Simulating an RF Cavity in Real Time using an FPGA

Jason O. Trinidad Pérez

Inter American University of Puerto Rico, Bayamón Campus

Science Undergraduate Laboratory Internship

Fermi National Accelerator Laboratory

Batavia, IL

August 6, 2008

Mentors: Warren Schappert and Yuriy Pischalnikov

Abstract

An RF cavity simulator can be used to test control electronics. One way to make such simulators is with the use of an FPGA. We configured a Xilinx® XtremeDSP Development Kit, Virtex-4 Edition, for doing such simulations. Also, am interface was created in MATLAB using C MEX-files. This allowed easy communication with the FPGA. After the FPGA was configured, it was tested. The output was that expected but still some minor fixes have to be done. Essentially, the core part of the simulator was successfully done, but some components need to be added to make the actual simulator.

Table of Contents

Introduction

Fermilab uses RF cavities, such as the one on Figure 1, to accelerate particle beams to high energies. To test the electronics used to control these cavities, a real-time simulator is useful. One way to do such simulations is with the use of a field-programmable gate array (FPGA). An FPGA is a device containing programmable logic components, and programmable interconnects. These components can be used to perform basic logical functions, such as AND, OR, and XOR, or more complex combinational functions such as mathematical functions.

An FPGA is usually slower than an application-specific integrated circuit (ASIC), cannot handle designs as complex, and draw more power, but they can be re-programmed in the field, and are cheaper to produce. FPGAs have a wide range of applications including ASIC prototyping, computer vision, speech recognition and digital signal processing. It can also be used for simulating many systems, including RF cavities.

The hardware used in this project is a Xilinx® XtremeDSP Development Kit, Virtex-4 Edition. To configure such system to do these simulations, firmware must be written, in this case using VHDL. The code must configure the module to process a signal using its ADC inputs and output it through a DAC. Also, the code must implement a set of equations capable of simulating an RF cavity.

Figure 1. An RF cavity.

The process of getting from a physical system to a firmware implementation is shown in Figure 2.

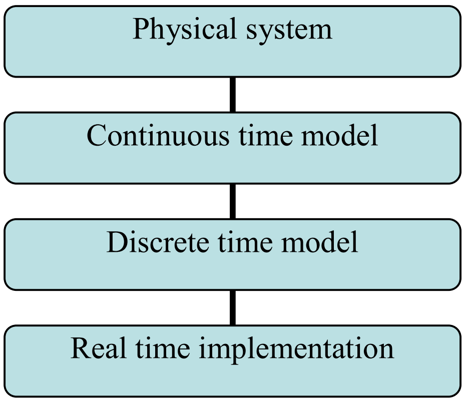


Figure 2. Process of getting from a physical system to a firmware implementation

A continuous time model of the physical system, in this case an RF cavity, consisting of differential equations representing the system's behavior is transformed into a discrete time model, consisting of difference equations. These difference equations are then implemented in firmware. Depending on these equation's coefficients, the FPGA can simulate or control various physical systems. Registers are used to change the values of theses coefficients.

An interface must be created to be able to manipulate the hardware. This interface will be created in MATLAB using C MEX-files. MEX stands for MATLAB executable. MEX-files are dynamically linked subroutines produced from C or Fortran source code that, when compiled, can be run from within MATLAB in the same way as a MATLAB M-file or built in functions.

Hardware

A Xilinx® XtremeDSP Development Kit, Virtex-4 Edition, shown in Figure 3, was used for this project. The XtremeDSP development board consists of a motherboard referred as the "BenONE-Kit Motherboard" populated with a module referred as the "BenADDA DIME-II module". The specifications are the following:

- BenONE-Kit Motherboard:
    - Supports the supplied BenADDA DIME-II module only
    - Spartan-II FPGA for 3.3V/5V PCI or USB interface
    - Host interfacing via 3.3V/5V PCI 32-bit/33-MHz or USB v1.1 interfaces
    - Status LEDs
    - JTAG configuration headers
    - User 0.1" pitch pin headers connected directly to user programmable FPGA I/O
- BenADDA DIME-II module:
    - Virtex-4 User FPGA: XC4VSX35-10FF668
    - 2 independent ADC channels: AD6645 ADC (14-bits up to 105 MSPS)
    - 2 independent DAC channels: AD9772 DAC (14-bits up to 160 MSPS)
    - Support for external clock, on board oscillator and programmable clocks
    - Two banks of ZBT-SRAM (133MHz, 512Kx32-bits per bank)
    - Multiple Clocking Options: Internal & External
    - Status LEDs
- External power supply (US Mains cable with separate UK, European or Australian mains adaptors)

• Wide ranging input (90 - 264Vac), multiple output, power supply, generating;

• +5 Volts @ 5A, +12 Volts @ 2A, -12 Volts @ 800mA

• USB v1.1 compatible cable, 2 meters long

• 5 MCX to BNC cables for connecting to the ADC / DAC and external clock connectors

• PCI Back plate and 2 screws

• 2x BNC jack to jack adaptors for use in loop back configurations

• Large blue Kit carrying case

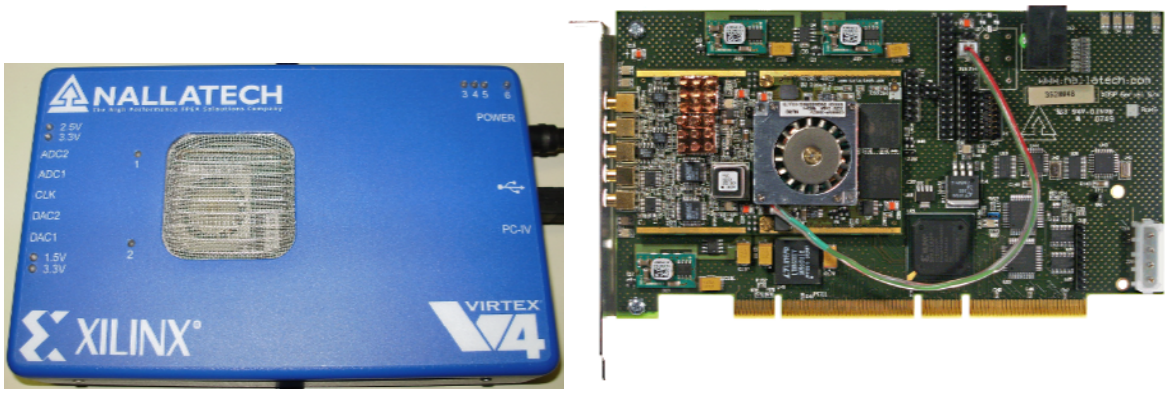


Figure 3. Xilinx® XtremeDSP Development Kit, Virtex-4 Edition. A) With case. B) Without case

Development software

In order to write, compile and run the code, the following software were used:

- Xilinx ISE Project Navigator Version 9.2i
- Nallatech Fuse Probe Version 210
- Microsoft Visual C++ 2008 Express Edition Version 3.5
- MATLAB Version 7.4.0.287 (R2007a)

Software

In order to simulate a physical system, several software components were created, as shown in Figure 3.

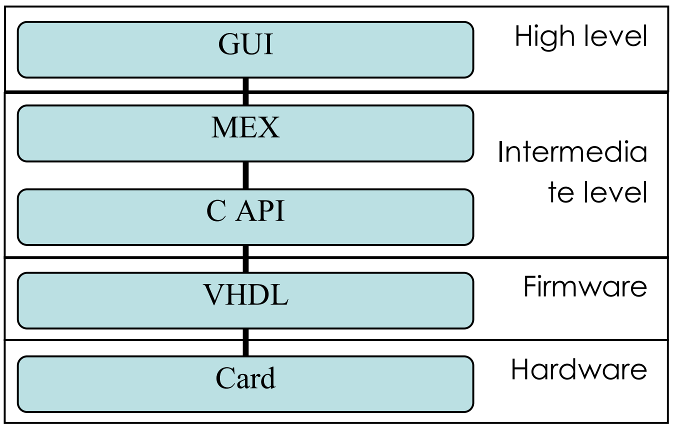


Figure 3. Software hierarchy

Firmware

To configure the card, firmware was written in VHDL. A program was written so the card would have registers to store data and be able to read from the ADC, process the input signal, and feed it through the DAC. To do this, we started using working examples provided by the manufacturer. One of the examples was an interface, which contained working registers and FIFO, and the other example contained an ADC to DAC connection which read a signal from the ADC and was outputted through the DAC. These two examples where then merged and modified. One 14-bit register called the address register and two 31-bits registers for input and output were added.

Also, a set of instructions were added to modify the data acquired from the ADC. The continuous differential equations for an RCL circuit,

$$\frac{d^2q}{dt^2}+\frac{R}{L}\frac{dq}{dt}+\frac{1}{LC}q(t)=\frac{1}{L}v(t) \qquad (1)$$

were transformed into a discrete difference equation,

$$\begin{bmatrix} x_0 \\ x_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ c_1 & c_2 & d \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ u_0 \end{bmatrix} \qquad (2)$$

suitable for numerical computing, where u_0 is the input from the ADC and y_0 is the output fed to the DAC.

The values of the coefficients can be modified to simulate many electrical systems, including RF cavities. In order to write a value to the variables a, b, and c, (will always be 0) the code was modified so that a value had to be given to the address register to specify the variable to be written and the value to be assigned to the variable had to be given to the input register. The next table shows which value of the address register corresponds to which variable:

Address register	Variable
1	a_{11}
2	a_{12}
3	a_{21}
4	a_{22}
5	b_0
6	b_1
7	c_0
8	c_1

Intermediate level

To communicate with the card, an interpreted language called DIMEscript provided by the board manufacturer is used. This language does not need to compile, and it is very fast to run a script, but has several flaws. Since this is an interpreted language, it is read line-by-line and the appropriate action is taken when the line is read. This causes errors to be found only when the line with the error is being executed. To remove these interfacing issues, a FUSE C/C++ API was used. Using libraries created by the manufacturer, a simple code was written to open the card, write and read the registers, and then close the card. Every time the code was executed, it took a fair amount of time to open and configure the card.

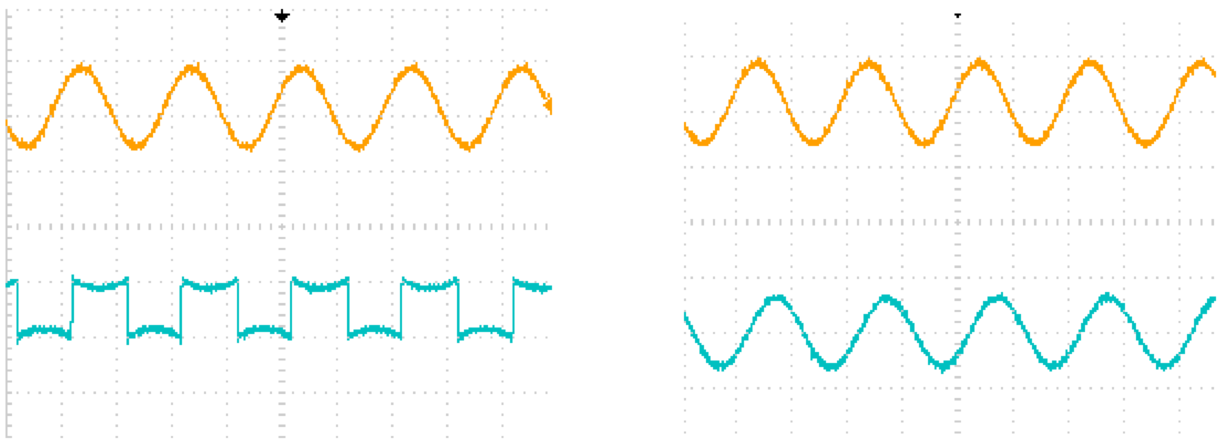
To solve this problem, MATLAB MEX-files were written. This allowed us to open the card, assign values to all of the variables and then close the card whenever desired with fair ease. The C code was divided into four parts: open card, close card, write register, read register. Each part is an individual MEX-file written in C and contained special MEX functions to pass variables to and from Matlab. Each file can be called as a function in Matlab. The open card simply opened the card and obtains the necessary handles to carry out all the necessary instructions, which are the Locate Handle and the Dime Handle, configured the FPGA with the bit files specified in the C code and returned the handles into a Matlab array. Close card has both handles as inputs and it simply closes the card. Write register has as first input the Dime Handle, second input as the address of the variable to be written, and third input as a number. Read register has as first input the Dime Handle, second input as the address of the variable, and returns the value of the output register. These functions allowed for easier I/O and to change the coefficients at anytime, without having to re-open and re-close the card.

High Level GUI

After all the functions have been defined, a graphical user interface (GUI) can be implemented in a high level language, like MATLAB. Unfortunately, we did not have enough time to develop this GUI.

Testing

To test if the hardware configuration and the interface were working correctly, we initialized all of the a's to 0 and the b's and c's to 2^{32} - 1, and the data inputted should not changed at all. To process the data, the code converted the all negative inputs into its 2's complement. But when the data was outputted, it was just treated as really large positive data thus giving an erroneous output (Figure 2.A) To solve this problem, a all the data to be outputted through the DAC was converted back from its 2's complement, thus giving the correct output.



(a) With 2's complement (b) Without 2's complement

Figure 3. Orange wave is the input and the blue wave is the output.

Then, to simulate a RF cavity, a rotation matrix (3) was used to process the input. The output expected is a decaying sine wave. The next MATLAB script was ran to assign the values to the coefficients of the difference equations:

$$\begin{bmatrix} x_0 \\ x_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 1 \\ -\sin\theta & \cos\theta & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ u_0 \end{bmatrix} \tag{3}$$

```
theta= 2*pi/10
lambda = .865
fullscale = 2^31-1

a11= uint32((fullscale)*(lambda)*(cos(theta)))
a12= uint32((fullscale)*(lambda)*(sin(theta)))
a21= uint32(mod(double(2^32)-double(a12),2^32));
a22= a11;
b0 = fullscale*(1-lambda)
b1 = 0
c0 = fullscale
c1 = 0

writereg(x(2),1,a11)
writereg(x(2),2,a12)
writereg(x(2),3,a21)
writereg(x(2),4,a22)
writereg(x(2),5,b0)
writereg(x(2),6,b1)
writereg(x(2),7,c0)
writereg(x(2),8,c1)
```

The value of lambda was varied and as it approximates 1, the time it took the sine wave to completely decay would increase. But when lambda was above 0.865, the output behaved unexpectedly. A pulse was given as an input and the output was observed.

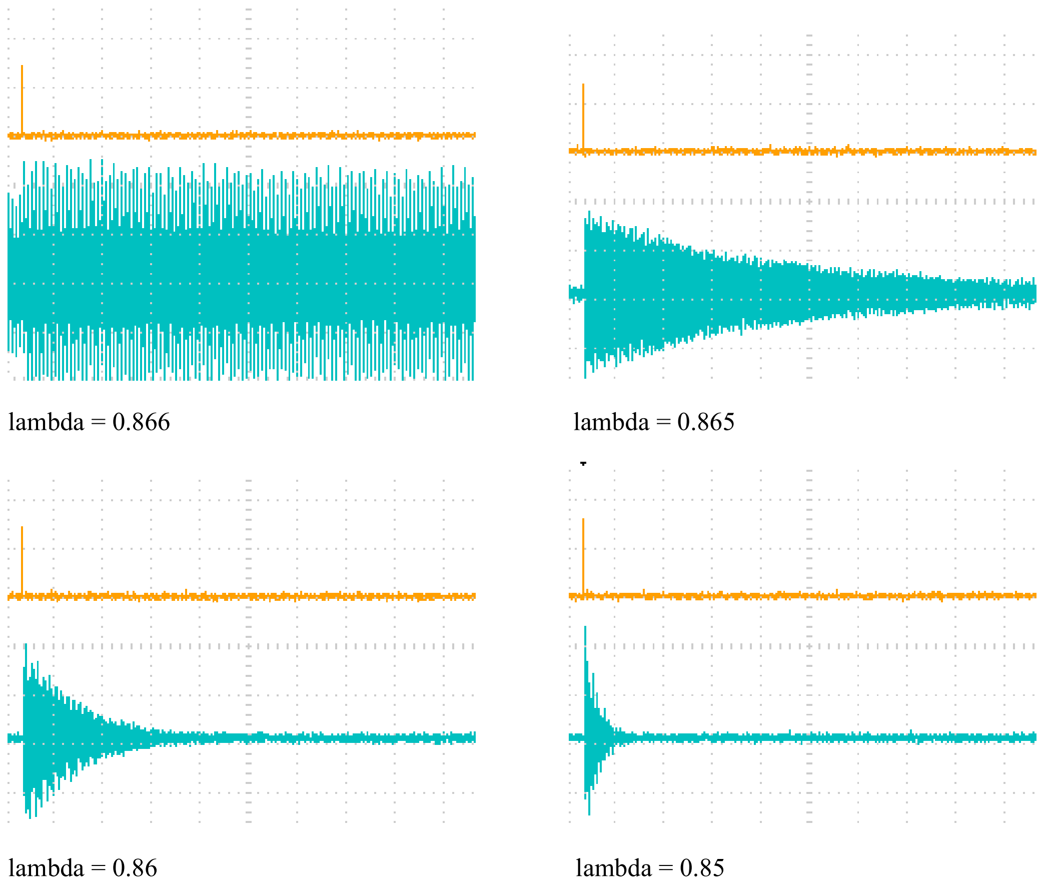


Figure 4. Decaying sine wave simulating an RF cavity.

Conclusion

The card was successfully configured to simulate a decaying sine wave even though more components need to be added for making the actual simulator. The equations were successfully implemented and tested. An interface was created in MATLAB to control the card with fair ease. Also, not only this system can simulate RF cavities, but it can be programmed to simulate many other physical systems.

Acknowledgments

I would express deep gratitude to my mentor who worked with me, Warren Schappert; Yuriy Pischalnikov for providing help to write this report; and to Carol Angarola, David Peterson and the Technical Department staff at Fermilab.

References

FUSE C/C++ API Overview. Issue 8. 17/01/2005 Nallatech Limited.

DIMEScript User Guide. Issue 3. 22/02/2005 Nallatech Limited.

XtremeDSP Development Kit-IV User Guide. Issue 1. 01/04/2005 Nallatech Limited

MEX-files Guide.
http://www.mathworks.com/support/tech-notes/1600/1605.html#intro

C++ with Matlab Tutorial. Oct 16 2007.
http://www.icaen.uiowa.edu/~dip/lecture/C++_with_Matlab.pdf

MEX-File Programming for Image Processing Using DIPimage. May, 11 2007. Luengo, C. L.
ftp://ftp.qi.tnw.tudelft.nl/pub/DIPlib/Download/docs/mex_file_programming.pdf