Tesla Report 2003-22

FPGA based Cavity Simulator for Tesla Test Facility

Wojciech M. Zabolotny[a], Karol Bunkowski[b], Tomasz Czarski[a], Tomasz Jezynski[a], Krzysztof Pozniak[a], Piotr Rutkowski[a], Stefan Simrock[d], Ryszard Romaniuk[a]

[a]Institute of Electronic Systems, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warszawa, Poland
[b]Institute of Experimental Physics, Warsaw University, ul. Hoza 69, 00-681 Warszawa, Poland
[d] Deutsches Elektronen - Synchrotron DESY, Notkestrasse 85, 22607 Hamburg, Germany

ABSTRACT

This paper presents a FPGA based DSP system for realtime simulation of superconducting accelerator's cavity. The superconducting linacs require sophisticated control systems for maintaining the constant amplitude and phase of accelerating field in the accelerator's cavities. The debugging of these systems on real hardware can be both difficult and dangerous.

To allow testing of the real LLRF systems in the real time and with different cavity parameters, the FPGA based system has been developed.

Keywords: TESLA, DESY, FPGA, DSP, analog systems simulation, high energy physics, superconducting accelerating cavities

1. INTRODUCTION

The TESLA is a linear e^+ e^- collider, using superconducting accelerating cavities. The electron and positon beams will be accelerated by standing wave cavities. To assure good focusing and monoenergetic beams, the constant amplitude and phase of the accelerating field is required. However the superconducting cavities are subjected to different factors influencing their geometry and resonance frequency - eg. microphonics, Lorentz force detuning and the others. Therefore a "Low Level RF Control (LLRF) System"[1] is required to keep the amplitude and phase of the accelerating field constant. Correct operation of the LLRF system is essential for the beam quality and availability and for the safety of the whole accelerator. Therefore the thorough testing of the LLRF system is required.

The FPGA based cavity simulator allows for safe realtime testing of the LLRF system for the wide range of the cavity parameters and operating conditions and can be used for training of the operators.

2. CONTINUOUS MODEL OF THE CAVITY

The cavity model used in the simulator was proposed by Stefan Simrock in.[2] The model consists of two components - the mechanical model, describing the cavity's detuning caused by the Lorentz force, and the electrical model describing the electrical field vector.

The mechanical properties of the cavity are described with the following equation:

$$\begin{bmatrix} \Delta\dot{\omega}_1 \\ \Delta\ddot{\omega}_1 \\ \vdots \\ \Delta\dot{\omega}_N \\ \Delta\ddot{\omega}_N \end{bmatrix} = \begin{bmatrix} 0 & 1 & \cdots & 0 & 0 \\ -\omega_1^2 & -\frac{1}{\tau_1} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & -\omega_N^1 & -\frac{1}{\tau_N} \end{bmatrix} \cdot \begin{bmatrix} \Delta\omega_1 \\ \Delta\dot{\omega}_1 \\ \vdots \\ \Delta\omega_N \\ \Delta\dot{\omega}_N \end{bmatrix} + 2\pi \begin{bmatrix} 0 \\ -K_1\omega_1^2 \\ \vdots \\ 0 \\ -K_N\omega_N^2 \end{bmatrix} \cdot V_{acc}^2 \tag{1}$$

Further author information: (Send correspondence to Wojciech Zabolotny or Ryszard Romaniuk)
Wojciech Zabolotny: E-mail: wzab@ise.pw.edu.pl, Telephone: +48 22 6607717, Fax: +48 22 8252300
Ryszard Romaniuk: E-mail: rrom@ise.pw.edu.pl, Telephone: +48 22 6607986, Fax: +48 22 8252300

Where

- $\Delta\omega_k$ - is the cavity's detuning associated with the *k*-th mechanical mode of the cavity
- ω_k - is the resonance frequency of the *k*-th mechanical mode of the cavity
- τ_k - is the mechanical time constant associated with the *k*-th mechanical mode of the cavity
- K_k - is the Lorentz force detuning constant for the *k*-th mechanical mode of the cavity
- V_{acc} - is amplitude of the accelerating electric field.

However the above equation may be decomposed in N independent equations of form:

$$\begin{bmatrix} \dot{X}_{m,2i} \\ \dot{X}_{m,2i+1} \end{bmatrix} = \begin{bmatrix} A_{m,2i,2i} & A_{m,2i,2i+1} \\ A_{m,2i+1,2i} & A_{m,2i+1,2i+1} \end{bmatrix} \cdot \begin{bmatrix} X_{m,2i} \\ X_{m,2i+1} \end{bmatrix} + 2\pi \begin{bmatrix} 0 \\ K_i\omega_i^2 \end{bmatrix} \cdot V_{acc}^2 \tag{2}$$

with coefficients defined as follows:

$$X_{m,2i} = \Delta\omega_i \tag{3}$$
$$X_{m,2i+1} = \Delta\dot{\omega}_i \tag{4}$$
$$A_{m,2i,2i} = 0 \tag{5}$$
$$A_{m,2i+1,2i} = 1 \tag{6}$$
$$A_{m,2i,2i+1} = -\omega_i^2 \tag{7}$$
$$A_{m,2i+1,2i+1} = -\frac{1}{\tau_i} \tag{8}$$

The electrical model is described with the following equation:

$$\begin{bmatrix} \dot{X}_{e,0} \\ \dot{X}_{e,1} \end{bmatrix} = \begin{bmatrix} -\omega_{1/2} & -\Delta\omega \\ \Delta\omega & -\omega_{1/2} \end{bmatrix} \cdot \begin{bmatrix} X_{e,0} \\ X_{e,1} \end{bmatrix} + R\omega_{1/2} \cdot \begin{bmatrix} U_{e,0} \\ U_{e,1} \end{bmatrix} \tag{9}$$

$$\Delta\omega = \Delta\omega_0 + \Delta\omega_1 + \cdots + \Delta\omega_N \tag{10}$$

where

- $\Delta\omega$ is the total detuning, defined by the equation (10)
- $\Delta\omega_0$ is the predetuning
- R is the shunt impedance of the cavity
- $\omega_{1/2}$ is the half bandwidth of the cavity

3. DISCRETE MODEL OF THE CAVITY

The cavity model presented above is however the "continuous model", and for digital simulator it is necessary to transform it into the discrete form. The parameters of the mechanical model are stationary during the simulation, so it could be discretized in advance using the standard Matlab discretization procedure. The discrete mechanical model is described by the set of N (11) equations, where the coefficients are calculated from the coefficients of the equation (2) with Matlab's "c2d" function.

$$\begin{bmatrix} X_{md,2i,n+1} \\ X_{md,2i+1,n+1} \end{bmatrix} = \begin{bmatrix} A_{md,2i,2i} & A_{md,2i,2i+1} \\ A_{md,2i+1,2i} & A_{md,2i+1,2i+1} \end{bmatrix} \cdot \begin{bmatrix} X_{md,2i,n} \\ X_{md,2i+1,n} \end{bmatrix} + 2\pi \begin{bmatrix} 0 \\ -K_i\omega_1^2 \end{bmatrix} \cdot \left[V_{acc}^2\right] \tag{11}$$

The parameters of the electrical model, however, depend on the state variables of the mechanical model, and therefore the electrical model coefficients must be updated and the model must be discretized again in each simulation step, when the new coefficients are available.

When choosing the discretization method for the electrical model it was necessary to consider the limitations of the target FPGA platform. Particularly the algoritm should not involve the division operation. Therefore the Euler forward method was chosen, to discretize the electrical model, which resulted in the following equation, where T_s is the sampling period:

$$\begin{bmatrix} X_{e,0,n+1} \\ X_{e,1,n+1} \end{bmatrix} = \begin{bmatrix} 1 - T_s\omega_{1/2} & -T_s\Delta\omega \\ T_s\Delta\omega & 1 - T_s\omega_{1/2} \end{bmatrix} \cdot \begin{bmatrix} X_{e,0,n} \\ X_{e,1,n} \end{bmatrix} + T_sR\omega_{1/2} \cdot \begin{bmatrix} U_{e,0} \\ U_{e,1} \end{bmatrix} \tag{12}$$

So finally the simulator was described with the equations (11), (10) and (12).

Another problem to be solved when implementing the simulator was the range of numbers. The original equations contain coefficients ranging from 10^{-12} to 10^9. The simulator had to be implemented in the fixed point arithmetics, and therefore scaling of the state variables and input/output values was necessary to limit the range of the numbers. After the scaling it was possible to implement the simulator using the 64-bit fixed point arithmetics with 40-bit fractional part.

4. IMPLEMENTATION

The design should be implemented in the single Xilinx XC V2E 3000 chip, available on the Xilinx DSP Xtreme board, produced by the Nallatech. Two high-speed inputs for the cavity stimulus are implemented with he 14-bit 65MHz A/D converters, and two high speed outputs for the accelerating field components (I and Q) are implemented with two 14-bit 65 MHz D/A converters). Other parameters are transmitted through the digital channel (eg. USB or EPP) and stored into the internal registers of the simulator.

The Xilinx System Generator was planned as the main development tool because of its features like:

- Easy integration with the Matlab environment, allowing for testing of the algorithms
- Easy management of the arithmetical block parameters ("wrap around" or saturation on the overflow, arithmetic's precision and so on).

Unfortunately, the Xilinx System Generator is mainly dedicated for the development of the systems with "datapath" architecture. Such an implementation of the cavity simulator, however, would be very inefficient. The calculation of the electrical model state can be performed only after the mechanical model state is calculated and both - coefficients and stimuli for the electrical model are available. Analogically the mechanical model state may be calculated only after the electrical gradient components are calculated.

To make efficient use of the FPGA resources, it is necessary to share resources between both models, which is not possible in the "datapath" architecture. To achieve both high speed of processing and efficient use of FPGA resources, a special DSP architecture has been developed.

The equations used in the cavity model (11), (10) and (12) may be calculated by the consecutive calculation of the following formula:

$$R = A_1X_1 + A_2X_2 + UB \tag{13}$$

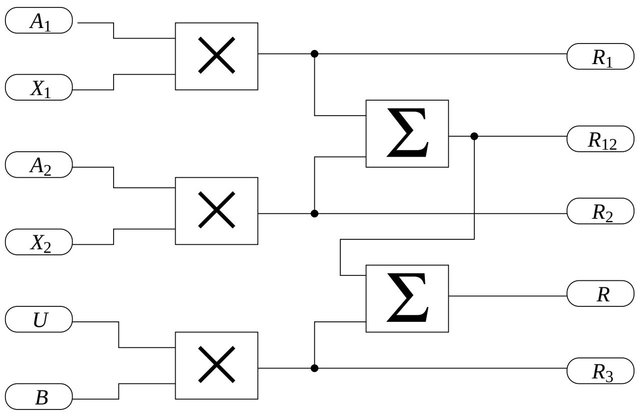

Figure 1. Elementary mathematical block

Which can be performed by the “Elementary Mathematical Blocks” (EMB) shown in the figure 1.

Such a block allows also for simultaneous calculation of the other sums of products:

$$R_1 = A_1X_1 \tag{14}$$

$$R_2 = A_2X_2 \tag{15}$$

$$R_3 = UB \tag{16}$$

$$R_{12} = A_1X_1 + A_2X_2 \tag{17}$$

The amount of available EMB’s depends on the precision of the arithmetics. The XC2V3000 chip contains 96 18-bit x 18-bit multipliers, which can be combined to build multipliers working with longer words.

For the cavity simulator the 64-bit precision has been chosen (the highest precision allowed by the Xilinx System Generator, probably the final version of the simulator will use more moderate word length).

The resources available in XC2V3000 chip are sufficient for six 64bit x 64bit multipliers (each consumes 16 elementary 18bit x 18bits multipliers, and there are 96 such multipliers in the chip), so it is possible to build 2 EMBs.

The FPGA performs the particular algorithm, by applying the proper data on the input of the EMB, and then, after certain number of clock pulses (depending on the latency of the EMB internal datapath), by reading the result from the proper EMB output.

To make efficient use of the 2 EMBs implemented in the chip it is necessary to transfer up to 12 input values, and (theoretically) up to 8 output values. Therefore it was impossible to use the standard data bus based architecture. Instead, the multiplexers have been used on each input of the EMB, and on each input of the register holding the modifiable coefficient or state variable (Fig. 2).

The multiplexers are controlled with two signals - MSEL, selecting the input data for the EMBs, and ISEL, selecting the input data for the data registers. Each program step has assigned the proper MSEL and ISEL values. Additionally the “decoded ISEL signals” (ISEL_0, ISEL_1, ... ISEL_N) are provided, as the write strobes to allow the particular data registers to latch their input data in the appropriate program cycles.

The interconnections between the EMB’s and data registers have been implemented using the “From” and “Goto” blocks available in Matlab/Simulink. Unfortunately tags to be used in “From” and “Goto” blocks used in XSG systems must be local, which impaired significantly possibility of hierarchization of the simulator. Some parts of the single sheet representing the simulator are shown in the figures 3 and 5. The resulting structure is not flexible and difficult to parametrize. One of important parameters which must be chosen in advance is the amount of inputs in the multiplexers. Big amount of inputs makes the design illegible, but too small amount of inputs can obstruct implementation of more

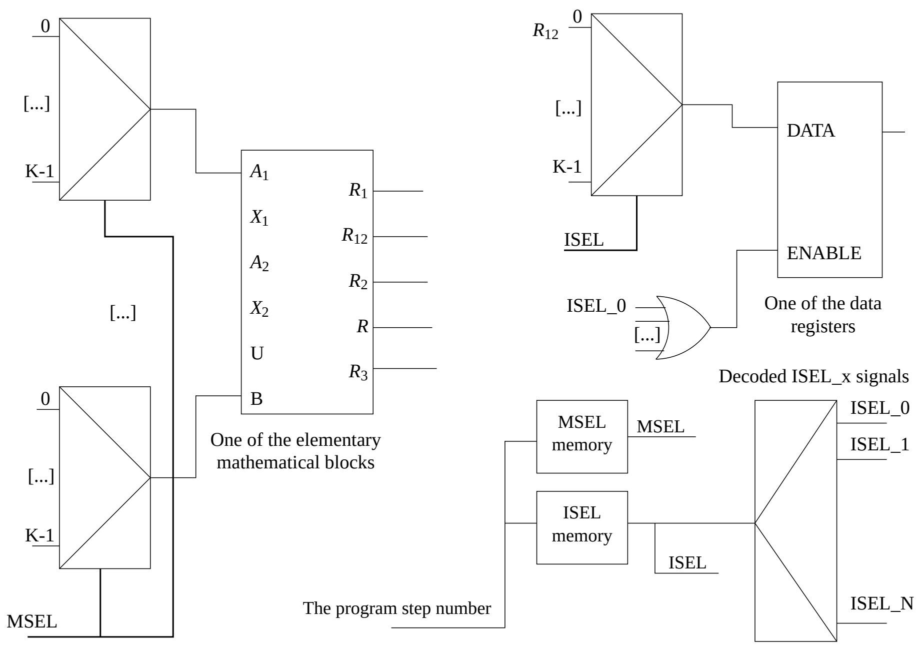

Figure 2. Structure of the simulator

complicated algorithms - because the increase of amount of inputs requires complete redrawing of the significant part of the diagram.

Another implementation of the block shown in the figure 5, based on the VHDL language, is shown in the figure 6. This implementation is much more flexible, the amount of conditions in the conditional signal assignment (corresponding to the amount of multiplexer inputs in the XSG implementation) can be easily changed.

However the pure VHDL based solution does not allow to utilize such advantages of the XSG system as easy creation of complex mathematical blocks and easy management of the arithmetics precision. Therefore another "hybrid" approach was also tried, where the basic mathematical blocks were created in the XSG and automatically converted to the VHDL, while interconnections between the EMBs and data registers were described in the pure VHDL.

5. PROGRAMMING OF THE SIMULATOR

The presented simulator may be considered to be a very simplified DSP processor. However, the programming of the system is very laborious.

To implement the program consisting of a few calculations one must first assign by hand the MSEL values corresponding to the different input combinations used in the program flow, and to put the proper tags in the "From" blocks at the appropriate inputs of the EMB input multiplexers.

Then, one has to assign the selected ISEL value to the particular combination of values on the outputs of EMB blocks, and to put the proper tags in the "From" blocks connected to the appropriate inputs of the data registers' input multiplexers. Additionally, one has to connect the proper decoded ISEL_x signal to the enable input of the data register.

The final step is to assign the selected MSEL and ISEL values, to the proper program clock cycles, and fill the MSEL and ISEL memories. Additionally one must consider the latency introduced by the EMB blocks.

The whole process is time consuming and error prone. This task could be accomplished by the dedicated compiler, but at the moment this is a work in progress.

The "graphical" representation of the algorithm, where the information about the single step of the program is scattered over the whole simulator's diagram is also quite illegible (figures 3 and 5).

The VHDL based implementation, shown in the figures 4 and 6 is more flexible and easier to maintain.

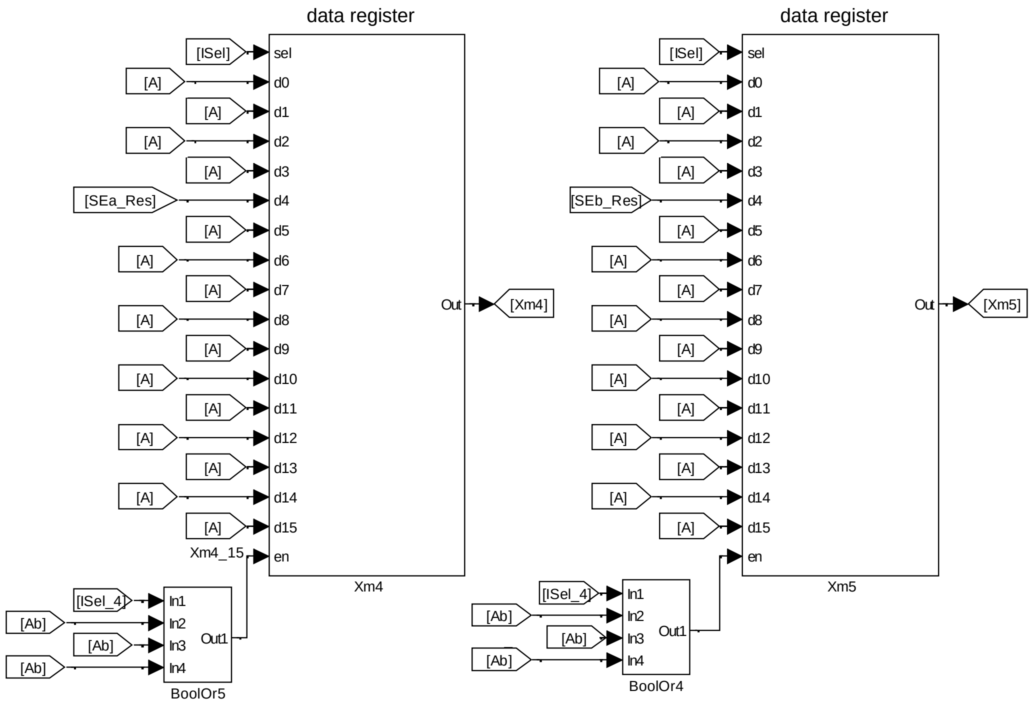

Figure 3. Two data registers with input multiplexer and "From" blocks

```
p_xm4: process (clk,clr)
begin
  if clr = '1' then
    xm4 <= (others => '0');
  elsif rising_edge(clk) then
    if isel=X"4" then
      xm4 <= res_a;
    end if;
  end if;
end process;
```

```
p_xm5: process (clk,clr)
begin
  if clr = '1' then
    xm5 <= (others => '0');
  elsif rising_edge(clk) then
    if isel=X"4" then
      xm5 <= res_b;
    end if;
  end if;
end process;
```

Figure 4. The VHDL code implementing the structure equivalent to the one shown in the fi gure 3.

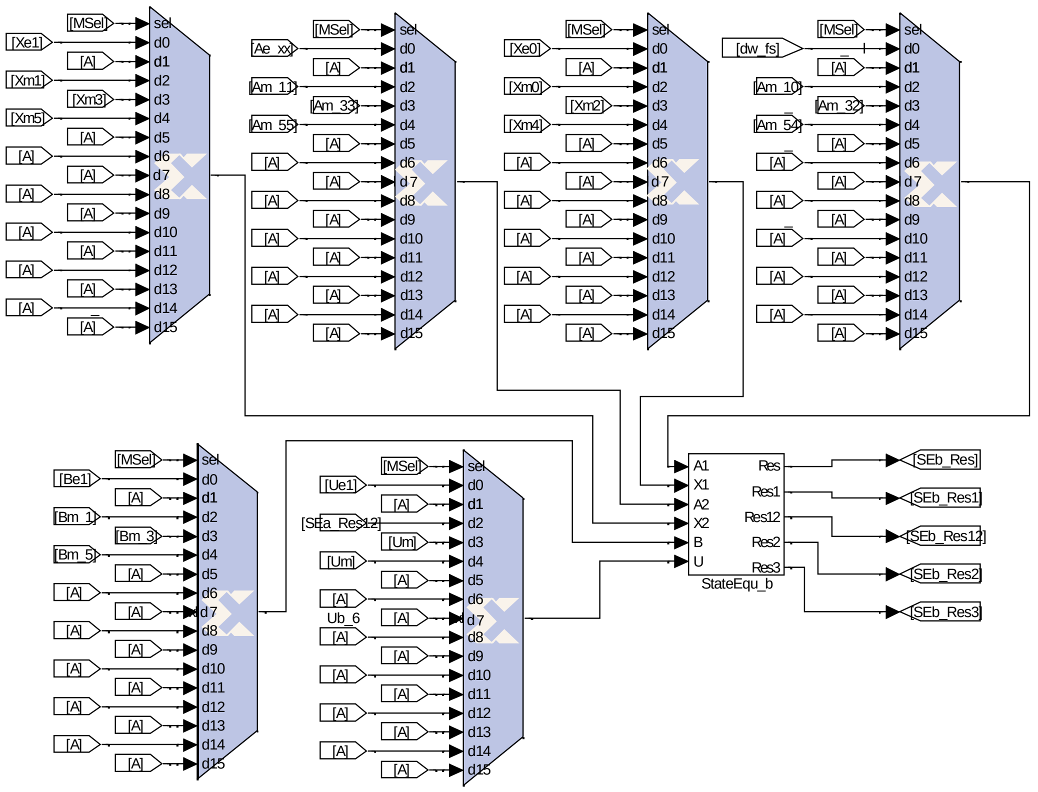

Figure 5. Single EMB block with input multiplexer and "From" blocks

```
a1_b <= dw_fs when msel = X"0" else
        (others => '0') when msel = X"1" else
        am_10 when msel = X"2" else
        am_32 when msel = X"3" else
        am_54 when msel = X"4" else
        (others => '0');
x1_b <= xe0 when msel = "0" else
        (others => '0') when msel = X"1" else
        xm0 when msel = X"2" else
        xm2 when msel = X"3" else
        xm4 when msel = X"4" else
        (others => '0');
a2_b <= ae_xx when msel = X"0" else
        (others => '0') when msel = X"1" else
        am_11 when msel = X"2" else
        am_33 when msel = X"3" else
        am_55 when msel = X"4" else
        (others => '0');
x2_b <= xe1 when msel = X"0" else
        (others => '0') when msel = X"1" else
        xm1 when msel = X"2" else
        xm3 when msel = X"3" else
        xm5 when msel = X"4" else
        (others => '0');
b_b <=  be1 when msel = X"0" else
        (others => '0') when msel = X"1" else
        bm_1 when msel = X"2" else
        bm_3 when msel = X"3" else
        bm_5 when msel = X"4" else
        (others => '0');
u_b <=  ue1_s when msel = X"0" else
        (others => '0') when msel = X"1" else
        res12_a when msel = X"2" else
        um when msel = X"3" else
        um when msel = X"4" else
        dw_1000 when msel = X"5" else
        (others => '0');
```

Figure 6. The VHDL code implementing the structure equivalent to the one shown in the figure 5.

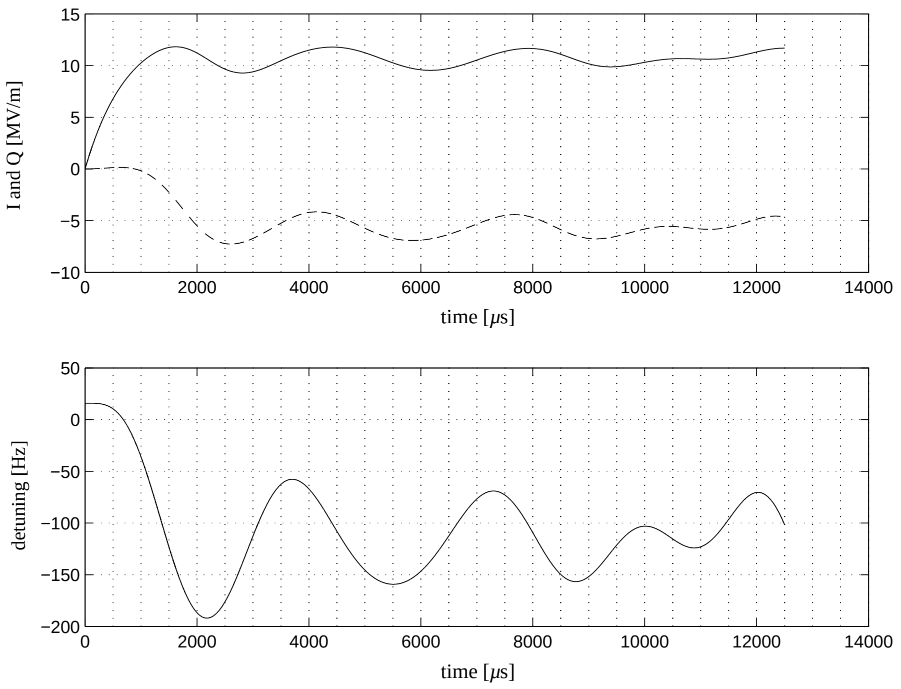

Figure 7. Output from the simulated cavity (FPGA based simulator, the Matlab model gave the same results, solid line - I, dashed line - Q, predetuning 100 Hz, step stimulus, low current)

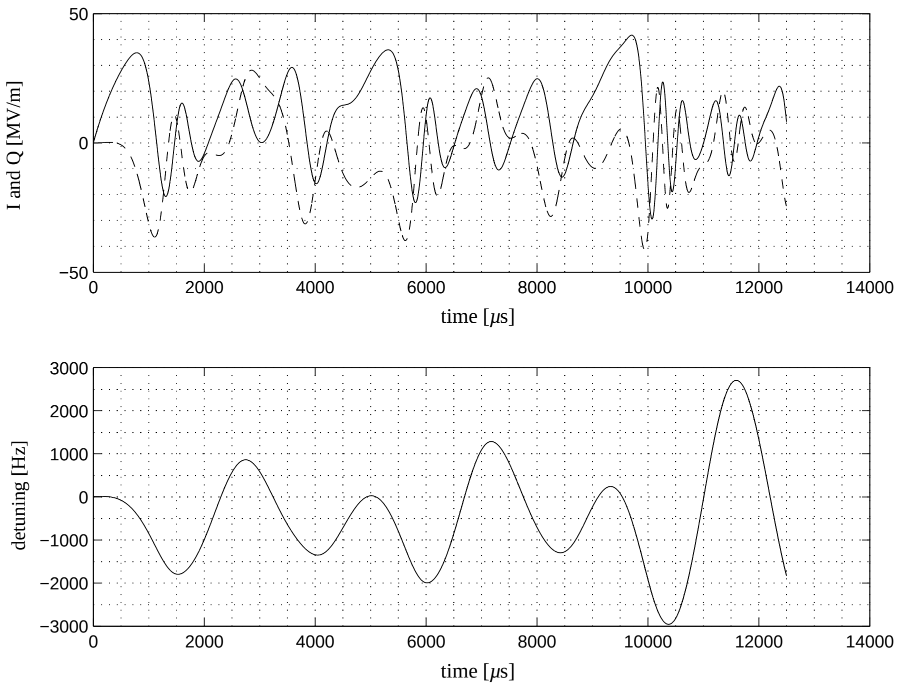

Figure 8. Output from the simulated cavity (FPGA based simulator, the Matlab model gave the same results, solid line - I, dashed line - Q, parameters chosen to emphasize the Lorentz force detuning effect, predetuning 100 Hz, step stimulus, high current)

6. RESULTS

The cavity simulator based on equations (11), (10) and (12) for N=3 (3 mechanical modes) has been successfully implemented. The simulation cycle required 51 clock pulses for maximum clock frequency above 40 MHz.

The test performed in the XSG Matlab/Simulink environment have shown, that the FPGA version of the simulator behaves similar to the floating point version implemented in the pure Matlab. The figures 7 and 8 show results of the simulation where Q component of the klystron current was changed stepwise at time 0. In the first simulation the klystron current was so low that the Lorentz force detuning moderately influenced the build-up of electric field. In the second simulation the klystron current was significantly increased to emphasize the detuning effect.

The simulation speed of the final hardware version (ca. 1.275μs per simulation cycle) allows for real-time work of the simulator.

However the software simulation speed in the XSG Matlab/Simulink was merely acceptable - the simulation of 500 000 cycles (equivalent to the 12.5 ms in the realtime) took 2000 seconds on the Pentium 4 1.7 GHz PC. The simulator is scalable, adding next cavity mechanical modes just increases the length of the program and reduces the simulation speed. The scalability is limited by the amount of resources available in chip, but (in the XSG implementation) also by the amount of inputs predefined in the multiplexer blocks.

Another problem found during the test was the superfluous graphical representation of the system in the XSG. Implementation of a typical algorithm results in a structure, where most multiplexer inputs are unused (ie. connected to the logical “zeroes” - denoted by tag [A] in the figures 3 and 5). This unused inputs are optimized out during the synthesis, but their presence significantly lengthens the compilation time and increases the amount of the memory needed to synthesise the design.

This problem is additionally boosted by the generation of ineffective VHDL code by the XSG itself (eg. each instance of the library block was represented by another component, instead of another instance of the same component). Therefore some system crashes (due to 2GB data memory limit on the Windows 2000 machines) have been experienced, unless special measures have been taken.

7. CONCLUSIONS

- The presented architecture of the Cavity Simulator allows to build an useful tool for realtime simulation of the accelerator’s cavity
- Description of the presented architecture with the XSG leads to inefficient source representation. The compilation of such design is time and memory consuming, however the final representation is efficient.
- Use of pure VHDL to describe interconnections between the blocks allows to obtain much more efficient source representation, which is much easier to mantain and synthesise.
- The simulation capabilities offered by the XSG Matlab/Simulink packet are too slow for long simulations of huge high-speed systems. Additional hardware allowing for real hardware debugging[3] may be required.

REFERENCES

1. “Low level RF,” in *TDR Part II The accelerator*, R. Brinkmann, K. Flöttmann, J. Roßbach, P. Schmüser, N. Walker, and H. Weise, eds., ch. Main linac, pp. 97–100, Kluwer Academic, Dordrecht, 2001.
2. S. Simrock, “Low level rf control system for tesla,” **Internal TESLA Documentation**, DESY, Hamburg, 2002.
3. P.Rutkowski, T.Czarski, K. Pozniak, W.Zabolotny, and R.Romaniuk, “Integrated simulation and testing environment for fpga based tesla cavity controller,” DESY, Hamburg, 2003, In preparation.