SPARC-CS-12/008
October 10, 2012

THE SPARC LOW LEVEL RF CONTROL SYSTEM DESIGN AND OPERATIONAL REFERENCE

Marco Bellaveglia
INFN, Laboratori Nazionali di Frascati, P.O. Box 13, I-00044 Frascati, Italy

Abstract

The SPARC RF low level control and synchronization system has been entirely designed and assembled in Frascati National Laboratories of INFN. This document accurately describes the control system software that handles the various RF devices involved in the cited systems. The first section is dedicated to an overview of the whole SPARC control system, written in LabVIEW^{TM} developing environment. We describe the main architecture, where all the objects with different interfaces and functions are identically inserted. After that we give a precise description of the RFS software class, that handles the RF low level and synchronization system and slightly differs from the other classes. In fact its architecture is more complicated, because some signal acquisition and object 'cross-talking' problems. Also a description of the control room software is given to provide a sort of operational manual for the RF low level control system user. Many upgrade on the LLRF system have been made since the beginning of the compilation of this manuscript both in the hardware and software architecture. Nevertheless this document can be used as reference to understand the basic working principle of the system.

1 Introduction

This document has been realized to precisely describe the functioning of the RF low level control system of the SPARC FEL project at LNF [1], mainly from a software point of view. In particular we want to give both an operational manual for the control room users and a detailed reference for the RF control system developer. In section 2 we report about the general control system architecture of the accelerator, that is completely designed and developed in a LabVIEW^{TM} [2] environment. Then we accurately describe the RF control system, that has been developed with a structure different from the other devices, due to some dedicated tasks it have to accomplish. This matter is discussed in section 3.1.1, that describes the main software classes that have been designed and gives the interesting correspondence between software and real objects. Moreover in section 3.2 we describe the tasks that the front-end CPU accomplish during each program cycle with emphasis on the main RF control system LabVIEW^{TM} VIs that are part of a precise software architecture. Next section 3.3 is dedicated to give a sort of operator manual for the RF actions that one can perform using this system. We give a precise description of each software window that can be opened from the control room consoles and a "troubleshooting" as a walk trough for common problems that often occur. This document can be treated as an extension of [3], where the description of hardware devices and measurement performed using this software are extensively treated.

2 The SPARC control system

2.1 Overview

The first part of the installation (RF gun and emittance-meter) allowed to test the architecture of the control system [4] from the hardware and from the software point of view. Control application for magnetic elements, vacuum equipments, RF cavities and some diagnostics have been developed and debugged on line. In order to improve the machine operations we have included in the system some operation service. An electronic logbook has been used since the first phase of the operation contributing to share the information between all the members of the collaboration. We began to develop an automatic system to monitor the accelerator status periodically or when some variable is changed. This system is based on a PostgreSQL database server. The SPARC Control System is in charge of managing devices (Tab. 1) distributed over an accelerator area. To develop the whole system we have short time and few people. We decided to use commercial technologies as much as possible in order to optimize the development time. A commercial product is characterized by a broad distribution, which means a lot of feedback from the users and,

Table 1: Devices controlled in SPARC

Device	e-meter	SPARC	Interface
Magnet P.S	9	30	serial
Vacuum pump	15	23	Fieldpoint
Vacuumeter	2	6	serial
Modulator	2	2	Ethernet
RF	5	23	serial, Ethernet
Camera	5	12	IEEE1394
Flag	5	12	Serial, CAN
Current Monitor	1	2	Multimeter
Position Monitor	0	12	Ethernet

consequently, deep debugging. Furthermore the wider is the distribution of a product the more reliable is its support from the producer. Another criterion was to privilege 'easy development and maintenance'. We decided to use:

- LabVIEW from National Instrument is used development as environment for all the software;
- Industrial Personal Computer with PCI bus to house the front-end hardware.

2.2 Software

In order to reduce the time of development of the SPARC control system, we decided to use well known software. Labview became the natural choice for the following reasons:

- in the Frascati laboratory the use of National Instrument software is diffused (we can say it is a 'standard');
- Labview is used as development software in the DAΦNE control system [5]. This choice allows us to re-use, when possible, the software;
- Labview is considered reference software by a lot of hardware manufacturers that write interface divers in Labview.

The DAFNE control system is working since 10 years and now is well defined and debugged. This encourages us in using the same architecture and software, when possible, for the SPARC control system. Analyzing the two systems we have found two main differences: first of all the acquisition bus is PCI for SPARC VME for DAΦNE. Furthermore the communication between the system levels is different. For these reasons it is

necessary to rewrite some software. We began the porting of the software starting from the communication mechanisms. In DAΦNE the shipment of the commands happens using a mailbox written on a shared memory. In the SPARC control we do not have a shared memory but a LAN. A server program, running in parallel, receives the commands and makes them available to the acquisition program through a global variable. In the DAΦNE control system a second communication channel is present that allows reading the state and the variations of the single elements under control. Also this communication is through shared memory. In the SPARC case we have realized a second server that bundles the information and sends them upon request to the console. Presently the data are transferred without coding them but we are thinking of using XML coding system [6] in the future. The usage of PCI bus instead of VME, in the front-end CPU, forced to replace the acquisition board drivers. In some cases this was not necessary because the acquisition happens through a secondary fieldbus. This allowed simpler re-use of the DAΦNE software to that element.

2.3 The SPARC network

In a distributed system the interconnection bus between the different CPUs is important to allow maximum performance. First of all we don't want to have any evident bottleneck in the data transfer. Furthermore the bus system has to be reliable and affordable. Today in every personal computer the Ethernet connection is a standard: this means that it can be a robust channel of communication. We use the Gigabit Ethernet to obtain the necessary bandwidth in the data transfer between the different parts of the system. The realization of a switched LAN (figure 1) gives the possibility to use the network also as a fieldbus infrastructure to reduce at maximum the interconnection between the devices and the acquisition system. In table 1 we can see the elements of the acquisition system: some of them can be directly connected to Ethernet some other fieldbus can easily connected to it.

2.4 System architecture

The main operation in an accelerator control system is data taking, display of information, analysis, command execution and expandability. To reach these goals we need to use a well defined system structure. We chose a simple but efficient three levels architecture (see figure 2) that is constituted by: (i) a front-end level (CPUs near the machine), (ii) an intermediate level (that controls the flow of commands and the CPUs status) and (iii) a console level (control room CPUs). The reasons and the main advantages of this choice are listed below:

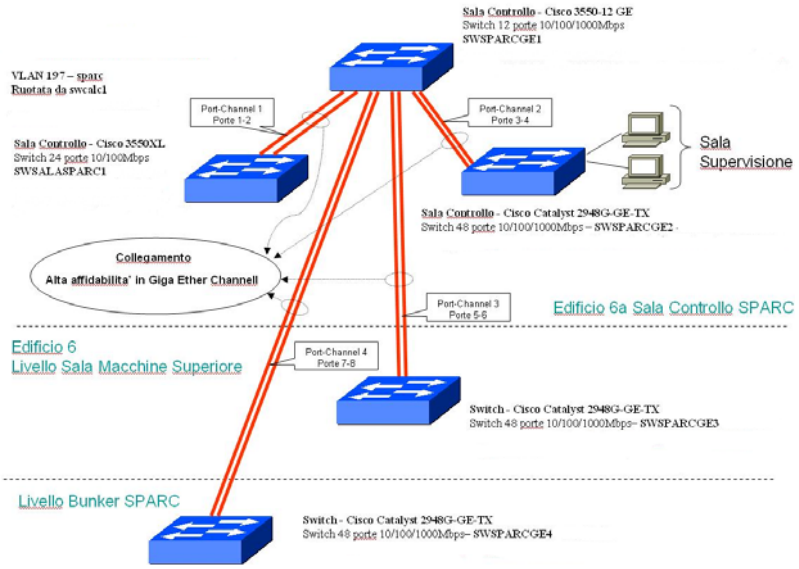


Figure 1: The SPARC LAN architecture

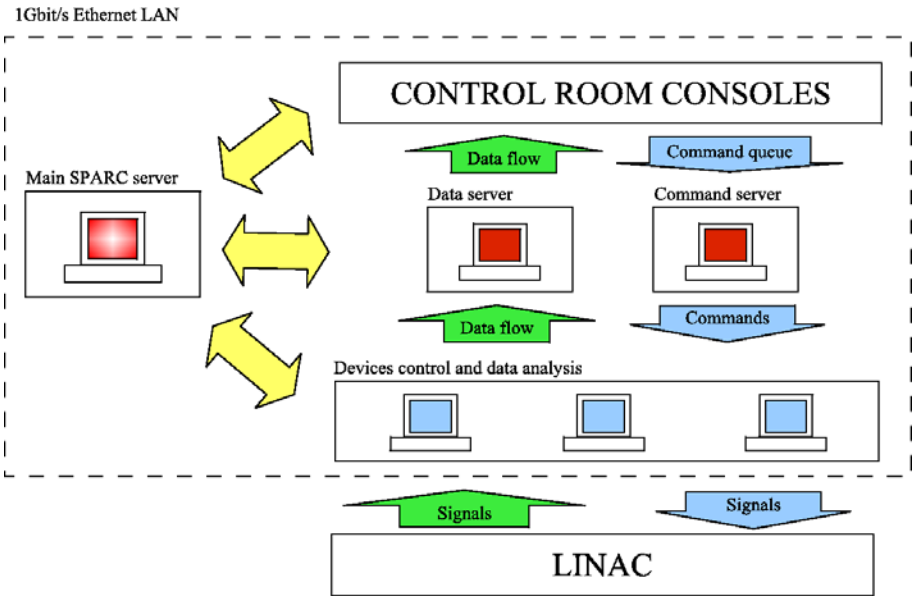


Figure 2: The SPARC control system architecture

- Different devices and interfaces can be handled directly by local programming the CPUs present in the front-end of the machine;
- The console application development is independent from the various devices connected to the system;
- The architecture is transparent to the operators, i.e. an operator only needs to press a button or read an indicator to control a device;
- We need to control devices that cannot be connected with very long wires, so the PC with instrument drivers has to be situated in the machine bunker;
- One needs a flow control of commands and a CPU status check.

2.4.1 Database

Recently we started an upgrade of this part of the system to remove the old text file database and create a new one with PostgreSQL. However the first run of SPARC used the old database architecture, inherit from DaΦne. An important node of the database is a global definition file that identifies the CPUs present in the system with their IP address and communication port, included the data, software and eLogbook servers. It is read from all the three levels at the startup of the system and 'teach' to the CPUs of the system how to communicate each other. Moreover, every CPU placed in the bunker must acquire information about the devices that it has to handle. This information is divided into two types: a static information, that never change after the startup, and a dynamic information, that can change every CPU cycle, or only when an operator send a command. These information are stored in static and dynamic variables and they are initialize at the system startup reading specific formatted text files placed in the server. In this way every device is seen by the driving CPU as a virtual device (called element) identified by two (static and dynamic) variables. There is no unambiguous correspondence between software elements and hardware devices, so it is possible to cumulate in a single element properties from different devices. Identical devices are identified by a software class and every class has specific variable, commands, and drivers. An example is given in figure 3 where the static and dynamic variables related to the SIP class (vacuum pumps) are shown. Some kind of data cannot be stored in these variables because sometimes one need a direct and faster information transfer to have a real-time monitor of the events. This happens for example for beam images taken from cameras. To avoid this problem we created some optional commands that query the desired information without storing it in the variables.

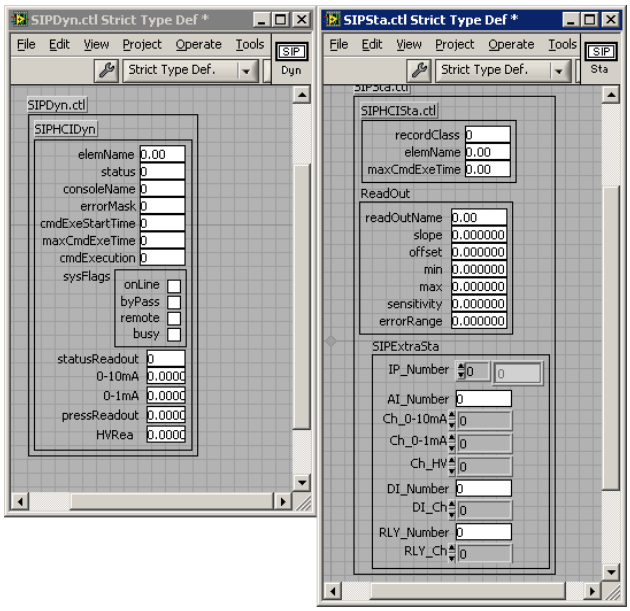


Figure 3: Static and dynamic control types (variables) of the SIP class

2.4.2 The front-end (or 3rd) level

This level is constituted by the front-end CPUs, where the signal are sent and received from the devices under control. Every CPU is positioned close to the controlled device. It has to control a certain number of elements, members of one or more classes. The operative system installed in these machines is Microsoft Windows XP © to minimize the time of device drivers implementation. There is a main LabVIEW^TM Virtual Instrument (VI) running to control the hardware attached to the machine and it is called SPARC13XX (where XX is a number from 01 to 99), plus two other VIs that work as data and command TCP/IP servers. The command server communicate with the intermediate level, while the data server send data (after request) directly to the console level. It is planned to develop a new server for each SPARC13XX that will be the SQL server for long time data logging. Every cycle, the SPARC13XX do the following job:

- it accepts commands sent from a console (through the intermediate level), sending electric signal to the hardware device or changing some software setting in the corresponding element;
- it receive data queries (from console level) and send back the requested data
- it send an alive signal to the intermediate level to monitor the CPU status.

Few main VIs characterize one class (commonly identified by three letters, here YYY) handled by a SPARC13XX:

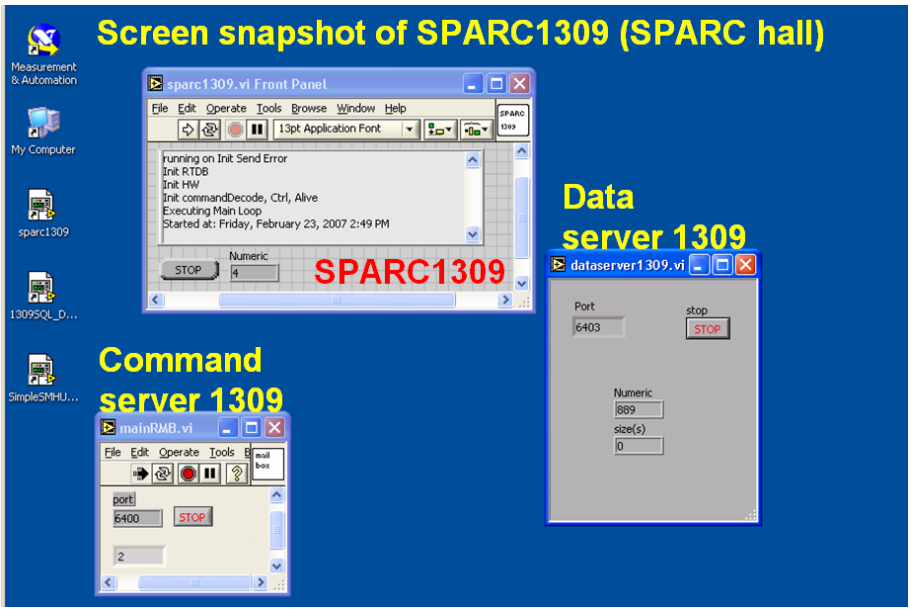


Figure 4: A screen snapshot of the SPARC1309 operating in the SPARC bunker

- loadRTBDYYY: it runs only before the first cycle, loading the elements information from the database and initializing the corresponding variables;
- initHWYYY: it runs only before the first cycle and is the hardware initialization procedure. It set up the devices (if necessary) for the routine operations;
- closeHWYYY: it runs only before closing the SPARC13XX, releasing the device after the routine operations.
- CTRLYYY: it runs every SPARC13XX cycle and commonly read the information that have to be displayed at the SPARC repetition rate in the control room.
- CMDYYY: it runs only when a command request arrives from a console (through the intermediate level) and it perform a decoding and it execute the command sending to the hardware the correct signal.

2.4.3 The intermediate (or 2nd) level

The intermediate level is constituted by a CPU where a VI called Caron is running. Its task is to receive the commands from the console, check if the syntax is correct and send them to the correct SPARC13XX. It is important to note that the command syntax, thanks to the global database, take into account only the element name and not the number of its SPARC13XX. This task is accomplished using a command server similar to the one

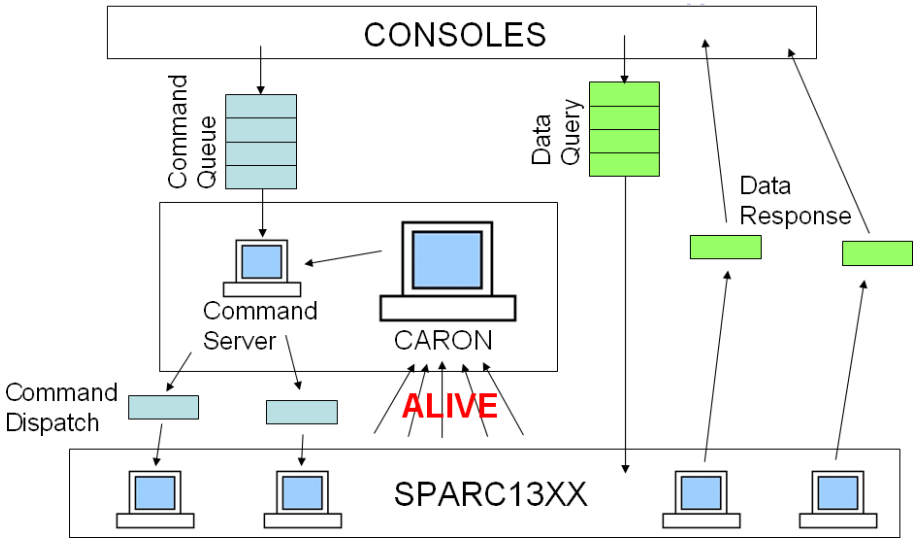


Figure 5: Intermediate level function explanation

of the front-end level. The difference is that it has TCP communication ports opened toward both the consoles and the SPARC13XX servers. The other task accomplished in the intermediate level is to collect the 'alive' signals coming from the front-end CPUs and to determine whether the SPARC13XXs are working or not. The status of every SPARC13XX is displayed in the control room and it can be: (i) Alive (red smile), if the CPU is working properly, Dead (blue smile), if the CPU is not responding and Berr (green smile) if there is an error in the communication. Figure 5 well explain the role of the intermediate level inside the control system architecture.

2.4.4 The console (or 1st) level

The consoles are identical desktop PC with fedora Linux as operative system. They are able to display information taken from each SPARC13XX and the way to request data from the front-end level is identical for all the software classes: one can do a query with the keywords 'STA' and 'DYN' respectively for the static and the dynamic variable. There are some exceptions mentioned before for data directly sent to the console level without store it in the main variables. Commonly they are big array of data (waveforms, images, etc.) and special query keywords has been created for them. The final operator that normally controls the machine has to know only few button combinations to send commands to the devices and to open software windows that automatically display the desired data. The system architecture is thus completely transparent to the final users. A typical screen snapshot of a SPARC control room console during operation is depicted in figure 6.

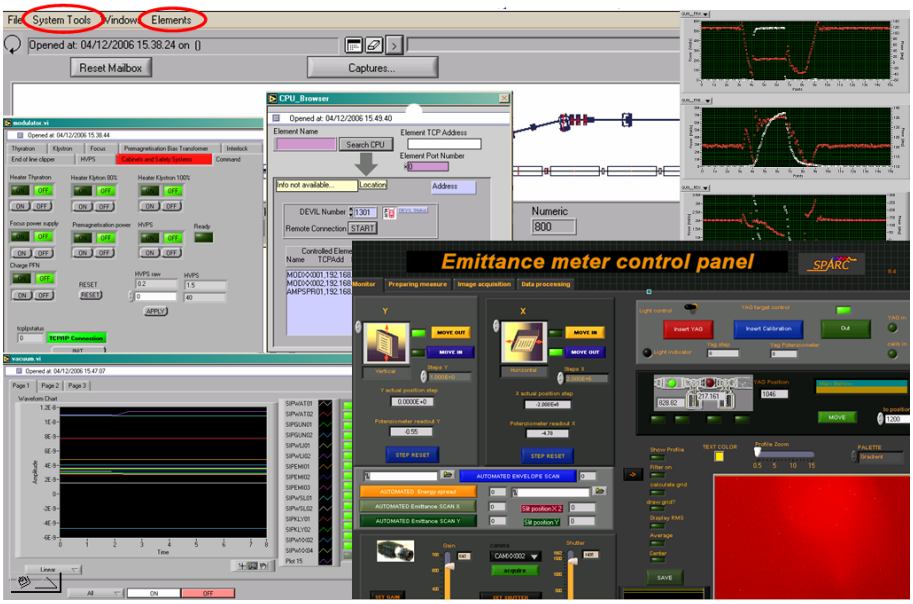


Figure 6: Screen snapshot of a console during SPARC operation

3 The low level RF control system

The RF low level control system is intended as the collection of software programs that controls the devices dedicated to manipulate and distribute the RF main clock signal to the accelerator power stations and to the signal monitor equipment. It also controls the signal monitor itself analyzing and acquiring the signals coming from the power RF devices present along the machine. Figure 3 is a rough description of the hardware layout including the devices under control and the signals acquired.

3.1 Virtual world and real world

For virtual world is intended the collection of classes and objects that represent the the real world in a convenient way to be used in the software control system. In particular software and real object are not in a unambiguous relation in the sense that they rarely coincide. Here we want to give a description of the relations between the two different worlds, reporting how the classes and objects have been designed.

3.1.1 RFS: the main class

In the SPARC control system a single object (as the GUN chiller) or a collection of objects that work together to accomplish a given task is represented by a single software class, as described in the section 2 for the case of the SIP class. Of course the RF low level system

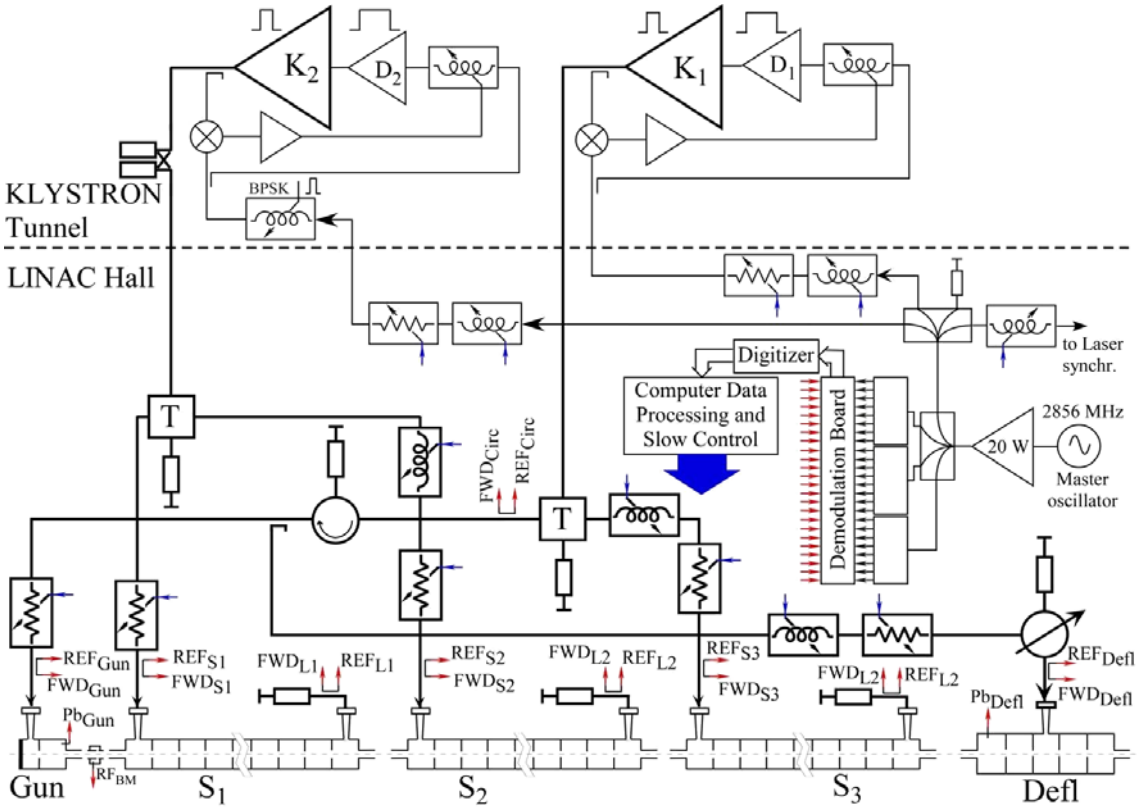


Figure 7: Layout of the SPARC RF system

is included in the latter case but it was included in the control system in a custom way. In fact, as you can read below, we designed this software class (called RFS) to satisfy a precise requirement: the signals coming from the RF accelerator devices like waveguides or accelerating sections, have to be demodulated and acquired at the same time, i.e. when the main 10Hz trigger arrives to the signal digitizers front end. So the control system have to send a command to arm all the sampling board at the same time and before the trigger occurrence. This is not possible with the standard SPARC class architecture because the elements (i.e. the objects of a class) are treated one after another and this mean that some sampling board (treated as an element) would have lost a trigger event. To overcome this problem we realized some software subclasses: this way the software objects are distinct, but a single cycle of the front-end CPU can treat them at the same time. In figure 3.1.1 we report the dynamic (DYN) and static (STA) control types of the RFS class, i.e. software templates to be filled out to create an element of the RFS class with its dynamic and static variables. As can be seen, not only simple fields, like the HCI, exist in the RFS class. Also array fields are shown and they represent the lists of the objects members of an RFS subclass. They are: DEM, SHS, ATS and KLP and they are described in the next section.

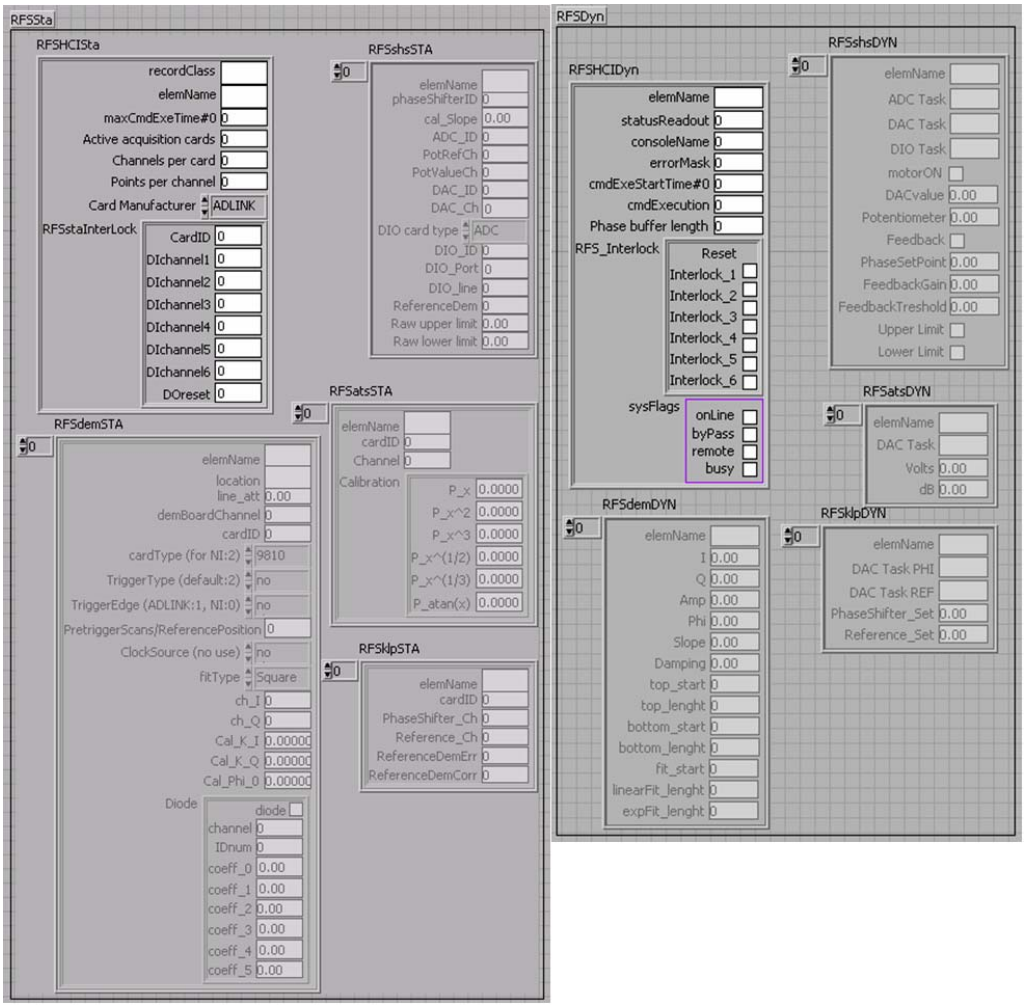


Figure 8: The static and dynamic LabVIEW^{TM}control types for the RFS class

Before continuing with a description of the RFSHCIsta and RFSHCIdyn fields, we give table 3.1.1 that describe the representation of data and their abbreviation, as used in the LabVIEW^{TM} development system and in the SPARC database. Here we describe the fields of the HCI static control type:

- record class (string): is the name of the class (in this case RFS);
- element name (string): the name of the element (for example RFSXX001) and it is represented by a string;
- maxCmdExeTime (U32): maximum command execution time. Is the time within the 3rd level CPU has to finalize a command sent by the consoles;
- Active acquisition cards (I32): is the number of sampling cards actually installed

Table 2: Data representations with abbreviations

LabVIEW	SPARC	Explanation
Boolean	TF	Boolean number
Un	DUn or HUn	Unsigned decimal or hexadecimal n bit integer
In	DIn or HIn	Signed decimal or hexadecimal n bit integer
SGL	not used	Single precision (32 bit) floating point number
DBL	DBL	Double precision (64 bit) floating point number
String	abc	Array of characters (string)

and used in the 3rd level CPU;

- Channels per card (I32): is the number of channel used in each sampling card;
- Points per channel(I32): is the number of points per channel that the cards acquire after each trigger.
  These last three fields are used to create a three dimensional array with proper sizes, where the waveforms data are stored. This data array is updated every CPU cycle in the RFSCTRL and is sent to the 1st level when requested. More details are presented in the continuation of the document;
- Phase buffer length (I32): this number is used only when the low level software feedback is ON. It permits to choose the average factor in the calculation of the actual phase of the e.m. field inside a single RF device(see [3] for details about feedback loop performances and section 3.2.4 for a more accurate description of the software);
- Card manufacturer (U16): It is an Enum LabVIEW^{TM} type and can assume two values: 0, if the cards used are made by ADLINK; 1, if the cards are by National Instruments (NI). This field is the selector of a case structure in the RFSCTRL.
- RFSstaInterlock (all I16): The RFSstaInterlock cluster has been designed to drive an interlock card that is installed but not yet used.

Next item to be described is the RFSHCIdyn. This item involves the fields that can be updated during the operation and that refer to the whole RFS class and not to the single subclasses. Its fields are described below:

- elemName (string): the name of the element (for example RFSXX001);
- statusReadout (I32): represents the status of the element;

- consoleName (I32): this field represent the number of the console that is remote controlling the element. In the SPARC control system it is not used.
- errorMask (I32): this field contain information about the type and the location of an error (if occured).
- cmdExeStartTime (I32): this is not used in the RFS class;
- cmdExecution (I32): this is not used in the RFS class;
- RFS_Interlock (all Boolean): this cluster has been designed to drive an interlock card that is installed but not yet used;
- sysFlags (all Boolean): every class has this cluster field. It gives to the system a communication about the status of any virtual object. In the SPARC control system this cluster is rarely updated. Here is reported a short description of the cluster boolean items:

  onLine: the element is fully operational if tis field is TRUE (the default state);

  byPass: the element is not controlled in the 3rd level CPU cycle if this field is TRUE;

  remote: if TRUE (as default) it is possible to control to the element from remote consoles;

  busy: if TRUE the element can not execute a new command.

3.1.2 Subclasses

The RFS subclasses are arrays of clusters placed inside the main RFS static and dynamic control type. Each element of the array represents an element of a subclass. Since this architecture is an extension of the main control system architecture, the subclasses are not recognized at the global level, that only can address commands and request information concerning the main RFS class. So a more complex command syntax and dynamic data update has been designed for the RFS environment, respect to the standard classes. These differences are extensively treated in the next 3.2 and 3.3 sections. Here we present the static and dynamic field description of the 4 RFS subclasses.

DEM: DEMoulator This subclass element represents more than a single real device. In fact, as shown below, it refers to: electronic S-band mixers, RF peak detectors, RF power devices and cables and sampling boards. A detailed description of a RFSdemSTA element is here reported:

- elemName (string): the name of the subelement;
- location (string): the location where a probe is inserted to acquire and analyze a signal;
- line_att (DBL): line attenuation. It is the sum of the calibrated RF devices (directional couplers, cables, fixed attenuators) from the signal location to the hardware front-end;
- demBoardChannel (I16): it is the channel used in the demodulation board (see [3]) to process the incoming signal;
- cardID (I16): it is the ID number of the sampling card used to acquire the demodulated signal;
- cardType (U16): this field is used to determine which kind of digitizer is used to acquire a signal. For NI DAQ cards PXI5105 (60Msamples/s, 12bit) it simply assume the value of 2, for ADLINK cards it can be 0 if the digitizer is the PCI9810 (20Msamples/s, 10bit), 1 if the digitizer is the PCI9812 (20Msamples/s, 12bit) and 2 is the digitizer is the PCI9820 (60Msamples/s, 14bit);
- Trigger Type (U16): it is normally set to the default value (2, external trigger) and one can change it if needed. In this case one has to take care during the field modification, because the Enum strings are relative only to the ADLINK cards. One can refer to the acquisition VIs in the RFSCTRL (see section 3.2.4) to acquire the needed informations about NI cards;
- Trigger Edge (U16): also in this case one must avoid errors due to the Enum strings that refers only at the ADLINK case. Nevertheless this field is normally set to the default (1 for ADLINK and 0 for NI), and it don't need to be modified;
- PretriggerScans/ReferencePosition (U32): this field has a double mean. The Pretrigger scans refer to the ADLINK cards and represent the number of points shown before the trigger occurs. Reference position is relative to the NI cards and it is the position of the signal acquired respect to the trigger event and it is expressed in percentage (50 means that the same quantity of data are acquired before and after the trigger event, 1 is the default and it means that all the data are acquired after the trigger event);
- ClockSource (U16): it is no more utilized because the sampling clock is permanently set to internal both for NI and ADLINK cards;

- fitType (U16): this item refers to the signal analysis after the acquisition and permit to choose if an exponential fit (needed for decaying field signal, for example) or a simple average (needed for square pulses, for example) is performed;
- ch_I (I16): the sampling card channel used to acquire the part of the demodulated signal in phase (I) with the original one;
- ch_Q (I16): the sampling card channel used to acquire the part of the demodulated signal in quadrature (Q) with the original one;
- Cal_K_I (DBL): it is a calibration coefficient relative to the RF mixer used to demodulate the signal and it is used in the algorithm of phase and amplitude calculation to overcome systematic errors;
- Cal_K_Q (DBL): same as the previous;
- Cal_Phi_0 (DBL): same as the previous;
- Diode (cluster): this cluster is used if we acquire the signal coming from a peak detector that substitute the RF mixer, but only measure the amplitude of the signal. The contents of the cluster is described below:

  diode (Boolean): it is TRUE if a peak detector is used to process the signal;

  channel (I16): it is the number of the channel of the sampling card used by this subelement;

  IDnum (I16): is the ID number of the diode;

  coeff_0 ÷ **coeff_5** (DBL): these are calibration coefficient and they are relative to a polynomial used to fit the calibration data of the diode. They are used to obtain the correct amplitude value starting from the measured voltage.

Now we describe the RFSdemDYN that represent the part of the DEM subelement. It can be updated during the operations automatically (see section 3.2.4) or after a command execution (see section 3.2.5)

- elemName (string): the name of the subelement;
- I (DBL): it is a number representing the part of the demodulated signal in phase with the original one and it is calculated during each 3rd level CPU cycle;
- Q (DBL): it is a number rep resenting the part of the demodulated signal in quadrature with the original one and it is calculated during each 3rd level CPU cycle;

- Amp (DBL): it is a number representing the amplitude of the signal calculated from the I and Q values during each CPU cycle;
- Phi (DBL): it is a number representing the phase of the signal calculated from the I and Q values during each CPU cycle;
- Slope (DBL): if a strait line is used to fit the RF pulse phase behavior, this is the the slope of that line;
- Damping (DBL): if an exponential curve is used to fit the amplitude of an RF pulse, this represents the damping factor that appear in the exponent;
- top_start, top_length (I16): modifying these items, one is able to choose the part of the signal where an average is performed to calculate a number representing the flat top of a square RF pulse. They are used in combination with bottom_start and bottom_length to calculate Amp and Phi. These fields are integers because they express the samples number (see section 3.2.4);
- bottom_start, bottom_length (I16): modifying these items, one is able to choose the part of the signal where an average is performed to calculate a number representing the area where the RF pulse is not present. These fields are integers because they express the samples number;
- fit_start, linearFit_length, expFit_length (I16): they are used in the same way as top_start and top_length, but they concern the case of an exponential fit performed on the RF demodulated pulse. They are used in combination with bottom_start and bottom_length to calculate Slope and Damping (see section 3.2.4);

SHS: phase SHifter at Signal level The SHS subclass represents the signal phase shifters placed in the SPARC hall and with the features of accepting incoming commands to move forward or reverse and performing an automatic phase locking movement when the software feedback system is ON (see [3] and section 3.2.4). The RFSshsSTA element controltype is described below:

- elemName (string): name of the SHS subelement;
- phaseShifterID (U32): the ID number of the phase shifter (visible in the board inside the SPARC hall rack);
- cal_Slope (DBL): the calibration coefficient (potentiometer reading VS phase shift);

- ADC_ID (U32): the ID number of the ADC reading the motor potentiometer;
- PotRefCh (U32): the channel of the ADC used to read the fixed voltage (about 10V) applied to the potentiometer;
- PotValueCh (U32): the channel of the ADC used to read the voltage at the potentiometer central pin. Togheter with the previous field it is used to calculate the potentiometer changing voltage in a way not sensitive to the main voltage fluctuations (see section 3.2.4);
- DAC_ID (U32): the ID number of the DAC controlling the DC motor;
- DAC_Ch (U32): the channel of the DAC used to control the DC motor;
- DIO card type (U16): this field has to be set properly depending on the type of card (ADC or DAC) where the digital input and output are located. It is only used in the case of NI cards;
- DIO_ID (U32): the ID number of the digital input and output card used to switch ON or OFF the phase shifter DC motor(acting on PWN driver electronic card inputs);
- Dio_Port (U32): the port of the DIO used to switch ON or OFF the DC motor;
- Dio_line (U32): the line of the DIO port used to turn ON or OFF the DC motor;
- Gain (DBL): the gain of the software phase feedback loop (see section 3.2.4);
- ReferenceDem (U32): the number of the DEM subelement where the phase is read to calculate the error signal in the feedback loop. The reference DEM has to be located on the same CPU of the SHS;
- Raw upper limit(DBL): is the software upper limit switch implemented only to give an alarm to the operator in the control room and not to stop the motor. It is expressed in volts and it is the maximum value that potentiometer can assume minus the 1% of the total phase excursion (that is about 520°);
- Raw lower limit(DBL): is the software lower limit switch implemented only to give an alarm to the operator in the control room and not to stop the motor. It is expressed in volts and it is the minimum value that the potentiometer can assume plus its 1% of the total phase excursion (that is about 520°);

Here the RFSshsDYN element control type is presented:

- elemName (string): it is the DEM subelement name;
- ADC task (string): it is used only with NI cards and it is the task opened at the first 3rd level CPU cycle to make available the ADC inputs;
- DAC task (string): it is used only with NI cards and it is the task opened at the first 3rd level CPU cycle to make available the DAC outputs;
- DAC task (string): it is used only with NI cards and it is the task opened at the first 3rd level CPU cycle to make available the DIO;
- motorON (Boolean): it is TRUE if the motor is switched ON and ready to move;
- DACvalue (DBL): the actual set of the PWM motor driver control voltage;
- Potentiometer (DBL): the actual absolute position of the motor expressed in degrees of RF phase;
- LockedPhase (DBL): the phase to follow in the software feedback loop;
- Feedback (Boolean): it is TRUE if the software phase feedback is ON;
- Upper Limit (Boolean): it is TRUE if the software upper limit is reached;
- Lower Limit (Boolean): it is TRUE if the software lower limit is reached.

ATS: ATtenuator at Signal level The signal level attenuators are represented in the control system by the subclass ATS. They are electronic voltage controlled attenuators and change their state when a command is executed, when an error occurs or when the 3rd level CPU is turned OFF. The Static control type RFSatsSTA is described below:

- elemName (string): the ATS subelement name;
- cardID (U32): the ID number of the DAC that controls the attenuator;
- Channel (U32): the channel of the DAC that controls the attenuator;
- Calibration (cluster of DBL): here are reported the calibration coefficient that are used to convert the dB setting wanted by the operator into a proper voltage sent to the device. The algorithm is more complex than a simple polynomial due to the strong not linearity of the attenuators characteristic: $V = P_0 dB + P_1 dB^2 + P_2 dB^3 + P_3 dB^{1/2} + P_4 dB^{1/3} + P_5 \arctan dB$;

The dynamic control type RFSatsDYN has the fields reported below:

- elemName (string): the ATS subelement name;
- DAC task (string): it is used only with NI cards and it is the task opened at the first 3rd level CPU cycle to make available the DAC outputs;
- Volts (DBL): the actual volts set at the DAC output;
- dB (DBL): the actual set attenuation.

KLP: Klystron phase feedback LooP The klystron fast electronic phase locking loop (see [3] for details) need some remote knob to be properly close the loop, so it is included in the control system inside the subclass KLP. The static control type is here described:

- elemName(string): the KLP subelement name;
- card ID (U32): the ID number of the DAC card;
- PhaseShifter_Ch (U32): the DAC channel used to control an electronic phase shifter that close the loop;
- Reference_Ch (U32): the DAC channel used to control the bias voltage of the amplification stages;
- ReferenceDemErr (U32): the number of the DEM subelement that acquire the error signal of the feedback loop;
- ReferenceDem Corr (U32): the number of the DEM subelement that acquire the correction signal of the feedback loop.

The description of the RFSklpDYN control type is reported below:

- elemName (string): the name of the KLP subelement;
- Dac Task PHI, Dac Task REF (string): it is used only with NI cards and it is the task opened at the first 3rd level CPU cycle to make available the DAC outputs;
- PhaseShifter_Set (DBL): the voltage set at the DAC output controlling the phase shifter;
- Reference_Set (DBL): the voltage set at the DAC output controlling the bias voltage.

3.1.3 RFS data types

In the RFS class, there are 3 types of data that can be requested from a console: they are STA, DYN, WAV. The request is made in the same way than the other classes sending to the data server the correct command. The STA and DYN data types simply represent the RFS main class static and dynamic control types, respectively. The WAV data type represents the three dimensional array that each RFS 3rd level CPU updates every cycle and where the acquired raw waveform data from all the active sampling card are stored.

3.2 The 3rd level VIs: a manual for the developer

In the SPARC control system the 3rd level is the real interface with instruments and devices of various types. The 3rd level CPUs control the objects as they are defined in the control system and normally read informations every cycle about the device status and execute commands that change the status. Only often these CPUs are used to perform a first information processing executing simple algorithms on the acquired data during each cycle.

3.2.1 Overview and actual hardware arrangement

Actually, the SPARC photo-injector is working in its final configuration and all the hardware, included the RF system one, is installed and functioning. In particular, the RFS class is using two 3rd level CPUs that are named SPARC1306 and SPARC1307 that respectively controls the RFSXX002 and RFSXX001 elements. The SPARC1306 is installed in the accelerator hall and it has to accomplish the following tasks: (i) signal phase shifter control; (ii) implementation of the slow phase feedback; (iii) acquisition of the demodulated signals from the linac RF devices using the sampling cards NI PXI 5105; (iv) a first data analysis to extrapolate amplitude and phase of the acquired signals. So, the subclasses used in the SPARC1306 are SHS and DEM. The second CPU, the SPARC1307, is installed in the RF power station hall and it has to finalize the following tasks: (i) signal attenuator control; (ii) acquisition of the demodulated signals coming from the waveguide directional coupler just after the klystrons output and from the fast phase feedback loop (the sampling cards are in this case ADLINK 9812); (iii) klystron loop working point remote control. The subclasses involved in the SPARC1307 tasks are DEM, ATS and KLP.

3.2.2 *RFSLoadRTDB, GRFSsta and GRFSdyn: loading the database and handling global variables*

We want to spend here few words about the database loading and the static and dynamic global variables updating during the operations. The main working principles for the RFS class are the same as the other classes, except for the handling of the subclass elements. Figure 3.2.2(a) shows the core of the GRFSdyn that is the VI responsible of the dynamic and static RFS variables. It is used in the RFSloadRTDB to create the STA and DYN global variables transferring the information out from the database (before the SPARC13XX main loop) and it is also used to read and update the RFS class variables (during the SPARC13XX main loop). The main difference from the other control system

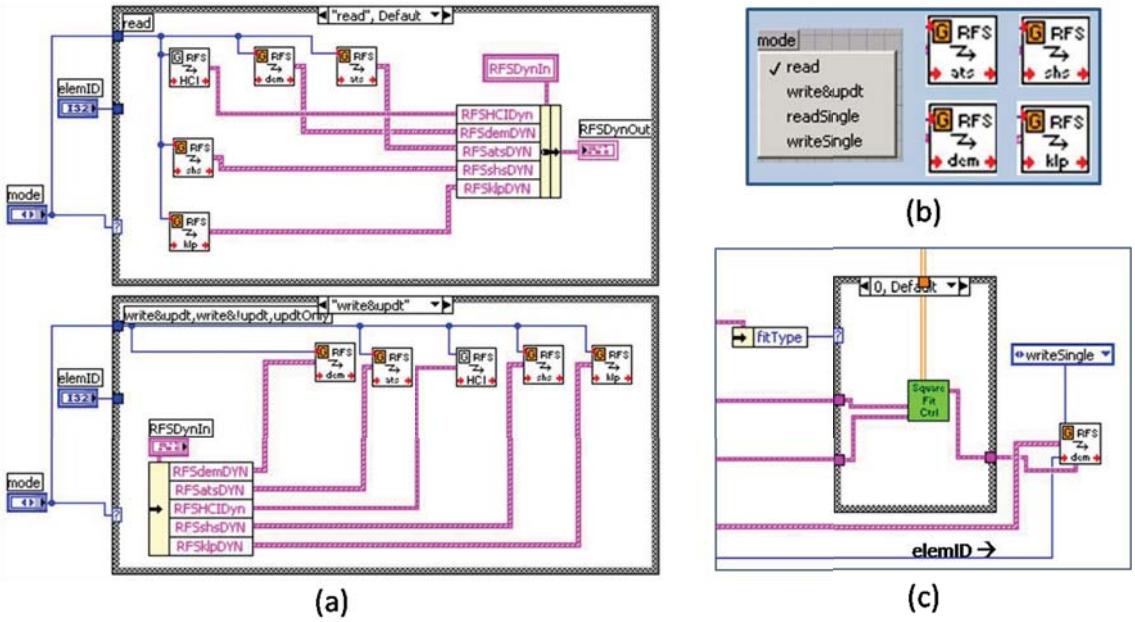


Figure 9: Global RFS static and dynamic variables handling: (a) the GRFSdyn VI block diagram; (b) the modes designed for the GRFS subclasses VIs; (c) an example of utilization of the GRFSdemDYN VI

classes is that the element ID number is not linked to the input of the GRFSxxxSTA or DYN inside the loadRTDB VI, but the default value of 0 is used because only a single RFS class element per CPU is supported. Nevertheless the elemID field is maintained because it is used in the subclasses global dynamic variables handlers GRFSdem, GRFSshs, GRFSats and GRFSklp. In particular new modes of utilization for these VIs has been designed and they are readSingle and writeSingle, in addition to the normal modes read and write&updt (see figure 3.2.2(b)). This new modes are used to have a smart and simple update procedure of the sub-elements during the machine operations. An example

of this usage is shown in figure 3.2.2(c), where the GRFSdemDYN is called in the CTRL to update a subelement of the DEM subclass (more details are reported in section 3.2.4). It is important to point out that this method is very fast, but it needs caution to be used. In fact the element ID number refers to a subelement that is identified by its position in the subclass element array, so when a developer writes down the database, he must pay attention to write the subelements in strict increasing number order (i.e. DEMXX001, DEMXX002, ., DEMXXn, DEMXX(n+1), .). This happen because the GRFSxxx VIs are called inside FOR loops and because a search inside an array to find the correct subelement number is nowhere implemented (too slow).

3.2.3 RFSinitHW: Hardware initialization

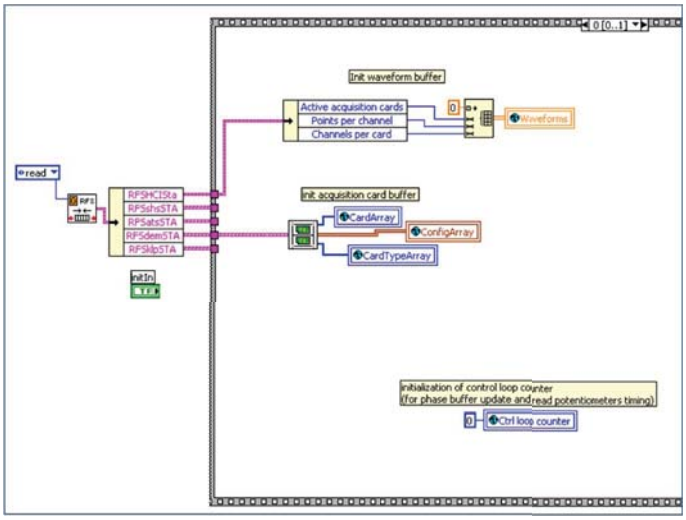


Figure 10: RFSinitHW: global variables initialization

The hardware initialization happens in the RFSinitHW VI that is called inside the SPARC13XX before the main loop execution. It has to accomplish some tasks, that are here described. Figure 3.2.3 shows the first frame of the sequence present in the RFSinitHW VI. This frame initialize some software variables that are used in the main loop:

- the waveform buffer: a three dimensional array that contains all the data coming from the acquisition cards. The sizes of the array are set by reading the static global variable and they are equal to: (i) the number active cards, (ii) the number of point per channel, (iii) the number of channels per card. This way, all the acquired waveforms have a place ready to store them during the operation. The global variable *Waveforms* is updated every 3rd level CPU cycle (see section 3.2.4);

- the global variables concerning the sampling cards configuration. This variables are array that contains information about the sampling card ID number, configuration and type. These items are necessary because one needs to have a sorted list of the cards and their configurations, stored in the DEM subelements without a precise order.

- the Ctrl loop counter. Simply initialize a counter of the SPARC13XX loop cycles. It is needed because the structure of a SPARC13XX VI must be identical for every 3rd level CPU. So also the counting of the executed cycle has to be included in the class specific VIs that we describe in this section.

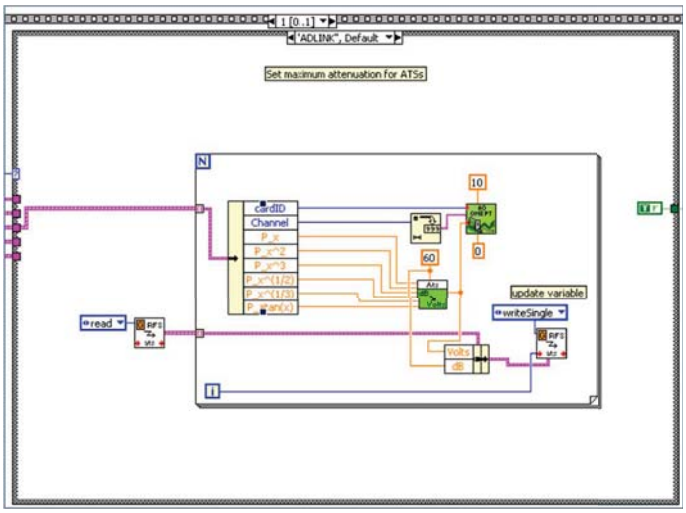


Figure 11: RFSinitHW: initialization of the ADLINK hardware to the default set point

The hardware initialization is composed by two parts. The first one is the initialization of the ADLINK cards. These devices don't need initialization since it happens when LabVIEW^{TM} loads a VI containing the ADLINK drivers. Then, the only thing to do is to set the proper working point for the attenuators, i.e. set the maximum possible attenuation ($60dB$) before the begin of the operations (see figure 3.2.3). Concerning the NI cards, they are not all directly initialized, since some cards need tasks to be created. The frame 0 of the NI initialization sequence (see figure 3.2.3) manage the digitizer cards. They need a classical initialization, i.e. the creation of references, each one associated with a single device. Concerning analog or digital i/o, they must have an associated task that identify it and make it available for operations. We create tasks for SHS analog and digital i/o, ATS analog output and KLP analog output in frames 1 to 6. Figure 3.2.3 only shows the example of task creation for the SHS analog inputs (frame 1). The last frame (frame 7)is

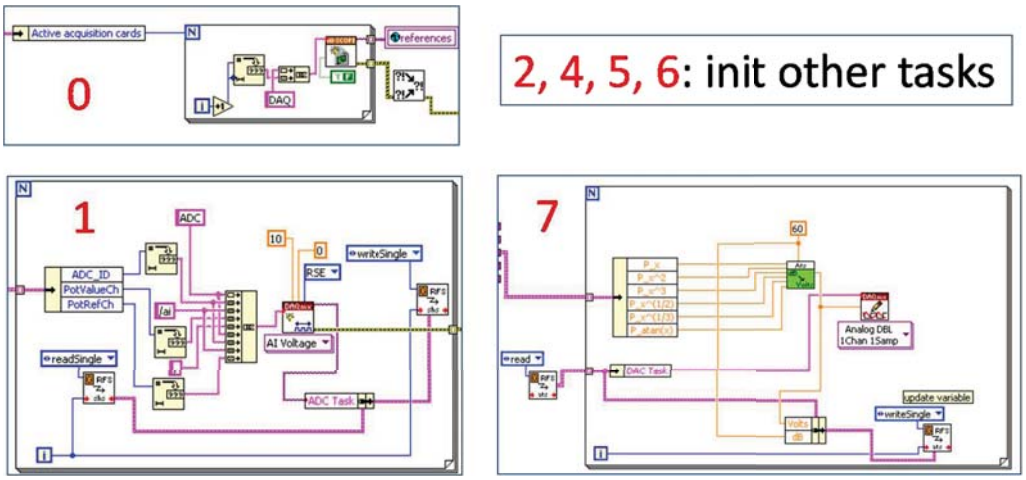


Figure 12: RFSinitHW: creation of the NI tasks and initialization of the NI hardware to the default set point

dedicated to set the correct ATS working point ($60dB$) where the created ATS DAC task is used for the first time.

3.2.4 CTRL: CPU cycle description

THe RFSCTRL VI (as in the other classes) handle all the tasks that a 3rd level RFS CPU has to accomplish during a single cycle. Below are presented and discussed the various RFSCTRL operations, showing snapshots of the corresponding frames of the main CTRL stacked sequence.

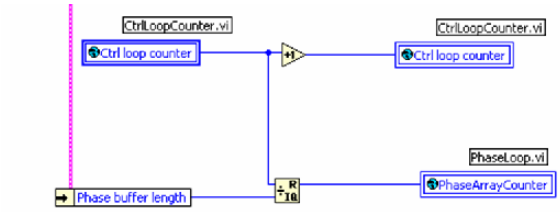


Figure 13: The first frame of the RFSCTRL main sequence updates the counters

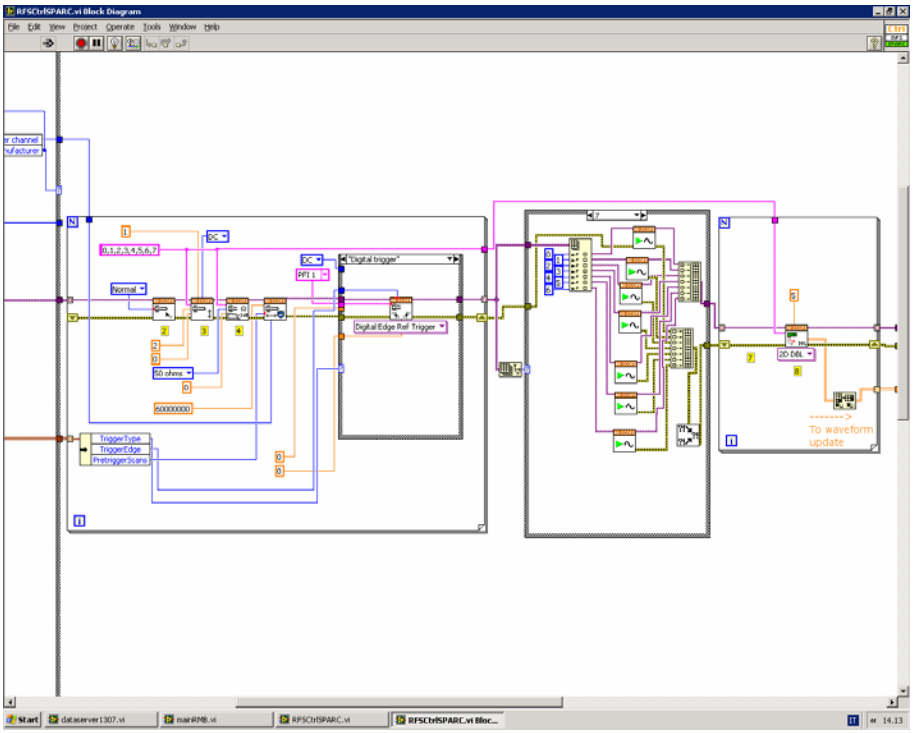
Update counters In figure 3.2.4 the first frame of the main sequence is shown. Here two counters are updated: (i) the Ctrl loop counter that simply counts the CPU cycles and (ii) the Phase buffer length counter that memorize which position has to be updated in the phase buffer array. This array is used to calculate the actual phase of the signals that are used to implement the software phase locking feedback (see in this section the paragraph

about the feedback). The dimension of this array is fixed by the field 'Phase buffer length' of the RFSHCIsta control type and the counter is reset to 0 when it reaches this value.

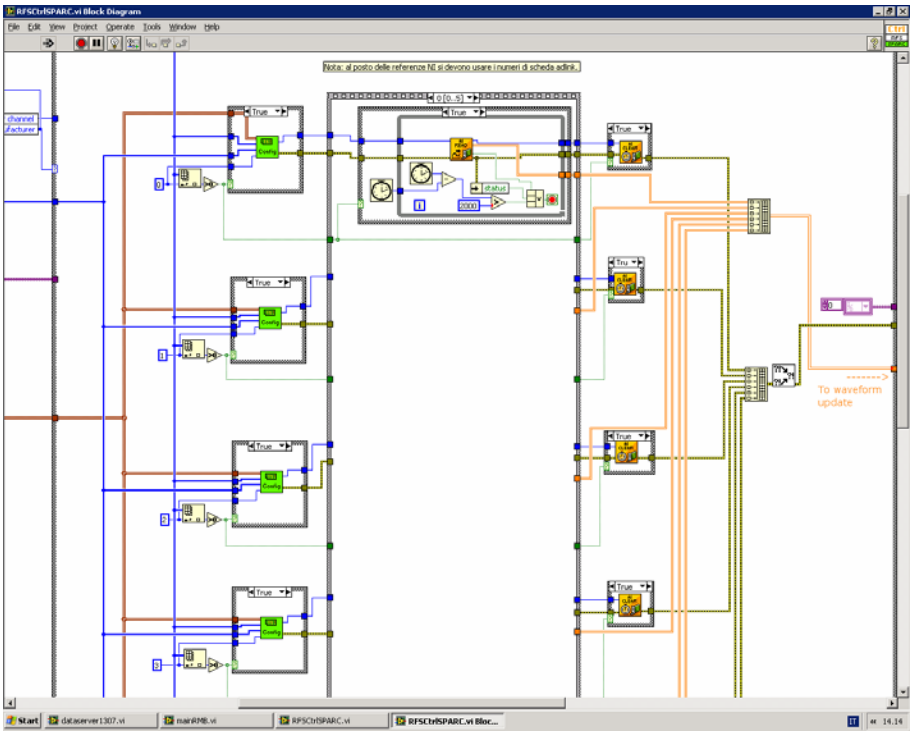
Read waveforms This paragraph concern the demodulated signal acquisition performed with sampling cards. These devices, as the normal I/O ones, have been acquired by two different manufacturers: ADLINK Technology and National Instruments. So, the CTRL frame that handles this task is provided with two distinct cases. Figures 3.2.4 (a) and (b) show these two cases and they are described below.

- NI: the card initialization is made in the VI RFSinitHW as described in section 3.2.3, where the card references are created. Here They are recalled and used to configure the cards and to arm the trigger before the acquisition begins (first FOR cycle in figure 3.2.4(a)). After that, the data acquisition take place: all the acquisition cards have to contemporaneously acquire the data, so, knowing the number of active cards (stored in the RFSHCIsta element), the correct number of instances of the 'NIscope initiate acquisition' VI are called (see the case structure in figure 3.2.4(a)). This must be done to not lose any trigger event in any acquisition card and we want to remember that this fact is the cause of the RFS special structure with subclasses: the elements can not be treated separately using a FOR cycle, but have to be handled simultaneously. After the trigger event the data are transferred from the card internal memories to the 3rd level CPU memory, using the VI 'NIscope MULTIfetch' that stores the data in a 2D array of floating point numbers. As shown in figure 3.2.4 This array is then used to update the variable 'waveforms' that acts as a buffer and its content can be requested by the first level sending a command to the RFS data server (see section 3.1.3 and 2).

- ADLINK: a similar frame is used to perform the waveform acquisition with the ADLINK cards, but some important differences are here discussed. First of all the ADLINK cards don't need any initialization because they are directly initialized by the manufacturer driver when a VI concerning them is loaded in memory. So the ADLINK case in the acquisition frame simply accepts in input the card numbers to configure them. The configuration is made inside a custom VI that contains some ADLINK sub-VIs that prepare the cards for the acquisition. After the configuration the acquisition is performed: it take place automatically when the trigger even occurs and the data are stored in the card memories. One only have to fetch the data from the memory, to store them in a array of floating point numbers and to clear the operation (signaling that the acquisition ended and make the card ready to receive a new configuration). All this operations are described in figure 3.2.4(b), where you



(a)



(b)

Figure 14: The second frame of the RFSCTRL main sequence acquire the demodulated signals

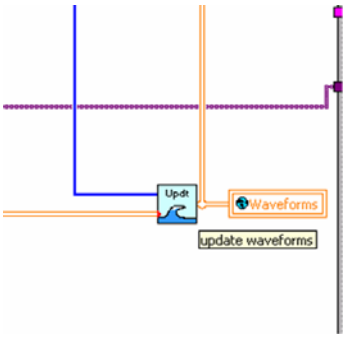


Figure 15: The second frame of the main CTRL sequence ends with the waveform variable update

can see that the fetch have to happen in a different instant for each card. Nevertheless, the data acquisition happens contemporaneously for all the digitizers, so they have to be ready to acquire simultaneously. So, also in the case of ADLINK drivers, the RFS subclass structure had to be realized. As in the case of NI cards, the data are finally stored in the 'Waveform' variable (see figure 3.2.4)to be available from the 1st level machines trough the data server.

Read potentiometer To have a continuous display of the signal phase shifters position, we implemented in the RFSCTRL a frame in the main sequence that read voltage of the potentiometers. In figure 3.2.4, this frame is shown. Two channels per potentiometer are used to acquire the voltage in the central pin and the bias voltage ($10V$) applied at the potentiometer ends. The ratio between these voltages gives the value of the potentiometer stored in the SHS dynamic variable. This method is used to be insensitive to possible ripples or noise affecting the $10V$ source.
After the voltage acquisition, this frame check if the DC motor has reached the limit positions and memorize the information in the proper 'upper limit' or 'lower limit' field of the dynamic sub-element variable (see section 3.1 for an accurate description of these fields).

Feedback The next frame of the CTRL main sequence is dedicated to control the software phase locking feedback that acts moving the signal phase shifters. This frame only acts if the field 'Feedback' of the RFSSHSdyn element is TRUE. Figure 3.2.4(a) report the first feedback frame tasks that are: (i) read the actual phase of the SHS reference DEM (i.e. the signal that the phase shifter has to follow) and (ii) calculate the difference between this value and the optimal phase value chosen (stored in the 'LockedPhase' field

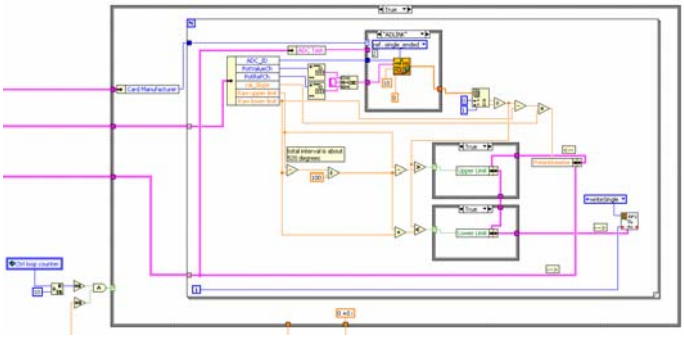


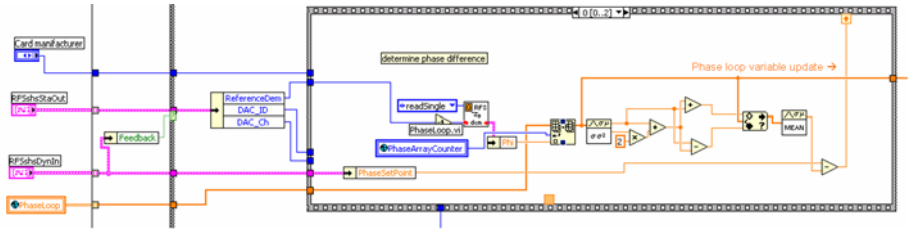
Figure 16: The third frame of the main CTRL sequence: SHS potentiometers reading

of the RFSSHSdyn subelement).This way a sort of feedback error signal is calculated. Before continuing we want to make a note on the actual phase value: it is calculated averaging the lasts values read from the DEM reference elements and the number of samples that one considers doing this average is equal to the 'Phase buffer length' field of the RFSHCIdyn variable. The average is then refined cutting the values more distant from the mean to be less sensitive to some instability like an arc inside a linac RF power device. Next step in the feedback frame is shown in figure 3.2.4(b). Here the calculated error signal is used inside the 'Control Voltage' VI, where also the 'FeedbackGain' and 'FeedbackThreshold' RFSshsDYN fields are used as an input variable. We don't report the content of this VI, but only report that the voltage applied at the motor driver using a DAC channel is calculated proportionally to the error signal.After this frame, the field 'DAC value' of the RFSSHSdyn variable is updated with the current voltage value applied to the PWM motor driver (see figure 3.2.4(c)).

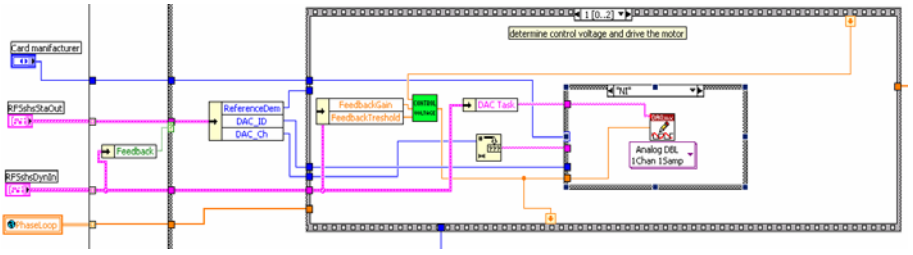
Error handling Last frame of the CTRL main sequence id dedicated to the error handling. It simply check if an error is occurred inside the CTRL VI and in this case turn off the SHS motors and set the maximum attenuation in the ATS.

3.2.5 CMD: receive and execute commands

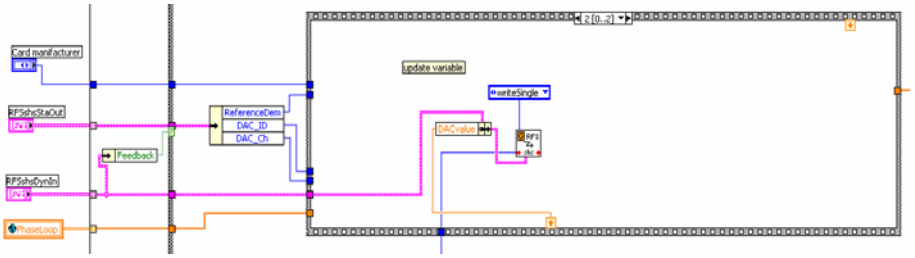
In this section we only report the command syntax without showing the command decoder VI to save a lot of figure space. Nevertheless here we describe the internal structure of this VI, that is very similar to the other classes CMD VIs. When a command is sent to the 3rd level machine, a series of case structures is called one after another. Like in a matrioska doll, every single word is analyzed in each case structure until the command line is finished and only the correct command is chosen to be executed. A description of the syntax for all the allowed commands is reported below. We report only the command



(a)



(b)



(c)

Figure 17: The fourth frame of the RFSCTRL main sequence: software feedback control

parameters that are placed after the class element name (in this case RFSXX00n) and after the command descriptor (SETT, SWTC, etc.) in the command line sent from the console level.

SETT decoder This VI is called if the command identifier in the command line sent from the console level is 'SETT'. Here we report the command parameters and their meaning, i.e. the physical action that the 3rd level CPU perform after receiving it.

- DEMXXnnn topstart n (integer number): sets in the RFSDEMdyn variable the average beginning position, expressed in sampling points, to calculate the top value of a square pulse;
- DEMXXnnn toplength n (integer number): sets in the RFSDEMdyn variable the number sampling points used to calculate the top value of a square pulse;
- DEMXXnnn botstart n (integer number): sets in the RFSDEMdyn variable the average beginning position, expressed in sampling points, to calculate the bottom value (when RF is OFF) of a demodulated pulse;
- DEMXXnnn botlength n (integer number): sets in the RFSDEMdyn variable the number sampling points used to calculate the bottom value of an RF pulse;
- DEMXXnnn fitstart n (integer number): sets in the RFSDEMdyn variable the fit beginning position, expressed in sampling points, to calculate the phase or amplitude value of an exponential decaying pulse;
- DEMXXnnn lnfitlen n (integer number): sets in the RFSDEMdyn variable the number sampling points used to calculate the phase value of an exponential decaying pulse;
- DEMXXnnn exfitlen n (integer number): sets in the RFSDEMdyn variable the number sampling points used to calculate the amplitude value of an exponential decaying pulse;
- SHSXXnnn REV2: sets the maximum reverse speed for a signal phase shifter motor ($0V$ at the input of the PWM motor driver) and updates the RFSSHSdyn variable;
- SHSXXnnn REV2: sets the medium reverse speed for a signal phase shifter motor ($1.5V$ at the input of the PWM motor driver) and updates the RFSSHSdyn variable;
- SHSXXnnn STOP: stops a signal phase shifter motor ($2V$ at the input of the PWM motor driver) and updates the RFSSHSdyn variable;

- SHSXXnnn FRW1: sets the medium forward speed for a signal phase shifter motor ($2.5V$ at the input of the PWM motor driver) and updates the RFSSHSdyn variable;
- SHSXXnnn FRW2: sets the maximum forward speed for a signal phase shifter motor ($4V$ at the input of the PWM motor driver) and updates the RFSSHSdyn variable;
- SHSXXnnn PHAS n (floating point number): sets the phase that the software feedback follow and stores this value in the RFSSHSdyn variable;
- SHSXXnnn GAIN n(floating point number): this command sets the feedback gain for an SHS device changing the 'FeedbackGain' field inside the RFSshsDYN variable. See section 3.1 for further details;
- SHSXXnnn TRSH n(floating point number): this command sets the feedback threshold for an SHS device changing the 'FeedbackThreshold' field inside the RFSshsDYN variable. The threshold is the minimum phase difference for the feedback intervention. See section 3.1 for further details;
- SHSXXnnn BUFF n(integer number): this command sets the phase buffer length for the SHS devices changing the 'Phase buffer length' field inside the RFShciDYN variable. Pay attention using this command because it affect ALL the phase shifter devices at the same time (also if the command is called with a single device name). See section 3.1 for further details;
- ATSXXnnn dB n (floating point number): sets the chosen attenuation to a signal attenuator and updates the RFSATSdyn variable with the output voltage value and the corresponding attenuation value (expressed in dB);
- KLPXXnnn PHI: sets the fast klystron phase feedback working point acting on an electronic phase shifter voltage input and updates the RFSKLPdyn variable;
- KLPXXnnn REF: sets the fast klystron phase feedback amplifiers reference voltage and updates the RFSKLPdyn variable.

READ decoder This VI is called if the command identifier in the command line sent from the console level is 'READ'. Here we report the command parameters and their meaning, i.e. the physical action that the 3rd level CPU perform after receiving it.

- SHSXXnnn POT: reads a signal phase sifter potentiometer voltage (doing a ratio between the central pin voltage and the bias voltage) and updates the RFSSHSdyn variable corresponding field.

SWTC decoder This VI is called if the command identifier in the command line sent from the console level is 'SWTC'. Here we report the command parameters and their meaning, i.e. the physical action that the 3rd level CPU perform after receiving it.

- SHSXXnnn ON: switches a signal phase shifter PWM motor driver ON using a digital output and updates the RFSSHSdyn variable;
- SHSXXnnn OFF: switches a signal phase shifter PWM motor driver OFF using a digital output and updates the RFSSHSdyn variable;
- SHSXXnnn FDB ON: switches a signal phase shifter software feedback ON, initializes the 'PhaseLoop' buffer with the actual measured phase at the reference DEM element and updates the RFSSHSdyn variable;
- SHSXXnnn FDB OFF: switches a signal phase shifter software feedback OFF and updates the RFSSHSdyn variable.

3.2.6 RFS close hardware

Also in this case we chose to not report images of this VI, because its functioning is very intuitive and there are a lot of frames that don't need a graphical explication. In fact the RFScloseHW VI check whether NI or ADLINK cards are used and perform the following actions:

- it sets the maximum attenuation in the ATS elements;
- it stops and turns off the SHS motors;
- in the case of NI hardware it also closes all the opened ADC, DAC and DIO tasks and the sampling card references.

Here we want to remember that the ADLINK cards don't need to be initialized and released because their driver automatically do it when a pertinent VI is open or when a LabVIEW^{TM} instance is closed.

3.3 1st level VIs: a manual both for the developer and operators

The aim of this section is to describe the main RFS class related control panels (1st level VIs), that any operator can open using a control room console. These VIs are able to send commands to the CPUs simply pressing a button. This way the machine control become very simple and all the tasks can be performed by an operator that does not know the control system architecture.

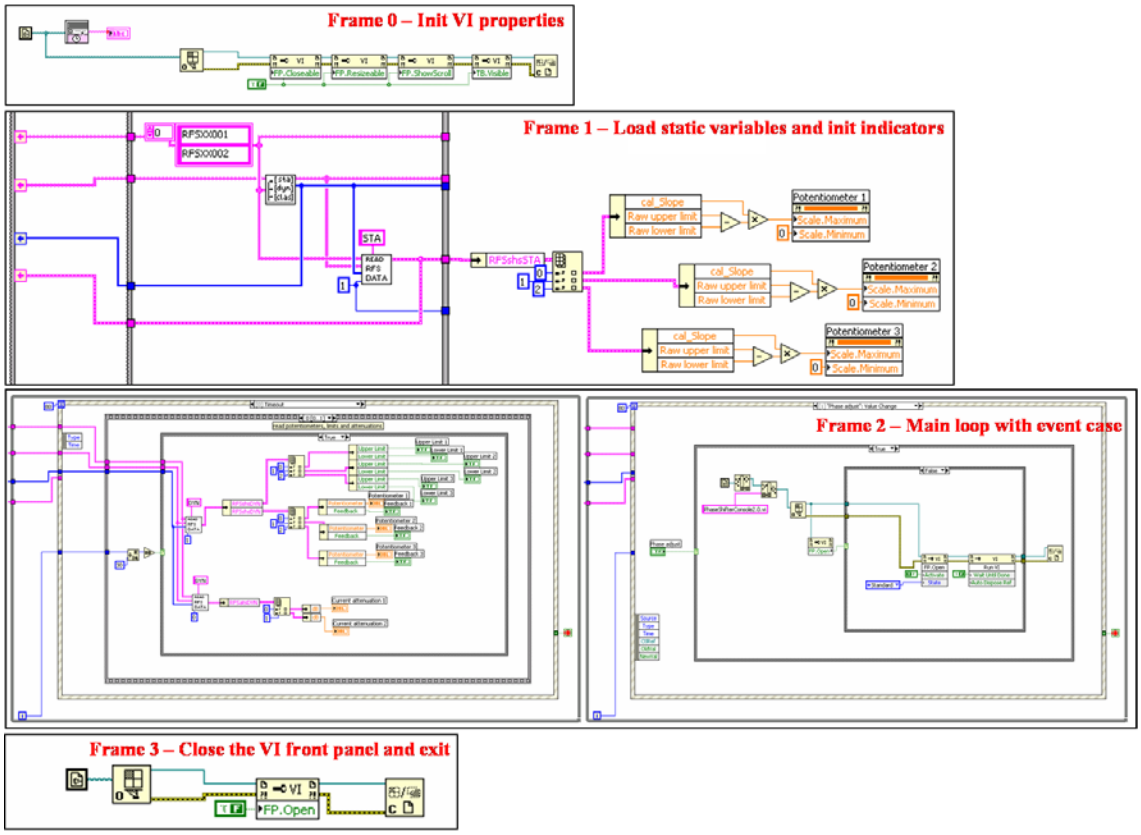


Figure 18: The typical sequence architecture of a 1st level VI: the main cycle is constituted by an event structure placed inside a while loop

3.3.1 Introduction to the 1st level panels architecture

The great majority of the 1st level VIs (and in particular all the RFS panels) have the same block diagram architecture, shown in figure 3.3. It consist the main stacked sequence described below:

- The first initialization frame set the VI fixed properties (not resizable VI, hidden scroll bars, hidden emergency stop button and so on) to prevent any VI modification performed by a control system 'non-developer' user;

- The second frame is another frame of initialization, where the needed STA variables are loaded from the 3rd level CPUs and some properties of the front panel indicators are set;

- The third frame contains the main while loop that remains active until the VI is closed. Inside this while loop a event structure is placed. An event can be related to mouse moves or clicks or to other software trigger that one can choose from a list. This kind of structure has a mandatory 'Timeout' frame that waits for 'events' a set amount of time and then execute its content. In practice here the developer puts the commands and tasks executed every while cycle. The other frames are identified by an event and are executed only if this event occurs before the timeout. The developer will create a number of event frames equal to the number of commands the VI can send to the 3rd level CPUs. In fact the panel is normally designed inserting buttons, knobs and other GUI objects and linking a mouse action performed on them to an event frame. So if no mouse action is performed the VI continue to execute only the 'Timeout' frame (that is normally used to display some DYN variable field values); else, if an operator wants to execute a command, for example pressing a button, the related event frame is executed (normally it sends a command to the 3rd level or modifies an information display property);

- Finally, when the while cycle is stopped (using a special custom button), the last frame of the main sequence is executed. It closes the front panel, protecting the VI from further modifications.

In the following sections we will precisely describe the aspect and functioning of all the 1st level VI front panels that control the RFS class objects.

3.3.2 RFS console: the main window

This panel is dedicated to give a quick overview of the RF system showing the main signals from the machine and the set point of the main RF devices, like attenuators and phase shifters. The operator can not send any command from this panel, but he has to press one of the rectangular buttons (named 'Phase adjust', 'Attenuator console', etc.) to open a VI dedicated to the task he want to perform. Precisely, these buttons open VIs if they are not open and bring to the front of the screen the VI front panel if it is already open. This means that in a console one can open only a single instance of a RFS control panel VI.

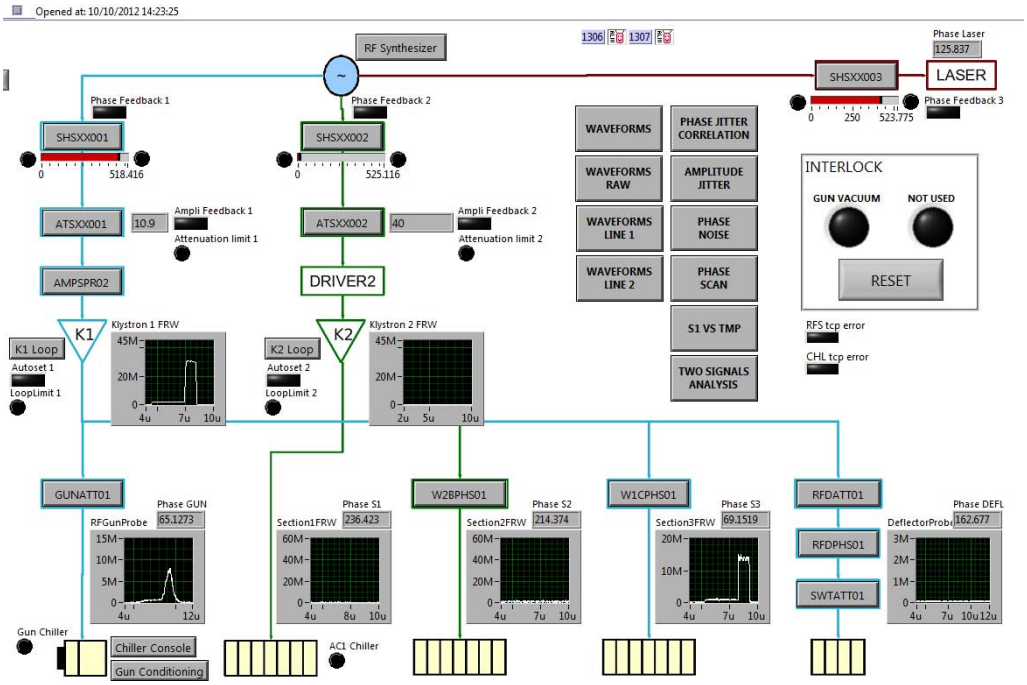


Figure 19: The front panel of the RFSconsole VI, the main control panel of the RFS system

3.3.3 RFS console: Waveform monitor

This is the most simple panel to use for an operator, because it only shows the signals coming from the RF devices along the machine. The only actions allowed are: (i) press the button above each graph to choose the location of the signal to display and (ii) press the 'print' button opening the SPARC logbook prompt to insert data and graphs into it. The waveforms displayed are white or red. The white waveform represents the amplitude of the signal while the red one represents the phase of the signal. Note that not all the signals have the red graph active because some of them are obtained from a device not able to detect the phase (for example an RF peak detector).

3.3.4 RFS console: Attenuator console

This is another very simple VI that controls the signal attenuators placed before the final amplification stage that feeds the waveguides with the radiation. The panel is divided into two identical regions because there are two low level attenuators, each one before a klystron. To set a random attenuation (from $0dB$ to $60dB$) the operator must write the desired value in the right numeric control and press the arrow button. After a while, if no error is occurred, the command is executed and the new set point is shown in the left

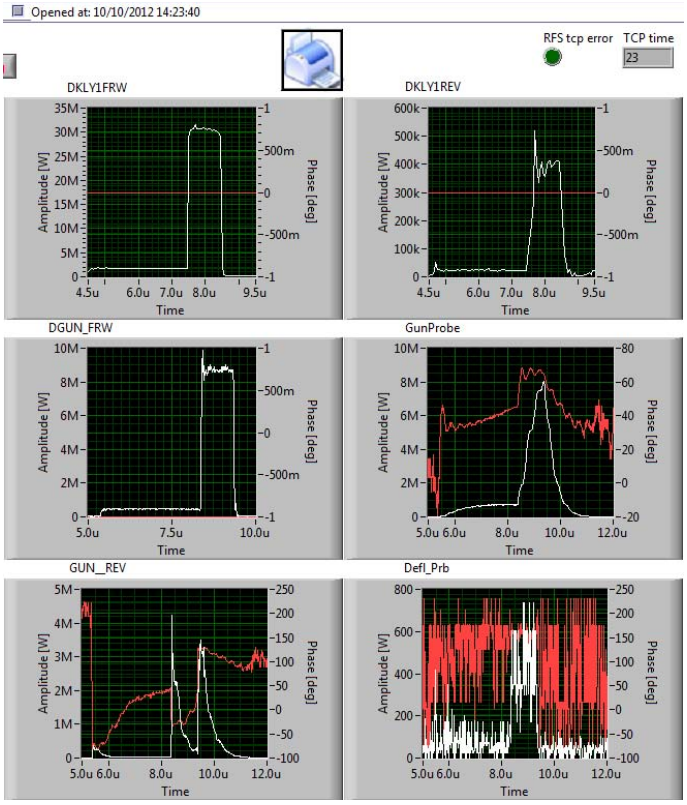


Figure 20: The front panel of the waveform monitor VI that displays up to 6 signal from the machine at the same time

numeric indicator. The indicator reads the value from the RFSatsDYN variable, so it is susceptible to variations performed from every console and from a software interlock (if active). The two buttons below the first row are used to quickly increase the attenuation level to prevent RF devices damage. The left button sets an attenuation $3dB$ above the current one, while the right button sets the maximum attenuation ($60dB$), shutting down in practice the RF signal.

3.3.5 RFS console: Phase shifter console

This VI act as interface toward the signal phase shifters device that are placed in the two RF power lines and in the laser synchronization line. It is used both for manually set the phase and to configure and activate the automatic phase locking system. Its front panel is reported in figure 3.3.5. When the VI is opened it sets SHSXX001 as default device to be controlled. One can easily change it pressing the button ‘Element’ and choosing from a list. The other indicators above the graphs show what is the demodulator associated to the device in use (‘ReferenceDem’) and the location where signal is spilled to measure the

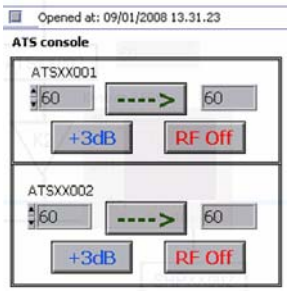


Figure 21: The front panel of the attenuators console VI

phase ('ReferenceDemLocation'). The window 'Waveform fit settings' is used to choose the points of the RF pulse where the phase is calculated.This can be done looking at the 'PhasePulse' graph. The meaning of these controls has been yet described in section 3.1. Once the fit parameters are correctly set,the 3rd level CPU calculates a single value for each machine trigger and the 'PhaseAverage' chart activates showing the phase behavior in the time. The tab in the bottom left part of the panel is used to controlling the DC motor of the phase shifter. If the 'Manual' mode is selected, the operator only can manually adjust the phase moving the motor at the 4 allowed speeds. This way the phase can not be set with accuracy. To do this, one can select the 'Auto(feedback)' mode, opening a new panel window. In this mode the basic commands are: (i) set the phase working point of the 'ReferenceDemLocation' (i.e. the phase of the RF line where the phase shifter is placed); (ii) switch the feedback ON or OFF. If the operator is an RFS system expert, he can press the 'Show Advanced Settings' and use more advanced features to configure the feedback functioning. They are:

- set the phase buffer length: this sets how many phase values (taken from different shots) are averaged to calculate the actual phase value. This parameter decide how much the feedback is sensitive to fast phase variation: the more the phase buffer is large, the more the sensitivity is small. Setting this parameter needs some care because one wants to be insensitive to beam instabilities (gun field arcs), but sensitive to other phase drifts. The phase buffer length obviously affect the feedback bandwidth;

- set the feedback loop gain;

- set the feedback threshold: this is a software threshold needed to avoid the feedback intervention for small phase errors. The feedback will work not so often to preserve the motor duration.

In the bottom right part of the panel some useful indicators concerning the motor and feedback status are displayed.

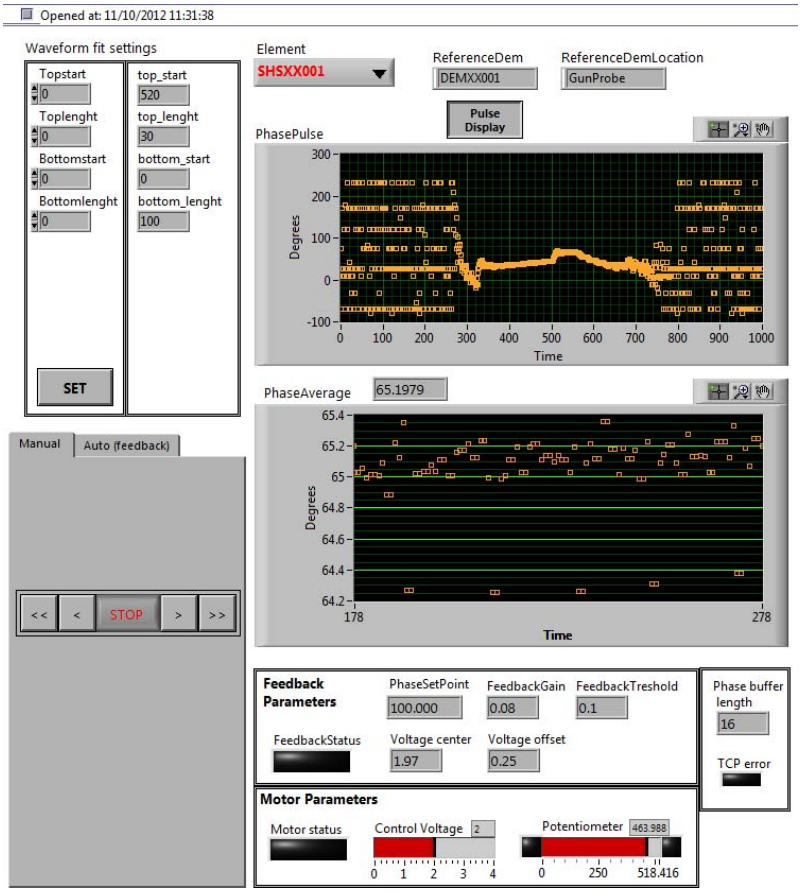


Figure 22: The panel where the signal phase shifter devices are controlled

3.3.6 RFS console: Phase noise measurement

This panel provide a simple tool to measure the phase noise of an RF signal or a time jitter of a incoming optical or RF fast pulse, respect to the main RF oscillator (see [3] for further details on hardware and synchronization system). Some front panel objects of this VI are very similar to those placed in the phase shifter console, described in the previous section. They are the parameter fit window, the 'PhasePulse' graph, the 'ProcessedPhase' chart and the button at the top left corner of the panel, used to select the signal under test. An histogram is present below the phase chart to show distribution of the acquired phase samples. Also a mean and a standard deviation is calculated on those values measure of the RMS phase noise (or time jitter) of the signal. The phase fitting and calculation algorithm depends on the type of signal one is analyzing (i.e. square or exponential decaying

pulse). Details about these algorithm are given in [3]. The 'Refresh histogram' button clear the buffer used to store the acquired phase values and in practice it permit to start a new measurement not affected by the previous phase calculations. The last object is the print button that, as usual, opens the SPARC logbook prompt to save images and spreadsheet data files. All the saved items store data after the last 'Refresh histogram' event.

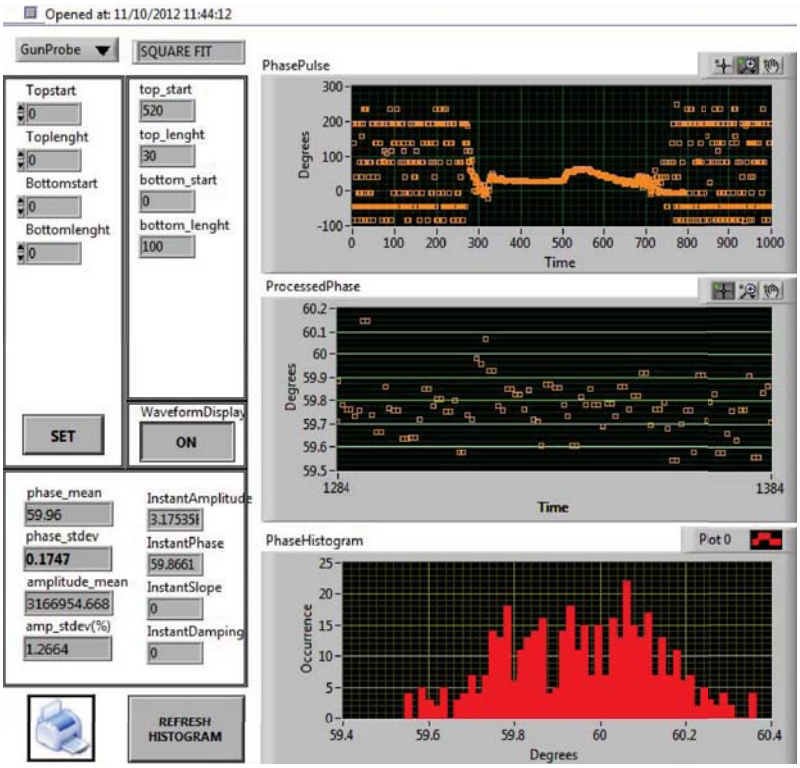


Figure 23: The panel to perform phase noise (or time jitter) measurements

3.3.7 RFS console: Klystron loop console

The SPARC klystrons are working with a fast intra-pulse phase locking system (see [3] and [7]) that is remotely controlled by the 'KlystronLoopConsole' VI, shown in figure 3.3.7. This loop can correct the phase inside a single $2.5 \div 4.5 \mu s$ RF pulse with a transient of about $1 \mu s$ (i.e. a bandwidth of $1 MHz$). For each klystron, two signal are monitored and two parameters can be changed to improve the loop performances. The monitored signal are the error signal (difference from the working point and the instantaneous phase) and the correction signal (that drives an electronic phase shifter). The two controls on the left part a klystron control window are used for the following purposes: (i) set the PLL working point controlling the driving voltage of an electronic phase shifter (the 'phase'

control value) and (ii) optimize the transient behavior, changing the mean value of the electronic components bias voltage (the 'reference' control value). A good setting for a klystron loop is identified looking at the shapes of the two monitored signals. The 'ERR' signal obviously has to be minimized, so in the left graph one wants to see null signal, as shown in the 'ERR' graph of figure 3.3.7'. The 'CORR' signal has to be more or less at the center of the allowed interval $0 \div 4V$, to be sure that the locking amplifiers work well inside their linear region and are out of the saturation region (a value of $2 \div 3V$ is good), as reported in the 'CORR' graph of figure 3.3.7. An operator can reach these correct settings only using the 'Phase' control, but he will see a large spike in the transient region of the 'ERR' signal. To minimize the transient effect, one can use the 'Reference' control and bring the 'ERR' signal to be about zero also in the first μs interval. Note that this procedure can modify again the 'ERR' and 'CORR' behavior after the transient, so an operator can reach the refine the working point set doing an iterative procedure. To simplify this task, an automatic routine to control the optimal feedback parameters is under development.

4 Conclusion

The document has been published to share the knowledge on the SPARC low level RF control system. To reach this goal, the first part of the document (section 2) has been dedicated to an overview of the general architecture of the SPARC control system. This is useful for the developer to understand how the various objects with different physical interface and functions are inserted in the control system in an identical way. In fact only few LabVIEW^{TM} VIs identify a single software class. The second part of the document (in particular sections 3.1 and 3.2) has been written to accurately describe how these class specific VIs are designed and programmed for the RFS class that is the one that handles the majority of the RF low level system devices. This class needed an accurate description for its special design. In fact its architecture is more complicated, because some signal acquisition and object 'cross-talking' problems. The last 3.3 section is about the programs that runs on the control room consoles. They are at disposition of every user, more or less qualified concerning the RF system architecture. So, this section wants to be a manual for the operator, that gives the procedures to correctly set the RF low level system devices. Every panel that can be opened and that involves the RFS software class is accurately described and the function of any front panel control and indicator is given.

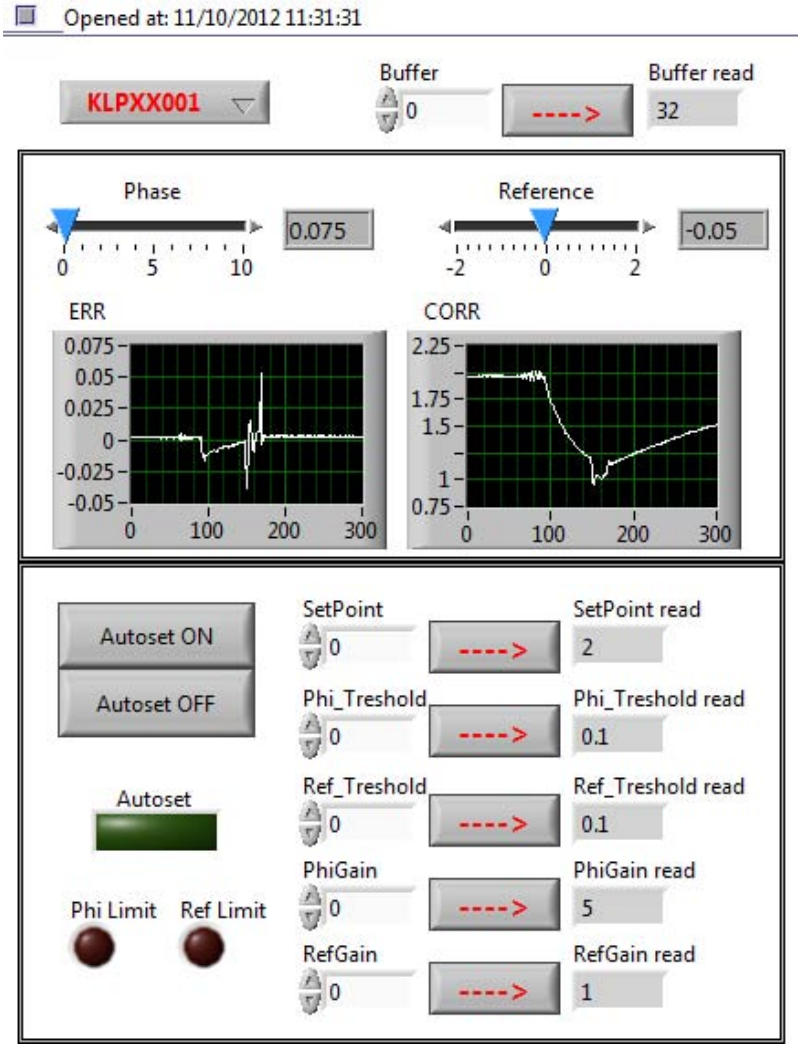


Figure 24: The panel to set the klystron fast phase feedback working point

Acknowledgments

I want to thank Alessandro Gallo and Giampiero Di Pirro for their constant support in developing the LLRF control system computational algorithms and general architecture.

References

[1] SPARC Project Team. Sparc injector tdr, www.lnf.infn.it/acceleratori/sparc. Technical report, INFN - LNF, 2004.

[2] National instruments corporation website. *http://www.ni.com/labview*, 2007.

[3] M. Bellaveglia. *Radio frequency control system and synchronization in high brightness photo-injectors driving FEL experiments*. PhD thesis, Unversita' degli studi di Roma 'La Sapienza', 2006.

[4] M. Bellaveglia et al. First operation with sparc control system. *Proc. of PcaPAC2006, CEBAF Center Jefferson Lab Newport News, VA USA*, Oct. 2006.

[5] G. Di Pirro et al. Dante: Control system for dafne based on macintosh and labview. *Nuclear Instrument and Methods in Physics Research A 352 455-475*, 1994.

[6] L. Catani. A communication protocol for a distributed control system with labview. *Proc. of PcaPAC2006, CEBAF Center Jefferson Lab Newport News, VA USA*, Oct. 2006.

[7] G. Gatti C. Vicario A. Gallo, M. Bellaveglia. Laser and rf synchronization measurements at sparc. *Proceedings of PAC07, Albuquerque, New Mexico, USA*, 2007.

Contents