PIG for programmers

Some reasons for establishing PIG in the present form are:

- PIG works well and independently with other devices like the positioning sensor, the encoders, and the driving motors and guides the telescope reliably and accurately to desired positions.
- PIG can works with many users at the same time. Users from different places can receive the reports from PIG anytime. PIG supports the collaboration of different observatories and helps the participants easily to share the information.
- PIG strictly separates the different levels hardware software. The scientists can freely develop their observing programs without thinking about the hardware anymore. Through the network their programs send commands to PIG and "virtually control" the hardware.
- The scientists can work comfortably with their ordinary platforms: Windows, Unix/Linux, ..., with text-based interface like telnet, putty, ..., or with graphical user interfaces.

It is possible to port PIG to other solar observatories by substituting few levels.

The architecture PIG is well modulated with many sub-functions (in LabVIEW the sub-functions are called SubVIs). Each SubVI is (relatively) easy to understand and to be extended. New SubVIs can be added easily if the scientists need new commands. With document, you will learn (nearly) all about the functions of the PIG-software. The concept of the program will be described in detail. After reading this part, you will understand how PIG is built and how the SubVIs work together. You will be able to extend the program by some new SubVIs. Furthermore the following text may serve as a tutorial how to establish a LabVIEW-software-system for controlling complex technical devices.

1. Overview

The main VI of PIG is the VI named "MainControl.vi", all other SubVIs are called directly or indirectly by "MainControl.vi". There are many SubVIs and they can be logically divided in 4 groups:

1. PIG is a guiding program and thus has to work with hardware. So the first group of SubVIs is responsible for accessing the hardware. The SubVIs working with the network as a multi-client server and the client of the axis encoders are also belonging to this group. Below the list of SubVIs of this group are shown:

   Sensor.vi
   Sensor_data.vi
   sensor_von_PIG.vi
   MyFPGACode.vi
   MotorControl.vi
   MultiServer.vi
   messenger.vi

2. As mentioned above, PIG is deeply modulated. All modules should work well with each other. In order to enable this, there must be a nerve centre for all these

modules. The second group builds such a centre. This group contains SubVIs not working directly with the hardware and the commands. They play the role of sharing platforms for other SubVIs. The most important SubVIs in this group are those that contain global variables:

command_list.vi
command.vi
setup.vi

Some SubVIs that don't really work as a sharing platform, but they change the variables in the platforms or work as tools for other SubVIs:

setup_load.vi
automatic_setup_load.vi
blxy.vi
xybl.vi
sunrot.vi

3. The biggest group contains all SubVIs that receive commands from users and execute the corresponding actions. This group can again be divided into two smaller groups based on the property of the commands:

3.1 Group of SubVIs responding for querying or executing commands. With these commands users can e.g. query the position of the telescope or select a set position for the telescope. The SubVIs here are mostly small and send immediate reports to the users:

query_actual_x.vi
query_actual_y.vi
query_actual_intensity.vi
select_x.vi
query_selected_x.vi
select_y.vi
query_selected_y.vi
select_delta.vi
query_delta.vi
select_imin.vi
query_min_intensity.vi
query_status.vi
query_sb.vi
query_pigsi.vi
query_pibsx.vi
query_pigsy.vi
query_pigsi.vi
query_pigsit.vi
select_loop.vi
query_loops.vi
encoder_query.vi
change_encoder_IP.vi
Encoder_connect.vi
encodercommand.vi

gethour_command.vi
setup_load_command.vi

3.2. Group of SubVIs that respond for guiding commands. These SubVIs receive the commands and then change the variables in SubVIs of the second group. These variables then obtain the values needed by the SubVIs of the fourth group for their activities. You will learn more about this later:

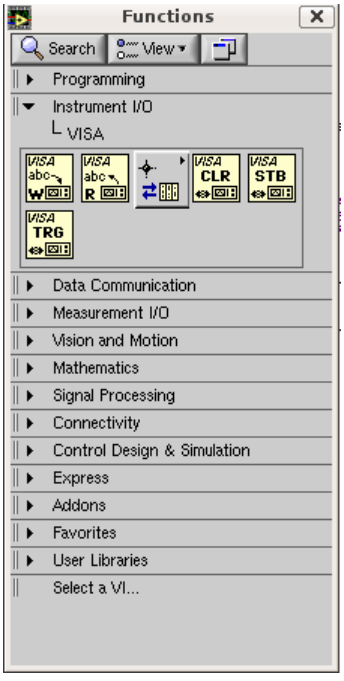
PigGo_commander.vi
PigGoFollow_commander.vi
gotosun_commander.vi
piggohome_commander.vi
pigstartffm_commander.vi
pigstopffm_commander.vi
GuideOff.vi
pig_abort.vi

4. The last group contains guiding executer SubVIs:

Go_to_point.vi
Hold_position.vi
Piggo_executer.vi
PigGoFollow_executer.vi
piggohome_executer.vi
piggosun_executer.vi
pigstopffm_executer.vi
flatfieldmodus_executer.vi

2. Hardware relevant SubVIs

The positioning sensor can be considered as the eye of the guiding system. The SubVI that directly sends queries to the sensor and receives reports from the sensor is "sensor.vi". Other SubVIs use the values from this VI. The functions used to connect with the sensor via the RS232 port are "VISA functions".



VISA functions

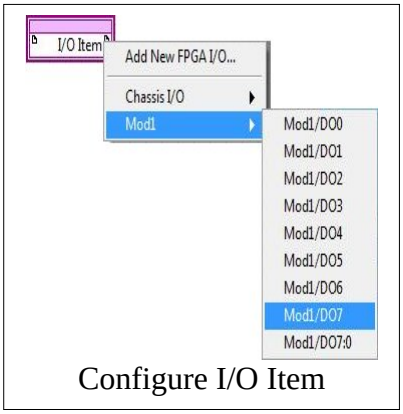
Sensor.vi		
input	init?	True: the sensor will be initiated
	command	Command to be sent to sensor
output	receipt	Report from the sensor
	error?	True if error happens
	vertical	x-position
	horizontal	y-position
	intensity	Intensity of the solar image

Further important hardware components are the two driving motors of the telescope. The motors are controlled by the NI 9472 module. In order to program this module, special functions from Nation Instruments must be installed: the NI LabVIEW FPGA functions. These functions can be found on the CD delivered with the module. If you use the program “Measurement & Automation Explorer” to install software, these functions are already installed.

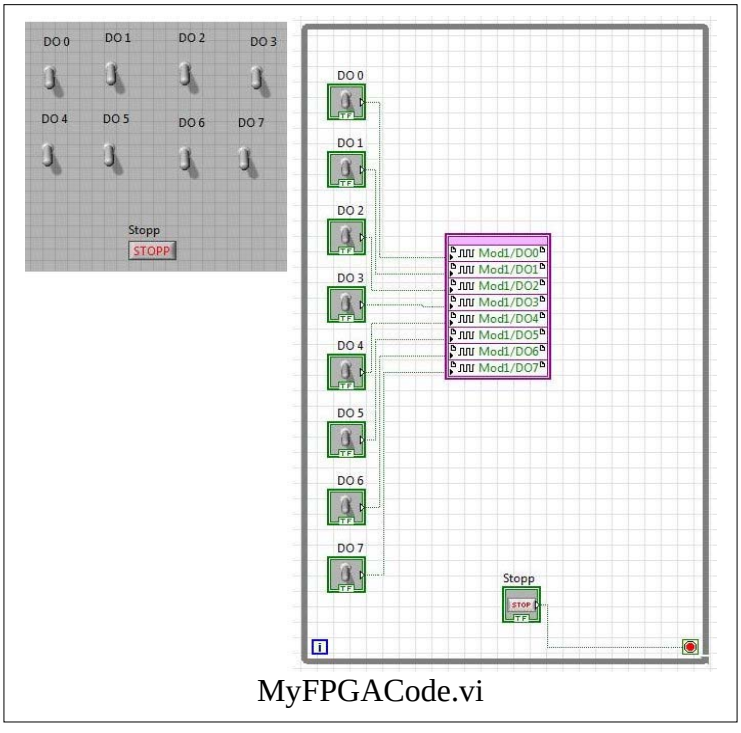
FPGA functions

FPGA is the abbreviation of “field-programmable gate array”. The NI LabVIEW FPGA Module uses LabVIEW embedded technology to extend the LabVIEW graphical development and target FPGAs on NI reconfigurable I/O (RIO) hardware. With the LabVIEW FPGA Module you can create custom measurement and control hardware without low-level hardware languages or board-level design. Furthermore programming with LabVIEW FPGA is as simple as with other LabVIEW functions.

The adjacent picture shows the FPGA functions that are used to control the FPGA module. In this project the FPGA module has only one digital output with 8 output channels. To program each channel the function “FPGA I/O Node” is used (see picture below). Firstly a new VI must be added to the Vi-list of the module (can be seen by the “project explorer”). Add “FPGA I/O Node” in the block diagram and right-click on the symbol. A pop-up menu will be displayed. Now you can choose the corres-ponding IO channel. In this project the VI that directly controls the output channels is “MyFPGACode.vi”. To each channel module belongs a boolean switch.

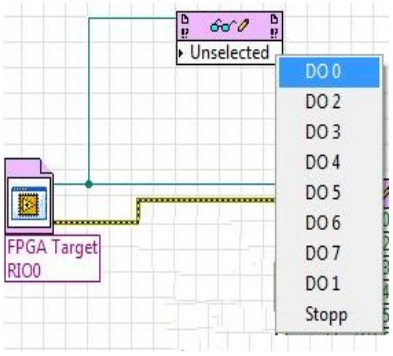


Configure I/O Item



MyFPGACode.vi

A VI that wants to change the status of a channel first calls the “MyFPGACode.vi” with a “FPGA Reference” and subsequent sets the status of channels with a “Read/Write Node”.



FPGA Reference and Read/Write Node

The Reference calls the VI and outputs a “reference number” that will be connected with the “Read/Write Node”. After this the user can easily choose the boolean switch that he wants to set on a new value.

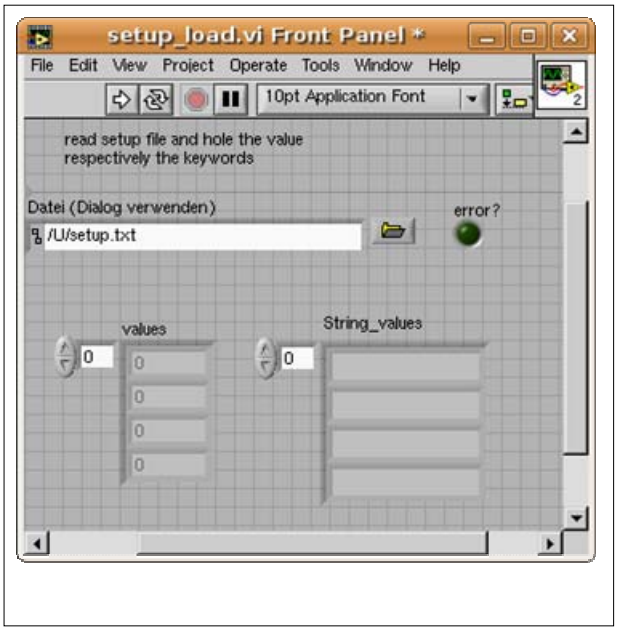
3. Concept of PIG

In order to understand the concept of the guiding system PIG, it is useful to observe the flow of processes. The main VI of PIG is “MainControl.vi”. In fact this VI is a combination of many SubVIs. After being loaded into the memory and being started, the VI “MainControl.vi” loads the setup values from the setup file of the USB-stick (The controller cRIO has an USB-port named “/U”). The SubVI called in order to load the setup values is “automatic_setup_load.vi”. But the real core of the loading setup values is “setup.vi”. The setup values are saved in a text file with the format:

<keyword>$_{space}$<values><CR>

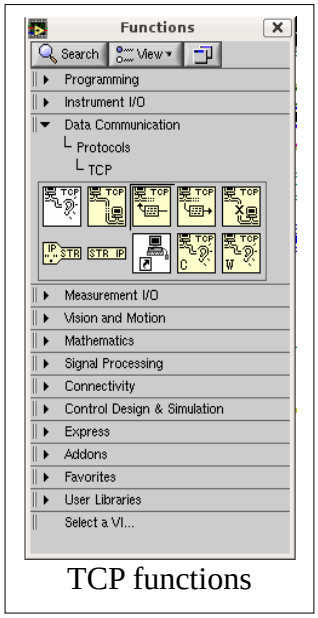
All keywords now will be saved in an array in “setup.vi”. Here you can see a technique that is widely used in PIG to save constant reserved keywords: a global array with default values. An array of type string is firstly created. Then the keywords will be entered as elements of the array. After that all values are defined as default values by right-clicking the array and choose of “Data Operation → Make Current Values Default” from the pop-up menu.

“setup_load.vi” compares the elements of the keyword array with each line in the setup file. When a keyword appears at the beginning of a line, the respective value will be taken and saved in the returning value arrays, an array for numerical values and a string array.

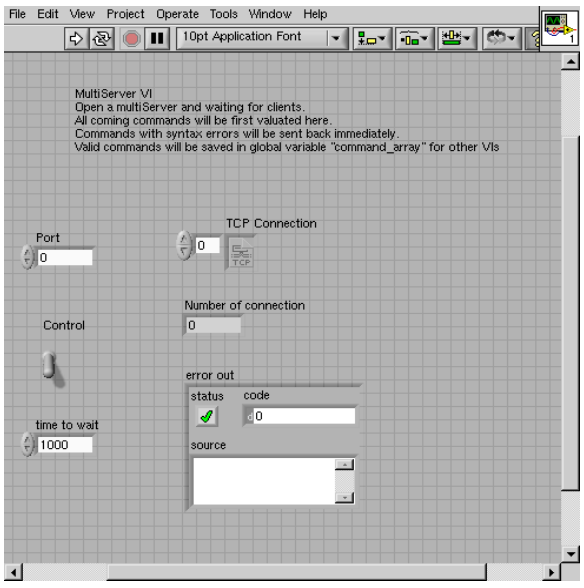


“automatic_setup_load.vi” stores values within the respective variables of in these “returning array”, e.g. speed in north-south direction, delta, latitude, longitude , etc.

Now the FPGA SubVI to control the driving motor will be loaded. All channels of the digital output module are reset to “off”.
PIG is a multi-client server, so the next important VI is the server VI “MultiServer.vi”. In order to enable many users to work parallel with PIG a TCP Listener is created. The used functions are located at the end of the list of TCP functions. Firstly the function “TCP Create Listener” with a symbol of an ear and a letter “C” is used to create a listener. Then in a “while-loop” the function “TCP Wait On Listener” waits for connections with users. Each valid connection will be registered with an ID. All IDs will be stored in an array.
Another “while-loop” is waiting for commands from each user. All users can work parallel. A “for-loop” divides the connection-ID-array in single IDs and waits for command from each ID. The time to be waited is assigned by the control “time to wait”. The default value is 10 milliseconds. That means for each connection, the server waits for a command 10 milliseconds. If there is no byte sent, the function “TCP Wait” returns an “error” and the “for-loop” steps forward to other connection ID. The error code for “TCP overtime” is 56. At the end of the “for-loop” the error code from each connection is checked. If there is an error and the error code is different to 56, it is clear that there really is an error regarding this connection and the respective connection ID will be removed from the ID array. Otherwise the ID stays in the array and is served by the “for-loop” again.



TCP functions



MultiServer.vi

If a user (client) sends a command to PIG the command will be read byte by byte. Subsequently the command will be tested coarse. The server compares the first word of the commands with the elements of the global array “command_list” of “command_list.vi”. If any of the elements meets the command it is considered “valid” and will be stored in “command.vi”. That is another global variable of PIG which is accessed by all other command relevant processes.
In “command.vi” not only the command is stored but also the ID of the respective client. The storage of the termination character of the command guaranties for the platform independence of the system. All clients always receive their specific termination character. The stored connection ID is used later by other processes in order to send an answer to the client.

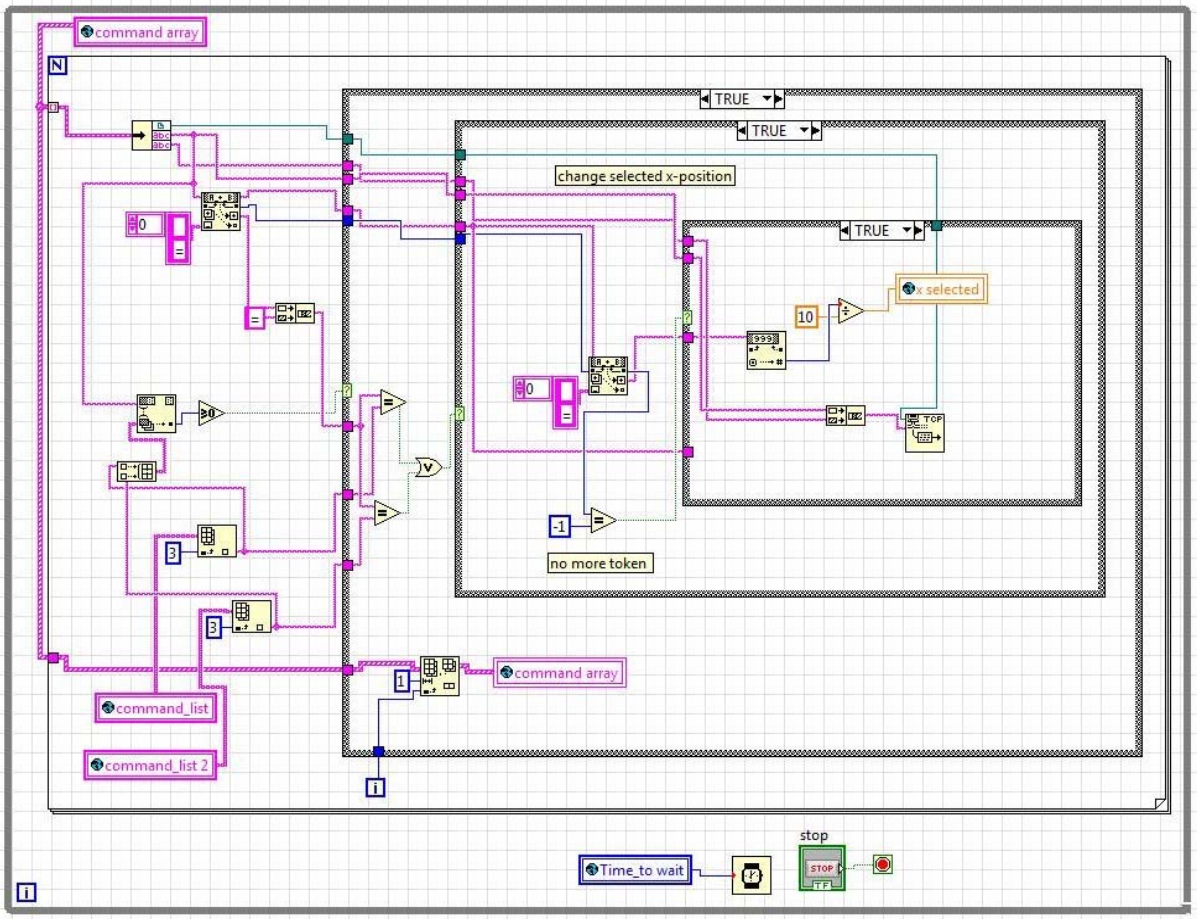
For each command exists at least one VI that is called by “MainControl.vi”. It runs parallel to other VIs. This VI checks the array of commands in order to find the complete command. When the command is found in the array, this VI performs the action caused by the command and then erases the command from the array.

4. How to add new commands

PIG is constructed in a way that a programmer can easily add new commands, which correspond to new tasks. First of all the string of the new command must be added to the “command list” array. In order to assure the old commands to work properly, the programmer must apply some rules:

- The new commands must be added at the end of the array. PIG uses the index of commands in the array to identify received commands. If the new command is added between old commands, all commands following the new one will not work properly any longer.
- After a new command is added, the current value of the “command list” array must be set to standard value. Otherwise the information of the new command would be lost when the module is started again.

Now the new VI is added to the project explorer. This VI will use the index of the new command to detect the command in the “received command array”. An example of such a VI is show below. This example-VI changes the selected x-value, which was entered by the user. The command string has the index “3”. Like all other command related VIs, this VI runs in a permanent while-loop. In this loop the array for received commands “command.vi” will be inspected: Each element of the array (accessed by the for-loop) is compared with the 4^{th} string within the “command list”. If the command is valid and thus found in the array (the case-structure has value “true”), the VI performs the task “save the x-value and send a confirmation back to the client”. The received command will also be erased from “command.vi”. In principle any new VI can always be copied from an existing one. The programmer only has to change the index of the new command and of course to exchange the activities that should be performed by the new VI.



Example of a VI related to a command