TESLA Report 2005-19

WARSAW UNIVERSITY OF TECHNOLOGY
Faculty of Electronics and Informational Technologies
Institute of Electronic Systems
ELHEP Laboratory

Piotr Dominik Pucyk

M.Sc. Thesis

DOOCS based control system for FPGA based cavity simulator and controller in VUV FEL.

Mentor:

Ph.D. Krzysztof Poźniak, WUT

Tutors:

D.Sc. Ryszard Romaniuk, WUT

Ph.D. Stefan Simrock, DESY

Warsaw, September 2005

Contents

Abstract

The X-ray free-electron laser XFEL that is being planned at the DESY research center in co-operation with European partners will produce high-intensity ultra-short X-ray flashes with the properties of laser light. This new light source, which can only be described in terms of superlatives, will open up a whole range of new perspectives for the natural sciences. It could also offer very promising opportunities for industrial users.

SIMCON (**SIM**ulator and **CON**troller) is the project of the fast, low latency digital controller dedicated for LLRF [1] system in VUV FEL experiment It is being developed by ELHEP [2] group in Institute of Electronic Systems at Warsaw University of Technology. The main purpose of the project is to create a controller for stabilizing the vector sum of fields in cavities of one cryo module in the experiment. The device can be also used as the simulator of the cavity and test bench for other devices.

Ths paper descrbes the concept, implementation and tests of the DOOCS based control system for SIMCON. The designed system is based the concept of autonomic and extendable modules connected by well defined, unified interfaces. The communication module controls the access to the hardware. It is crucial, that all modules (this presented in thesis and developed in the future) use this interface. Direct access to the control tables let the engineers to perform algorithm development or diagnostic measurements of the LLRF system. Default control tables generator makes the whole SIMCON an autonomic device, which can start immediately the operation without any additional tools.

[1]Low Level Radio Frequency
[2]Electronics for High Energy Physics

1 Introduction

The X-ray free-electron laser XFEL that is being planned at the DESY research center in co-operation with European partners will produce high-intensity ultra-short X-ray flashes with the properties of laser light. This new light source, which can only be described in terms of superlatives, will open up a whole range of new perspectives for the natural sciences. It could also offer very promising opportunities for industrial users. The overall layout of the X FEL linear accelerator and laser facility is shown in Fig. 1.

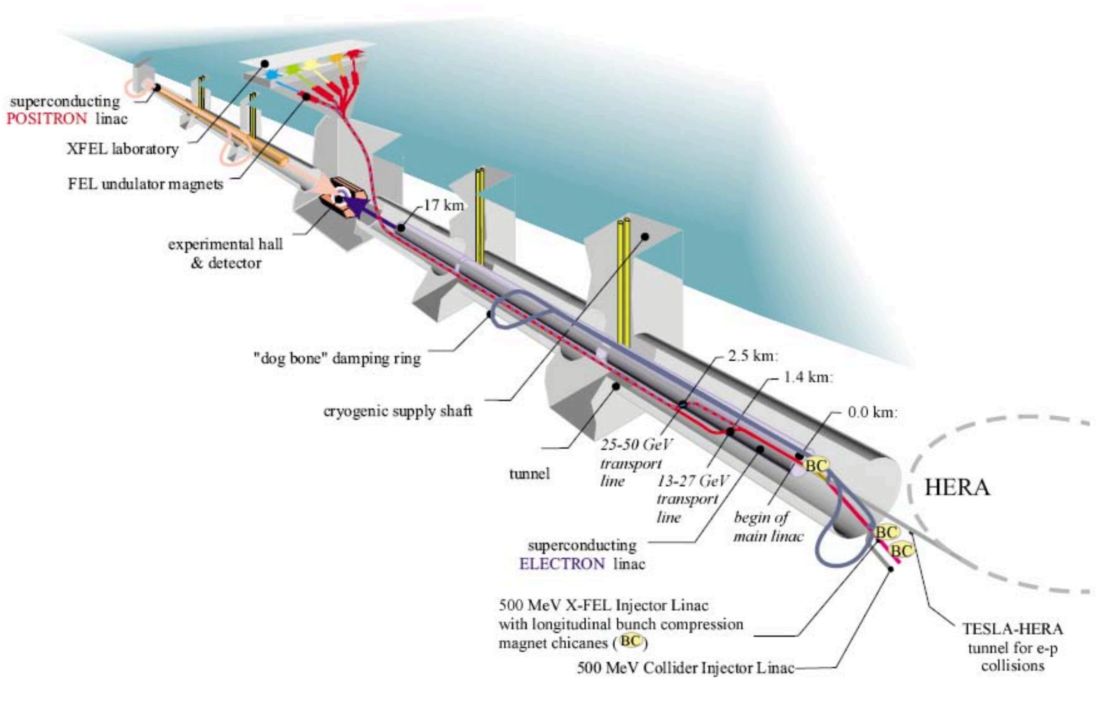

superconducting POSITRON linac
XFEL laboratory
FEL undulator magnets
17 km
experimental hall & detector
"dog bone" damping ring
cryogenic supply shaft
tunnel
2.5 km
1.4 km
0.0 km
25-50 GeV transport line
13-27 GeV transport line
begin of main linac
BC
HERA
superconducting ELECTRON linac
500 MeV X-FEL Injector Linac with longitudinal bunch compression magnet chicanes (BC)
500 MeV Collider Injector Linac
TESLA-HERA tunnel for e-p collisions

Figure 1: The layout of the XFEL facility.

The main X FEL parameters are:

- Total length of the facility: approx. 3.4 km
- Accelerator tunnel: approx. 2.1 km
- Depth underground: 6 - 38 m

- Experimental hall : 10 experimental stations at 5 beamlines, floor area approx. 4500 mÂ˛
- Scope for expansion: Second experimental hall with an additional 10 experimental stations
- Wavelength of X-ray radiation: 6 to 0.085 nanometers (nm) corresponding to electron energies of 10 to 20 billion electron volts (GeV)
- Length of radiation pulses: below 100 femtoseconds (fs)

The planned facility will include a superconducting linear accelerator that brings tightly bundled "bunches" of electrons to energies of several billion electron volts. At that point, the electrons race at almost the speed of light along a slalom course through a special arrangement of magnets called the undulator. As they go, they emit X-ray radiation that amplifies itself during the flight. The results are brilliant: Extremely short and intense X-ray flashes with laser properties. For such an X-ray laser to work, an electron beam of extremely high quality is required. And the TESLA superconducting accelerator technology is already making it possible to generate this kind of electron beam today.

Before the electrons can emit X-ray flashes, they must first be accelerated to energies of several billion electronvolts. That's exactly what happens inside the resonators, where electromagnetic fields accelerate the particles. The resonators are made of niobium and are superconducting: When they are cooled to a temperature of -271 ^{o}C, they lose their electrical resistance. Electrical current then flows through the resonators with no losses whatsoever - and that's an extremely efficient and energy-saving method of acceleration. Nearly the entire electrical output is transferred to the particles. Moreover, the superconducting resonators deliver an extraordinarily fine and even electron beam of extremely high quality. In the X-ray laser, each of several billion free-electrons needs to have the same energy and direction. They also need to be combined into bunches with a diameter of no more than one tenth of a millimeter. Unless the electron beam meets these very special requirements, the X-ray laser cannot be operated.

At present the present VUV-FEL [3] , a free electron laser to generate light in the vacuum ultraviolet part of the spectrum, is built at DESY in TTF facility. The project bases on the TTF

[3]Vacuum Ultra Violet - Free Electron Laser

infrastructure and is the pilot project for the X-FEL [4] . Both VUV-FEL and X-FEL accelerators bases on the same superconducting technology. In the VUV-FEL (Fig. 2.) cavities are grouped in cryomodules. Each cryomodule consists of 32 cavities. In general, four cryomodules are driven by one klystron (some klystrons drives one or two modules). In order to accelerate the beam, the electromagnetic field inside the cavity must be stabilized (in order to minimize the energy spread during beam transmission) and have appropriate phase(in order to accelerate and not deaccelerate the beam). The regulation of the field is performed by LLRF [5] system (Fig. 3. The system controls I and Q components of the cavity field (which corresponds to real and imaginary part of the field vector). Because one klystron drives many modules module, the LLRF system is used to stabilize a vector sum of 8 to 32 cavity fields. The LLRF consists of many devices from which one can mark out: downconverters, digital feedback controllers, vector modulators, piezo controllers, timing modules, and many ADC boards for monitoring the signals in the system.

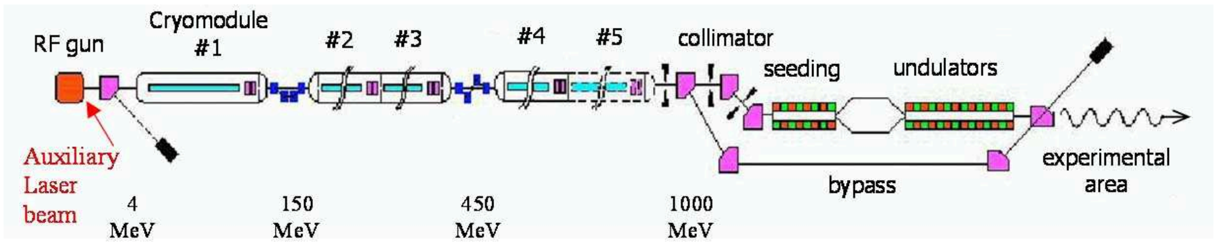


Figure 2: The layout of the VUV FEL tunnel.

[4]X-ray Free Electron Laser
[5]Low Level Radio Frequency

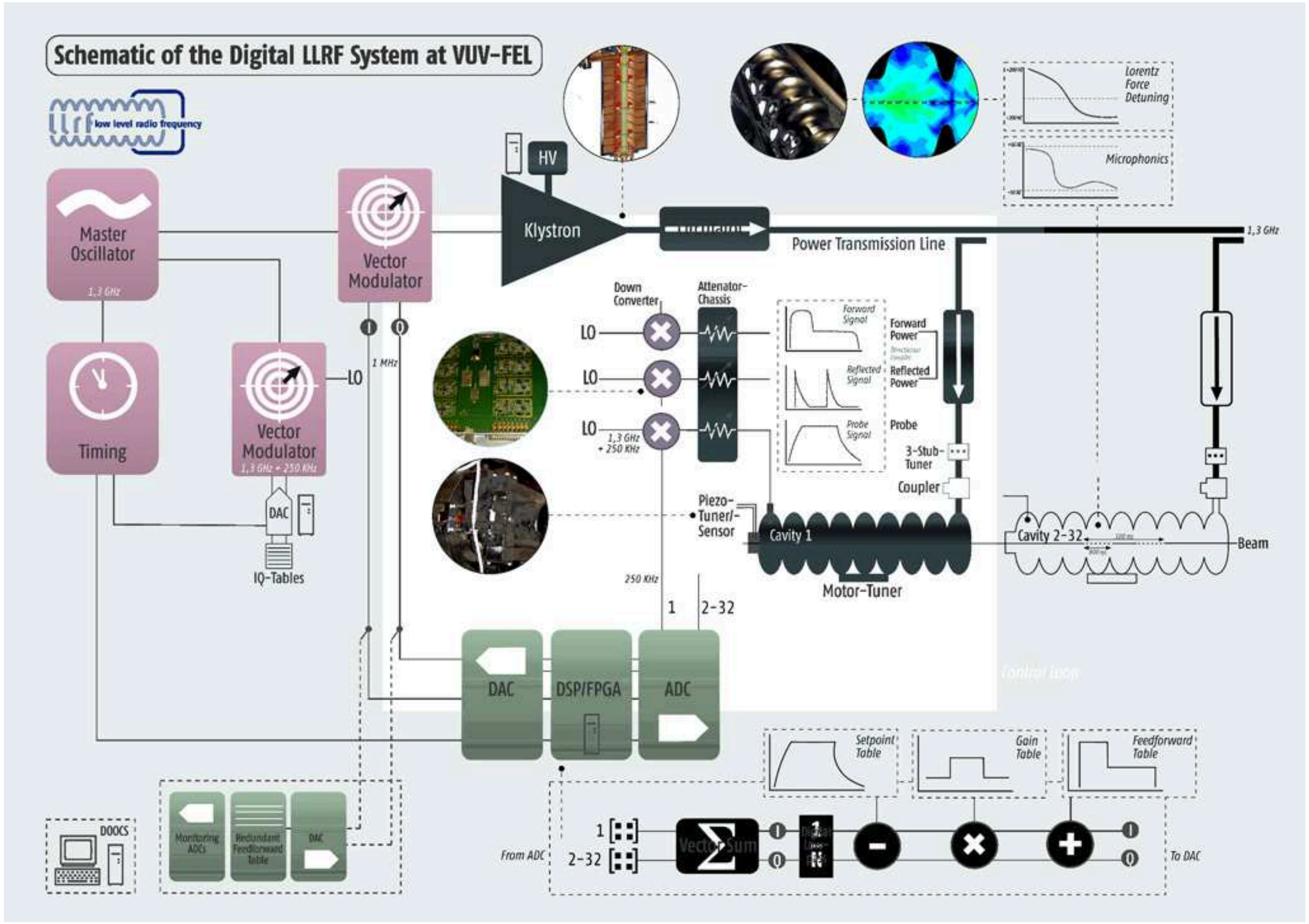


Figure 3: The LLRF system in VUV-FEL [12].

The main control loop in the LLRF system starts from the cavity probe. The signal from each cavity (1.3GHz) is downconverted to an intermediate frequency of 250KHz. Eight downconverted signals are inputs of the digital controller, which samples the probe signal with a frequency of 1MHz. Inside the controller (currently it is a DSP based system) the signal is decomposed into I and Q components. The controller is able to drive the system using Feed-Forward or/and Feedback algorithms. At the output I and Q signal is produced for driving the klystron. These two signals drive through vector modulator to reconstruct the complex signal form I and Q components. The output of the vector modulator drives klystron. With a sampling rate of 1MHz, every new probe sample is measured every 1 $1\mu s$. In order to stabilize the field using feedback algorithm, one must use high gain. High gain in the loop can make the system unstable if the feedback latency is to high. The maximum estiamted latency for the feedback algorithm has been estimated to about $1\mu s$ (for the whole control loop including latency of system (mainly cables) and controller board). The actual estimated system delay is about 500ns which forces the controller delay not bigger than 500ns. The requirements for the stability of the amplitude and phase of the field are tight: for the amplitude $3 * 10^{-4}$ and for the phase 0.1 degree. The control loop has many nonlinear elements like klystron, vector modulator or preamplifiers. Every conversion from analog to digital signal adds noise to the system. The temperature changes cause phase delay drifts in cables. All these distortion are added to the control signal and force the controller to perform sophisticated algorithms, which can compensate noises, drifts and nonlinearities. This require a calculation power of the controller and big data throughput.

The current solution bases on system with DSP processor (TMS320C67 from Texs Instruments). The controller is split into three boards - ADC board with 14bit analog to digital converters sampling with a frequency of 1MHz, the DSP board with TMS320C67 processors and DAC board (14 bit, 1MHz). All boards are connected through gigalink interfce. Because of the DSP the programmers can only optimize the software for the DSP processors but cannont optimize its architecture. Actual computation capabilities of the system are close to the limit. Also the computation power is closed to limit - the algorithm is performed with a time of more than $1\mu s$. The system operation requires the delay of the algorithm start according to the trigger.

In order to control the whole experiment, in which the LLRF is one of the systems, there

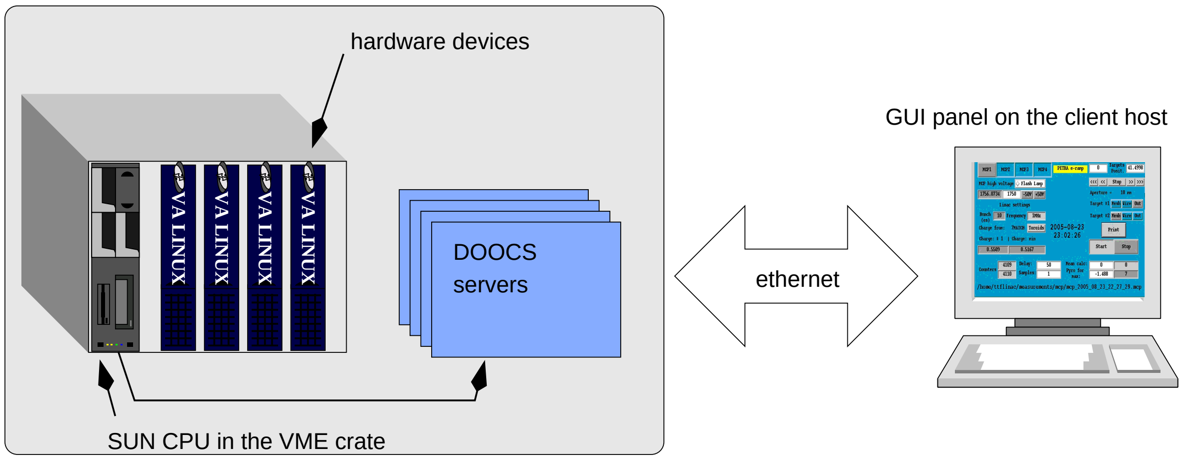


Figure 4: DOOCS as a remote, distributed system.

must be a dedicated software provided. In DESY for the TTF operation a complex control software system has been developed. It is named DOOCS (Distributed, Object Oriented Control System). The goal of this project is to provide an environment through which operators in the control room are able to remotely change the settings of the hardware and perform experiments. The idea of the system is to represent every device as a separate instance of the server (Fig. 4). Server, similar to the real device has some properties, which correspond to the real device parameters. On the client side, there are GUI based applications which provide user interface. The communication is performed through Ethernet, using RPC protocol.

The hardware background of DOOCS are SUN embedded computer placed in VME crates. VME is the official bus standard in VUV FEL, therefore most of devices are also places in the VME crate. Every SUN is connected to a fast, gigabit network. Because of the dimensions of the linac, crates with the devices are spread along the tunnel. The system is therefore really distributed in logical and physical way. DOOCS is object oriented environment. The DOOCS server itself is only the skeleton of the application. User must design the virtual device, provide the access to the hardware and create "virtual knobs" - the properties of the server which will be available to the client. Every device can be represented as a separate C++ class. There is one root class for all devices. One must extend it in order to create its own version, suitable for the particular device. In the device class, user declares the list of DOOCS properties. Every property is also a class which can represent the basic data types: integer number, float number,

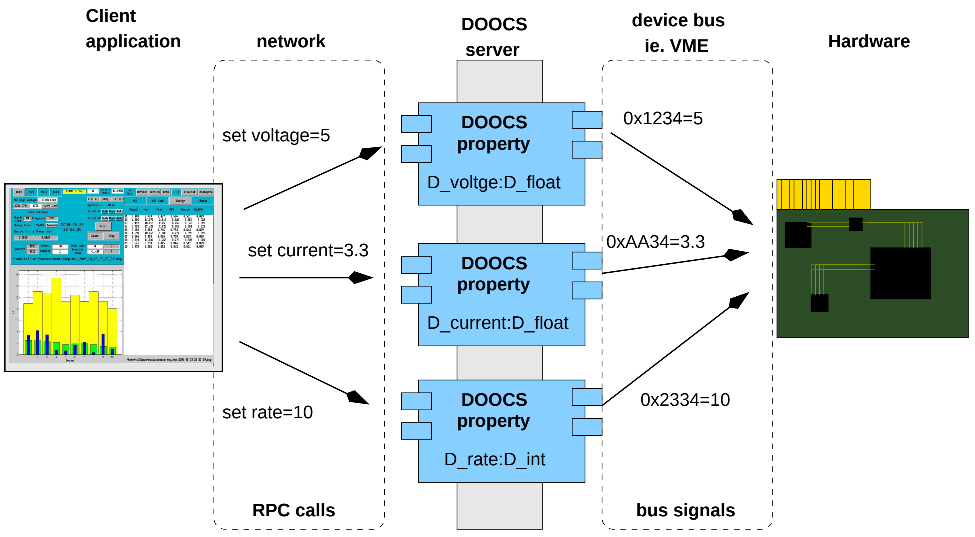


Figure 5: DOOCS properties are the bridge between the user which operates the virtual knobs and hardware.

or more complex structures like spectrum class, or history class (Fig. 5). DOOCS property class provides the access to the data from the network. It takes care of the all communication issues like RPC or XDR [6] protocols. Programmer must provide the hardware access for each property. The system provides archiving of the properties on the local hard drives. This is especially important in case of network problems. If for some reasons the network is down, the server continues the operation, however its properties cannot be adjusted. If for some reasons the server will be stopped, during the next start up it will load the latest values of the properties from local archive file and will continue the operation. DOOCS provides also client environment for creating virtual Panels. This tool is called DDD (DOOCS Data Display). It provide user friendly graphical interface (drag & drop) with a set of widgets. Using them, one can built its own graphical representation of the operation panel for the particular device. The important issue is the way that DOOCS addresses devices and properties in the network. As it was mentioned before, the implementation of the network protocol is hidden inside the application. Therefore DOOCS provides its own naming service called ENS (Equipment Name Service). Every property in DOOCS environment has its own unique address. It has the following form: *facility/device/location/property*. The facility and device are logical addresses stored in the ENS

[6] eXternal Data Representation - standard which allows to exchange data between computer system with different hardware architecture

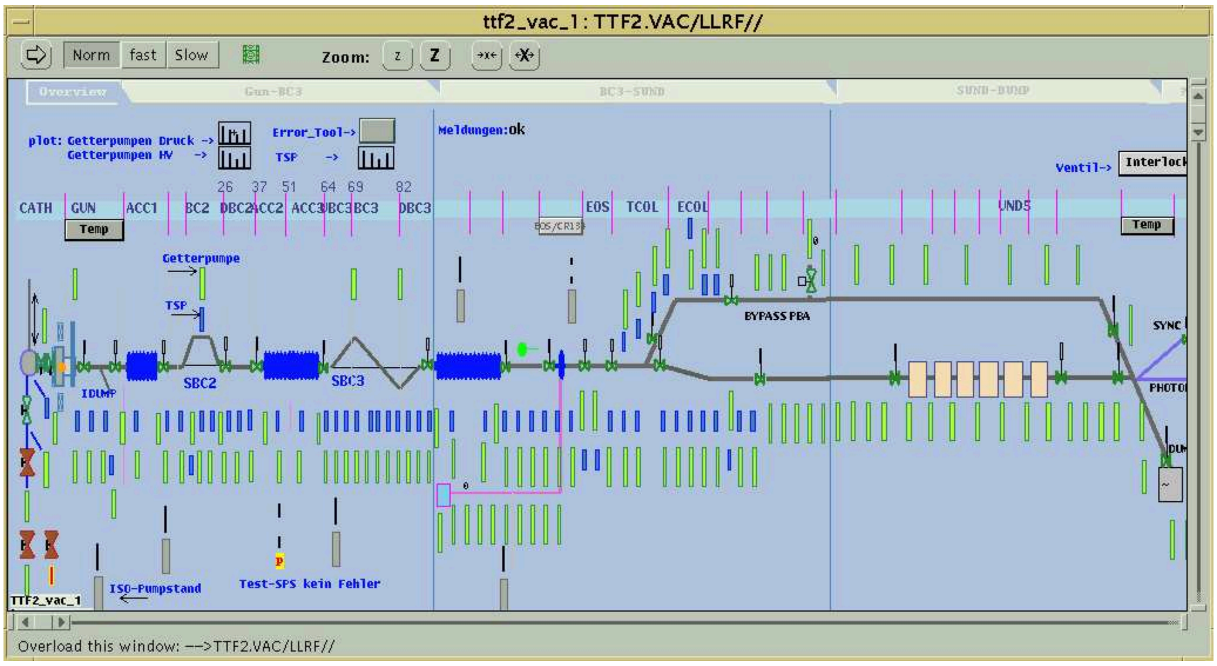


Figure 6: Virtual device panel with the part of the VUV FEL linac.

server. They are used to group servers including the system they belong (LLRF, magnets, cryo, etc), and subsystems (klystron 1, ACC1, ACC2, etc). *Location* is the name of the server instance and *property* is the particular property name. They are stored on the server side. The example address of the property can look like this:

```
TTF2.RF/SIMCON/CONTROLLER/VOLTAGE
```

From the address one can read that the property is a part of *controller* server which controls the *simcon* device, and this device belongs to RF system of TTF2. This semantic is very useful when the system is big, and there are many servers controlling different devices.

DOOCS is not a closed environment. It provides API for many engineering tools like Matlab, LabView, LabWindows and popular programming languages like C, C++, Java, Fortran. It has also tools and libraries for the integration with a different control systems like EPICS. Currently DOOCS environment is controlling over 100 servers and provides few thousands of properties. In DOOCS system many devices interacts with each other, exchange data, send control signals or monitor the work of the other systems. Therefore it is crucial to integrate every new device with the existing infrastructure. One of the currently developed devices, which is planned to be a part of the VUV FEL is SIMCON.

1.1 Cavity field controller and simulator - SIMCON

SIMCON (**SIM**ulator and **CON**troller) is the project of the fast, low latency digital controller dedicated for LLRF [7] system in VUV FEL experiment It is being developed by ELHEP [8] group in Institute of Electronic Systems at Warsaw University of Technology. The main purpose of the project is to create a controller for stabilizing the vector sum of fields in cavities of one cryo module in the experiment. The device can be also used as the simulator of the cavity and test bench for other devices.
SIMCON design is based on FPGA [9] technology which allows to create fast hardware devices inside the chip, dedicated for the particular purposes and therefore faster than currently used controllers based on DSP processors. FPGA technology offers also integrated peripherals for the fast communication (optical links) and calculations (embedded PPC or DSP). All these features create powerful platform for control system development. The flexibility of FPGA technology used in SIMCON makes this device multipurpose system which can perform many sophisticated algorithms and its capabilities are limited only by the board architecture (esp. the number of inputs and outputs).
The SIMCON as the device can be split into three main parts:

- **Hardware** - which is the board with FPGA chips, ADCs and DACs, hardware interfaces (VME slots, ethernet sockets etc.). Several versions of the SIMCON board have been developed yet and is still being developed. They have different capabilities, different architecture, number of inputs and outputs, etc.

- **Firmware**. Every logic inside the FPGA chip is described with VHDL (Very High speed integrated circuits Definition Language) code. FPGA chip itself does not include any "ready to use" logic [10] so it must be defined. In case of SIMCON the firmware includes

[7] Low Level Radio Frequency
[8] Electronics for High Energy Physics
[9] Field-Programmable Gate Array
[10] Some new FPGA chips (like Xilinx Virtex II pro) have embedded processors and other sub-components implemented inside, however these components are useless until the FPGA is configured

components for communication with other chips on the board as well as fast control algorithms described in VHDL. Firmware can be developed in the way that it can run on different boards. The modularity of VHDL code offers the possibility to split the project into small pieces and work separately on particular tasks, and then join the components into one final project.

- **Software**. The main algorithm development process for SIMCON is done using MATLAB environment. This advanced tool offers the easy way of manipulating the matrices and vectors which are widely used in control algorithms. Also the most of tests of firmware are done using MATLAB. To provide direct communication with the hardware from MATLAB, the dedicated laboratory software environment has been developed [1]. It bases on Internal Interface described in [3]. In the Internal Interface registers, bits and memory in FPGA are accessible from software level not through physical addresses, but through mnemonics (Fig. 7). A dedicated library loads at the application start up the IID file (Internal Interface Description) with declarations of mnemonics, its data types, size etc. Then it calculates the physical addresses of these mnemonics in the FPGA memory. The same file is used in the VHDL project to synthesize the structure of the FPGA using the same address calculation algorithm as used in the software library. This ensures the proper device addressing. The software library that is responsible for providing the communication based on mnemonics make use of another library to perform read or write operation based on translated addresses. There are many ways to connect the SIMCON boards to the control computer (VME bus, LPT, Ethernet, etc.), therefore, to keep the modularity of the system, a communication channel concept has been applied to the system (Fig. 7). In this concept, the II library uses a configuration file to read the name of the channel library. Then it loads the library and uses the unified interface to provide the communication with the hardware. The control software is independent from the channel implementation. There is no need to recompile the software if the channel changes. Only the configuration file of the II library has to be changed in order to use the new channel. The limitations of MATLAB reduces the usage of this environment only to the laboratory purposes. The Matlab provides only single-threaded API for the user. Therefore the

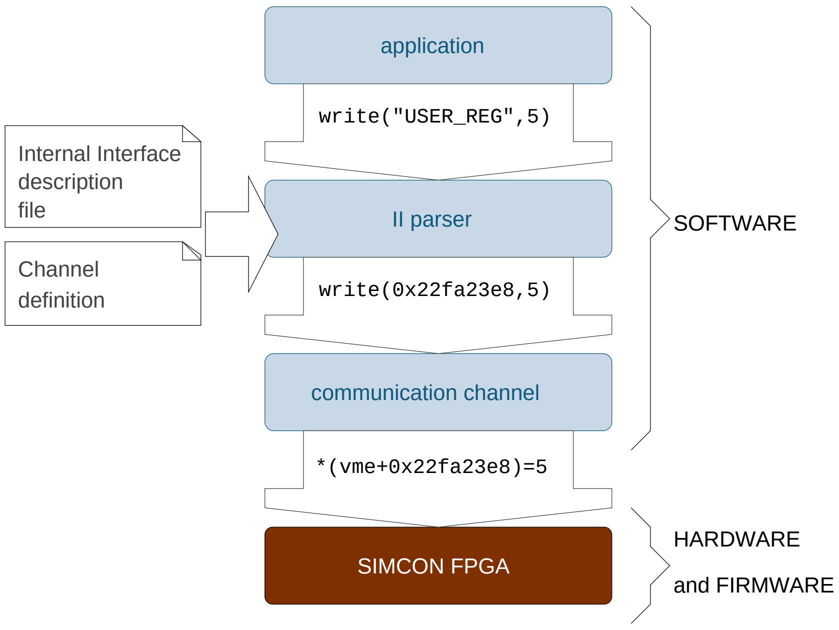


Figure 7: SIMCON mnemonic to address translation process using Internal Interface.

Graphical interface runs in the same thread as the rest of the application. This causes performance degradation of the system and makes it almost useless for testing fast, time critical algorithms.

1.1.1 SIMCON 3.0

The current version of SIMCON (3.0) (Fig.8) was designed for controlling the vector sum of fields in one cryo-module (8 cavities). The main features of the board are:

- Xilinx Virtex II chip.
- Eight 14bit ADCs (50 - 100 Msps) and four 14bit DACs (40 - 160Msps).
- 2 inputs for external clock and trigger signals (these signals can be also generated internally inside FPGA).
- 2 outputs for providing the clock and trigger signal (if they are generated inside the FPGA).

- Modular design (the SIMCON board is made as the daughter-board and is placed on the motherboard with VME interface).

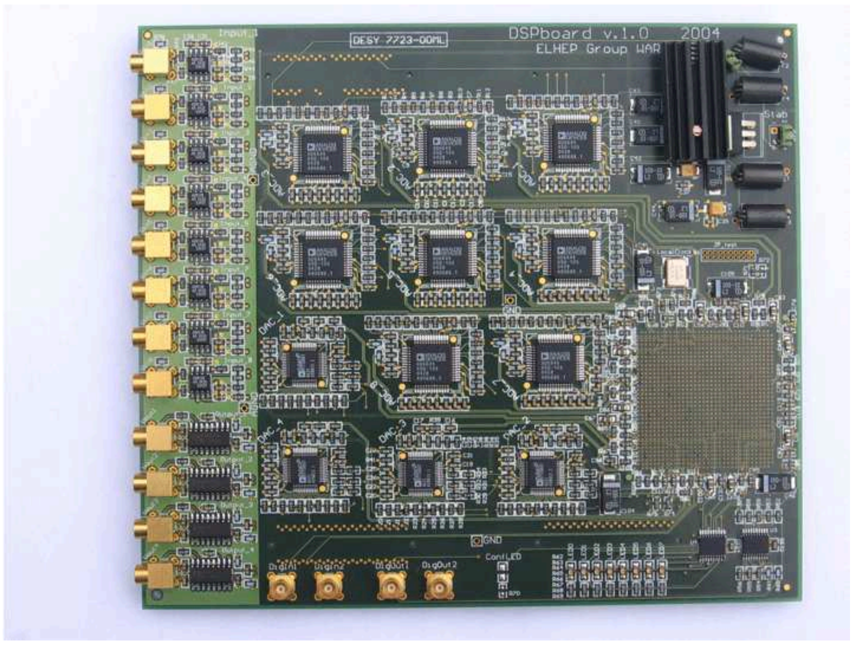


Figure 8: SIMCON 3.0 board with 8 ADCs and 4 DACs. It has Xilinx Virtex II chip on-board.

For the SIMCON 3.0 several firmware versions have been prepared. The main purpose of the system is vector sum regulation in superconducting module, but the board can be also used (with the appropriate firmware) as the controller for the Radio Frequency Electron GUN field stabilization (with normal conducting cavity).

The 8 cavities controller algorithm diagram has been shown in figure 9 and the detailed information about its implementation can be found in [1] [2]. The main features of the algorithm are:

- Programmable FeedForward, Feedback and SetPoint tables.
- I and Q detection.
- rotation matrices for the each input and output channel.

- Proportional feedback.
- Exception handling.

The algorithm implemented in SIMCON 3.0 provides a big number of internal signals which can be access through Internal Interface. The speed of the SIMCON as a complete device (including board and FPGA chip latency) is less than 500ns. This operation speed together with the high pulse repetition rate (5 - 10Hz) causes generation of many, high rated data which should be available to the user on-line. The existing MATLAB based environment is not sufficient for the real-time device operation. The next chapters describe requirements, concept and implementation of the software system for controlling SIMCON device in operation environment.

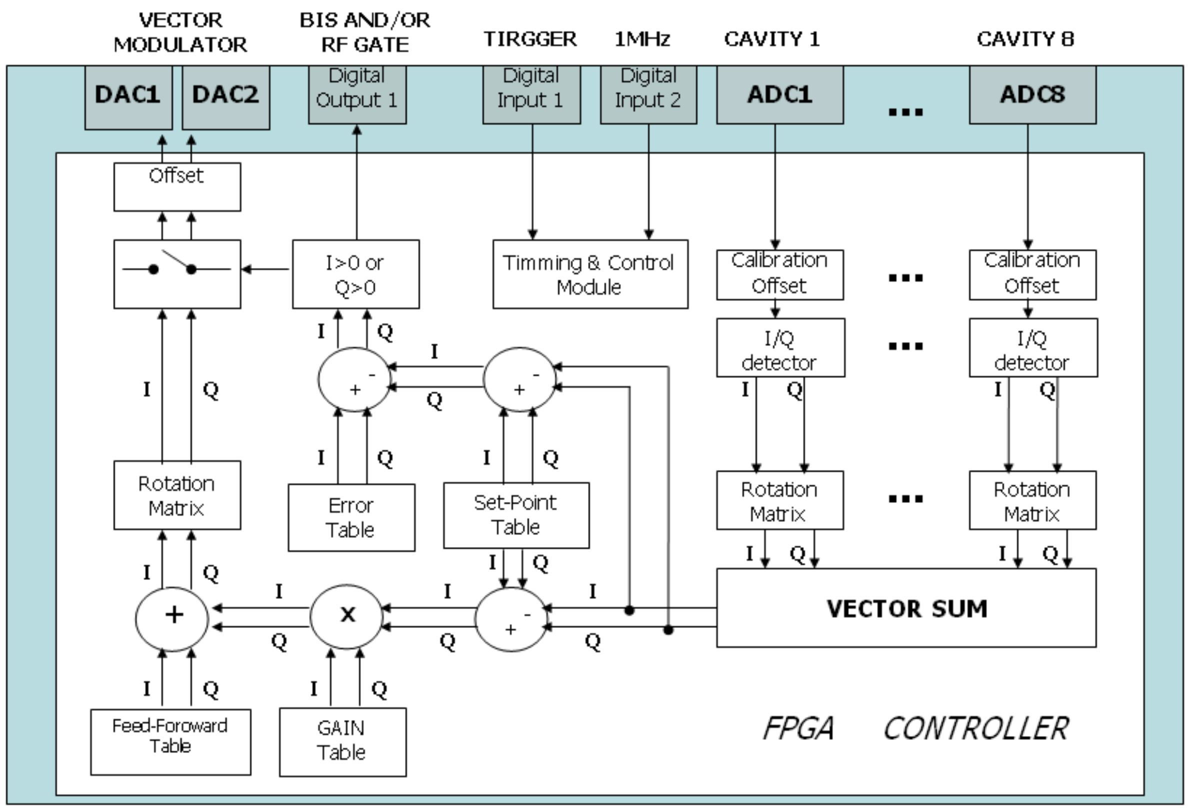


Figure 9: SIMCON 3.0.2 control algorithm scheme.

2 System requirements

The complexity of SIMCON , especially the flexibility of the firmware and the number of operation and configuration parameters of the device make the hardware almost unusable without appropriate control software support. The fast progress of the algorithm development and implementation in SIMCON board requires the unified solution which can be easily adapted to the particular need. The SIMCON must also be compatible on every design level (hardware, firmware and software) with the existing experiment infrastructure for the proper integration. As it was [resented in the introduction, all VUV FEL systems and its devices have to be controlled through DOOCS. This is the requirement which cannot be fulfilled by Matlab based control system.

The aim of the thesis is the design and implementation of the software control system for SIMCON controller. Additional requirements for the project are:

- DOOCS based design.
- Easy integration with the experiment.
- Flexibility in integration with different firmware versions.
- Control software must provide full functionality of the SIMCON device.
- The system must be expandable.

3 The control software concept for SIMCON 3.0

The designed system is based the concept of autonomic and extendable modules connected by well defined, unified interfaces. Some interfaces and modules described in this chapter have been only adapted from the existing MATLAB-based environment [[1]] for DOOCS requirements. This solution allow to use the same core elements of the system in these two (DOOCS and MATLAB) environments and simplify the software development process. The next chapter presents the concept and the architecture of the control system for SIMCON with a focus on the modules design and interaction between them.

The version 3 of SIMCON [4] provides more than just the possibility to control the cavity field using implemented algorithm, but also offers possibility to elaborate new control algorithms. In order to do this, the programmer must have many possibilities for integrating the algorithm with SIMCON. The general concept of the system has been presented in the figure 10. The whole system consists of three main parts:

- **Client applications**. DOOCS provides APIs for many programming languages, engineering applications and includes its own GUI tool - DDD. Every client can use the same, well defined interface consisting of the set of DOOCS properties and control the device using GUI panels or command line tools. For the SIMCON 3.0 needs a dedicated DDD panels have been created (described further in paper).

- **DOOCS server** is the core of the designed system. This is the only application which should have the direct access to the hardware. It consists of three main operation modules and one communication module (described in details below). Each operation module provides a part of DOOCS user interface. The communication module is the gateway through which operation modules can access the hardware.

- **SIMCON board**. This is also the part of designed software system, because from the software point of view the hardware is a module connected through ta defined interface which interacts with the rest of the system. DOOCS server cannot work without the hardware.

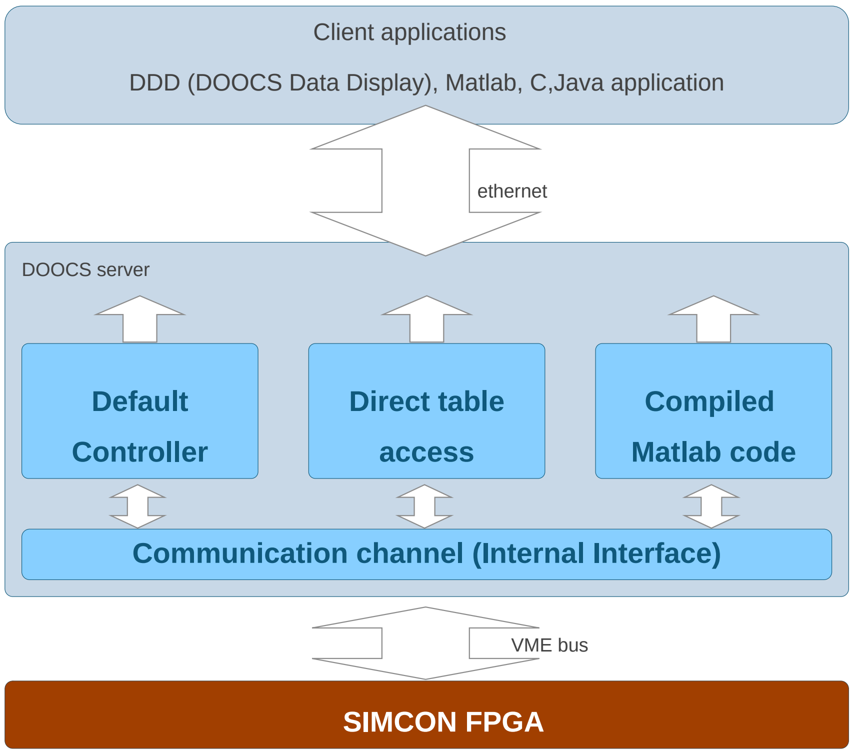


Figure 10: General concept of the DOOCS based control system for SIMCON 3.0.

3.1 Communication module

The communication module is based on the Internal Interface [3] and originally is used in Matlab control software [1]. The module provides the unified interface to access all elements of II declared in the IID file. It consists of two sub-modules: II engine - which translates the mnemonic names into addresses and channel module which implements the particular hardware bus access method using channel interface [1]. The communication module does not include any logic related to the specific algorithm or firmware version. It is only the bridge with translation engine for accessing hardware logic elements (bits, registers, memory).

The main feature of the DOOCS system is the distributed access to the server. Therefore the server must accept request from many client at the same time. In the DOOCS server, unlike in the Matlab control environment, every request is served as a separate, asynchronus thread. If two clients want to perform two procedures which use the hardware, and if these procedures uses the same registers or bits (Fig. 11) the application may set the SIMCON in the mode that will interrupt its operation or provide wrong data which is not permitted during the experiment. Therefore the communication module must also provide mechanism for blocking the access to the hardware. One module can block the access to the device for the time it is performing necessary operations. After that time the module can unblock the device. When the device is blocked by one module, another module which is requesting the access should wait until the access will be granted. The system provides the way for queuing requests from server to communication channel (see *implementation* chapter for details).

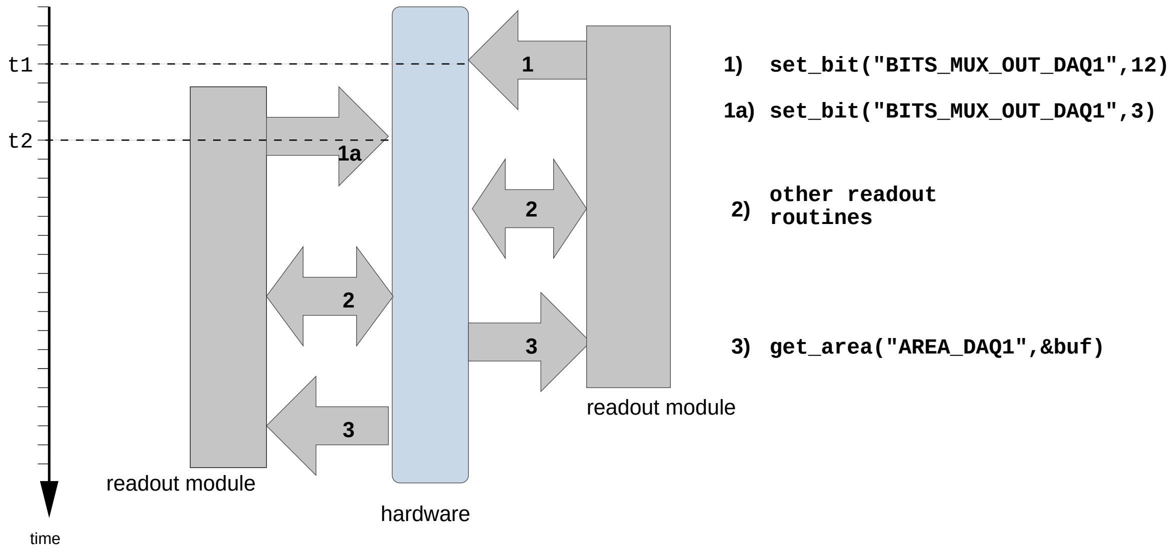

t1
t2
time
1
1a
2
2
3
3
readout module
readout module
hardware
1) set_bit("BITS_MUX_OUT_DAQ1",12)
1a) set_bit("BITS_MUX_OUT_DAQ1",3)
2) other readout routines
3) get_area("AREA_DAQ1",&buf)

Figure 11: Simplified time diagram of two readout procedures performed at the same time in the system without device access lock. The left readout module was triggered before the right one. The right one sets the number of internal SIMCON signal it wants to read by setting the signal number in bit "BITS_MUX_OUT_DAQ". After a while the left module also starts the readout procedure and set the same DAQ bit to 3 and changes the settings of first module. Both modules perform other routines and at the end both perform the readout from DAQ block. Both modules will read the signal number 3, because the settings were overwritten by left module. Usage of blocked access to the hardware could prevent such situation.

3.2 Module for direct control tables access

Control algorithms implemented in VHDL have many operation and configuration parameters (Fig. 9.). In the current version of the firmware, the algorithm does not calculate any parameters or control tables itself. This must be done in the software and than loaded into FPGA. The most important parameters are control tables. The *direct table access* module provides the DOOCS interface for the control tables implemented in the FPGA. Control tables in the FPGA can be accessed in read/write mode, while normal tables can be only accessed in read-only mode. This module is firmware-dependent, because different firmware can provide different tables. In the current firmware version of SIMCON firmware, one can have access in read/write mode the following controller tables:

- Two setpoint tables - for I and Q component separately.
- Two feed forward tables - for I and Q component separately.
- Two gain tables - for I and Q component separately.
- Two error tables for exception handling algorithm (I and Q component separately).

The rest of tables are available in read only mode - since they represent the internal algorithm signals (see appendix for full table list). The module takes care of blocking the device for the time, the readout or write operation is performed. The important issue of the direct control concept is the proper procedure of new tables upload. From the FPGA point of view, new tables are switched between pulses which ensures the proper functionality of the controller. The SIMCON switches always tables in pairs - I and Q component at the same time. As it was mentioned before, every request to the DOOCS server for updating the data is realized as a separate thread. On the client site DOOCS API does not offers the possibility to update two properties at the same time. Therefore, setting the I and Q components is done at the client site as two RPC calls. In the designed module, tables are also grouped in pairs (Fig. 12). One of the tables is called trigger table. It triggers the procedure of switching the tables in the hardware. Programmer should always execute the update of the DOOCS property corresponding to the trigger table as a second RPC call. There is a general rule in the designed system which sais:

If two associated tables (tables which are switched together in the FPGA) correspond to I and Q components of the signal, the trigger table is always the "Q table". In other cases the trigger table is always the table with the larger position number on the signal list (see appendix for the signal list).

This rule will ensure, that the controller will be updated properly. The buffered tables updated by client (Fig. 12) are loaded into the hardware not in parallel mode but also in serial mode. In the FPGA there is a mechanism for switching tables which is triggered by setting the appropriate bit. Server loads new data into the backup tables in FPGA. when data are ready, server sets the switch bit, and the FPGA switches itself to use the backup as a operation tables while the previously used tables become new backup tables for the next exchange. From the server point of view there is only one set of tables. FPGA takes care of writing new data to actual backup tables and not to currently used.

The direct control table access can be used especially through Matlab, where one can generate its own setpiont or feed forward tables and than load them into FPGA. Using this interface one can perform slow feedback regulation, or just collect data for later analysis.

3.3 Module for default control tables generation

Using only the direct control tables access one must provide third party software for tables calculation and setting algorithm parameters. After running DOOCS server, the system would not be ready for SIMCON operation. Therefore an addition module has been designed in order to enable device operation from the start-up of the server. *Default control tables generation module* includes simple algorithms for control tables calculation and allows to operate the SIMCON with its all features.The general diagram of the *Default control tables generator* has been presented in figure 13. There are three main layers in the scheme:

1. **First order parameters layer** - which correspond to parameters adjusted by user ie.: amplitude and phase of the setpoint, gain of the input signal of the ADC, etc. Some of these parameters have the exact equivalents in FPGA (green blocks), some of them are

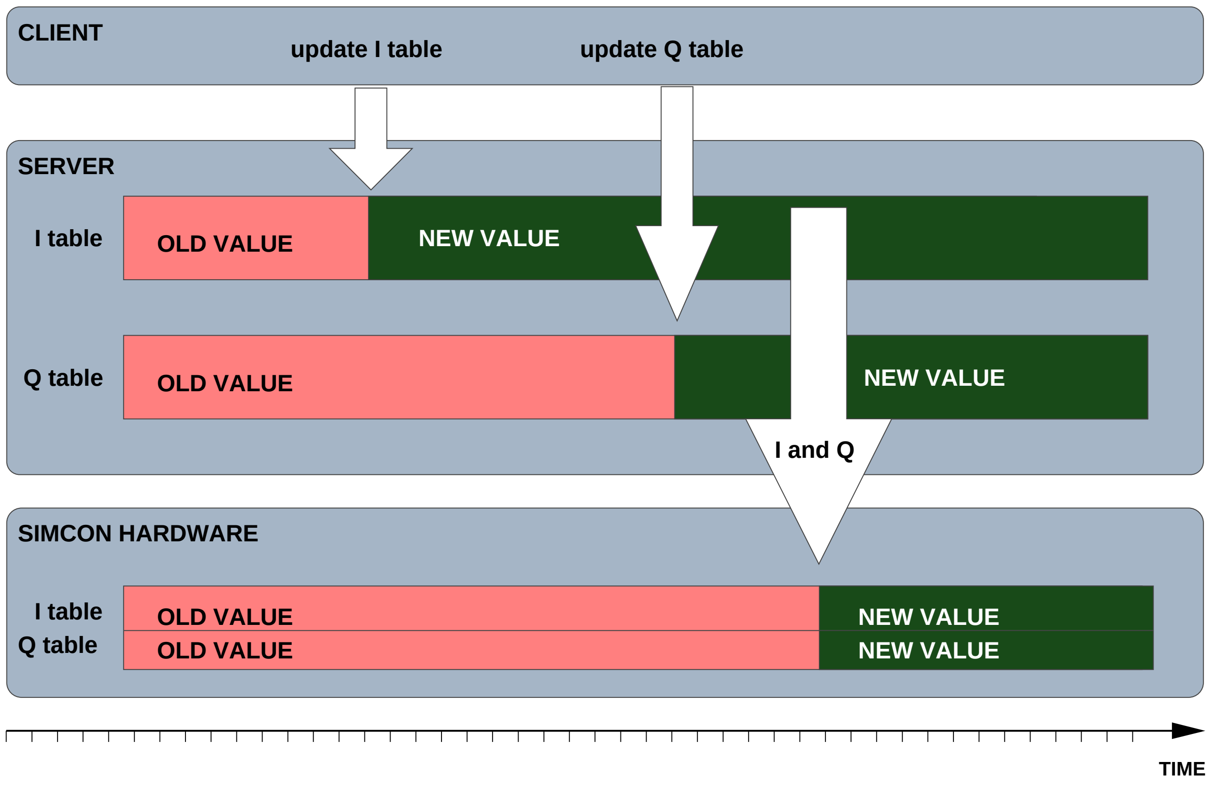


Figure 12: Triggering of the tables switching process in the *direct control tables access module*. When the first table is updated by the client, the server buffers the new data until the second, *trigger* table is updated. The update of the second property causes the loading of two tables into the FPGA and setting the switch bit which which tells the FPGA, that new tables are ready to use.

only initial parameters for further calculations and are not directly loaded into FPGA.

2. **Second order parameters** - are parameters, which are of size and type which can be loaded into SIMCON (green blocks).

3. **Hardware layer** with the FPGA representation of the algorithm register, bits and tables.

Black arrows in the picture show which *first order parameter* is used to calculate particular *second order parameter*. The diagram shows, that some of the *first order parameters* are directly loaded to FPGA (green blocks), while the rest (white blocks) is used for further calculation. This solution is caused by the limitations of FPGA firmware. In FPGA the implementation of particular operations like floating point operations, dividing (fixed and floating point) or calculating sine and cosine is difficult for implementation. The calculation of new control tables in most cases is not time critical. Therefore this calculation are done in the software. The goal of the *Default control tables generator* is to provide the basic, not the optimal field stabilization. Therefore the algorithm uses the same equations as used in existing DSP [5] based LLRF system. The complete list of user parameters used in the module has been presented in the table 1.

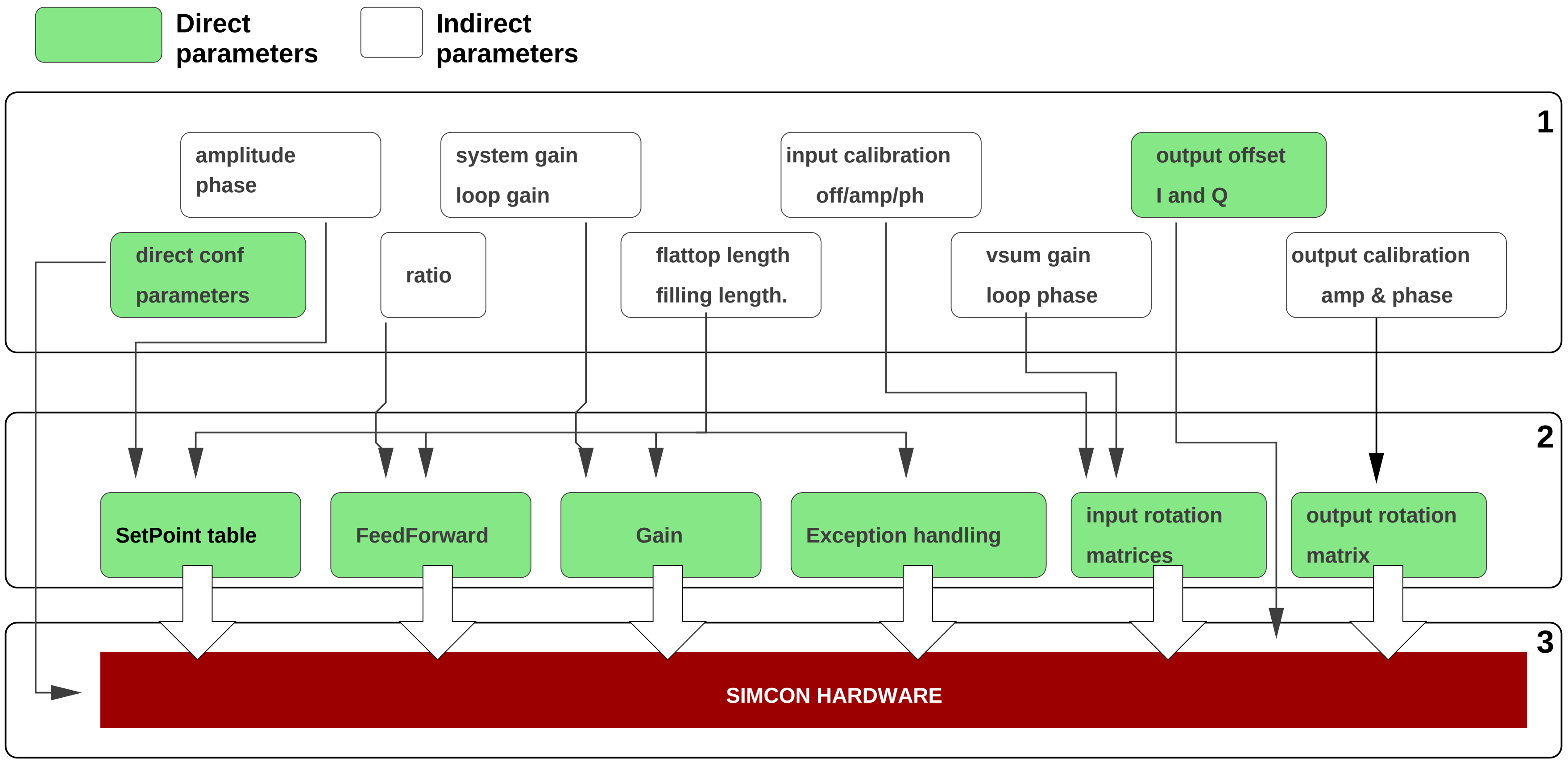


Figure 13: The general scheme of the default controller module. One can see three main layers with first and second order parameters. First order parameters are set directly by user. Second order parameters are calculated from the first order parameters and then loaded to FPGA. Black arrows show which first order paraemter is used to generate second order parameter.

Using the variables listed in table 1, The following equations have been applied to generate control tables:

1. **setpoint** tables are generated using the equation:

$$SP_I[n] = \begin{cases} A_{SP} * \frac{1-e^{\frac{n}{\tau}}}{1-e^{\frac{t_{fill}}{\tau}}} * cos(ph_{SP}) & \text{for } 0 < n < t_fill \\ A_{SP} * cos(ph_{SP}) & \text{for } t_{fill} < n < t_{fill} + t_{flatt} \\ 0 & \text{for } n > t_{fill} + t_{flatt} \end{cases} \tag{1}$$

$$SP_Q[n] = \begin{cases} A_{SP} * \frac{1-e^{\frac{n}{\tau}}}{1-e^{\frac{t_{fill}}{\tau}}} * sin(ph_{SP}) & \text{for } 0 < n < t_fill \\ A_{SP} * sin(ph_{SP}) & \text{for } t_{fill} < n < t_{fill} + t_{flatt} \\ 0 & \text{for } n > t_{fill} + t_{flatt} \end{cases} \tag{2}$$

   where SP_I is the setpoint table for I component, SP_Q is the setpoint table for Q component and n=0..2047 is sample number (1 sample = 1μs).

2. **feed forward** tables are generated using the equation:

$$FF_I[n] = \begin{cases} \frac{A_{SP}}{R_{FF}} * cos(ph_{SP}) & \text{for } 0 < n < t_fill \\ A_{SP} * cos(ph_{SP}) & \text{for } t_{fill} < n < t_{fill} + t_{flatt} \\ 0 & \text{for } n > t_{fill} + t_{flatt} \end{cases} \tag{3}$$

$$FF_Q[n] = \begin{cases} \frac{A_{SP}}{R_{FF}} * sin(ph_{SP}) & \text{for } 0 < n < t_fill \\ A_{SP} * sin(ph_{SP}) & \text{for } t_{fill} < n < t_{fill} + t_{flatt} \\ 0 & \text{for } n > t_{fill} + t_{flatt} \end{cases} \tag{4}$$

   where FF_I is the feed forward table for I component, FF_Q is the feed forward table for Q component and $n = 0..2047$ is sample number (1 sample = 1μs).

3. **gain** tables are generated using the equation:

$$G_{I,Q}[n] = \begin{cases} G_{SYS} * G_{LOOP} & \text{for } 0 < n < t_{fill} + t_{flatt} \\ 0 & \text{for } n > t_{fill} + t_{flatt} \end{cases} \tag{5}$$

   where $G_{I.Q}$ are Gain tables for I and Q components and $n = 0..2047$ is sample number (1 sample = 1μs).

Parameter name	**Unit**	**Description**
A_{SP}	MV	The setpoint amplitude.
ph_{SP}	$degree$	The setpoint phase.
R_{FF}	–	Ratio of "two steps" in the Feed Forward table.
G_{SYS}	–	System gain.
G_{LOOP}	–	Loop gain.
t_{fill}	μs	The filling time.
t_{flatt}	μs	The flaattop time.
$offset_{ADC}$	$samples$	Offset of the ADC channel (8 channels)
$gain_{ADC}$	–	Gain of the input rotation matrix (8 channels)
ph_{ADC}	$degree$	Phase of the input rotation matrix (8 channels)
$gain_{vsum}$	–	Gain multiplied with each $gain_{ADC}$ for scaling the whole vector sum.
$lphase_{vsum}$	$degree$	Phase added to each ph_{ADC} for rotating the whole vector sum.
out_offset_I	$samples$	The offset for the I output channel.
out_offset_Q	$samples$	The offset for the Q output channel.
$gain\_out$	–	The gain for output rotation matrix.
ph_out	$degree$	The phase for output rotation matrix.
τ	μs	Cavity time constant.

Table 1: List of variables used in default controller.

Each input channel in SIMCON has its own matrix rotation. It is implemented in the hardware as two registers ($c1, c2$) to which one must load values calculated with the following equation:

$$c1_m = gain_{ADC_m} * gain_{vsum} * cos(phase_{ADC_m} + lphase_{vsum}) \tag{6}$$

$$c2_m = gain_{ADC_m} * gain_{vsum} * sin(phase_{ADC_m} + lphase_{vsum}) \tag{7}$$

where m is the channel number (1 to 8).

The reason of this solution is the complexity of implementation of *cos* and *sin* functions in FPGA. In the future, this problem will be solved by using the floating point operation VHDL core. Similar equation has been used for calculating the output rotation matrix:

$$c1 = gain_{out} * cos(ph_{out}) \tag{8}$$

$$c2 = gain_{out} * sin(ph_{out}) \tag{9}$$

The offsets of the input and output channels have its direct equivalents in the IID [11] elements list so they are directly loaded into FPGA without any sophisticated equations [12] .

In addition to described user parameters, there are some configuration parameters for SIMCON, which are not directly related to the main algorithm, but are essential for the proper SIMCON operation. These parameters are:

- **FPGA mode**. One can set SIMCON in three main states: As a controller, as a simulator, and in the internal loop. Controller mode uses external intermediate frequency signals (8 channels) for controlling the vector sum. The simulator mode switches SIMCON into simulator of 8 cavities. In the internal loop, SIMCON is working as a controller and simulator - the controller drives simulated cavity.

- **Timing mode**. SIMCON can work with external timing and trigger, or can use internal clock and trigger.

[11] Internal Interface Description

[12] The only conversion is the default conversion to binary coding and extending the range to 18 bits - see implementation chapter for details.

- **Sample index**. Currently on SIMCON 3.0 board ADCs are sampling the input signal with frequency of 40MHz. In the algorithm samples have $1\mu s$ resolution, it means that from sampled signal only the every 40th sample is taken for further calculation. This parameter determines which sample out of these 40 should be taken to the algorithm.

- **samples averaging**. Because ADCs samples the signal much faster than it is needed, the is possibility to average few samples and in order to eliminate noises. This parameter tells, which number of samples should be taken to the averaging. filter.

- **Trigger delay**. SIMCON can delay the start of the whole algorithm by number of microseconds from the beginning of the trigger.

The presented default control tables generator module is only the example of possible solution. Due to modularity of the system, one can easily replace this algorithm with its own implementation. The disadvantage of this solution is that algorithm, which is in most cases developed in Matlab, must be rewritten to C/C++ code. Sometimes it is very complex process. For this case, software system for SIMCON has been equipped with the next module, which allows to include compiled Matlab code into the DOOCS server.

3.4 Module for compiled Matlab code

Matlab offers to the user sophisticated tools and language for engineering calculations and algorithm development. In order to use developed algorithm later in DOOCS, it must be programmed in C/C++. Simple routines can be rewritten by hand. Complex calculations may be impossible to rewrite in a short time, without ready C libraries or additional tools. Matlab offers a compiler which can compile m-functions into C/C++ source code, dynamic library or executable application. In all these cases in order to use the compiled code, user must provide access to specific Matlab libraries.

In most cases, during the algorithms development, one can access the SIMCON hardware through Matlab using the laboratory control system [1]. After initial tests using Matlab, there is a need to either implement ready algorithm in the FPGA or in DOOCS. In this second case one can compile Matlab code and use it as a module inside DOOCS server.

Moving algorithms from Matlab environment into DOOCS is not only the matter of compiling the Matlab code. Those two environments have different data types, data flow and the architecture. DOOCS server unlike the Matlab (from the programmer point of view) is a multithreaded application. It uses the communication module for accessing the SIMCON hardware. In Matlab one can access directly the particular register in FPGA from the body of the m-function or script. This is not permitted in the presented DOOCS system. Figure 14 presents the model of the SIMCON hardware access used in Matlab and DOOCS. In the first solution every Matlab routine can use inside its body functions to access the hardware. Since Matlab scripts cannot be run in the multithreaded environment there is a guarantee that one Matlab script will not be interrupted by another script during running the procedure requiring multiple accesses to the device. In DOOCS solution there is an arbitrary unit through which all calls to the hardware should be done. Compiled Matlab function which is put into the DOOCS server and contain direct calls to the device is not thread safe. It can be called by many clients at the same time so many instances of this function will run at the same time and cause controller algorithm crash. The second problem is the lack of Matlab workspace inside DOOCS environ-

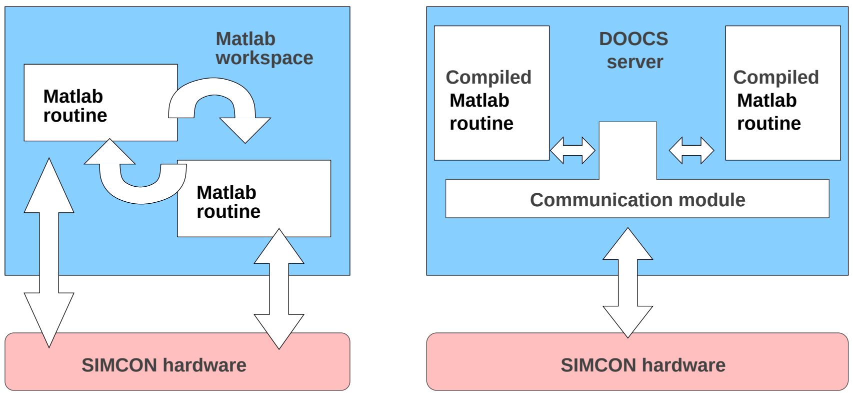


Figure 14: Matlab hardware access model versus DOOCS hardware access model.

ment. If function uses in Matlab global variables, they will not work in DOOCS . All function parameters must be passed in the function invocation. In order to use Matlab code in DOOCS one must fulfill the following requirements:

1. Every routine must be programmed as a m-function, not m-script. Matlab cannot compile m-scripts.

2. Access to the hardware must be done between functions calls. That means, there should be one root function of algorithm in which particular parts of calculations are performed and the data exchange with SIMCON is done on the root function level.

3. Any global variables are not permitted.

4. All functions must be constructed according to the interface described below.

3.4.1 Matlab - DOOCS interface

The Matlab module for DOOCS provides to the system an interface, through which compiled Matlab function can communicate with the rest of the DOOCS system. The purpose of this unification is to make the changes of the server as simple as it is possible. Interface bases on the firmware structure. It means, it must be changed if the firmware changes (which happens relatively seldom compering with the changes made in the software). The interface should be applied in Matlab m-function so after compilation the library will be easily integrable with server.

The idea of the interface is to limit the number of possible m-functions structures in Matlab and create inside the designed module a persistence environment which will simulate Matlab environment. There are free kinds of functions in the proposed interface (Fig 15) and four matrices.

1. **Init functions**. Generally for each algorithm there should be only one init function. This function is called only once by the DOOCS server at the very start up stage. Its goal is to return the special structure (see implementation chapter for details) with all algorithm parameters that can be adjusted by the user. Matlab module for DOOCS scans the structure returned by this function and automatically creates the DOOCS properties for the entire algorithm. If the properties in the structure have some initial values assigned, they are also loaded into DOOCS.

2. **Input functions**. Input function is triggered every time the user changes any of DOOCS property related to the compiled Matlab algorithm. As the output it returns three matrices: First is the U which represents the scalar parameters calculated for controller and

simulator. The second is Y which represents the table of vectors. Each vector is a single control table (FF, SP, etc). The third one is W - the virtual workspace, a structure which contain other data using in algorithm.

3. **Internal functions**. This function does not use directly user input data but performs calculations on data red from the hardware or workspace. As an input arguments it requires Y, and W and returns also these variables.

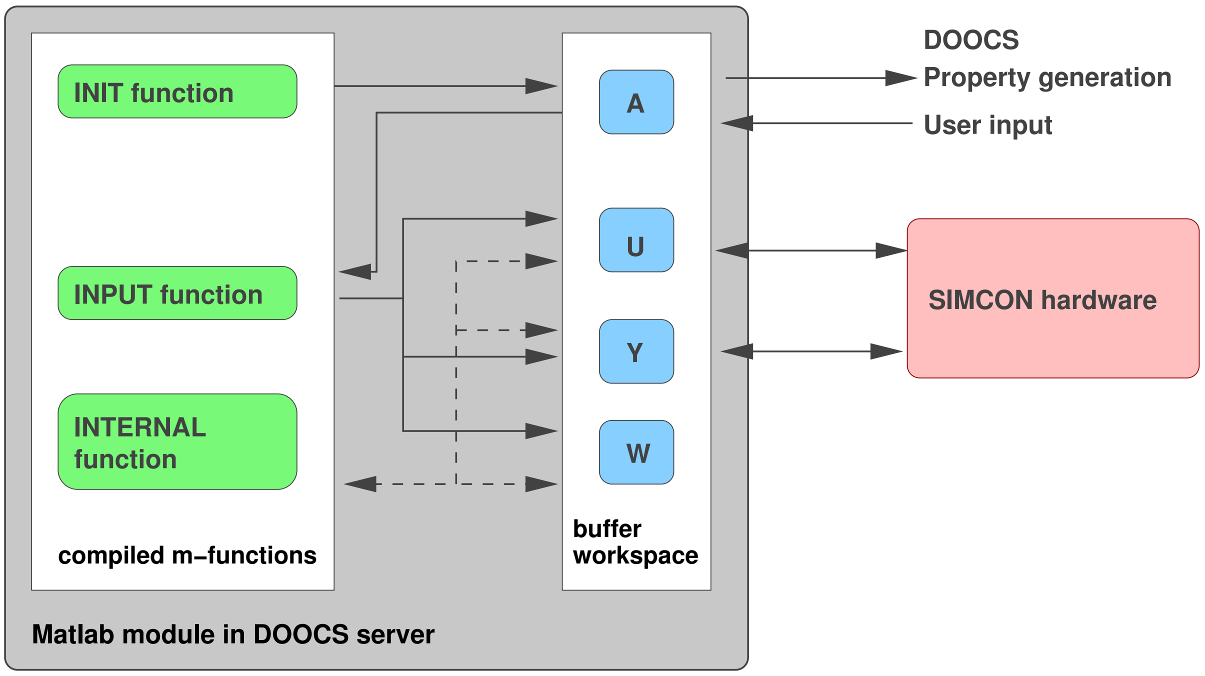


Figure 15: Matlab hardware access model versus DOOCS hardware access model.

The most important element of the module is the set of matrices and structures used as a buffer between Matlab and the rest of the system. The specification of these data structures are presented below.

- A - Matlab structure. Created dynamically during the start up of the server. It includes the list of parameters describing the algorithm. It is algorithm dependent strucutre, not firmware dependent. The system does know the content of this structure and can modify values of fields.

- U, Y - Matlab matrices with size and structure. These matrices are firmware dependent because represents the set of control parameters used in VHDL routines. When internal function is called, the Y matrix is updated - it is filled with actual data red from hardware. When the input function or internal function returns the actual Y and U matrices, they are immediately loaded into SIMCON. These two elements are therefore the real buffer between Matlab and SIMCON hardware.
- W - simulate the workspace, through which matlab functions can exchange data. It is the matlab structure. The DOOCS system does not know the content of the structure and cannot access it. It only keeps the persistent object in the memory. The algorithm can store there temporary data which are used by many algorithm functions, but does not corresponds to hardware properties.

Presented solution unify the way the algorithms should be developed in th Matlab and helps to automate the process of integration the Matlab code with DOOCS server. Additionally it separates these two environments, and simplify the design of the presented system. In cases, where C implementation of the algorithm is for some reasons impossible it can be the only solution. There can be multiple Matlab modules in one DOOCS server. The usage of one communication module ensures the thread safety.

3.5 Summary

The presented conception of the system covers all requirements presented in the system requirements chapter. The communication module controls the access to the hardware. It is crucial, that all modules (this presented in thesis and developed in the future) use this interface. Direct access to the control tables let the engineers to perform algorithm development or diagnostic measurements of the LLRF system. Default control tables generator makes the whole SIMCON an autonomic device, which can start immediately the operation without any third part tools. In the future version of the SIMCON, DOOCS will run on the embedded Power PC processor. The default control tables generator in that case can make the SIMCON a standalone device, which can run without VME crate, on the desk of the engineer. Finally the module for Matlab code integration is a solution for the cases, where sophisticated algorithm must be integrated with DOOCS, and there is no possibility to implement it in C language.

4 Implementation

Presented concept has been implemented according to the requirements of the system and limitations of DOOCS, Matlab and Internal Interface environments. A set of C++ classes has been developed for each operation module. Different data types and structures have been wrapped with DOOCS classes in order to provide them to the user. The focus has been put on the thread safe access to modules and effective DOOCS basic classes usage.

4.1 Data formats in the designed system

In the realized system, one can define three different software environment which interacts with each other. These are DOOCS system, Matlab code, and Internal Interface together with FPGA. All of them use different data types and formats. Detail specification of these types has been presented below:

1. **Matlab**. Matlab provides its own C API for compiled code. In Matlab one can distinguish between scalars, vector, matrices (multidimensional), structures , etc. Additionally it operates on complex numbers. In C language there is no such data structures. Therefore Matlab provides set of functions and structures, through which one can communicate with compiled Matlab code. There are defined functions for calling compiled functions, operating in return values, modifying them, etc. Natively Matlab operates on $double$ format (64 bit). In C code also the basic Matlab data format is $double$.

2. **DOOCS** has been written in C and C++ language. it operates mainly on $integer$ and $float$ format. DOOCS has various data types for representing different data structures. The main feature of these types is the availability through the network. All data types are C++ classes with virtual methods. Programmer has to overload these methods in order to provide its own data representation. Due to object representation of the scalar or vector values the access to the value is done through appropriate method, not by direct addressing.

3. **Internal Interface**. As it was described in the concept chapter, DOOCS server for SIMCON access the hardware using mnemonic names of the Internal Interface elements.

These can be: bits, words, and areas. Due to the unique capabilities of the FPGA, word can have different width then it is assumed in computer architecture. SIMCON operates on integer, 18 bits words. It is causes by built-in, 18 bit DSP blocks inside FPGA dedicated for fast calculations. Every single word in the server side, no matter if it is a scalar or element of a vector, is before loading to FPGA converted from U2 format to unsigned format. The reverse conversion is done on every word red from FPGA.

Presented data types differ on both levels; data structure and binary format. The important issue of the implementation is data conversion between these three environment. Next chapters presents implementaion realted details of the designed modules and DOOCS server.

4.2 Communication module

In the communication module libraries, which are originally used in Matlab, have been adapted for the DOOCS needs. Additionally in DOOCS, a dedicated C++ classes have been developed. The layout of the entire module has been presented in figure 16.

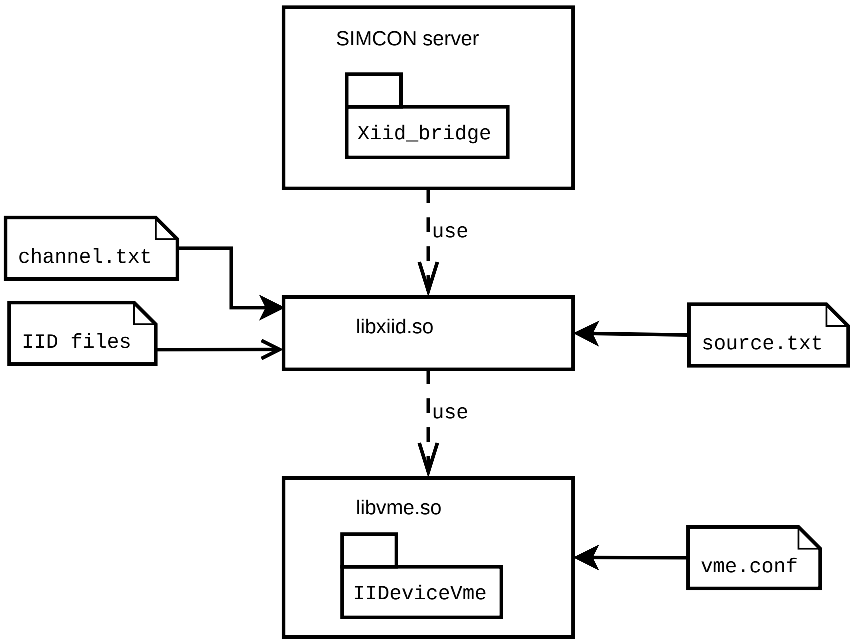


Figure 16: Layout of the communication channel implementation for DOOCS based SIMCON server.

In the DOOCS server, the communication module is implemented as a `Xiid_bridge` class.

It provides the interface for writing and reading to the every II element available in the `libxiid.so` library. This library is loaded by the `Xiid_bridge` class constructor. One can also obtain from the library the full list of mnemonics and its types. `Xiid_bridge` class provides also two methods for locking and unlocking the access to the hardware. Detailed description of the available methods has been presented in the table 2.

`Xiid_bridge` class uses `libxiid.so` library for translating the mnemonic names into physical addresses in FPGA. The library uses two configuration files. First is the `channel.txt` file which contains only one line with the name (with path) of the library with channel implementation. The `libxiid.so` opens directly pointed library and maps appropriate functions. The second configuration file is `source.txt` with names of all IID file that should be parsed by the library. The `libxiid.so` library communicates with the channel library using simple interface. There are two function defined for accessing the hardware:

```
write_data(const unsigned int addr, const unsigned long data)
read_data(const unsigned int addr, unsigned long * const data)
```

These functions allow to write or read from the hardware the single word. The IID parsing engine translates the mnemonic names to the physical addresses, but according to the II specification [3] data in the FPGA memory can be placed not continuously. Therefore one access to the hardware using mnemonic can cause many accesses on the channel level. The channel library `libvme.so` is responsible for direct accessing the VME bus using the system driver. It uses a configuration file `vme.conf` placed in the same directory as the library itself. The file contains three lines with configuration parameters which are:
`[base address]` - the base address of the device in the VME memory space.
`[memory size]` - the size of memory the device uses in order to map it into computer memory.
`[driver name]` - the VME bus is visible in SUN as a device placed in `/dev/` directory. In the name of the device the address width and data width is included. For example 24d32 means that the address has width of 24 bits and data are 32 bits.
The example configuration for SIMCON can look like this:

```
0xC00000
```

Method declaration	**Description**
`Xiid_bridge()`	Class constructor. Opens the `libxiid.so` library, maps appropriate functions.
`setWord(int id,u_long value)`	Writes word value to mnemonic with given id.
`getWord(int id,u_long *value)`	Reads data from mnemonic with given id and word type and store it in `value`.
`seBit(int id,u_long value)`	Writes value to bit element with given id.
`getBit(int id,u_long *value)`	Reads the value of bit element with given id.
`setArea(int id, u_long *buf,int num, int offs)`	Writes to the area with given `id` the `num` of words from buffer `buf`, starting from `off`.
`getArea(int id, u_long *buf,int num, int offs)`	Reads from the area with given `id` the `num` of words starting from `off` and stores it in buffer `buf`.
`int getID(char* name)`	Returns the numerical id of the element with given `mnemonic`.
`int getItemsNumber()`	Returns the number of all items in the parsed IID file
`getItem(int id, iid_item _t * item)`	Returns the structure describing the II element with given id
`lock()`	Locks the access to the hardware. If the hardware is already locked it the method waits until it is unblocked
`unlock()`	Release the access to the hardware.

Table 2: List of methods available in Xiid_bridge class.

```
0x1C000
/dev/24d32
```

The implementation of the communication module allows to use it in different servers just by including the `Xiid_bridge` class in the source code and copying appropriate libraries and configuration files.

4.3 Direct control table access

As the base type for control tables implementation in DOOCS, the `D_spectrum` class has been chosen. For every of the four tables sets (feed forward, setpoint, gain and exception) The appropriate class has been designed. Since the implementation of these tables is similar, the detailed description has been presented for `feed forward` class. For every main class (like FForward class) there are some helper classes assigned to (Fig 17). They base on D_spectrum class. The `feed forward` class has two instances of D_FForward_spectrum class (one for I table and one for Q table) and two instances of D_fforward_spectrum_ap class (one for representation of the amplitude spectrum and one for representation of phase spectrum).

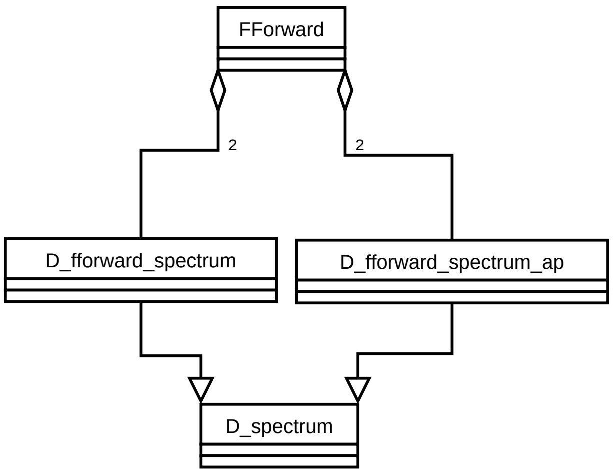


Figure 17: General UML scheme classes used in direct control table access module in SIMCON server.

As it was presented in the concept chapter, control tables have to be exchange in the hardware in at the same time and the software update process is performed in serial mode not in parallel. Therefore, only the main FForward class access the hardware (Fig 18). It includes four buffer tables for I, Q , amplitude and phase signals, and the helper classes access only these buffers.

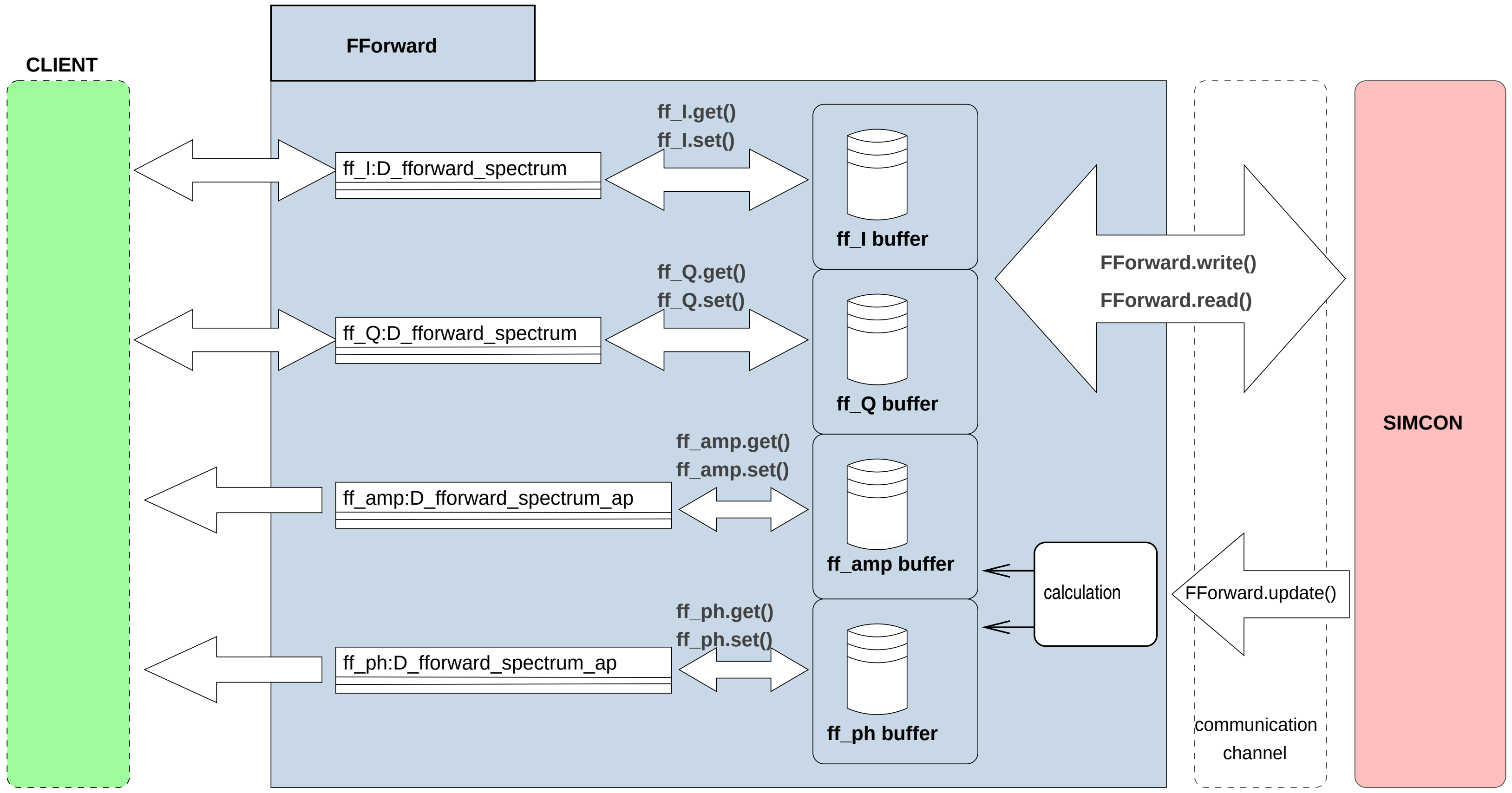


Figure 18: The general scheme of the direct control tables access module implementation.

During the initialization of the FForward class, the instances of helper classes (automatically adds the new DOOCS properties to the server) are also created and the FForward class constructor sets the trigger bit in one of them (according to the rule mentioned in the concept chapter). The whole process of updating new tables has been presented in figure 19. The client send a RPC call with new table values to the server. Server creates an executive thread. The thread triggers a method of `D_fforward_spectrum` class to set new data. The instance of `D_fforward_spectrum` class writes a new data to the buffer of `FForward` class. Next, the instance checks internal bit that decide if the instance is the trigger class or not. If it is the trigger table, according to the rule presented in the concept chapter, it triggers the `write` method of FForward class, and then FForward class updates two tables inside the FPGA.

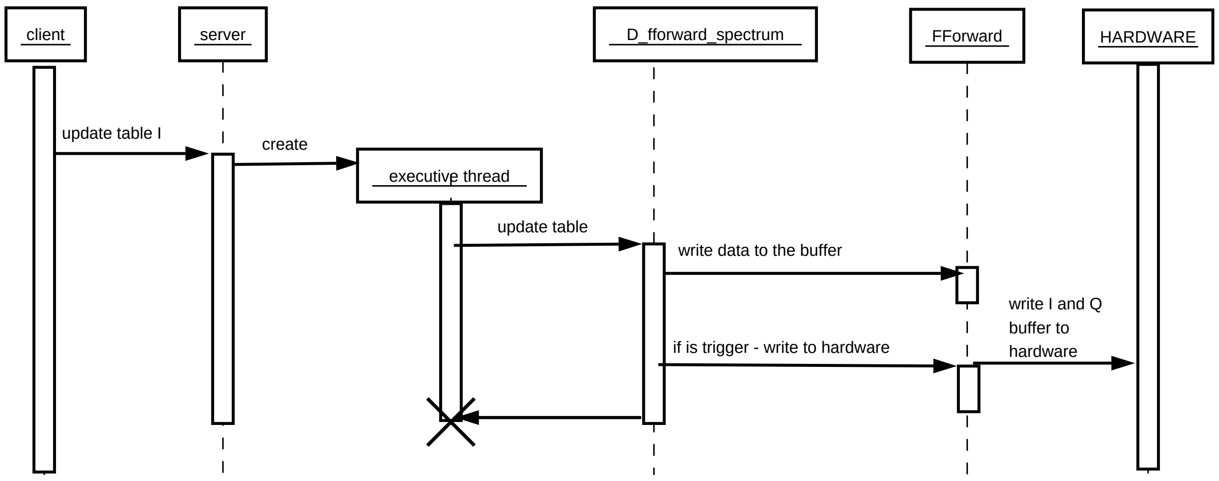


Figure 19: Sequence of methods calls during the procedure of updating the control tables.

The FForward class can be in several states. If the request from user appears, it can return data which are red from the hardware, or from buffer. When the I table has been updated, according to the rule presented in the concept chapter, it is loaded to the software buffer. If between update of table I and update of table Q a readout of table I will occur, the data must be send back to the user not from the hardware, but from the buffer. Reading the table from hardware would overwrite the updated data in the buffer, and after updating Q table the the system would send the old I table and new Q table to the hardware. This situation is presented in figure 20. Different situation should appear, when there is no table update. In that case the object is reading data from the hardware. It is realized by another flag in FFroward class - `iswrite`. This flag is initially set to 0 - this means there is no update procedure activated, every readout should read data from the hardware. When when table is updated (no matter if it is I or Q table) this flag is being set to 1. If any readout request appears it checks if the flag is set to 0 - if yes, it reads directly from hardware, if not, it reads data from the buffer. Additionally, the trigger table always calls `write` method of FForward class to update new tables in the hardware and resets the `iswrite` flag. This mechanism ensure the proper and safe control table exchange process.

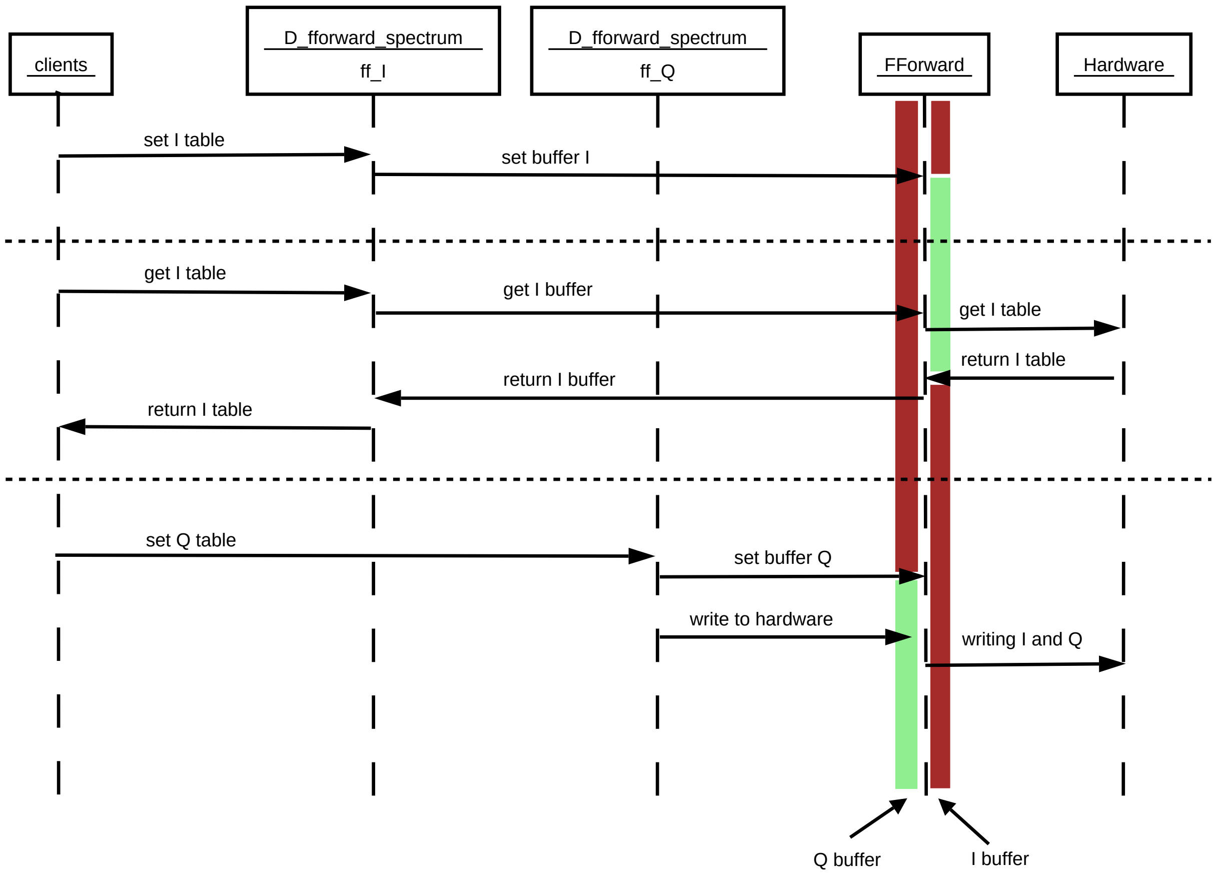


Figure 20: The effect of overwriting the buffered I table during the direct tables access. After updating the I component, the new table is stored in the buffer. If the readout request will appear in the system before updating the Q component - which flushes any changes to the hardware, the readout from SIMCON will overwrite the new I value. This will cause incorrect table exchange and algorithm failure.

There are two additionally instances of `D_fforward_spectrum_ap` class in FForward class. They provide amplitude and phase plots of the particular signal, such a feed forward or setpoint. These classes are read-only objects, which means one can only read the plot of the property but has no possibility to change it. Update of amplitude and phase plots is done also through FForward class. `D_fforward_spectrum_ap` class call during readout procedure the FForward's `update` method. This method reads from hardware actual I and Q signal and calculate amplitude and phase which are next stored in buffers. `D_fforward_spectrum_ap` object reads then values from buffer and return it to the client.

Presented solution is flexible - the particular class like `FForward` uses only the main server class and communication module. It does not depend on any other part of the system (especially the default control tables generation module). The module provides secure hardware access via communication module.

4.4 Module for default control tables generation

This module implements basic algorithm presented in the concept chapter using several DOOCS classes. The module itself is realized as a separate class `DefaultController`. Similar to the previous module, there are also helper classes implemented, and the access to the hardware is done only through the root class which is `DefaultController`. The basic DOOCS classes which have been extended for this module needs where `D_int` and `D_float`. The list of all helper classes with description have been presented in table 3.

Helper classes have `set_value` method overloaded in which they invoke the appropriate method of `DefaultController` class for generating the table (Fig. 21). The invoked method reads then the needed properties and using equations described in concept chapter it generates particular control tables set (I and Q) and then loads it into FPGA using communication module. The important issue is, that this module does not use direct control table access module but has own uploading procedures included.

The module calculations product are almost always a floating point values. Because the SIMCON operates on integer values only, every result of calculation is cut to the integer number after changing its range in order to limit the error. However the inverse operation does not return

class name	**Description**
`defcon_ff_float`	The class representing float parameters used for FF table generation.
`defcon_ff_int`	The class representing integer parameters used for FF table generation.
`defcon_fb_float`	The class representing float parameters used for FB table generation.
`defcon_fb_int`	The class representing integer parameters used for FB table generation.
`defcon_sp_float`	The class representing float parameters used for SP table generation.
`defcon_sp_int`	The class representing integer parameters used for SP table generation.
`defcon_int`	The class representing float parameters used for FF and SP table generation.
`defcon_float`	The class representing int parameters used for FF and SP table generation.
`defcon_in_flaot`	The class representing float parameters used for input rotation matrix calculation.

Table 3: List of helper classes created for Default Controller module.

the exact value which was the result of the original calculation. This effect is visible especially in rotation matrices, where in the FPGA there is no direct representation of amplitude and phase of rotation matrix, but recalculated values. In order to read actual amplitude and phase of input matrix set in the hardware, it is necessary to perform inverse calculations. Since the original result has been rounded before loading to SIMCON, the value in FPGA has error and inverse calculation cannot return the exact value. The errors which occurs are at the level 10^{-6}, so they does not have strong influence on algorithm performance.

The default control table generation module does not include any DOOCS properties which allow to display generated control tables. All readouts can be done only through the direct control table access. The reason for that solution is to avoid doubling DOOCS properties and overloading the server with frequent readout procedures of the same property. In case of lack of the direct control table access module in the server, one can provide appropriate readout plot using universal readout properties described later in chapter. The implementation of the presented module allows changes in the algorithm control tables as well as modifying the interface DOOCS interface. Helper classes are universal and can be the used for extending the number of input parameters.

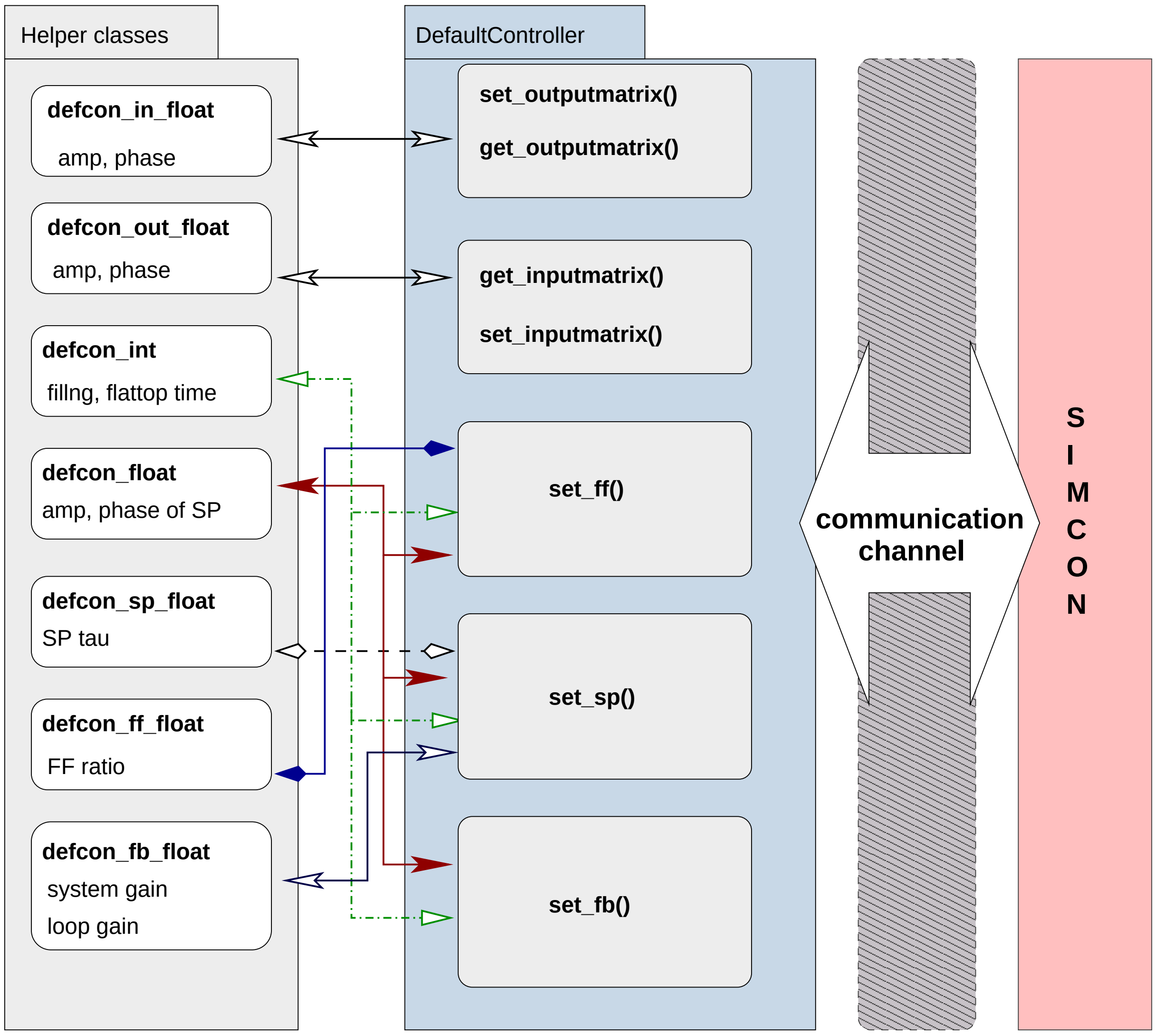


Figure 21: The relation diagram between helper classes and DefaultController class. Arrow shows which helper class triggers the appropriate root class method. The same arrows show which data are red for particular table or rotation matrix coefficients calculation.

4.5 Module for compiled Matlab code

The implementation of Matlab module bases on dynamic libraries. This solution was chosen because of the unique usage of Matlab code in DOOCS. Matlab algorithm can be frequently changed by developer without changing the interface to DOOCS server. Therefore compiling m-functions into dynamic library allow to avoid recompilation of the server without need. The Matlab code before compilation must be correctly formatted to the specific interface which was introduced in the concept chapter. This chapter describes details concerning Matlab compilation and integration with DOOCS environment.

As it was mentioned before, there are three types of functions which can be used in Matlab:

1. Init function: `A = INPUT_INIT().`
2. Input function: `[U,Y,W]=FOO(A).`
3. Internal function: `[Y,W]=FOO(Y,W).`

Init function is the first function called during start up of the server. There should be only one init function in the library and should be named exactly `INPUT_INIT`. It returns the structure with a list of all user parameters in the algorithm. These parameters are used for automatic created DOOCS properties creation. The returned structure must follow the several rules:

1. The structure can have only fields of scalar type and real only value. If one of the parameter is a matrix or vector, it must be spited into singles elements and then put into the structure. Later, inside the Matlab code (Input functions) one can join single elements back to the matrix or vector and use it in calculations.
2. The name of the field will be the DOOCS property name. Therefore, only capital letters are allowed, and maximum length of the name should not extend 15 characters. There cannot be two fields with the same name.

The structure returned by this function persists in the server memory until server is quited. It is modified by user every time he changes any of input parameters.

Input function should also have one instance in the library. It is the function which is invoked after the user changes any input parameter. It recalculates the input parameters in order to return control tables and rotation matrix coefficients (it can be compared to default control tables generation module). It returns three elements: vector U, matrix Y and structure W. Vector U corresponds to all scalar values which are loaded into FPGA, matrix Y is a table of vector. Each vector is a single control table. One can compare the Y matrix to the set of tables in the direct table access module. The matrix Y will be later modified by the internal functions. Both U and Y elements depends on actual firmware. They should correspond to the particular elements in II file. DOOCS server does know the structure of these elements and will expect the proper data format. At the server start up, the wrapper library (see further in the chapter) creates a buffer data structures of U and Y in order to provide communication with the rest of the server. The W is the private workspace structure of the algorithm. It can contain every kind of Matlab data inside. These data can be modified by algorithm in every way. Server only provides a pointer to this structure and take care of passing it between functions.

Internal function can be triggered by the user or by the DOOCS server internally. It operates only on Y and W element. There can be many internal functions in the algorithm.

If the Matlab code is properly formatted, it can be compiled using Matlab compiler. For SIMCON purposes, Matlab 6.x version has been used. The command which creates the shared library `libsimcon.so` from m-function `foo1` ,`foo2`, `foo3` is following:

```
mcc -t -L C -W lib:libsimcon -T link:lib foo1, foo2, foo3, ...
```

The compiled library is ready for the integration with DOOCS server. For this purpose, the special wrapper library `libmatcore` has been provided. It is not only providing compiled functions to the DOOCS environment, but also separates the Matlab API from DOOCS API. It is important, because DOOCS server can be therefore compiled without any Matlab library linkage. The wrapper library provides the class `DummyMat`. This solution is similar to the `Xiid_bridge` class in the communication module. `DummyMat` has a mechanism that allow the DOOCS server to create the properties from the structure returned by the Init function.

Server can read the total number of structure elements and then read step by step names of

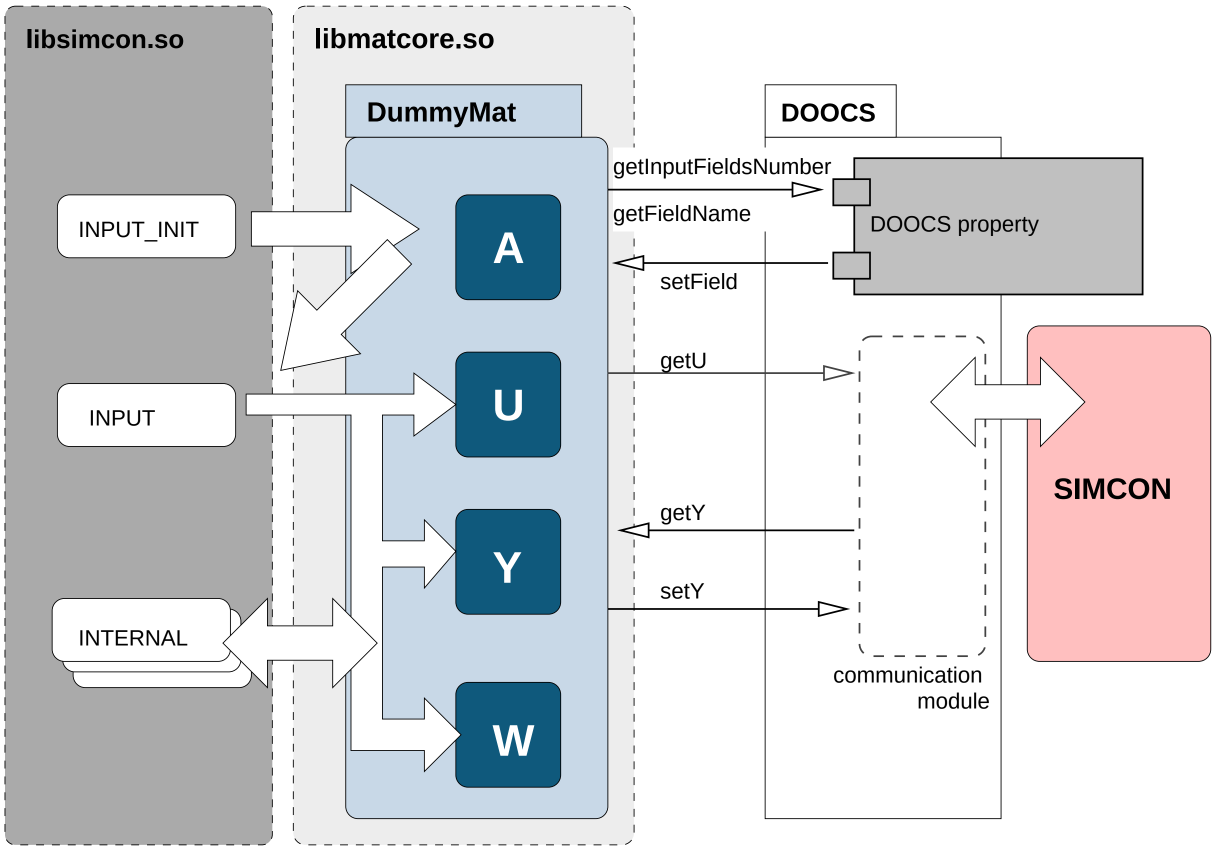


Figure 22: The implementation scheme of Matlab module for SIMCON server.

all fields. Using this name it creates new DOOCS property with name equal to the field name. As a base class for all Matlab-DOOCS properties the `D_matlab_reg` class has been used. It bases on D_float class. The overloaded `set_value` method of the `D_matlab_reg` accesses the structure inside the The wrapper library and change the value of the particular property.

There is no automatic procedure for mapping `libmsimcon` functions in the `libmatcore` library. This means, the programmer has to modify the source code of the `libmatcore` for particular `libsimcon` functions by hand. However it simple task since the interface in compiled Matlab functions is unified. Every function from `libsimcon` should be represented as a method of the `DummyMat` class. Inside this method there are only `DummyMat` internal buffers manipulations and the call of the function from the `libsimcon`. The typical method which implements the INPUT function can look like this:

```
void calcinput(){
     mxDestroyArray(W_out);
     mxDestroyArray(U);
     mxDestroyArray(Y_out);
     U=initt(&Y_out,&W_out,input_struct);
     mxDestroyArray(W_in);
     mxDestroyArray(Y_in);
     W_in=mxDuplicateArray(W_out);
     Y_in=mxDuplicateArray(Y_out);
}
```

First free lines clear old output buffers. Third line is the call of the mapped function from library. Next two lines clear the old input buffers. the last two lines copy the output buffers to the input buffers. This example shows the specific problem in Matlab API. In the function invocation, the output argument cannot appear in the input arguments list. If it happens, the output will not be filled with new data. As a solution for this limitation a two buffers have been used, and data are afterwords copied from out-buffer (the result of the function) to in-buffer(the input of the next function). Logicaly there is only one buffer, but physicaly the server can access two buffers. The doubled buffers are: W and Y. the U buffer is not used as an input argument, therefore it has one instance. The next example show the method of implementing the internal function in the `DummyMat` class:

```
 void calcrecontrol1(){
                mxDestroyArray(W_out);
                mxDestroyArray(Y_out);
                mlfAssign(&Y_out,recontrol1(&W_out,Y_in,W_in));
                mxDestroyArray(W_in);
                mxDestroyArray(Y_in);
                W_in=mxDuplicateArray(W_out);
                Y_in=mxDuplicateArray(Y_out);
}
```

These two examples shows important issue of implementation. The output buffers are destroyed after the entering to the method. This means, that after leaving the method both, output and input buffers contain the same data (since the input buffers has been filled with new data before exiting the method). This information can be used by DOOCS server. On the DOOCS side, `DummyMat` provides interface through which one can call every method, which is mapped matlab function and obtain pointers to internal buffers of the `DummyMat`. This is enough to perform the proper algorithm operation. The general sequence for calling the input and internal function from DOOCS can look like this:

```
mat->lock(); -  we lock th access to the matcore
xiid->lock(); - we lock the access to the hardware
mat->calcinput(); - we calculate the input function
LoadTables(mat->getU(),mat->getY_out(),0); we load the new data to FPGA
mat->calcrecontrol1(); we call the first internal function
ReloadTables(mat->getY_out()); update of new Y to hardware
ReadTables(mat->getY_in()); loading the actual table to Y
mat->calcrecontrol2(); call of the second internal function
xiid->unlock(); releasing the hardware lock
mat->unlock(); releasing the matcore lock
```

The `mat` object is the instance of DummyMat class and `xiid` is the instance of Xiid_bridge class. In this example another feature of the `DummyMat` class has been shown. This class, similar to `Xiid_bridge` provides the blocking of access to the matlab functions. DOOCS routines are blocking the access to the internal libmatcore tables and structures to ensure the data integrity. Different threads in the server could access the libmatcore tables in the same time. This would cause the failure of the whole controller. The example shows the flexibility of implemented solution. The input function is called from DOOCS without any parameters. DOOCS does not know nothing about the function, there is no Matlab API grammar on the DOOCS site. After input function invocation, new data are available for downloading to the hardware (function LoadTables) using methods for obtaining pointer to data. Internal function is also called without any arguments. Its result can be once again uploaded to the hardware. If the actual data are needed for the internal function they can be readout from hardware and loaded into `Y_in` table.

This table will be input argument of the internal function. In the next step we can call this function. On the DOOCS site, there is no W structure. It is hidden inside `DummyMat` class, because it is not related to the hardware. Only hardware related data of algorithm are visible to DOOCS.

Using this simple interface one can obtain fully functionality of the compiled Matlab code, equal to the original Matlab control environment. The module provides thread safe operation. Extending the interface of `DummyMat` class is simple, and changes in `libsimcon` library functions does not require recompilation of the server. Using the matlab module instead of rewriting algorithm to C language has impact on server performance. Compiled Matlab code uses many original Matlab libraries, which are not designed for real time calculations.

4.6 Monitoring and general purpose data modules

Beside main modules in the system, there have been many helper data structures developed. These includes helper classes for accessing basic IID elements like registers and bits and monitoring spectrum classes for readout of the internal signals from FPGA. The list most frequently classes with description has been presented below.

- **D_SIMCON_reg** is the general purpose class for representing the II register in DOOCS environment. It bases on D_int class. It offers the scaling and offset parameters. This means, one can define the number by which the register value will be multiply and number which will be added to the result of this multiplication before loading it to the hardware. The purpose of this feature is to provide scaling factor for registers than have different range in the hardware than in the software. The readout operation perform the reverse calculation to obtain the original value.

- **D_SIMCON_regf** is the same register class as D_simcon_reg, but operates on float values and is derived form D_float class instead of D_int.

- **D_simcon_area** can be used for readout of the particular signal form the signal list (see appendix). One can define the number of signal which should be red, the number of DAQ from which the signal should be red and as in he previous classes, the scale factor and offset.

- **D_simcon_complexarea** is used to calculate amplitude and phase from I and Q components. One has to pass in the constructor of the class the pointer to two D_simcon_area objects which are I and Q signals. Additionally one has to set the purpose flag which tells if the object will calculate amplitude (set 0) or phase (set 1) calculated out of these two signals.

In the server exists also small, dedicated for the particular purpose extensions of basic DOOCS classes. These are used for setting the timing mode, device state, etc. They function mostly as a bit switches or properties with few discreet values.

5 Tests of the designed system

The entire presented system has been tested partially during the implementation process. The SIMCON has been set to internal timing and trigger mode and and has been completely disconnected from the VUV FEL infrastructure. As the reference environment the Matlab system has been used. Through Matlab, the test and debug readouts of the SIMCON registers, bits and memory has been performed.

In order to confirm the realization of the thesis requirements, the dedicated tests have been done in the full operational environment. Three tests has been performed. For each, the appropriate test stand has been assembled. The test band configuration included:

- SUN embedded computer in VME crate, connected to the Ethernet.
- Solaris 2.8 operation system with DOOCS server and libraries.
- SIMCON 3.0 board.
- firmware files and upload tools.
- external timing (1MHz) and trigger signals.
- probe signal from cavity(or cavities).
- monitor ADCs for reference - controlled from DOOCS

The test in the operation environment required a special procedure for switching the system from DSP system into FPGA and after tests recovering the DSP operation. Following steps had to be done in order to perform proper and secure test:

1. **SIMCON booting procedure**. If the power of the VME crate was down, after turning on the system SIMCON is not booted, this means the FPGA is not configured. The booting procedure is done through VME bus from SUN computer. The complete SIMCON device consists of two board: the motherboard and the SIMCON daughter-board. On each board there is FPGA, and it mus be configured. In the configuration process the motherboard must be booted as the first one, because only through configured FPGA, SUN can "see"

the SIMCON FPGA chip. The tool that allows to configure the FPGA through VME is called *jambo*. It is a extended version of the tool provided by the ALTERA called *jam player*. ALTERA provides the source code for this tool. Therefore it was modified to use VME channel, the same which is used in DOOCS system and Matlab software. *Jambo* uses IID files with description of JTAG bits and jam-files which are the bit-files including the configuration of the chip. Because SIMCON uses Xilinx FPGA chip, before configuring the device with *jambo* the Xilinx bit file must be converted to jam file. Since the JTAG standard is the same for the both chip family, Xilinx Virtex chip can be configured using ALTERA tool. *Jambo* is the command line tool and as a parameter requires only the name of the jam file. After configuring the motherboard and SIMCON the device is ready.

2. **Running DOOCS server**. There are many ways to run DOOCS server. For the final integration, the server should be registered in the configuration file of watchdog server. This is the server which is responsible for keeping alive DOOCS server running on the same machine that it runs. For testing purposes DOOCS server has been started from console, and then switched to operate in the background. This solution is more flexible, because one can kill the process of the server whenever it is necessary. Killing the server which is under control of watchdog would cause it automatic restart which, in case of bug detection or server problem, is not permitted. After start up of the DOOCS, the system is ready for cable exchange. One can check the proper DOOCS and SIMCON interaction, by setting the SIMCON timing into internal mode and performing the ADC readout. The plots should show noise with offset. If ADC plots does not show noise it can be caused by the wrong setting of the input rotation matrix setting ie. setting the amplitude to 0. It is very important to set the FeedForward and Feedback off. SIMCON must be ready to drive system with offsets only!!

3. **Timing**. The next step is the procedure of connecting the external timing and trigger signal to SIMCON. After connecting signal it is necessary to switch SIMCON to external timing. The ADC readout should show the same noise (no input connected).

4. **Switching off the DSP controll**. As the next step one should switch off the Feedback

and Feedforward in the DSP. It is a good habit to save the actual settings in the logbook. The result should be confirm by monitoring ADC. They should show zero gradient (< 1MV). In this stage, the DSP is switched off but still is controlling the system by the compensated offset. If at the output of the controller is zero level of signal, after vector modulator there will be a constant, non zero signal. It is caused by the offset produced inside the vector modulator. In DSP the output ofsset is set to values which compensates the the output of the vector modulator to zero. In this step, after switching off the DSP, one can connect probe signals to the SIMCON.

5. **Setting klystron power to zero**. This step is necessary before switching off the cables in the output of the DSP system. After switching off, DPS will no longer compensate offset of the vector modulator. This will cause the power coming out of klystron. Therefore the voltage of the klystron should be set to zero.

6. **Connecting the SIMCON output**. After switching of the voltage of klystron, one can connect cables to the output of the SIMCON. The SIMCON is ready to start up the operation.

7. **Adjusting offsets in SIMCON**. The goal of this stage is to achieve the same offset compensation as in DSP. The voltage of the klystron can be slowly polling up. During this operation one should observer monitoring ADC. They are calibrated and will show the real gradient even if SIMCON is not calibrated. If gradient will achieve 1MV level it means that offsets are not calibrated. The compensation is done by setting the offsets in the SIMCON panel and observing the ADC readout. One should find separately for I and Q signal the value which give the minimum power at the output of the vector modulator. After compensation one can continue polling up the voltage up to the nominal level but in the same time observing the ADC readout. If gradient is exceeding the 1MV level the compensation should be repeated. At the end of the procedure, the level of signal in the ADC monitors should be similar to the one achieved with DSP.

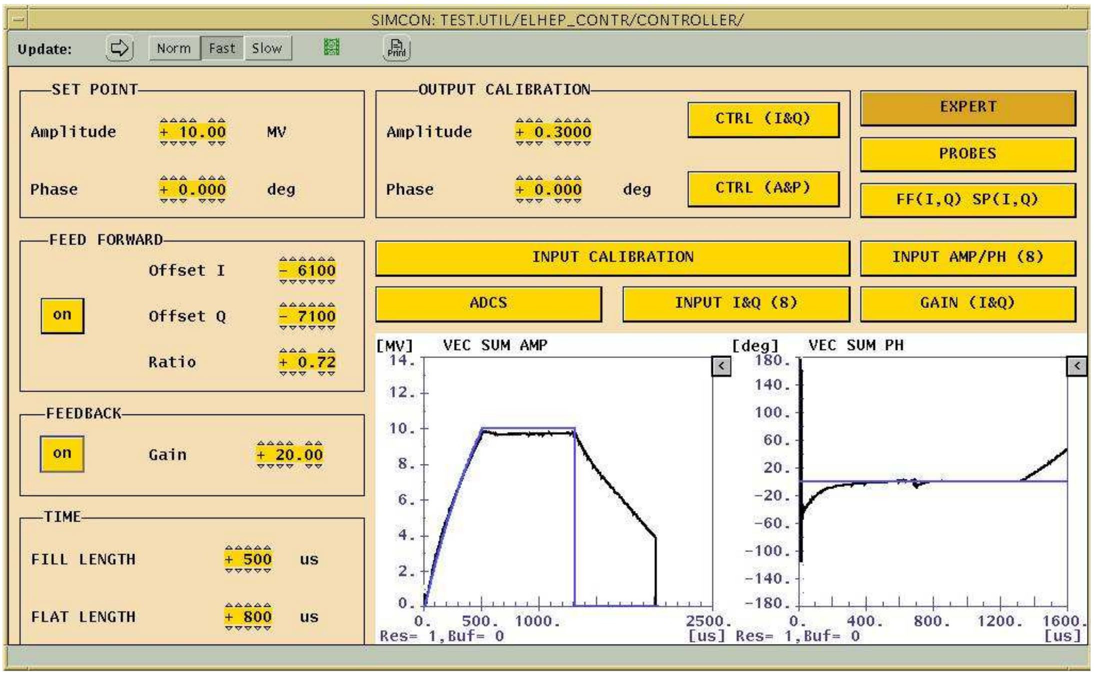
8. **Starting the operation**. After all these steps, SIMCON is in "stand by" mode which means it drives the module (or single cavity) to compensate the offsets. From this point

one can start the real test, setting the Feedforward, and Feedback.

After finished tests, one should exacly repeat the procedure from the last point to the first in order to bring the DSP back to operation. Presented procedure has been applied in every of three tests. Next subchapters describe the particular tests. Presented screens from DDD panel shows the DOOCS system working in real operation environment.

5.1 SIMCON 3.0 test in CHECHIA

CHECHIA is the test band for the single cavity. It is placed in the VUV FEL tunnel. CHECHIA is used for testing the cavities before mounting it in the module. It consists of the full environment: Cavity, klystron, RF system, local timing distribution, downconverter and DPS controller. The test has been performed using SIMCON 3.0. The probe signal has been connected to first ADC channel. During the test cavity was installed without piezo tuners, and therefore was very unstable. The following pictures show the panels with signals collected in CHECHIA.



SIMCON: TEST.UTIL/ELHEP_CONTR/CONTROLLER/

Update: Norm Fast Slow

SET POINT
Amplitude + 10.00 MV
Phase + 0.000 deg

OUTPUT CALIBRATION
Amplitude + 0.3000
Phase + 0.000 deg

FEED FORWARD on
Offset I - 6100
Offset Q - 7100
Ratio + 0.72

FEEDBACK on
Gain + 20.00

TIME
FILL LENGTH + 500 us
FLAT LENGTH + 800 us

Figure 23: Main panel for SIMCON controller. One can see SIMCON parameters and Feedforward with gain applied.

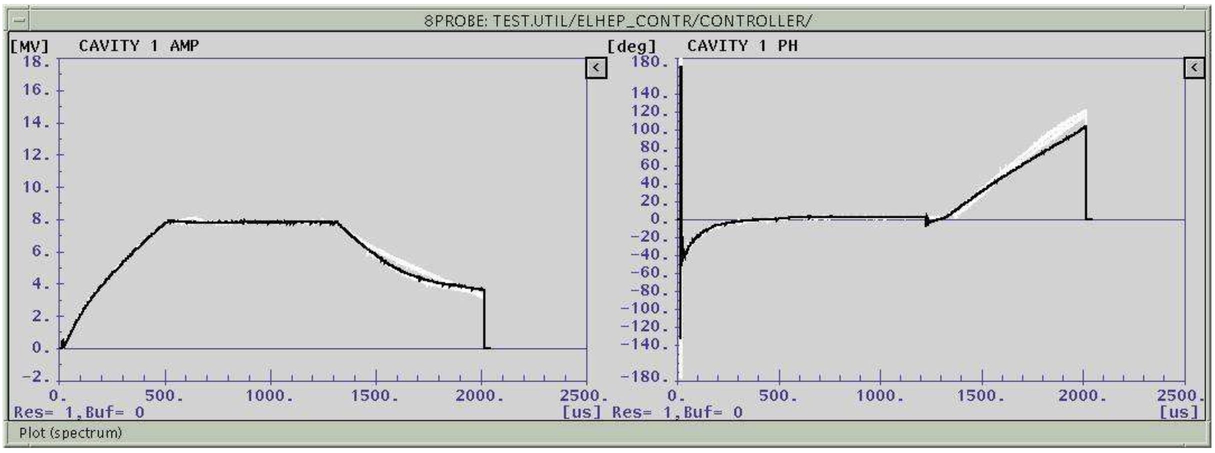
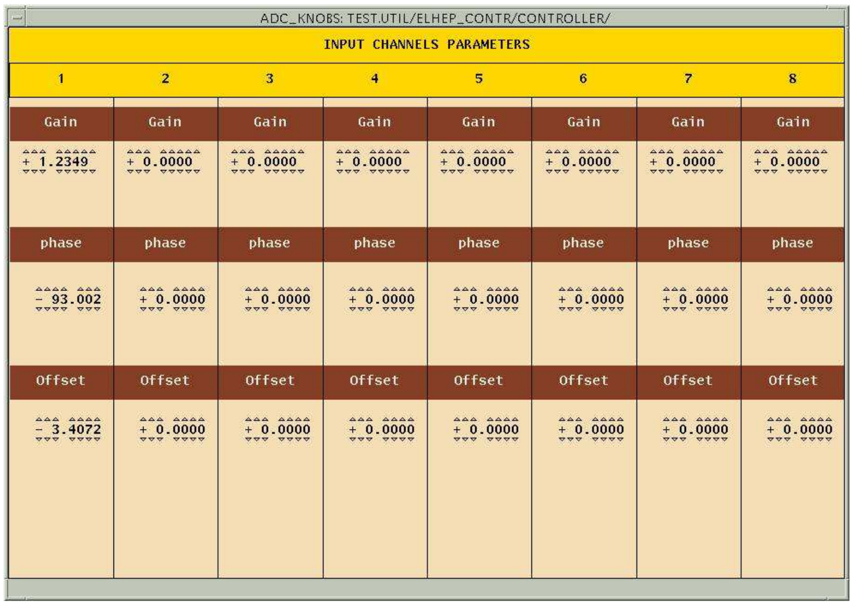


Figure 24: Amplitude and phase of the probe signal from cavity driven by SIMCON. The Feedforward and Feedback have been applied.


ADC_KNOBS: TEST.UTIL/ELHEP_CONTR/CONTROLLER/

INPUT CHANNELS PARAMETERS

1	2	3	4	5	6	7	8
Gain	Gain	Gain	Gain	Gain	Gain	Gain	Gain
+ 1.2349	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000
phase	phase	phase	phase	phase	phase	phase	phase
- 93.002	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000
Offset	Offset	Offset	Offset	Offset	Offset	Offset	Offset
- 3.4072	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000

Figure 25: The panel with configuration parameters of input channels. Only the first channel is active.

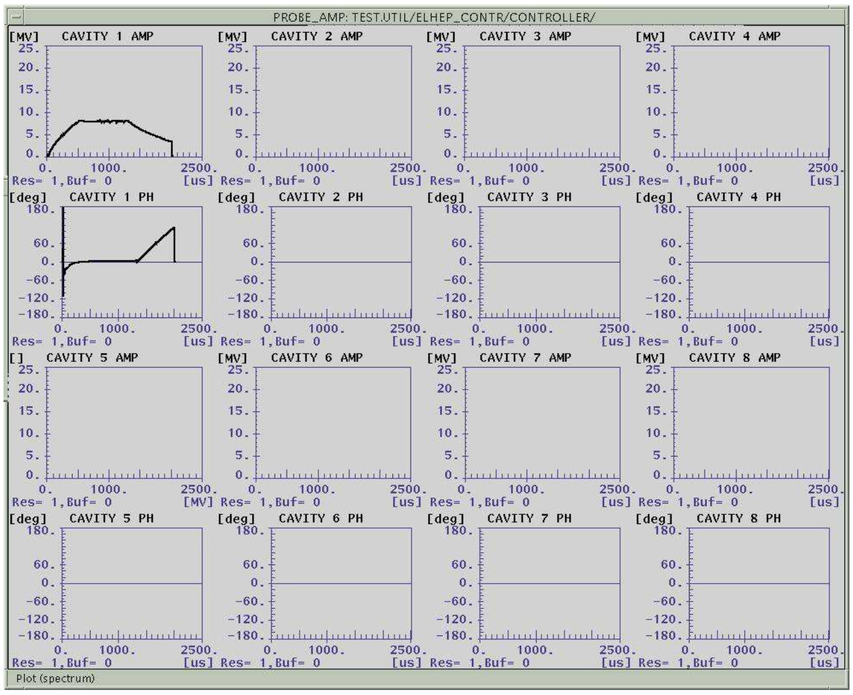
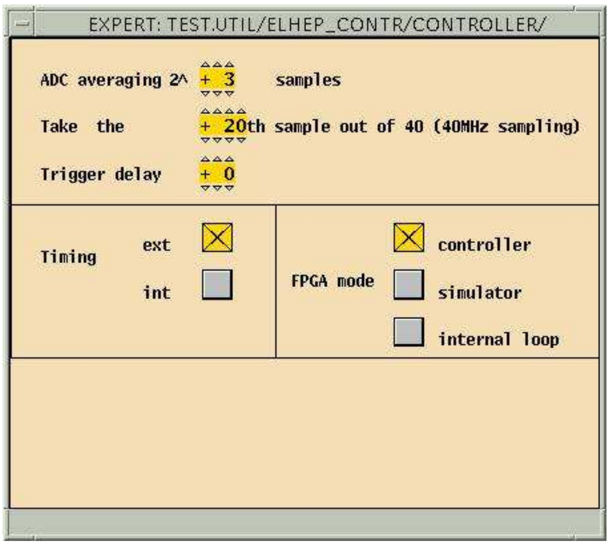


Figure 26: DOOCS allows to observe in real time amplitude and phase of all 8 channels. On the picture only the first channel is active


EXPERT: TEST.UTIL/ELHEP_CONTR/CONTROLLER/

ADC averaging 2^ + 3 samples

Take the + 20th sample out of 40 (40MHz sampling)

Trigger delay + 0

Timing ext [✓] int

FPGA mode [✓ controller] simulator internal loop

Figure 27: An expert panel in SIMCON server. The device is set to controller mode and uses external timing signals.

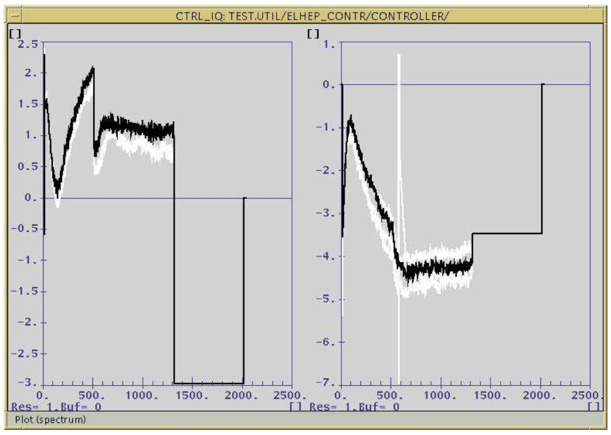


Figure 28: The output I and Q signals from the controller. The gain loop try to compensate the instability of the cavity.

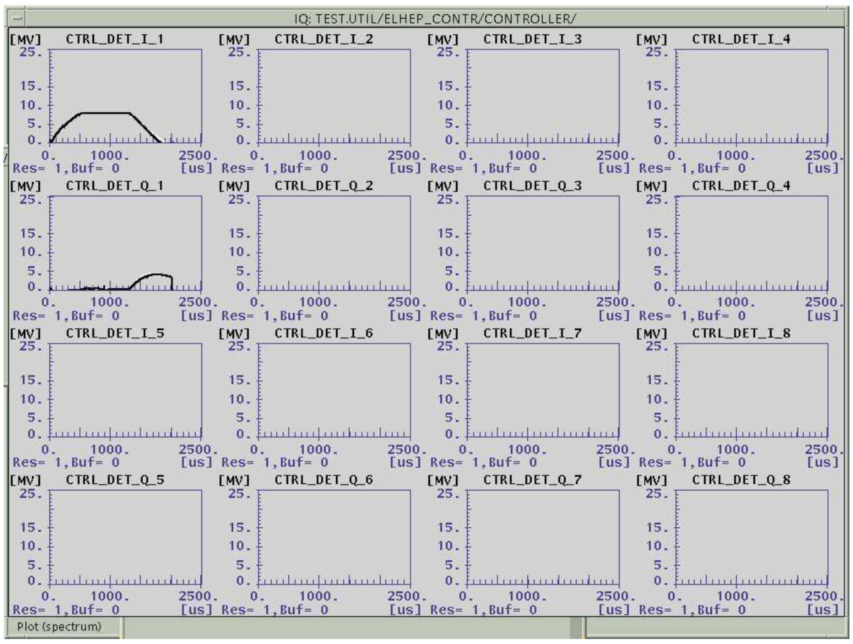
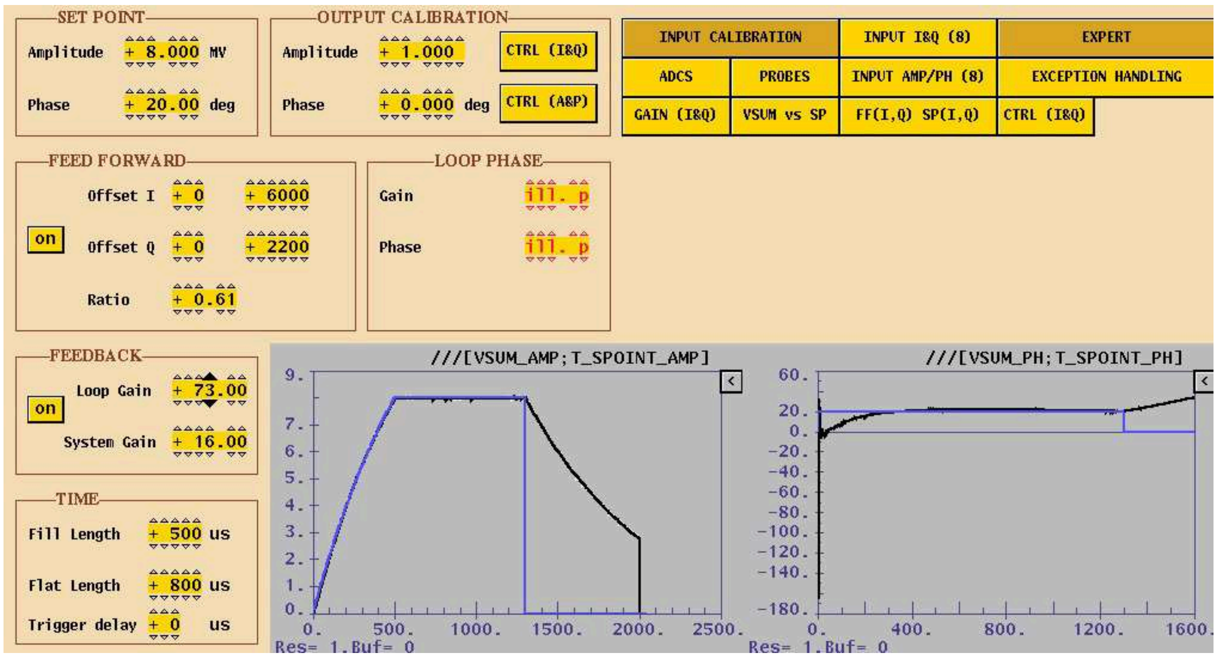


Figure 29: The I and Q signal detected by the SIMCON from probe signal.

5.2 SIMCON 3.0 test in ACC1

ACC1 is the first superconducting module after RF GUN. The tests have been performed without beam. The SIMCON firmware used in tests differed from the one used in CHECHIA but the server was not changed. Some changes has been made in GUI panel for operation optimization.

Following figures show ACC1 operation using DOOCS for SIMCON.



SET POINT
Amplitude + 8.000 MV
Phase + 20.00 deg

OUTPUT CALIBRATION
Amplitude + 1.000
Phase + 0.000 deg

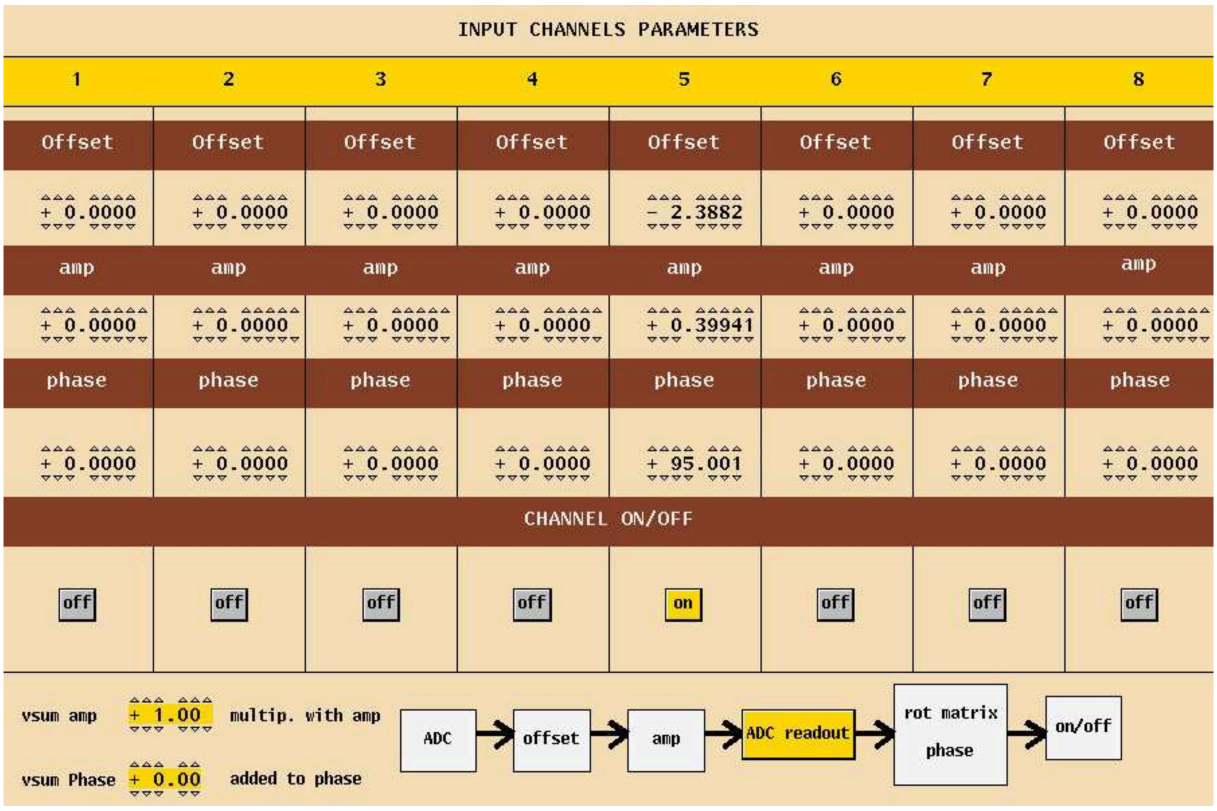
FEED FORWARD
on
Offset I + 0 + 6000
Offset Q + 0 + 2200
Ratio + 0.61

LOOP PHASE
Gain ill. p
Phase ill. p

FEEDBACK
on
Loop Gain + 73.00
System Gain + 16.00

TIME
Fill Length + 500 us
Flat Length + 800 us
Trigger delay + 0 us

Figure 30: Main panel of the server. The SIMCON is running with Feedforward and feedback.

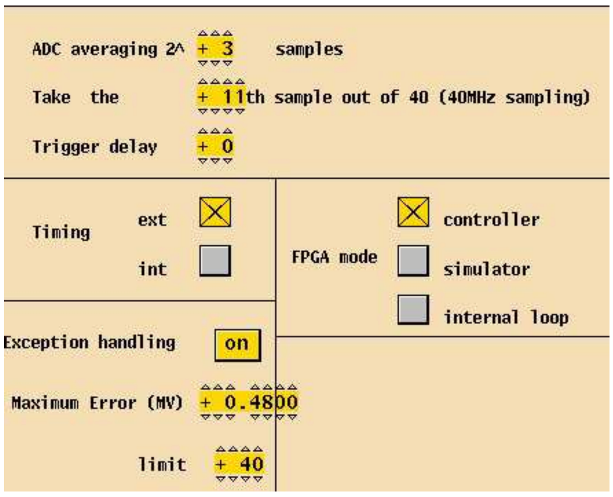

INPUT CHANNELS PARAMETERS

	1	2	3	4	5	6	7	8
Offset	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000	- 2.3882	+ 0.0000	+ 0.0000	+ 0.0000
amp	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.39941	+ 0.0000	+ 0.0000	+ 0.0000
phase	+ 0.0000	+ 0.0000	+ 0.0000	+ 0.0000	+ 95.001	+ 0.0000	+ 0.0000	+ 0.0000
CHANNEL ON/OFF	off	off	off	off	on	off	off	off

vsum amp + 1.00 multip. with amp

vsum Phase + 0.00 added to phase

Figure 31: Panel for input calibration. Different values for each channel are needed to calibrate the whole vector sum.


ADC averaging 2^ + 3 samples

Take the + 11th sample out of 40 (40MHz sampling)

Trigger delay + 0

Timing: ext [✓] int [ ]

FPGA mode: controller [✓] simulator [ ] internal loop [ ]

Exception handling on

Maximum Error (MV) + 0.4800

limit + 40

Figure 32: Expert panel for SIMCON.

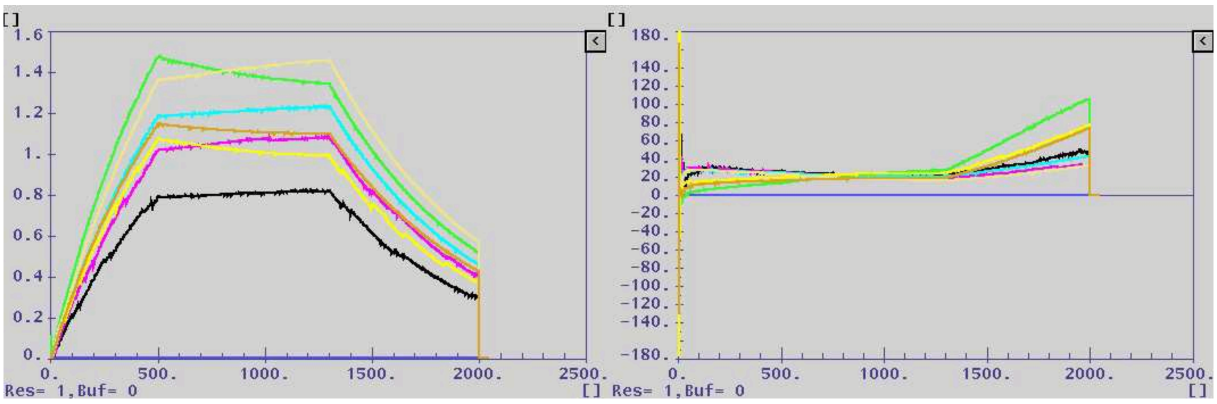


Figure 33: Grandient and phase plots for each of 8 cavities.

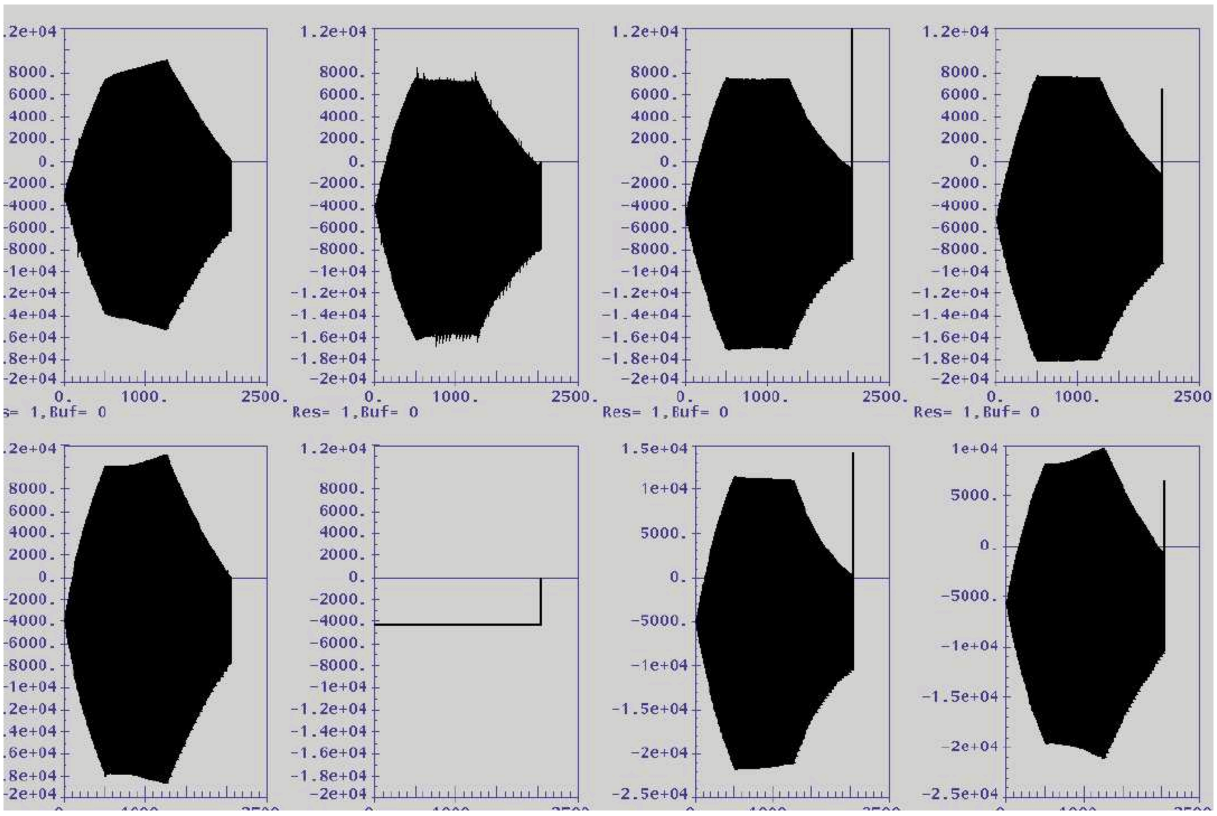
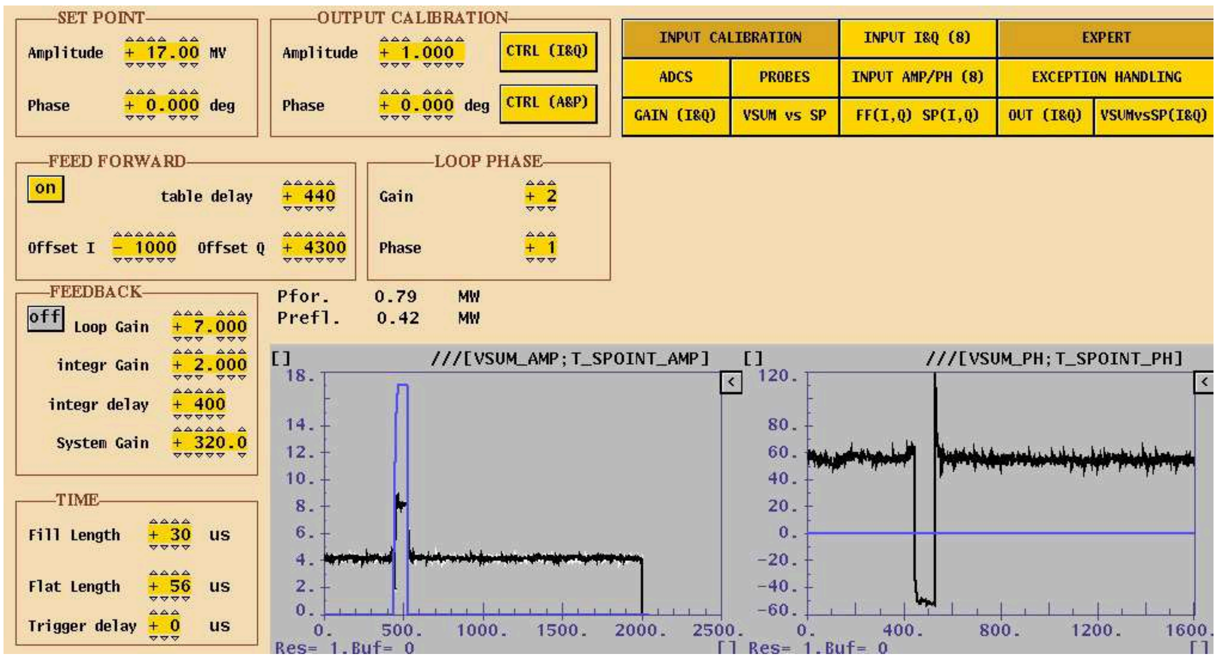


Figure 34: Eight probe signals from SIMCON ADC. One can observe directly input signal for controller.

5.3 SIMCON 3.0 test in RF-GUN

RF GUN is one cavity module at the beginning of the linac. It is used for electron generation which are next accelerated in superconducting modules. The cavity in RF GUN has different parameters. It is normal conducting cavity, cooled by water. The different behavior of the system needs different algorithm for controlling the cavity. Due to time limitation, the original firmware for ACC1 has been modified for RF GUN purposes and tested in fully operation environment. For this purpose also the DOOCS server has been changed. The main changes were done in default controller module, where the shape of generated tables has been modified additional registers have been added and GUI panels have been modified. Results showed that the firmware dedicated for ACC1 is not able to stabilize the RF GUN field. However the test showed also that DOOCS server can be modified and adapted to new firmware very easily. During 8 hours shifts, the server code was modified view times. This quick response for particular operator needs proved that implemented solution is flexible and expandable.


SET POINT
Amplitude + 17.00 MV
Phase + 0.000 deg

OUTPUT CALIBRATION
Amplitude + 1.000
Phase + 0.000 deg

FEED FORWARD
on
table delay + 440
Offset I - 1000
Offset Q + 4300

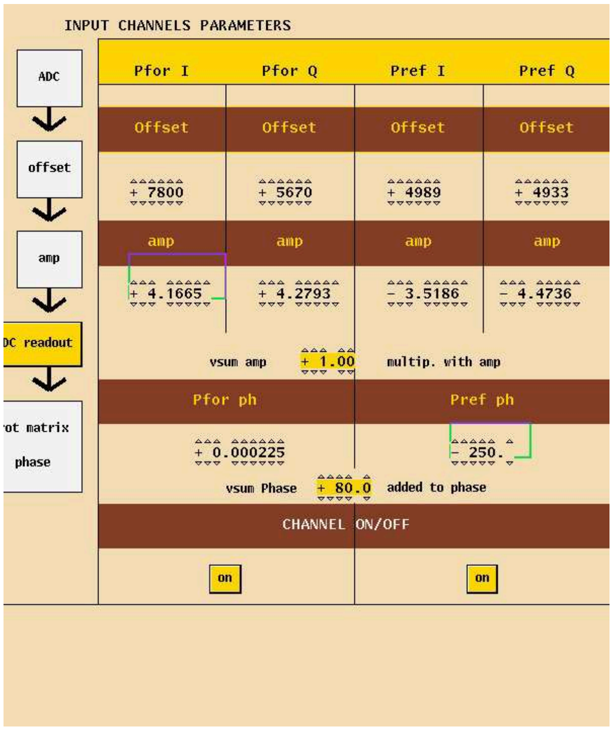
LOOP PHASE
Gain + 2
Phase + 1

FEEDBACK
off
Loop Gain + 7.000
integr Gain + 2.000
integr delay + 400
System Gain + 320.0

Pfor. 0.79 MW
Prefl. 0.42 MW

TIME
Fill Length + 30 us
Flat Length + 56 us
Trigger delay + 0 us

Figure 35: Main panel for RF GUN server. One can see modified ACC1 panel. An integrator gain was added, integrator start delay, and delay for all table start.



INPUT CHANNELS PARAMETERS

	Pfor I	Pfor Q	Pref I	Pref Q
Offset	+ 7800	+ 5670	+ 4989	+ 4933
amp	+ 4.1665	+ 4.2793	- 3.5186	- 4.4736

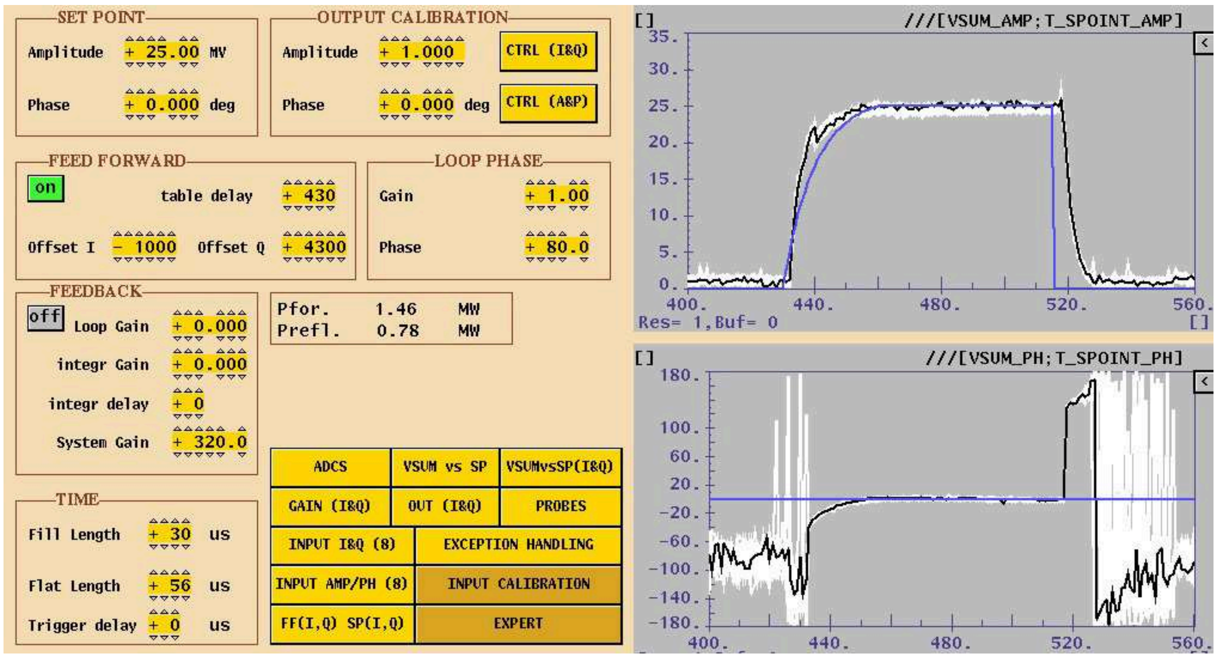
vsum amp + 1.00 multip. with amp

Pfor ph	Pref ph
+ 0.000225	- 250.

vsum Phase + 80.0 added to phase

CHANNEL ON/OFF: on on

Figure 36: RF GUN controller uses forward and reflected power for field stabilization. Therefore there are only four input channels used.



SET POINT
Amplitude + 25.00 MV
Phase + 0.000 deg

OUTPUT CALIBRATION
Amplitude + 1.000
Phase + 0.000 deg

FEED FORWARD
on
table delay + 430
Offset I - 1000 Offset Q + 4300

LOOP PHASE
Gain + 1.00
Phase + 80.0

FEEDBACK
off
Loop Gain + 0.000
integr Gain + 0.000
integr delay + 0
System Gain + 320.0

Pfor. 1.46 MW
Prefl. 0.78 MW

TIME
Fill Length + 30 us
Flat Length + 56 us
Trigger delay + 0 us

Figure 37: Eight probe signals from SIMCON ADC. One can observe directly input signal for controller.

6 Summary and conclusions

Presented DOOCS based control environment or cavity simulator and controller SIMCON 3.0 now in the testing stage. Initial tests showed, that applied concept is correct and can be used in future DOOCS applications. The following system modules have been realized and tested:

- Unified hardware-software communication model provides interface which can be easy integrated with DOOCS environment. The communication module, due to its flexibility allowed to maintain the DOOCS server during tests and simplify the source code modification. Applied thread safety procedures provided the proper communication protocol between software and hardware. Modular design of this system component, which bases on several shared libraries helped to debug the software.

- Default control tables generator is an example of the many possible algorithms, that can be implemented in DOOCS. Its flexibility and modularity has been used during tests in RF GUN, where there was a need to modify the control algorithm during the test. The algorithm implemented in this module allowed to control a superconductive module of 8 cavities, and calibrate a vector sum.

- Direct control table access module allowed to test the SIMCON firmware and other parts of the DOOCS server. The possibility of changing the control tables directly has been tested using Matlab. This application is the main tool used during algorithm development. The simplicity of using the presented module allow to integrate the SIMCON with other, non DOOCS applications which can contain sophisticated algorithms.

- Compiled Matlab code can be also integrated in the DOOCS server. Designed wrapper libraries provide easy to use interface, which allows to connect the Matlab data structures with DOOCS environment. Additionally they separate these two environment which ensure required modularity of the whole presented system. However this solution is not recommended for the future. C implementation of the algorithm gives possibility for easy interaction of the calculation routines with other modules of the DOOCS application, while Matlab code always need wrapper libraries and interfaces for system integration.

Tests of DOOCS applications with SIMCON device showed also a need for firmware optimization, especially in the readout components. Experience gained during tests will allow to avoid many software and firmware related problems in the future.

Both server, and GUI panels will be optimized for user operation. Additionally to the existing server for ACC1, the second server for RF GUN (for new FPGA firmware) is being developed. The integration of the designed systems with the existing VUV FEL infrastructure is planned to be finished in the fall of the year 2005. Solutions used during the development process gives the background for the future control systems, dedicated for SIMCON based control devices.

The next version of the SIMCON - 3.1 will have a possibility to run DOOCS server on embedded Power PC processor inside the FPGA chip. This solution will create new possibilities of SIMCON usage. With DOOCS embedded in the Power PC, SIMCON can be standalone device, which does not need VME crate for the operation. First tests showed, that DOOCS can operate in that new environment. However, changes will have to be made in order to provide access to the hardware. New channel library has to be developed, because DOOCS will no longer communicate through VME, but the communication core will remain unchanged - it excellent fits the embedded platform requirements. The module for compiled Matlab code will be also abandoned because of the lack of support from Mathworks for this particular Power PC (no run time libraries are available for this processor).

There are plans to develop unified Mathematical library for algorithm development in software under DOOCS environment. It will be a kind of substitute for Matlab libraries and provide unified data structures and functions helpful in the development process. Unified mathematical engine dedicated for DSP calculation can use floating point processor (placed as a separate chip on the board) or floating point VHDL core inside FPGA chip. In both solutions, the communication protocol with the float unit would be hidden inside the mathematical library, and user would have only simple and clear interface for using floating point calculations. Proposed solution would be easily integrated with DOOCS control environment or any other control application.

References

[1] Waldemar Koprek, Pawel Kaleta, Jaroslaw Szewinski, Krzysztof T.Pozniak, Ryszard S. Romaniuk: *Software Layer for SIMCON ver. 1.1., FPGA based TESLA Cavity Control System; USER'S MANUAL.* TESLA internal note 2005-05.

[2] T. Czarski, K. Pozniak, R. Romaniuk S. Simrock: *TESLA Cavity Modeling and Digital Implementation with FPGA Technology Solution for Control System Purpose.* TESLA internal note 2003-28.

[3] 5.K. T. Pozniak, M. Bartoszek, M. Pietrusinski, *Internal interface for RPC muon trigger electronics at CMS experiment*, Proceedings of SPIE, Bellingham, WA, USA, Vol. 5484, 2004, pp. 269-282.

[4] W. Giergusiewicz, W. Koprek, W. Jalmuzna, K. T. Pozniak, R. S. Romaniuk *FPGA Based, DSP Integrated, 8-Channel SIMCON, ver. 3.0. Initial Results for 8-Channel Algorithm.* Tesla internal note 2005-14.

[5] http://tesla.desy.de/doocs/server/dsp/DSPprsvr.pdf - Description of the DOOCS server for DSP system.

[6] T. Czarski, R. S. Romaniuk, K. T. Pozniak, S. Simrock: *Cavity control system: optimization methods for single cavity driving and envelope detection*, Proceedings of SPIE, Bellingham, WA, USA, Vol. 5484, 2004, pp. 99-110. bi

[7] Krzysztof T. Pozniak, Ryszard S. Romaniuk *Modular and Reconfigurable Common PCB-Platform of FPGA Based LLRF Control System for TESLA Test Facility*, TESLA Report 2005-04.

[8] Thomas Schilcher ,*Vector Sum Control of Pulsed Accelerating Fields in Lorentz Force Detuned Superconducting Cavities*

[9] http://doocs.desy.de - DOOCS homepage.

[10] http://sun.com - SUN MICROSYSTEMS homepage.

[11] http://www.mathworks.com - Matlab homepage.

[12] Diagram of the LLRF system done by Alexander Brandt.

[13] W.M. Zabolotny, T. Czarski, T. Jezynski, K.T. Pozniak, P. Rutkowski, R.S. Romaniuk, K. Bunkowski, *FPGA Based Cavity Siumulator for TESLA Test Facility*, TESLA Report 2003-22.

[14] Kernighan Brian W., Ritchie Dennis M. *Język ANSII C*, WNT 2001.

[15] Hall Carl L. *Techniczne podstawy systemów klient - serwer*, WNT 1996.

[16] Bentley Jon *Perełki oprogramowania*, WNT 2001.

[17] Knuth Donald E. *Sztuka programowania*, WNT 2001.

[18] Stevens W.Richard *Programowanie w środowisku systemu UNIX*, WNT 2001.

[19] W. Petersen *The VMEbus Handbook*, WNT 2001.

[20] J. Szabatin *Podstawy teorii sygnałów*, WKŁ 2003

[21] T. P. Zieliński *Od teorii do cyfrowego przetwarzania sygnałów*, Wydawnictwo AGH 2002.

[22] A. Silberschatz, P. Galvin *Podstawy systemów operacyjnych*, WNT 2001.

7 Appendixes

SIMCON 3.0 READOUT CHANNELS

channel 0: test signal from module TEST GENERATOR,

channel 1: simulator CAV_ OUT_I

channel 2: simulator CAV_ OUT_Q

channel 3: simulator CAV_ DETUN

channel 4: simulator CAV_VMOD

channel 5: controller MUX_OUT_SUMV_I

channel 6: controller MUX_OUT_SUMV_Q

channel 7: controller CTRL_I

channel 7: controller CTRL_Q

channel 9: controller TGAIN_I

channel 10:controller TGAIN_Q

channel 11:controller TSETPOINT_I

channel 12:controller TSETPOINT_Q

channel 13:controller TFEEDFORWARD _I

channel 14: controller TFEEDFORWARD _Q

channel 15: simulator TBEAM _I

channel 16: simulator TBEAM _Q

channel 17: simulator CAV_MODE1

channel 18:simulator CAV_MODE1D

channel 19:simulator CAV_MODE2

channel 20:simulator CAV_MODE2D

channel 21:simulator CAV_MODE3

channel 22:simulator CAV_MODE3D

channel 23: input signal ADC1

channel 24: input signal ADC2

channel 25: input signal ADC3

channel 26: input signal ADC4

channel 27: input signal ADC5

channel 28: input signal ADC6

channel 29: input signal ADC7

channel 30: input signal ADC8

channel 31: controller CTRL_DET_I_1

channel 32: controller CTRL_DET_I_2

channel 33: controller CTRL_DET_I_3

channel 34: controller CTRL_DET_I_4

channel 35:controller CTRL_DET_I_5

channel 36: controller CTRL_DET_I_6

channel 37: controller CTRL_DET_I_7

channel 38: controller CTRL_DET_I_8

channel 39: controller CTRL_DET_Q_1

channel 40: controller CTRL_DET_Q_2

channel 41: controller CTRL_DET_Q_3

channel 42: controller CTRL_DET_Q_4

channel 43: controller CTRL_DET_Q_5

channel 44: controller CTRL_DET_Q_6

channel 45: controller CTRL_DET_Q_7

channel 46: controller CTRL_DET_Q_8

Acknowledgments

I would first like to thank my tutors, Dr. R. Romaniuk and K. Pozniak for patient, all advices and encouragement that they have given me during these years.
I owe also sincere thanks to S. Simrock for giving me opportunity to work in the team of people with great experience and treating me (student) as a professional engineer. He always was ready to share with me with his knowledge and experience.
I would like to thank all my colleagues from PERG and ELHEP group especially J. Szewinski, W. Jalmuzna, W. Koprek, T. Jezynski for giving me support (mental and technical).
Finally I send my thanks to my family, which have been waiting for this moment so long.