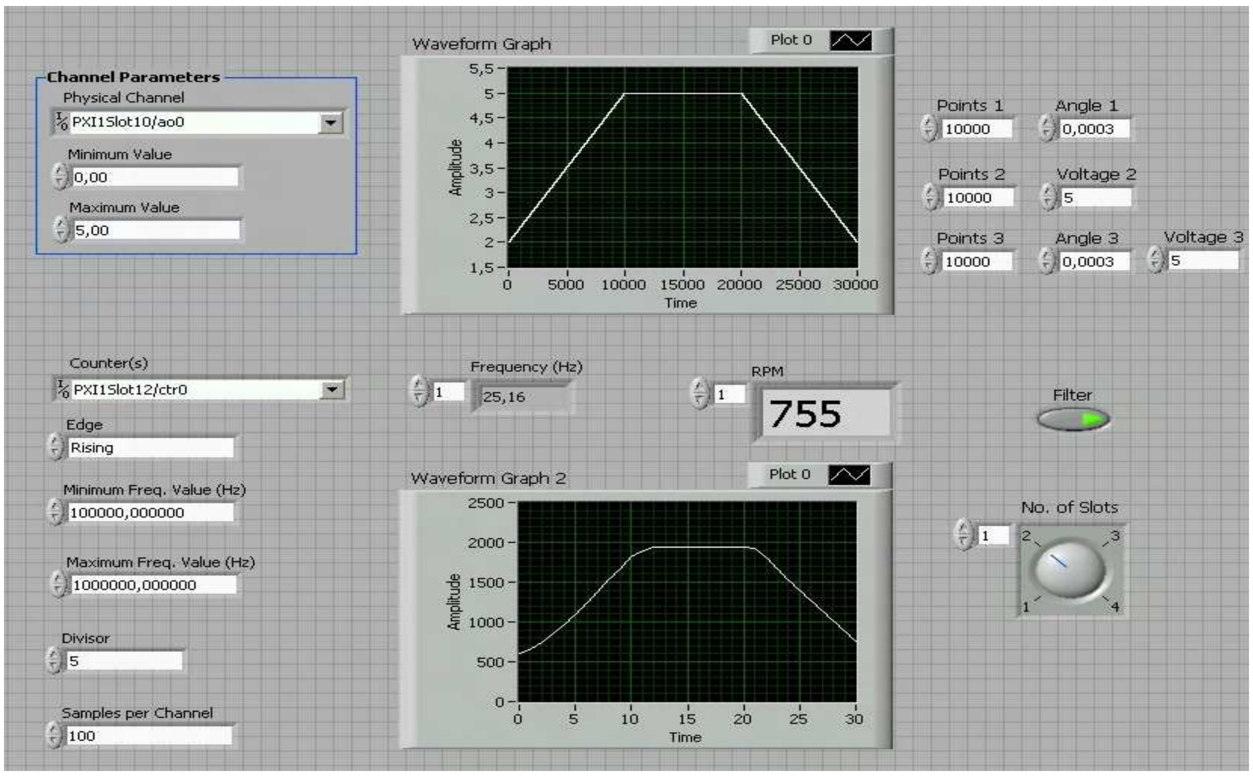


Fig. 23. Experimental Setup of the Simple Motor Control Program.

In Fig. 24. we have the Front Panel of the program in LabVIEW with the trapezoidal signal generation on Waveform Graph and the RPM readings represented on Waveform Graph 2. The signal generation method is similar to the method presented in paragraph 2.5.
In the upper part of the Front Panel we have the signal generation configuration, where we have the resource ID of the signal generation DAQ and the minimum value a t 0 V and the maximum value at 5 V.
The second part of the Front Panel is the counter configuration with resource ID, rising edge, minim frequency at 100 KHz and maximum frequency at 1 MHz, 100 samples per



Channel Parameters
Physical Channel: PXI1Slot10/ao0
Minimum Value: 0,00
Maximum Value: 5,00

Points 1: 10000 Angle 1: 0,0003
Points 2: 10000 Voltage 2: 5
Points 3: 10000 Angle 3: 0,0003 Voltage 3: 5

Counter(s): PXI1Slot12/ctr0
Edge: Rising
Minimum Freq. Value (Hz): 100000,000000
Maximum Freq. Value (Hz): 1000000,000000
Divisor: 5
Samples per Channel: 100

Frequency (Hz): 1 25,16
RPM: 1 755
Filter
No. of Slots: 1

Fig. 24. Front Panel of the Simple Motor Control Program.

channel and the sampling divisor is 5. We have also indicators for frequency and RPM. We have also a median filter which can be deactivated with a button and we have a slot setting dial, which must be set to be equal with the number of slots of the disk present on the motor. This should be correctly set to calculate the RPM.
In Fig. 25. we can see the Block Diagram of the program. We have two parallel while loops. The first loop is for the trapezoidal signal generation. We have first the creation of the channel, after we have the start of the generation process. Next we enter the While loop, we have here the trapezoidal signal creation similar to paragraph 2.5. This is the method how a signal is generated by a DAQ board; we provide the numbers (samples) of the signal and the use the write signal VI, which has the icon with the pencil. After we exit the loop with closing the instrument (delete task) and we finish the program with the simple error handler.
The second loop is for the counter configuration. We have channel creation for the counter, the timing settings (sample configuration), the start of the execution continues, followed by the signal reading, which is filtered and with a specific formula we calculate the RPM and represent it on a Waveform Graph.
Equation (4) shows how to calculate the RPM from the frequency.

$$RPM = \frac{f}{N_s} \cdot 60s \tag{4}$$

Finally we close the instrument and we handle errors. We put everything in the loop to be controllable during execution.

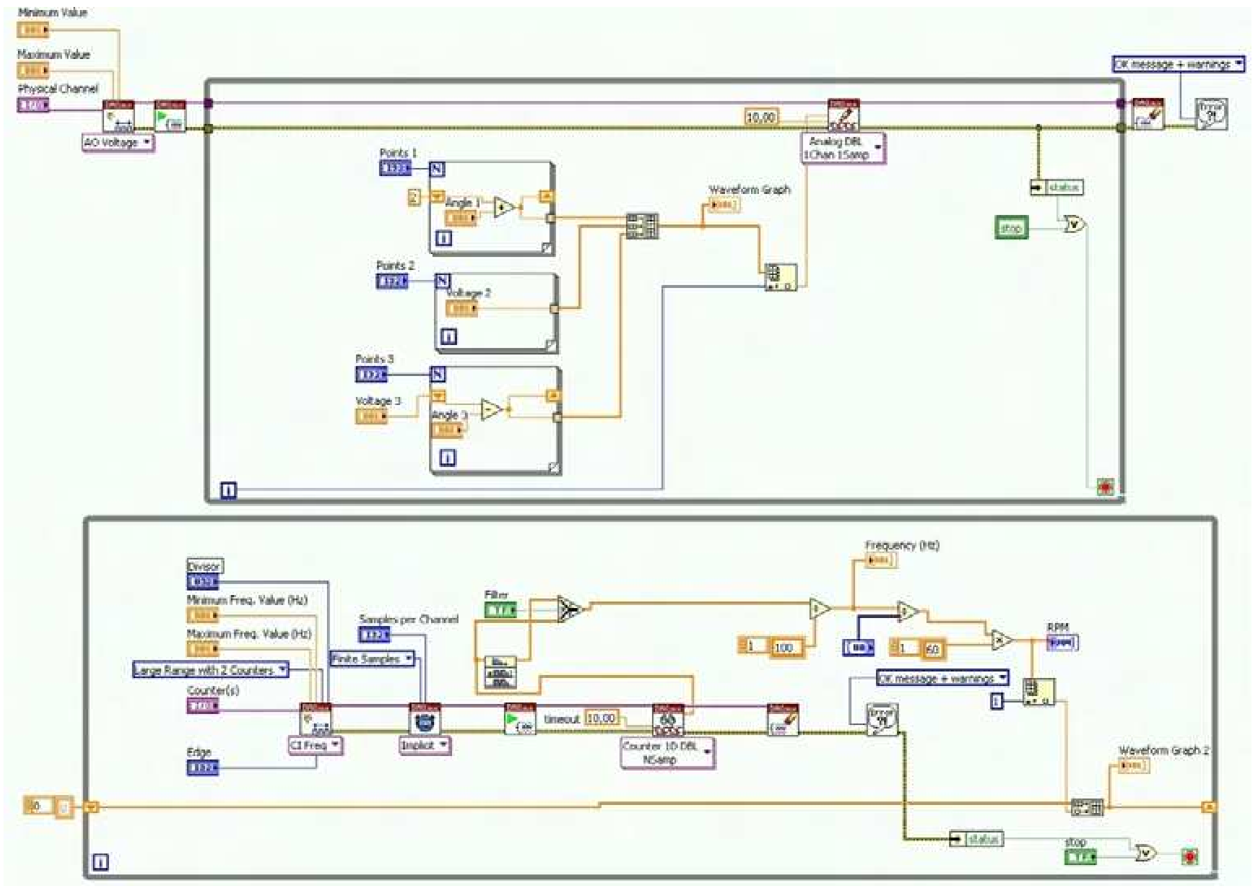


Fig. 25. Block Diagram of the Simple Motor Control Program.

3.5 Simple temperature measuring

In Fig. 26. we can see the setup for the temperature measurement. We have an LM35 centigrade sensor and an NI USB-6251 acquisition board. We also made the experiment with the NI USB-6009 smaller acquisition board and worked very well. The idea of this experiment is to measure voltage given from the temperature sensor and with the formula from the datasheet we convert the voltage to temperature.

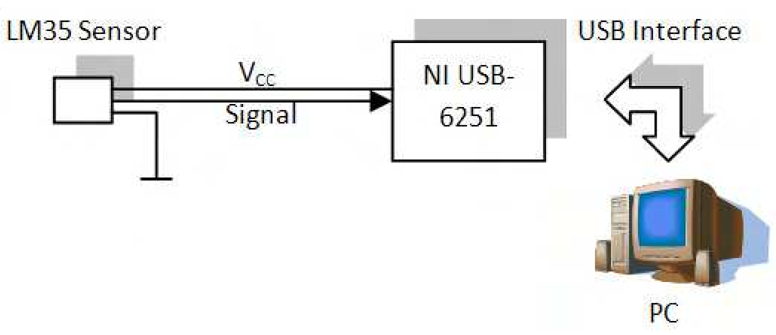
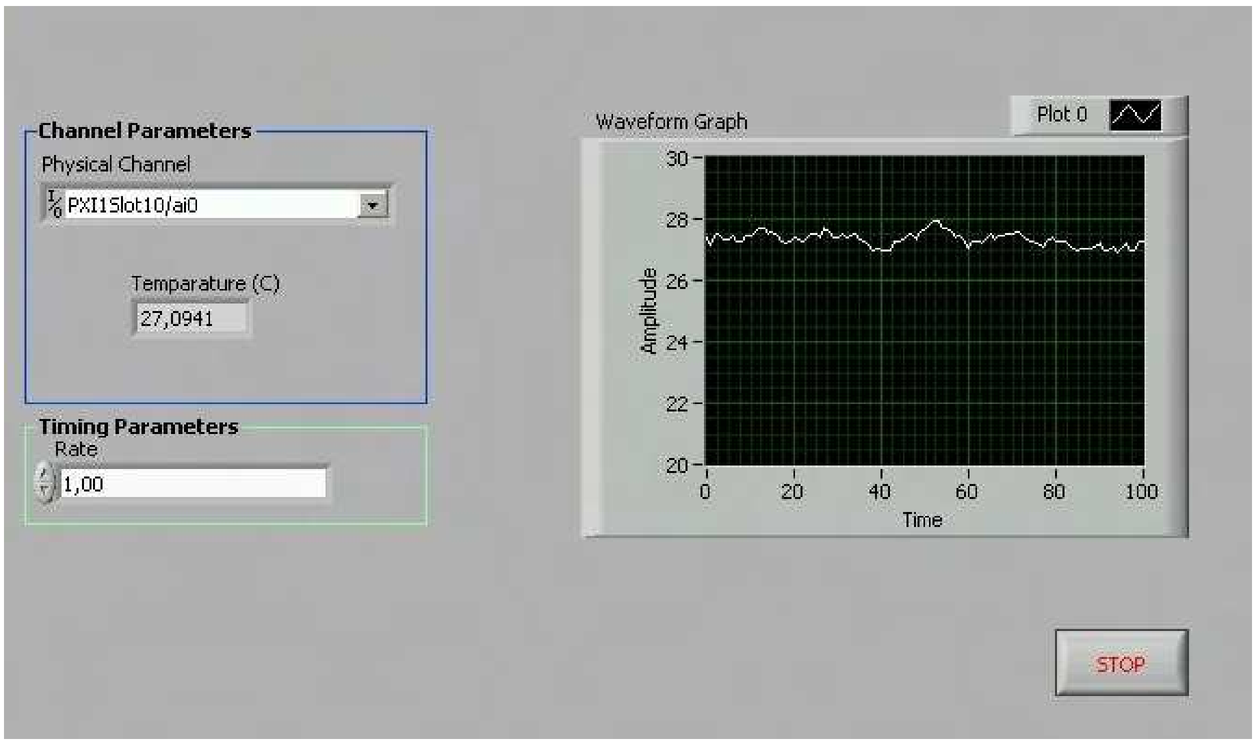


Fig. 26. Block Schematics of the Simple Temperature Measuring.

In Fig. 27. we have the Front Panel of the program. We have resource ID, timing rate at 1 and the actual temperature shown on an indicator and on a Waveform Graph.



Channel Parameters
Physical Channel
PXI1Slot10/ai0
Temparature (C)
27,0941
Timing Parameters
Rate
1,00

Fig. 27. Front Panel of the Simple Temperature Measuring.

In Fig. 28. we have Block Diagram of the program. We start with the channel creation, the setting of the timing (samples) and the start of the process. We enter in the While loop, here we have the reading of the samples and some mean value calculation over 10 samples. The temperature at the output of the LM35 centigrade sensor is its output voltage multiplied by 100. Finally we close the instrument and handle errors.

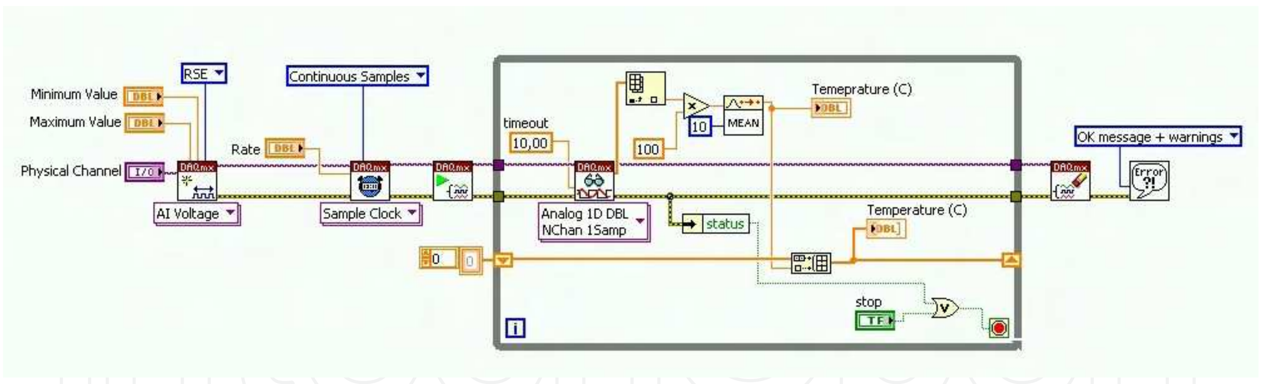


Fig. 28. Bock Diagram of the Simple Temperature Measuring.

3.6 Power supply testing

This experiment represents a more complex functional test. If we have many samples it's not the best solution. For many samples graphs are not indicated, but for only one sample this is the best test. With a graphs we can see really if the power supply is working.
The signal is generated with an Agilent 33250A function generator and the input and output signals are viewed using an NI PXI-5112 oscilloscope (Fig. 29.)

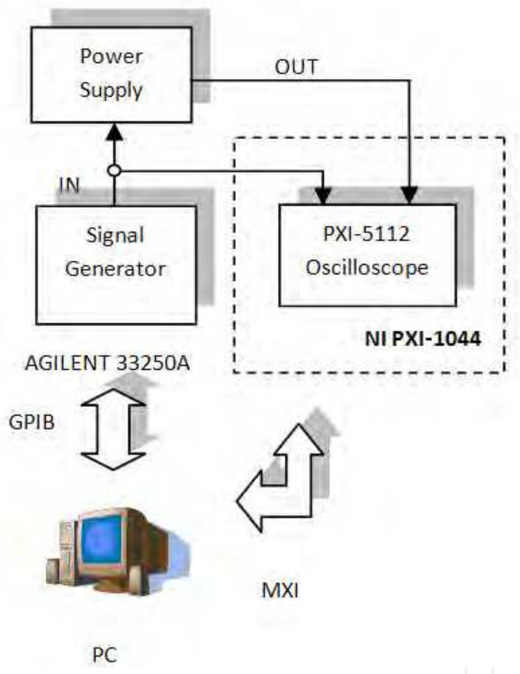


Fig. 29. Block Schematics of the AC-DC Power supply.
In Fig. 30. we can see the AC-DC power supply.

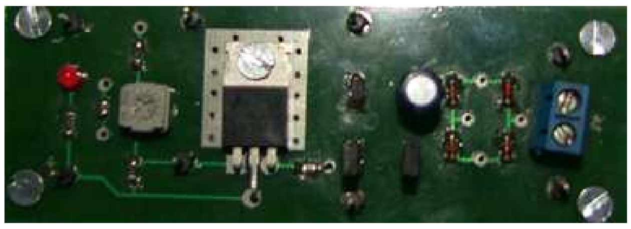
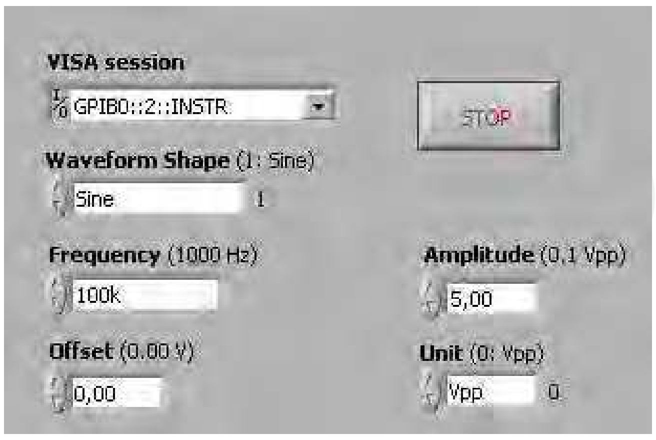


Fig. 30. Power Supply.

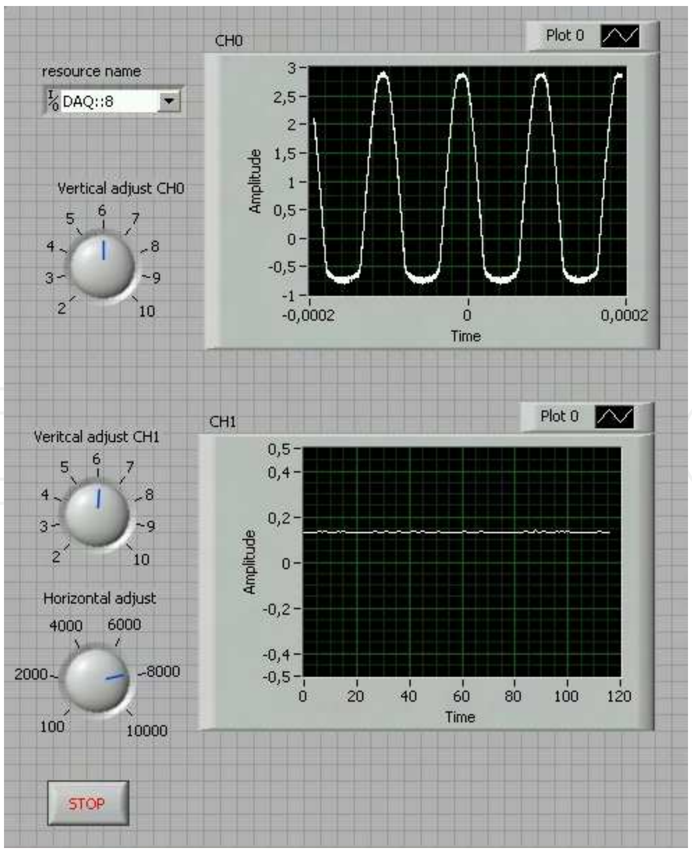
In Fig. 31. we have the Front Panel of the Agilent 33250A signal generator. As we can see we have resource ID, the waveform type is sine wave, the frequency 100 kHz, the offset is 0 V and the amplitude is 5 V.



VISA session: GPIB0::2::INSTR
Waveform Shape (1: Sine): Sine 1
Frequency (1000 Hz): 100k
Offset (0.00 V): 0,00
Amplitude (0,1 Vpp): 5,00
Unit (0: Vpp): Vpp 0

Fig. 31. Front Panel for the Agilent 33250A Function / Arbitrary Waveform Generator.

In Fig. 32. we can see the Front Panel of the oscilloscope application. We can see the resource ID of the oscilloscope and the horizontal adjust dial. We have two channels activated with two graphs and two vertical adjust dials. The first graph represents the input AC signal and the second graph represents the output DC signal. From this graphs we ca see that the power supply works correctly and we have an AC – DC power supply.



resource name: DAQ::8

Fig. 32. Font Panel for the Power Supply Testing with Oscilloscope.

In Fig. 33. we have the 33250A Agilent signal generator programming. We have between the initialization and closing a While loop. In the loop we have the waveform configuration VI and the output enable VI.

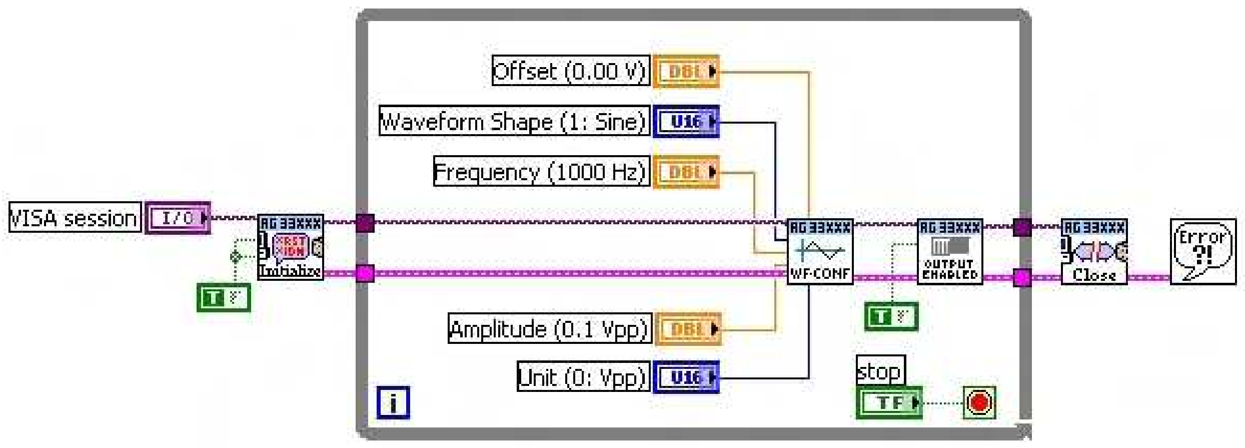


Fig. 33. Block Diagram for the Agilent 33250A Function / Arbitrary Waveform Generator.

In Fig. 34. we have the Block Diagram of the oscilloscope application.
We start with the channel creation. We enter in the while loop. We have some timing settings for the horizontal adjust and the vertical adjust and readings for the both activated channels (channel 0 and channel 1). We put a Bessel filter to the output signal of the voltage supply. When we exit from the while loop we close the instrument and handle errors.

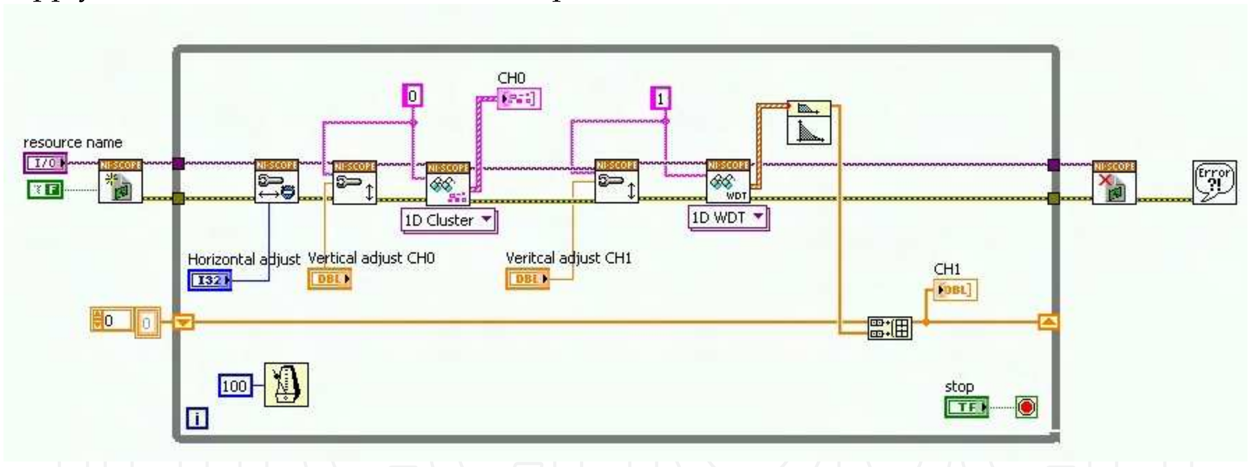


Fig. 34. Block Diagram for the Power Supply.

4. Virtual instruments for the CompactRIO chassis

4.1 Advanced motor control

In Fig. 35. we can see, we used both the PXI and the CompactRIO chassis. The trapezoidal signal is generated with the NI PXI-4110 Power Supply. The signal then is amplified with the NI 9505 H – bridge. The CompactRIO is programmed to make a PID loop too. The motor is connected to NI 9505 H – bridge. The motor has a disk with 100 slots, which rotates between an optocoupler pair.
The optocoupler is connected to the counter to read the frequency and calculate the RPM.

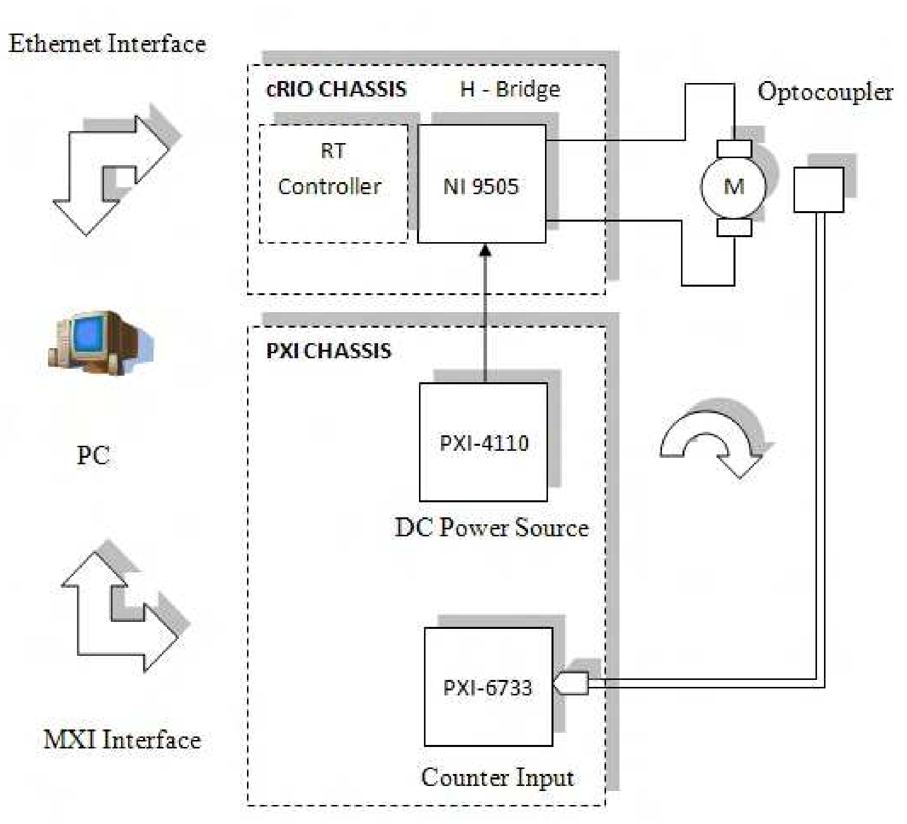


Fig. 35. Block Schematics of the Advanced Motor Control.

In Fig. 36. we can see the experimental setup with the motor and the CompactRIO.

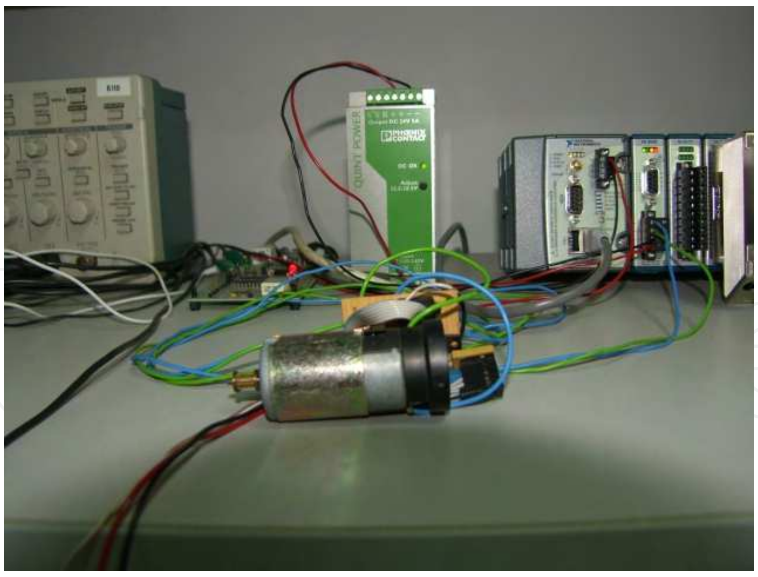


Fig. 36. Experimental Setup of the Advanced Motor Control.

In Fig. 37. we can see the front panel of the signal generation program, which is similar to the program presented in paragraph 3.4. The trapezoid generation uses the method presented in paragraph 2.5. In the signal generation program the NI PXI-4110 power supply is used. In the Front Panel we have the resource ID, channel 1 is activated, the current limit is set to 100 mA and the voltage is measured at the output of the power supply.

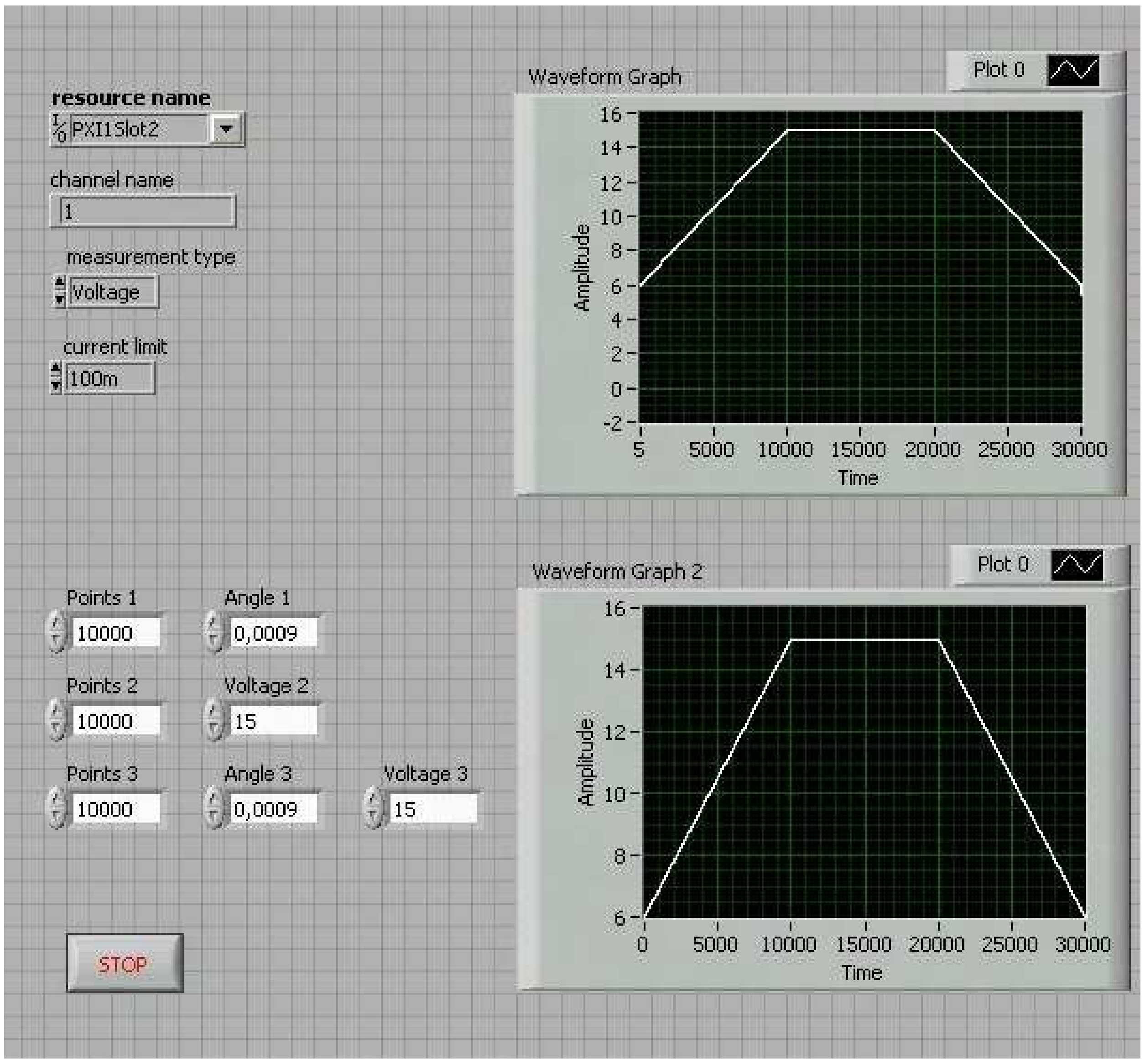

resource name: PXI1Slot2
channel name: 1
measurement type: Voltage
current limit: 100m
Points 1: 10000
Angle 1: 0,0009
Points 2: 10000
Voltage 2: 15
Points 3: 10000
Angle 3: 0,0009
Voltage 3: 15

Fig. 37. Front Panel of the Signal Generation.

In Fig. 38. we have the RPM reading program. We have resource ID, rising edge, minimum frequency set to 100 kHz and maximum set to 1 MHz, 100 samples acquisitioned at the channel and the sampling divisor set to 5. We have put indicator for frequency and RPM and a Waveform Graph to represent the RPM. This graph should be similar to the generated signal.

In Fig. 39. we have the Front Panel of the FPGA programming. We have a lot of controls and indicators, which are use to control some functions of the NI 9505 H – bridge, which are also shown on the card with some LEDs. We have Enable Drive, Disable Drive and Enable Emergency-Stop. We show the status of the Drive, the Drive Fault and the Overtemperature Fault, the supply current (Vsup Present) present and the presents of the analog input's trigger (AI Trigger). We have the Current Loop Rate set to 50 µs, this rate is the rate of the PID loop. We have the PID parameters: 350 for the proportional gain, 300 for the integral gain and 5500 for the derivative gain. We have a current setpoint at a negative value (in our case -16) this for rotating the motor in one direction, if it's positive the motor rotates in other

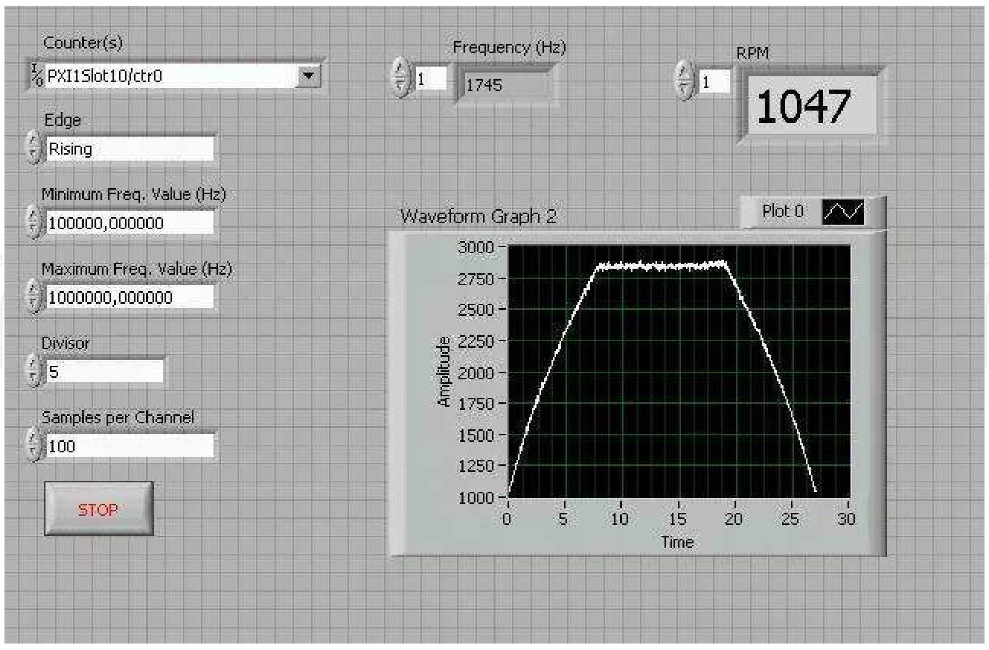

Counter(s): PXI1Slot10/ctr0
Edge: Rising
Minimum Freq. Value (Hz): 100000,000000
Maximum Freq. Value (Hz): 1000000,000000
Divisor: 5
Samples per Channel: 100
STOP
Frequency (Hz): 1 1745
RPM: 1 1047

Fig. 38. Front Panel of the RPM Reading.

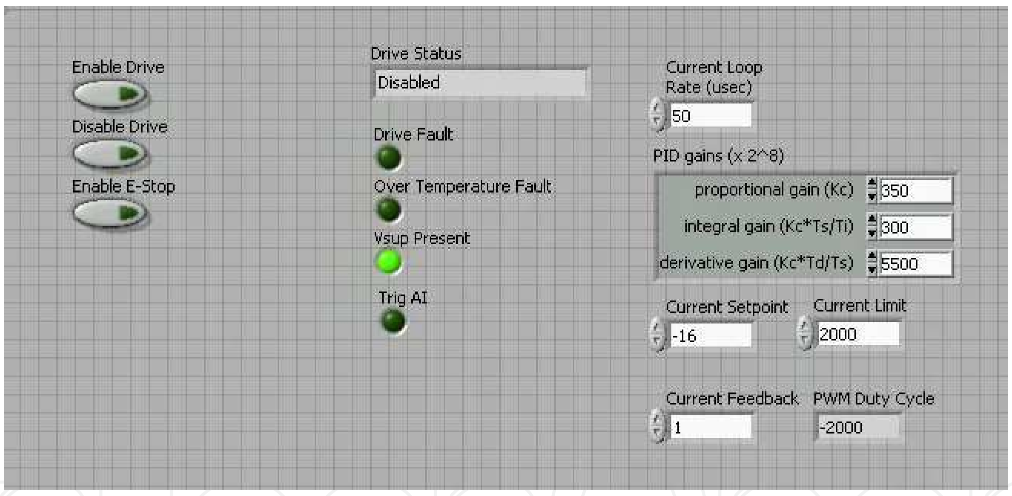

Enable Drive
Disable Drive
Enable E-Stop
Drive Status: Disabled
Drive Fault
Over Temperature Fault
Vsup Present
Trig AI
Current Loop Rate (usec): 50
PID gains (x 2^8)
proportional gain (Kc): 350
integral gain (Kc*Ts/Ti): 300
derivative gain (Kc*Td/Ts): 5500
Current Setpoint: -16
Current Limit: 2000
Current Feedback: 1
PWM Duty Cycle: -2000

Fig. 39. Front Panel of the FPGA Part.

direction. We limit the current at a certain value (in our case 2000) and we set current feedback to 1, after we read the PWM Duty Cycle. The PWM Duty Cycle has the same sign as the Current Setpoint, this way to show the direction of the motor rotation. The Front Panel is almost the same with this one for all Block Diagrams which will be presented next.
In Fig. 40. we can see the Block Diagram of the trapeze generation program. This program is similar to the first While loop from Fig. 25., but it's made with a power supply, not a DAQ. The trapeze is generated with the method presented in paragraph 2.5.
The programming starts with channel creation. After entering in the While loop we set the voltage. Here is connected the trapezoidal signal. After we set the current, we enable output and we measure the voltage at the power supply's ports. When we exit from the While loop, we disable output and close the instrument.

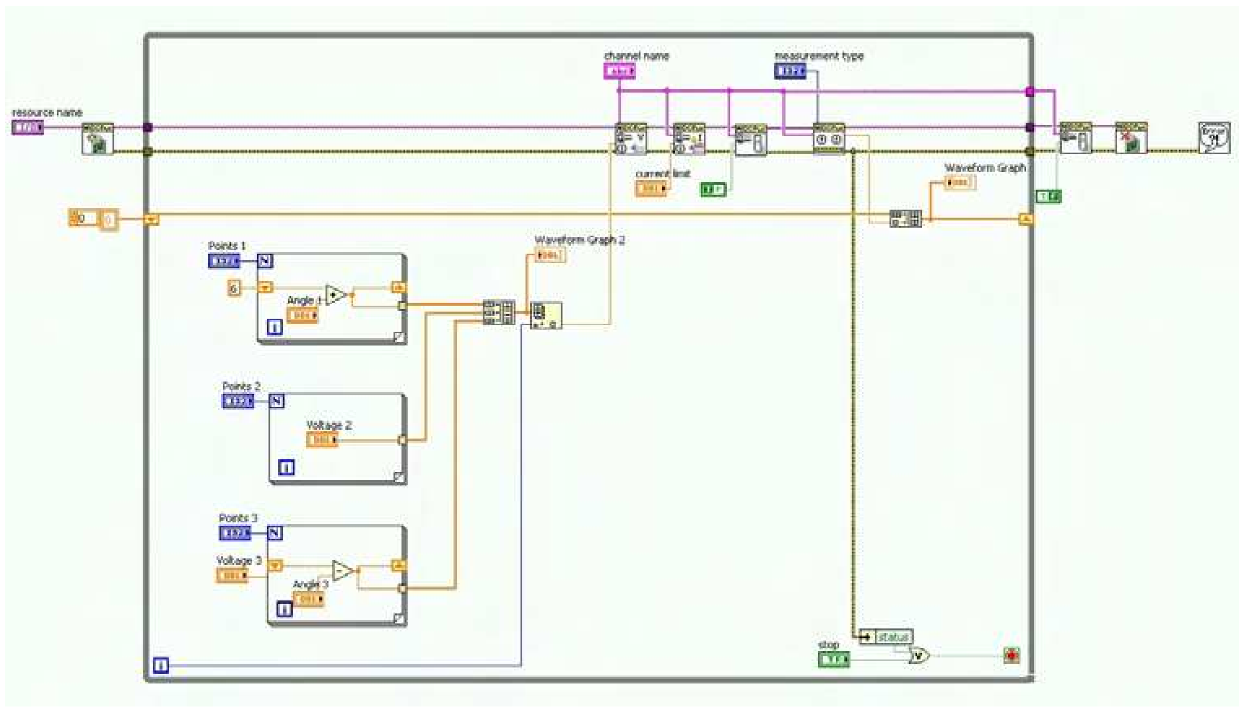


Fig. 40. Block Diagram of Trapeze Generation with Power Supply.

In Fig. 41. we have the RPM reading Block Diagram, which is same with the second While loop from Fig. 25.

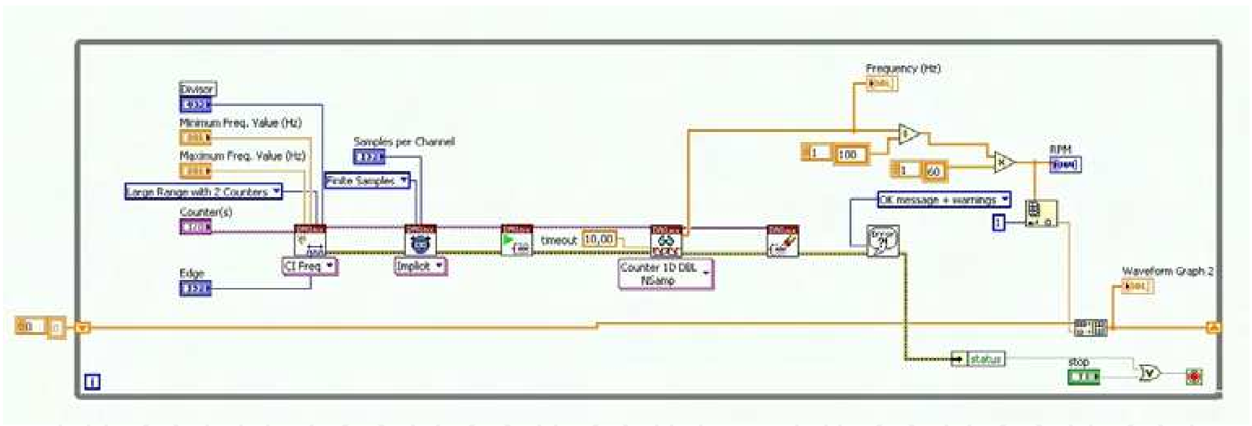


Fig. 41. RPM Reading Block Diagram.

In Fig. 42. we can see the we have a big While loop with all the controls used in the FPGA part and called by the Real-Time controller. At the left part we have loaded the FPGA part and at the output we closed it and we handle errors.
In Fig. 43. we have the block Diagram of the networked real-time host. This VI is made copying the VI form Fig. 41. and adding global variables to send the data trough network to the Windows host. We have the global variables also before of the FPGA part loading for initialization.
In Fig. 44. we have the Windows host with the same global variables as in the networked real-time host, but mirrored. This means if in the networked real-time host we had a control in the windows host we have indicator and vice-versa. We can imagine an invisible line between the global variables with the same name starting from the control to the indicator.

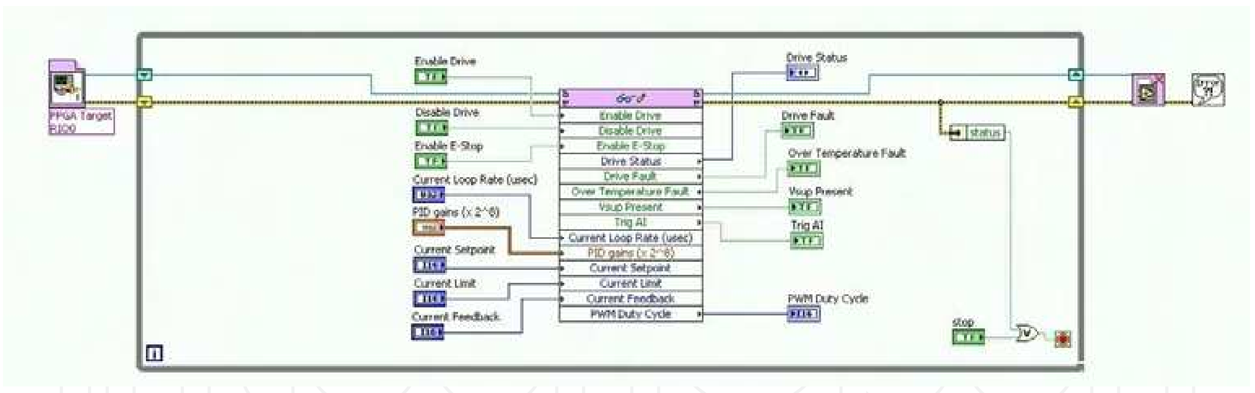


Fig. 42. Block Diagram of the Real-Time Host.

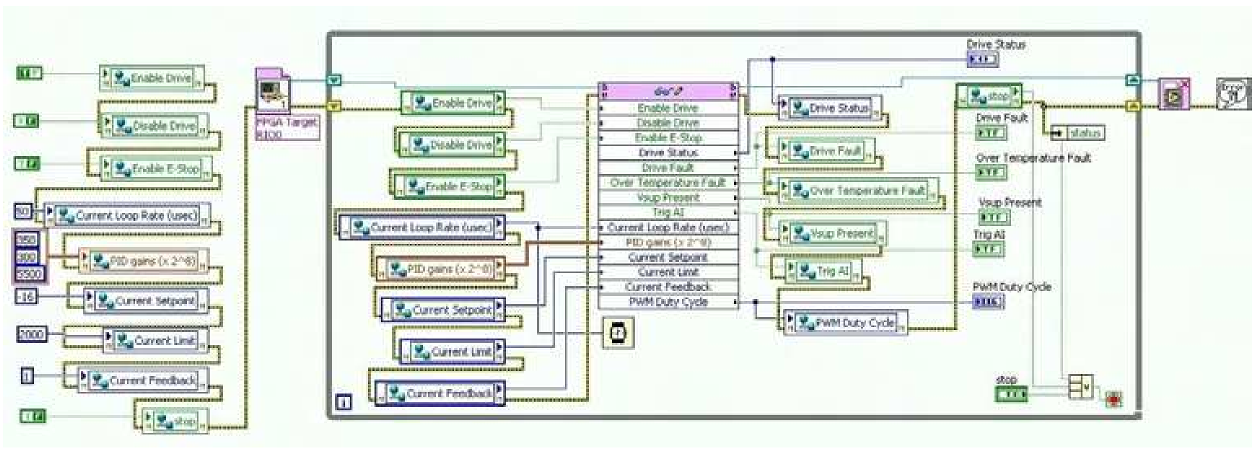


Fig. 43. Block Diagram of the Networked Real-Time Host.

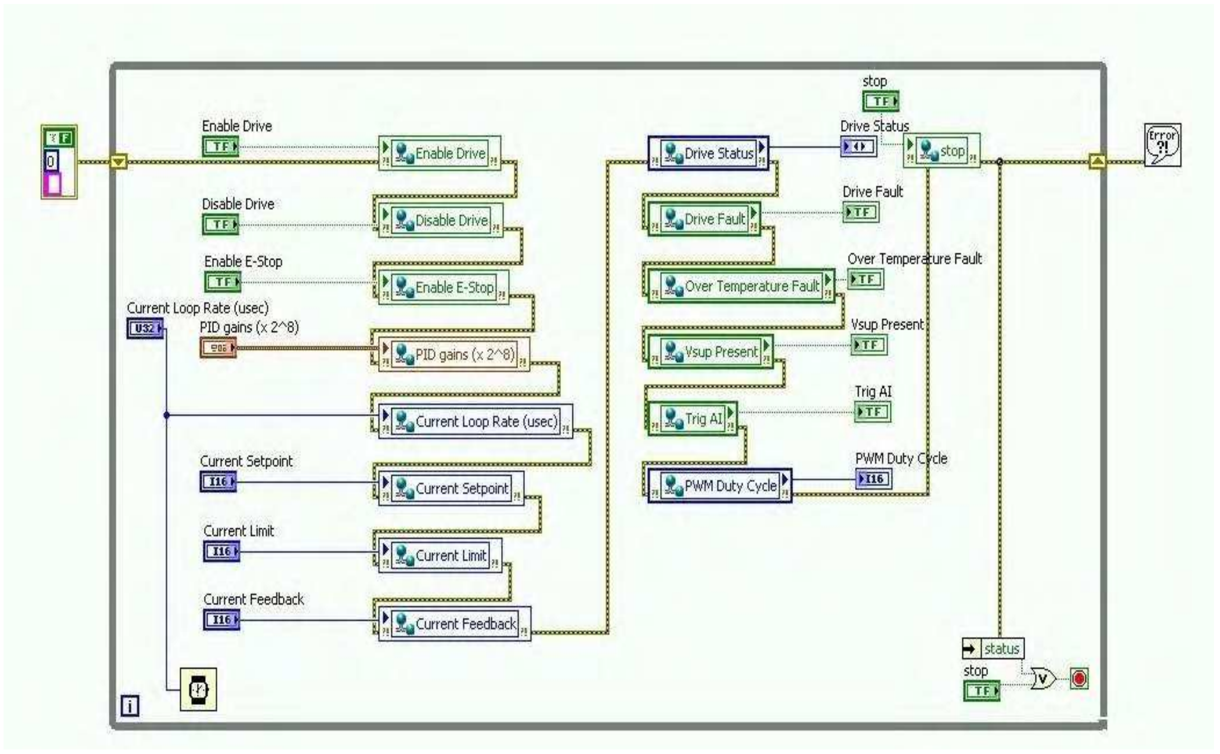


Fig. 44. Block Diagram of the Windows Host.

global variables communicates not only between different VIs on the same computer, but between VIs on different computers, connected via networks cables if they are in the same subnet.

4.2 Advanced temperature measuring

In Fig. 45. we can see the block schematics of the experiment.
We have the LM35 centigrade sensor connected to the NI 9201 analog input C series module which is connected to a CompactRIO. The CompactRIO is connected via Ethernet interface to the PC.

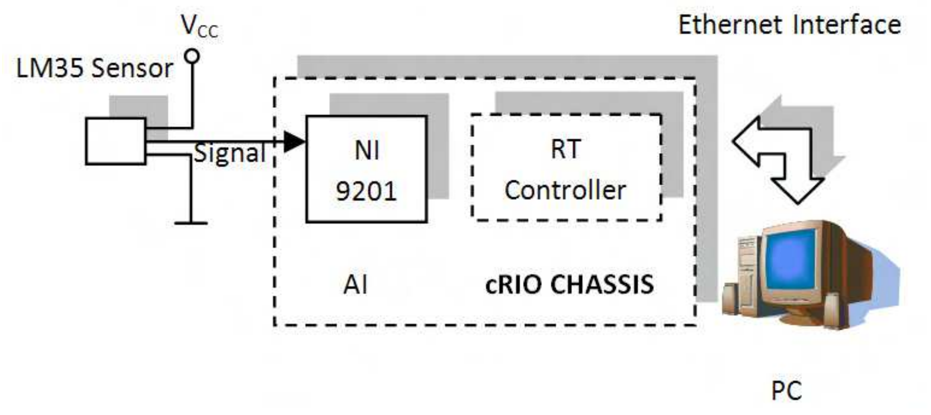


Fig. 45. Block Schematics of the Advanced Temperature Measuring.

In Fig. 46. we have the Front Panel of the FPGA part. We have only raw data collected here; we just measure the voltage in mV with the NI 9201 analog input module. We ca set the loop timing and we visualize errors.

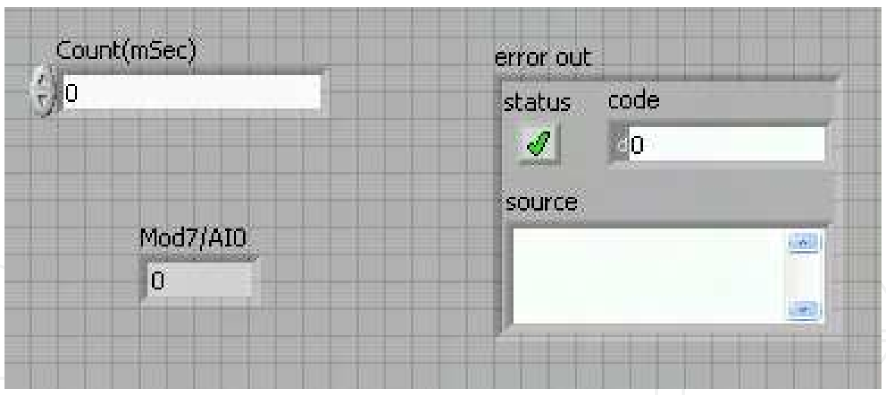

Count(mSec)
0
Mod7/AI0
0
error out
status ✓
code
0
source

Fig. 46. Front Panel of the FPGA part.

In Fig. 47. we have the Front Panel of the networked real-time host part. Here we have the voltage converted in temperature in ºC, we visualize errors and we display the log of the temperature o both Waveform Graph and Chart. Here we ca see a big log of the temperature over one day when turning the air conditioning system OFF and ON.
In Fig. 48. we can see the Front Panel of the Windows host part. We have here a special stop button (Stop Windows GUI). With this button we can stop only the windows part of the program, the temperature acquisition will continue on the real-time system. We have indicator for the Temperature and we have a Waveform Chart for graph. We have c dial control for the sample interval in ms.

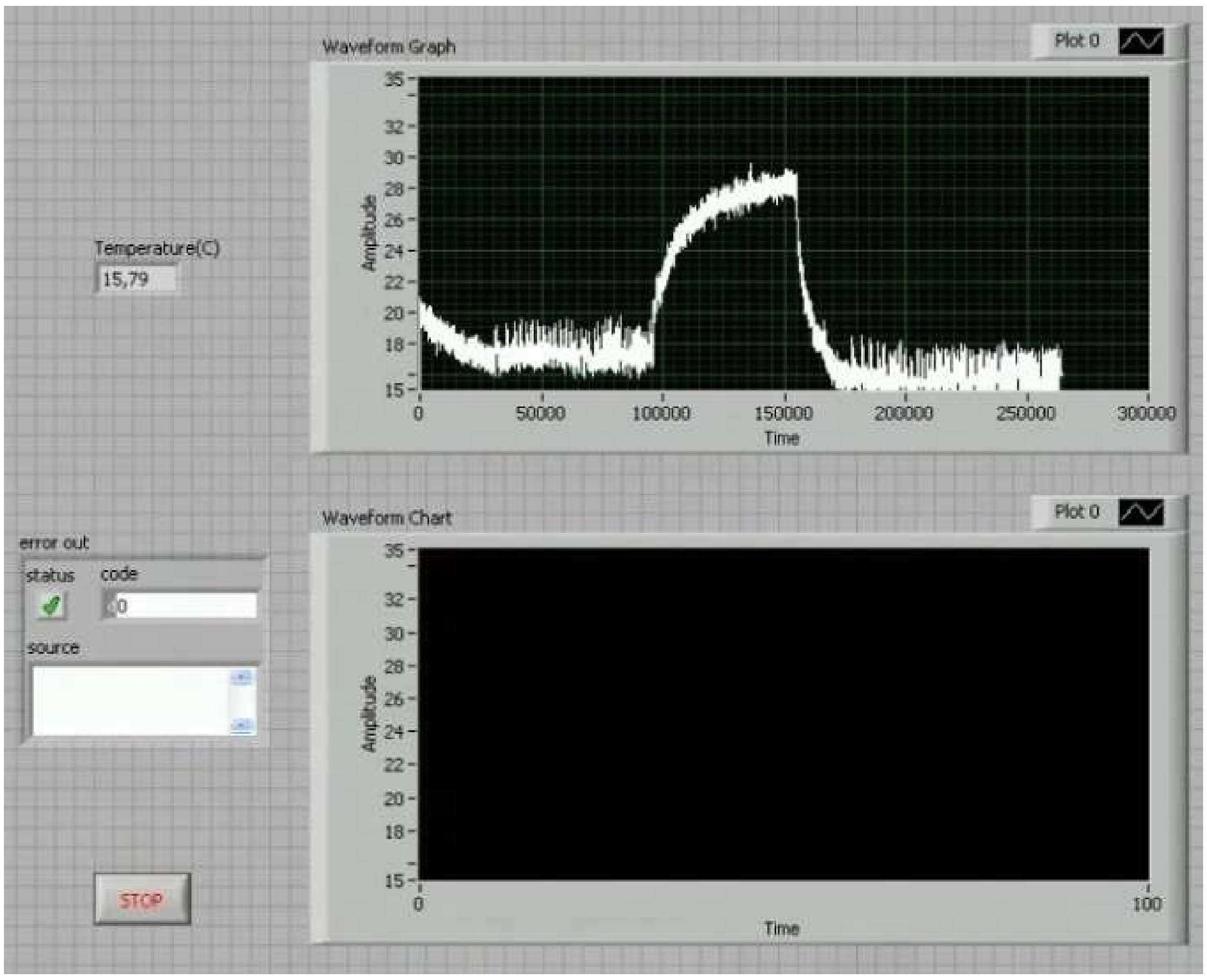


Fig. 47. Front Panel of the Networked Real-Time Host.

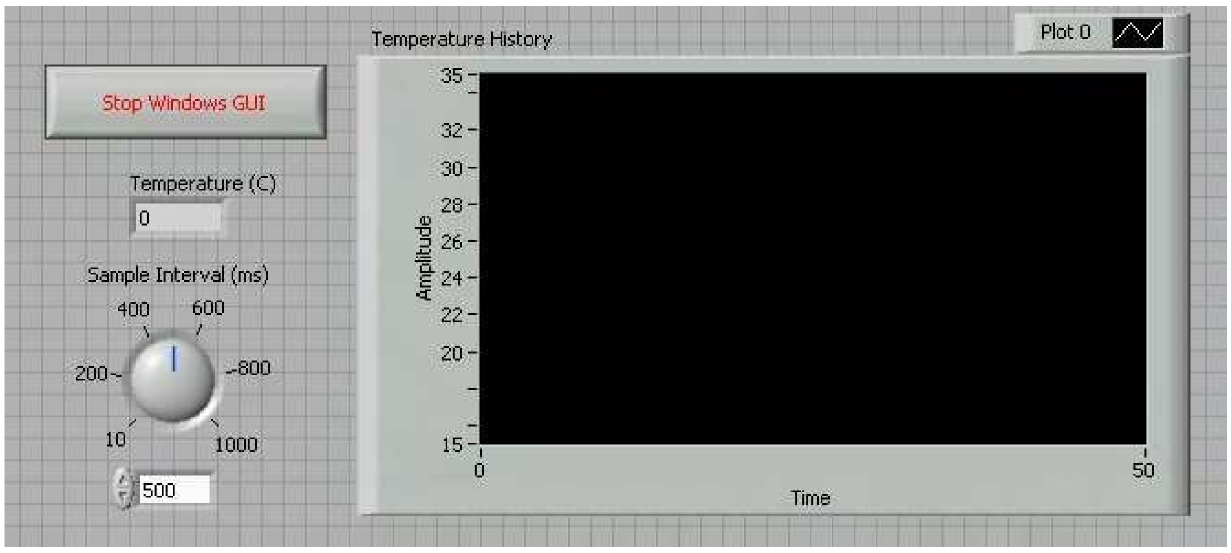


Fig. 48. Front Panel of the Windows Host.

In Fig. 49. we have the block diagram of the FPGA part. We have While loop and a Flat Sequence with some special FPGA timing and the acquisition with the FPGA node.

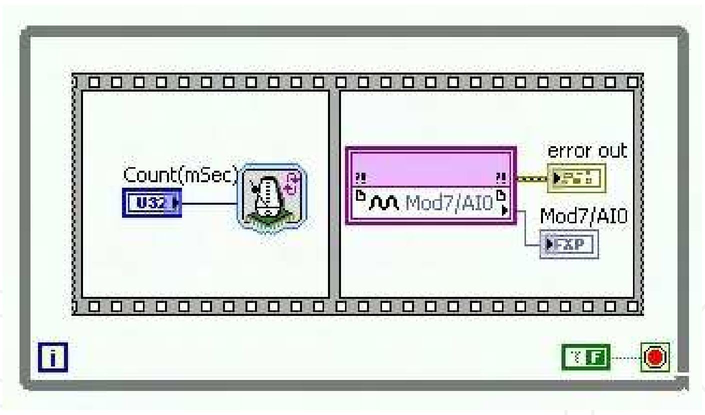


Fig. 49. Block Diagram of the FPGA part.

In Fig. 50. we can see the Block Diagram of the real-time host part. We have the FPGA program loaded in the real-time host (FPGA Target RIO0). We have a while loop with the used variables, the conversion from voltage to temperature in °C with multiplying with 100, a mean of 100 values. After the exit from the While loop we have the closing of the FPGA program loaded in the real-time host and the handling of errors.

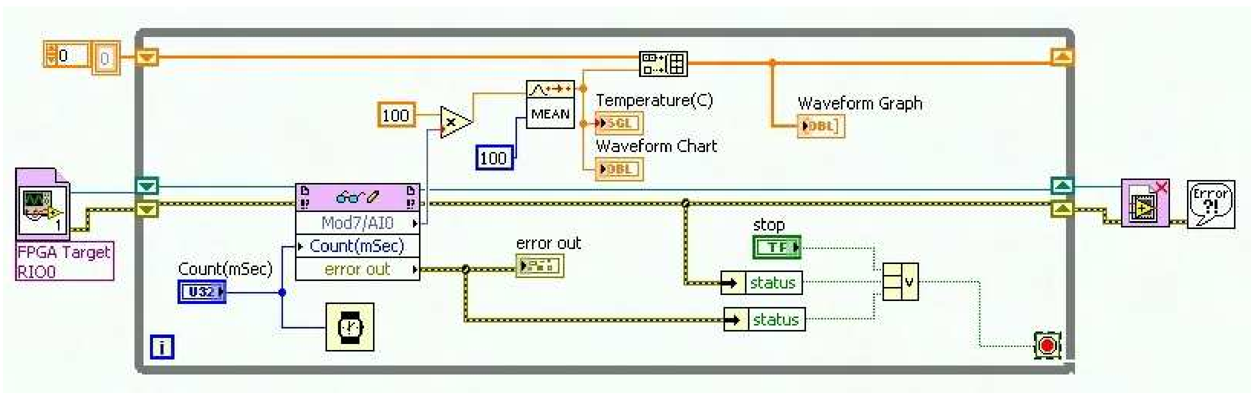


Fig. 50. Block Diagram of the Real-Time Host.

In Fig. 51. We have the Block Diagram of the networked real-time host, which is the copy of the real-time host plus the adding of the global variables to the controls and indicators.

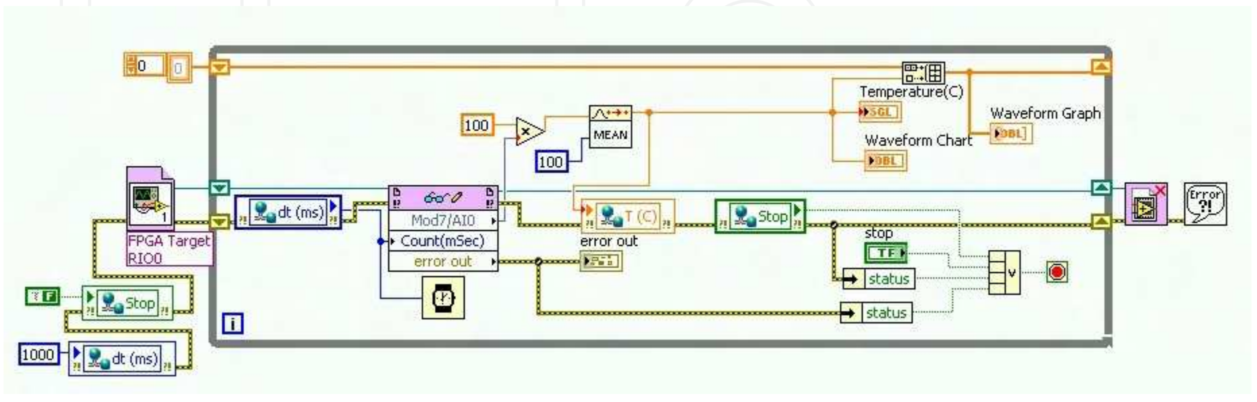


Fig. 51. Block Diagram of the Networked Real-Time Host.

In Fig. 52. we have the Block Diagram of the Windows host, which has the global variables as the mirror image of the networked real-time host. This means where we have control, we will have indicator and vice-versa. We can imagine an invisible line, connection between these global variables. These lines transport the data between the two VIs via Ethernet. The data is sent from the networked real-time host, but there is also feedback from the Widows host.

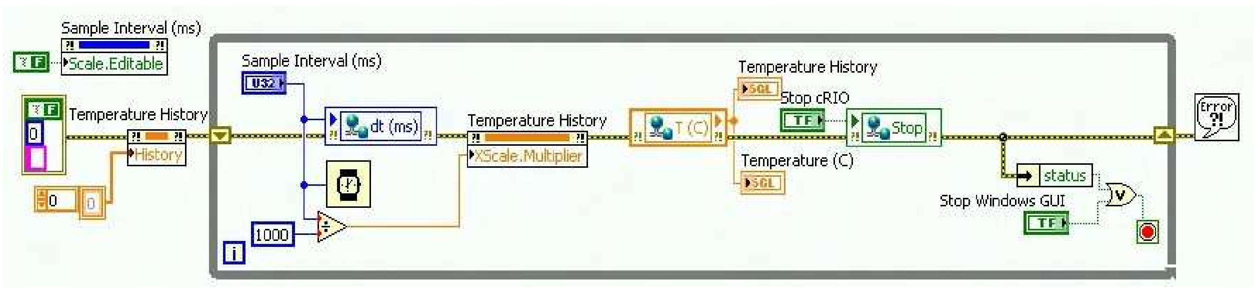


Fig. 52. Block Diagram of the Windows Host.

5. Conclusion

LabVIEW is a very complete programming language. We could say that if assembly is a first level programming language and C is a second level programming language, than LabVIEW should be a third level programming language. LabVIEW is graphical and has a lot of implemented blocks, as they say; you don't have to invent the wheel again. This concept makes programming really fast and could reduce very much an engineer's work.
As we could see in paragraph 2., LabVIEW has a lot of block, but not enough. It hasn't got everything that is needed. In paragraph 2. we presented only 7 programming tricks that we made and we thought that should be important to know. We are sure that are much more tricks in this programming language. We encountered and solved much more programming problems, but we decided not to present those tricks, because, they are not so general. After all LabVIEW is a very complex programming language, but there is place for more.
After the tricks we presented 6 applications made with the PXI industrial system and 2 applications made with CompactRIO. They have similarities; some of them are made with both of the PXI and ComapctRIO platforms. The applications were made for didactical purposes and for publishing them on the LabVIEW Remote Lab. All the applications are published and accessible from the http://plst.etc.upt.ro address with a user name and password.

6. References

Antonovičs U. & Priednieks Ē. (2006). Interactive Learning Tools for Electrical Engineering and Electronics Course, *Electronics and Electrical Engineering*, ISSN 1392-1215 – Kaunas Technologija, No. 7(71), pp. 29–34, 2006.

Auer, M.E. & Gallent, W. (2000) The Remote Electronic Lab as a Part of the Telelearning Concept at the Carinthia Tech Institute, *Proceedings of the ICL2000, Villach/Austria*, 2000.

Garbus, R.U.; Aguirre, I.J.O.; Sanchez, R.C. & Pureco, O.R. (2006). Virtual Remote Lab for Control Practic, *Electronics, Robotics and Automotive Mechanics Conference*, vol. 2, pp. 361-366, 2006.

Gontean, A.; Lie, I. & Szabó, R. (2009 a). LabVIEW powered remote lab, *Design and Technology of Electronics Packages, (SIITME) 2009 15th International Symposium*, pp. 335-340, Gyula, Hungary, September , 2009.

Gontean, A.; Lie, I. & Szabó, R. (2009 b). Using a Low Cost Programmable Power Supply for Automated Testing, *Design and Technology of Electronics Packages, (SIITME) 2009 15th International Symposium*, pp. 341-346, Gyula, Hungary, September , 2009.

Gontean A. & Szabó R. (2009 a). Remote temperature measurement with and without FPGA, *TELFOR 2009 – 17th Telecommunications Forum*, pp. 775-778, Belgrade, Serbia, November, 2009.

Gontean A. & Szabó R. (2009 b). Comparison between PIC and CompactRIO remote motor control, *TELFOR 2009 – 17th Telecommunications Forum*, pp. 747-750, Belgrade, Serbia, November, 2009.

Grober, S.; Vetter, M.; Eckert, B. & Jodl, H.-J. (2007). Experimenting from a distance – remotely controlled laboratory, *Eur. J. Phys.*, vol. 28, pp. 127-141, 2007.

Hercog, D.; Gergic, B. & Matko, V. (2005). Remote Lab for Electric Drives, *Industrial Electronics – ISIE 2005 Proceedings of the IEEE International Symposium*, vol. 4. pp. 1685-1690, 2005.

Khalil, A.; Hasna, M.; Benammar, M.; Chaabane, M. & Ben Amar, C. (2009). Development of a remote lab for electrical engineering program*, Signals, Circuits and Systems (SCS) – 3rd International Conference*, pp. 1-5, 2009.

Mihela Lascu (2007). *Advanced programming techniques in LabVIEW*, Politehnica Publishing House, Timişoara, 2007.

*** National Instruments (2008). *LabVIEW™ Data Acquisition and Signal Conditioning Exercises*, 2008.

*** National Instruments (2008). *Data Acquisition and Signal Conditioning – Course Manual*, 2008.

*** National Instruments (2007). *LabVIEW™ Basics II: Development – Course Manual*, 2007.

*** National Instruments (2007). *LabVIEW™ Basics I: Introduction – Course Manual*, 2007.

*** National Instruments (2008). *CompactRIO™ and LabVIEW™ Development Fundamentals – Course Manual*, 2008.

Paladini, S.; da Silva, J.B.; Alves, G.R.; Fischer, B.R. & da Mota Alves, J.B. (2008). Using Remote Lab Networks to Provide Support to Public Secondary School Education Level, *Computational Science and Engineering Workshops – CSEWORKSHOPS '08 11th IEEE International Conference*, pp. 275-280, 2008.

Szabó, R.; Gontean, A. & Lie, I. (2010). Sound Based Coin Recognition and Clapper, *MENDEL '10 – 16th International Conference on Soft Computing*, pp. 509-516, Brno, Czech Republic, June, 2010.

Szabó, R.; Gontean, A.; Lie, I. & Băbăiţă, M. (2009). Oscilloscope Control with PC, NAUN 2010 – *International Journal of Computers and Communications*, pp. 33-40, Issue 3, Volume 3, September, 2010.

Uran, S.; Hercog, D. & Jezernik; K., (2006). Remote Lab Experiment RC Oscillator for Learning of Control, *International Journal of Online Engineering*, vol. 2, no. 4, pp. 1-8, 2006.