2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010)

Implementing Medical CT Algorithms on Stand-alone FPGA Based Systems Using an Efficient Workflow with SysGen and Simulink

Zhiqiang Que[1], Yongxin Zhu[1], Xuan Wang[2], Jibo Yu[1], Tian Huang[1], Zhe Zheng[1]
Li Yang[1], Feng Zhao[1], Yuzhuo Fu[1]
Email: quezhiqiang@sjtu.edu.cn, zhuyongxin@sjtu.edu.cn, eexwang@ust.hk,
{yujibo, huangtian, zhengzhe, yangli, zhaofeng}@ic.sjtu.edu.cn, yzfu@sjtu.edu.cn

[1] School of Microelectronics
Shanghai Jiao Tong University
Shanghai, China
[2] Department of Electronic & Computer Engineering
Hong Kong University of Science & Technology
Hong Kong, China

***Abstract*—Xilinx and Mathworks jointly proposed System Generator (SysGen) and Simulink to accelerate development of DSP (digital signal processing) style applications on Field Programmable Gate Array (FPGA) chips. However, most of developments with Simulink and SysGen end at simulation stage without complete stand-alone implementation on FPGA since these tools do not come up with sufficient IO system libraries. In this paper, we present the process of implementing a full system to reconstruct CT (computed tomography) images on FPGA using Simulink and SysGen. In this process, we solve technical issues regarding algorithm mapping, design of additional modules, system IO library and resource allocation. Our experience will be taken as useful reference for researchers working on FPGA applications and high resolution medical image applications.**

I. Introduction

In embedded computing domain, Field Programmable Gate Array (FPGA) chips have been adopted to accelerate varieties of applications, e.g. image processing, information security, medical diagnostics, and electronic design automation (EDA), since it was invented more than 20 years ago. Early speedup of 18 over general CPU for information processing was achieved [1]. Flexfilm [2], a digital movie system based on FPGA, provided a processing capability of 200 Gop/s. FPGA based systems were also applied to medical applications such as computed tomography (CT) [7], photodynamic therapy [17]. In EDA community, FPGA was also adopted to accelerate satisfiability solvers [8].

Due to the technical gap between hardware and software development, application developers with experience in high level languages still complain about the difficulty in programming at Register Transfer Level (RTL) on FPGA. Many hybrid languages were proposed for application developers to improve their productivity on FPGA. To name a few, these languages included are at least Catapult C [3], Impulse C [5], Mitrion-C [13], Spark [15], ROCCC [19], Trident [9] and MATCH [14]. These languages provide high level programming models and transparent device drivers for application developers. However, the compiled or generated RTL code from the high level languages cannot perform as efficiently as designs made directly at RTL level.

To warrantee better efficiency and improve productivity for developers at RTL level, Xilinx and Mathworks jointly proposed System Generator (SysGen) and Simulink to accelerate development of DSP (digital signal processing) style applications in early new millennium [6] [18] [12]. The workflow was evaluated in theory [19] and practice in image processing [14] and video processing [16]. However, none of them was completely implemented on physical FPGA chip or board.

In this paper, we shall present our implementation of FDK algorithm [10] on FPGA using Simulink and SysGen. We shall also show how to solve technical issues regarding algorithm mapping, design of additional modules, system IO library and resource allocation. Our practical experience in implementation of image processing application with Simulink and SysGen would be valuable for peer researchers in high performance computing as well as embedded computing domain.

978-0-7695-4108-2/10 $26.00 © 2010 IEEE
DOI 10.1109/CIT.2010.411

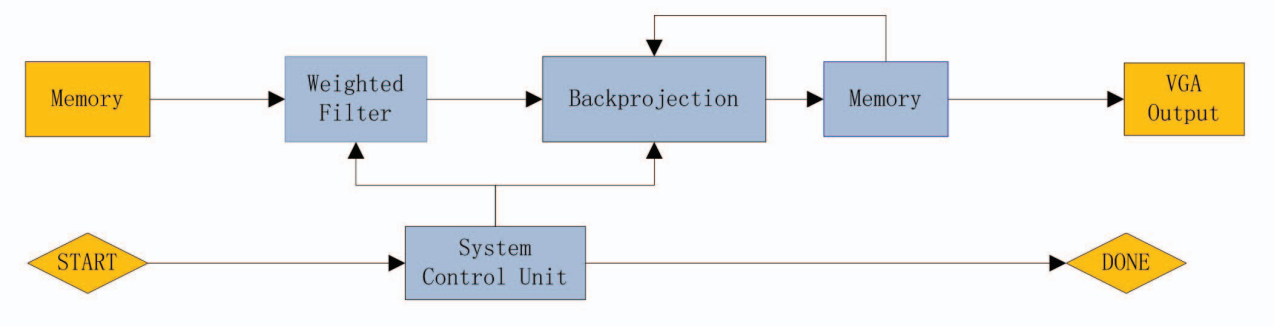


Figure 1. Schematic of the whole System

The remainder of the paper is organized as follows. In section 2, we present an overview of the system design. The details of the design and implementation will be shown in section 3. Results of hardware-software co-simulation will be presented in section 4. In section 5 we will explain the hardware-only implementation. The results and analysis will be presented in section 6, which is followed by concluding remarks.

II. Overview of the System Design and Implementation

In this paper, a cone-beam backprojection system under FDK algorithm using Simulink and Xilinx System Generator (SysGen) is implemented.

The FDK algorithm was firstly introduced by Feldkamp in 1984 [10] and was a good approximate reconstruction method of cone-beam backprojection. It could be decomposed into three major computational steps [4]. The first step is to generate weighted projection data $P'(u,v,\theta)$ by multiplying a factor related to the distance from the source to the central point of the object, just as the Equ. 1 shows.

$$P'(u,v,\theta) = P(u,v,\theta) * \frac{R}{R^2+u^2+v^2} \tag{1}$$

where (u,v) is the coordinate point on the projection plane, θ is the projection angle and R is the distance between the source and the central point of the object,

After the projection data are weighted, a one-dimensional (1D) FIR high pass filter is used to filter the projection data along the u direction. The Equ. 2 shows the details.

$$\bar{P}(u,v,\theta) = P'(u,v,\theta) * \frac{1}{2} V_{\Omega}(u) \tag{2}$$

where the $V_{\Omega}(u)$ is the function of Ram-Lak filter with Ω as the bandwidth of the image and u as the variable.

It is because that when the image is sampled, the samples of the image are very concentrated towards to the center, and very sparse near the edges. To compensate such mismatch in sampling, a filter should be applied to the projection data. Here we used the Ram-Lak filter, which is often used in filtered backprojection algorithm [2] and could easily be implemented with Fast Fourier Transform (FFT) in hardware.

The last step of the algorithm is backprojection where the image is reformed from the projections. The final value of voxel (x,y,z) or the intensity value f(x,y,z) could be achieved by summing all these filtered projection data $\bar{P}(u,v,\theta)$ of each projection angle θ with a weighting factor $\omega(x,y,\theta)$ multiplied. Here is the form for each intensity value:

$$f(x,y,z) = \sum_{\theta} \bar{P}(u,v,\theta) * \omega(x,y,\theta) \tag{3}$$

where

$$\begin{aligned} u &= R * s/(R-t) \\ v &= R * z/(R-t) \\ \omega(x,y,\theta) &= R^2 / (R-t)^2 \\ t &= x\cos\theta + y\sin\theta \\ s &= y\cos\theta - x\sin\theta \end{aligned} \tag{4}$$

According to the analysis above, the system could be divided into 4 sub-systems: (a) System IO Blocks; (b) Weighted Ramp Filter; (c) Backprojection; (d) System Control Unit.

As Figure 1 shows, the totally system is composed of an input memory interface, a weighted high-pass filter which filters the projection data, a backprojection calculation stage which reconstructs the volume from the filtered projections, a VGA interface as the image output, and a system control unit which is implemented using a Moore State Machine to control all the other components of the system.

All of the components mentioned above are built using SysGen. Though Simulink and SysGen are flexible and powerful in the field of algorithm implementation,

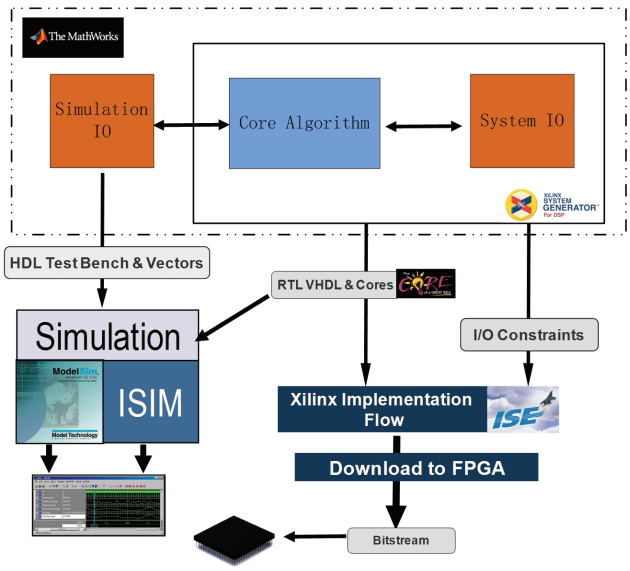


Figure 2. Design methodology

they do not have sufficient system IO blocks to interact with the FPGA peripherals. So we designed a system IO library with some commonly used IO blocks. After these IO blocks are inserted into the system, the user can use SysGen to generate the bit stream of a complete system for FPGA, which not only includes the processing blocks, but also includes some FPGA peripheral interface controllers. Figure 2 shows the whole design flow. Our methodology would improve the efficiency to verify complex systems in a complete FPGA environment, as traditional hardware software co-simulation or hardware-loop verification is very slow.

In a nutshell, we explore the parallelism in the FDK algorithm by implementing it with a flexible design based on a hardware pipeline to enable pipelining of the filtering, division calculation and accumulation under a delicate pipeline control unit. The pipeline is also scalable with multiple channel enabled to process streaming data in concurrent coordination.

In later sections, we shall further illustrate the effectiveness of our design with the final speedup of approximate 100 times over a software implementation on a 2.4GHz Pentium.

III. Details of Design and Implementation

A. Design and Implementation of the Weighted Filter

A look up table is used here to generate the weighted factor of R / $(R^2+u^2+v^2)$. After the projection data are weighted, they will be passed to the Ramp Filter. This filter is actually a one-dimensional (1D) convolution and can be easily implemented using FFT. One 256-point 16bit FFT block and one 256-point 16bit iFFT block are used here. As the input data range of the Xilinx FFT model is among (-1 ~ 1), a shift for the data is needed here.

B. Design and Implementation of the Backprojection Calculation Stage

After the ramp filtering, the projection data are weighted correctly, which will be fed into the next stage of image reconstruction named backprojection. Figure 3 shows the whole architecture of our implementation of the backprojection calculation in the form of Simulink models.

A pipelined array divider is carefully designed here. It is composed of only add and subtract blocks which are very mature in SysGen tool. All the other components are also designed to be pipelined to make sure the system could calculate one memory address of the pixel for filtered projection value in one clock cycle. The corresponding projection data are then pushed out and accumulated to the value for the same pixel reconstructed from the previous projection data. Two Block RAM (BRAM) are used here to store the reconstructed pixel data to solve the overlapping issue between the data fetching and accumulation processing [11].

After summing the projection data from all the projection angles, the final pixel data in the memory will not be updated anymore and a system done signal will be generated to make the state machine into an idle status. The system done signal will also drive a led on the FPGA board.

C. Design and Implementation of the System Control Unit

A Moore finite state machine (FSM) is designed to control the whole system. After a one-cycle pulse signal is fed to it as the start signal, the FSM generates the control signals for all the other components of the system, as Figure 4 shows. The Start Model, which is in orange color, is implemented using the Xilinx Gateway In block and will be finally mapped to a button on the FPGA board with a specified IOB location constraint.

D. System IO library

SysGen provides easy graphic interface and workflow to create algorithms conveniently and quickly, but it does not provide sufficient IO system libraries. In our system, a system IO library with some commonly used blocks are designed to make it convenient for creating a complete and complex system for FPGAs using SysGen. All these system IO blocks that are in the orange color are designed using SysGen low-level

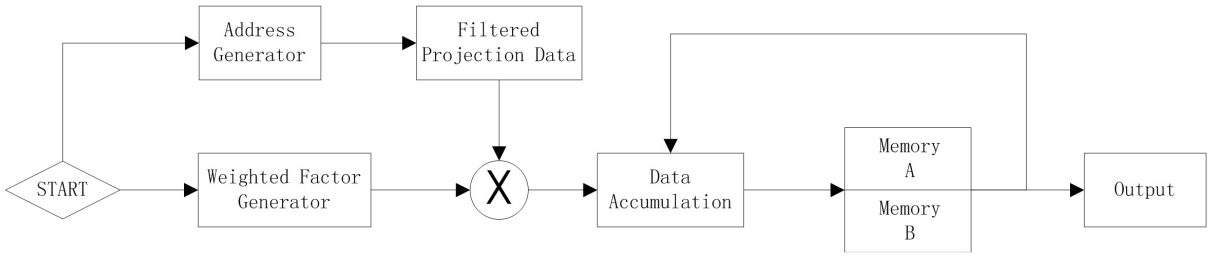


Figure 3. Overview of Backprojection Model

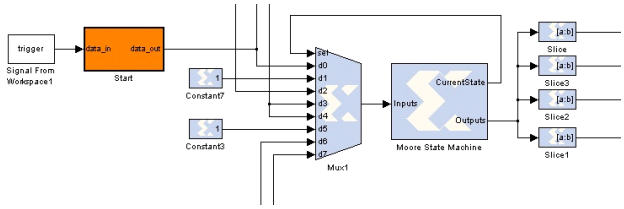


Figure 4. Overview of system control unit

blocks, Gateway In and Gateway Out blocks. Figure 5 shows the schematic of a VGA block. All the specified IOB location constraints of FPGA peripherals have been set in the Xilinx Gateway In and Gateway Out blocks according to the real FPGA peripherals pin assignment. The SysGen can generate HDL model of the FPGA peripheral controllers with the corresponding constraints which are translated into .xcf file under the project directory. At last they will be integrated into the whole system when generating the final physical configuration file of the complete system for FPGAs.

IV. Hardware-Software Co-simulation

After the whole system is created using Simulink and SysGen, a hardware co-simulation block can be generated. Generating a model for hardware co-simulation includes operations such as producing HDL code and netlists and invoking the Xilinx CORE Generator tool. After the SysGen models are generated, a compilation script is invoked, which invokes the necessary FPGA tools to produce a configuration file for the FPGA. This script will also create a new library and add a co-simulation block that is parameterized with information from the original subsystem. It can be driven by either Xilinx fixed-point data type or by Simulink double-signal data types. It behaves as a normal Simulink block and can be simulated in the Simulink with other Simulink blocks simultaneously.

In our co-simulation system, input and output data are combined with Matlab workspace, making the system convenient to debug. When the design is simulated in Simulink, the processing data from MATLAB workspace will be fed to the FPGA via JTAG connection. Then the results will be calculated in the real hardware and can be fed back to the MATLAB workspace and shown. This makes the hardware testing and verification of the interesting part of the system more convenient.

The system IO blocks are not included in co-simulation processing because some FPGA peripherals are driven by a much lower clock than the processing model, which will dramatically reduce the simulation performance and waste lots of time. The final simulation results obtained from the co-simulation agreed to the ones got from normal Simulink simulation. Figure 6 shows the simulation results.

V. Hardware-only implementation

We have managed to reconstruct a 132*132 Shepp-Logan phanton image using FDK algorithm on the XUPV2P Pro board bearing a Virtex 2 Pro FPGA and on the XUPV5-LX110T board bearing a Virtex 5 FPGA. Two channel data-stream processing are implemented only on XUPV5-lX110T board as the limitation of FPGA resources on V2P Pro. The projection data are generated from MATLAB and stored into the memory on the FPGA board before the system begins. These initial data were stored in SRAM on XUPV5-LX110T, but in on-chip BRAM of XUPV2P Pro because there is no SRAM on this board. The final reconstructed image is shown on the VGA displayer. For a better displaying, every pixel of the image is displayed as 4 pixel on the displayer. The VGA port on the V2P Pro is used while the DVI port of on the XUPV5-LX110T is used.

VI. Results and Discussion

The final output data width is 13 bits with 5 fraction bits. Only top 6 bits of the pixel value are passed to the VGA block and displayed in the screen, as Figure 7 shows. The final performance results and

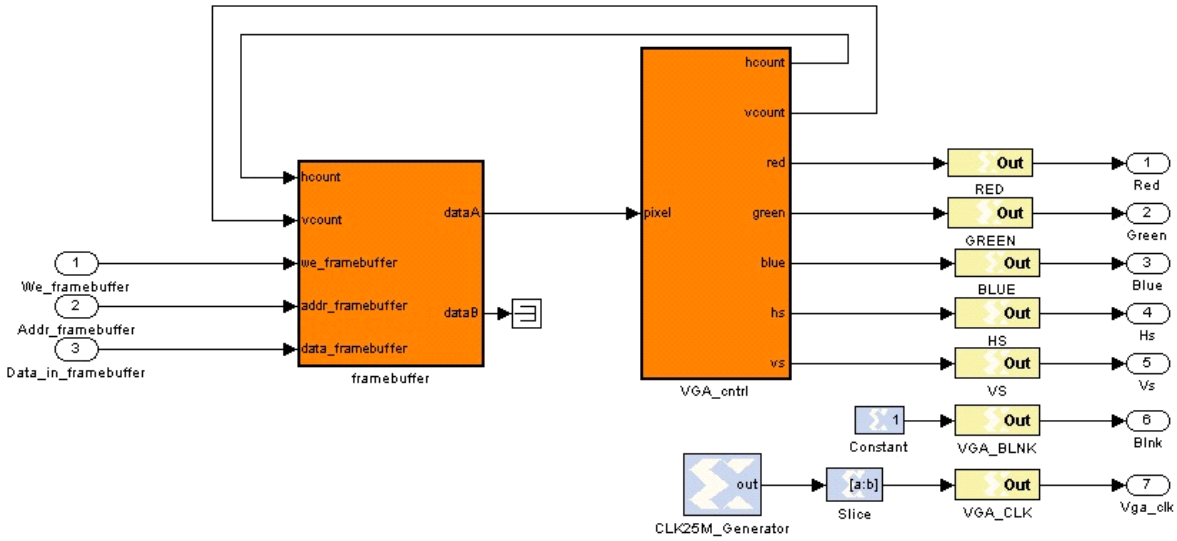
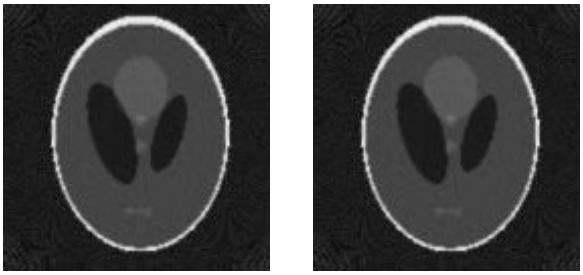


Figure 5. Schematic of VGA interface



(a) Simulink Simulation Result (b) Co-simulation Result

Figure 6. Simulation Results

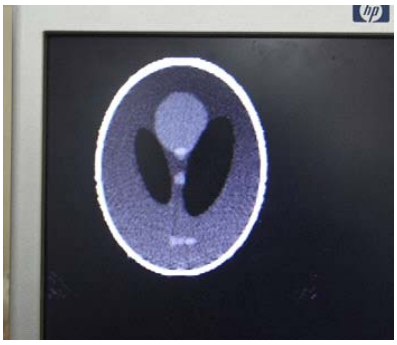


Figure 7. VGA output

FPGA resource utilizations can be found in Table I and Table II.

As there are only 64 DSP48e blocks on the XUPV5-LX110T board, some of the multipliers have to be implemented using FPGA LUTs and flip-flops, which limits the system's performance on Virtex 5. If some other FPGAs are used, such as the SX50T with larger on-chip RAM capacity and more DSP48e, the performance will be better.

Table I
PERFORMANCE AND FPGA RESOURCE UTILIZATION OF ONE CHANNEL SYSTEM FOR XUPV2P PRO

FPGA Chip	XC2VP30-7FF896
Maximum Frequency	100.1 MHz
Reconstruction Time	72.8 μs
Flip-flops	11,108 (40%)
4 input LUTs	7,881(28%)
Block RAMs	121 (88%)
MULT18X18s	56 (41%)

VII. CONCLUSION

In this paper, we evaluated the methodology to develop applications using Xilinx SysGen and MathWorks Simulink by implementing FDK algorithm for computed tomography on physical FPGA chip and board. Besides the intuitive mapping of tasks onto the workflow model described in Simulink, we carefully designed the system IO library and System Control Unit which were ignored in previous studies. We verified our design by running co-simulation on FPGA whose data were fed by Simulink. We further implemented one FDK algorithm channel on the XUPV2P Pro board

Table II
PERFORMANCE AND FPGA RESOURCE UTILIZATION OF TWO CHANNEL SYSTEM FOR XUPV5-LX110T

FPGA Chip	xc5vlx110t-1FFG1136
Maximum Frequency	100.3 MHz
Reconstruction Time	36.4 μs
Flip-flops	15,035 (21%)
6 input LUTs	14,946(21%)
Block RAMs	115 (77%)
DSP48e	64 (100%)

and two FDK algorithm channels on the XUPV5-LX110T board. Both the co-simulation result and the VGA outputs from the FPGA boards indicated the correctness of our design. Our implementation shows that the workflow is productive and effective to develop image processing applications in terms of ease of use and coding speed. Moreover, the performance of the hardware implementation significantly outperforms CPU based solution with a speedup of over 100x.

VIII. ACKNOWLEDGEMENTS

This paper is partially sponsored by the National High-Technology Research and Development Program of China (863 Program) (2009AA012201) and the Shanghai International Science and Technology Collaboration Program (09540701900).

REFERENCES

[1] A. J. Elbirt, C. Paar. An fpga implementation and performance evaluation of the serpent blocks cipher. *Eighth ACM/SIGDA International Symposium on FPGAs*, pages 33–40, February 2000.

[2] Amilcar do Carmo Lucas, Sven Heithecker, and Rolf Ernst. Flexwafe - a high-end real-time stream processing library for fpgas. *Proceedings of the 44th annual Design Automation Conference*, June 2007.

[3] Catapult C. http://en.wikipedia.org/wiki/catapult_c.

[4] H. Turbell. Cone-beam reconstruction using filtered backprojection. *Ph.D. dissertation*, 672, February 2001.

[5] Impulse C. http://en.wikipedia.org/wiki/impulse_c.

[6] James Hwang, Brent Milne, Nabeel Shirazi and Jeffrey D. Stroomer. System level tools for dsp in fpgas. *Field-Programmable Logic and Applications*, 2147/2001:534–543, January 2001.

[7] Jianchun Li, Christos Papachristou, and Raj Shekhar. An fpga-based computing platform for real-time 3d medical imaging and its application to cone-beam ct reconstruction. *The Journal of Imaging Science and Technology*, 2005.

[8] John D. Davis, Zhangxi Tan, Fang Yu and Lintao Zhang. A practical reconfigurable hardware accelerator for boolean satisfiability solvers. *Proceedings of the 45th annual Design Automation Conference*.

[9] Justin L. Tripp, Maya B. Gokhale, and Kristopher D. Peterson. Trident: From high-level language to hardware circuitry. *IEEE Computer Society*, 2007.

[10] L. A. Feldkamp, L. C. Davis, and J. W. Kress. Practical cone beam algorithm. *J. OPT. SOC. AM.*, (1):612– 619, 1984.

[11] M. Leeser, S. Coric, E. Miller, H. Yu and M. Trepanier. Parallel-beam backprojection: An fpga implemenation optimzied for medical. *The Journal of VLSI Signal Processing*, 39:295–311, March 2005.

[12] Mathworks Inc. Matlab@ student version: Learning simulink 4. 2001.

[13] Mitrionics AB Inc. The mitrion processor. *Product Overview*, 2005.

[14] P. Banerjee, N. Shenoy, A. Choudhary, S. Hauck, C. Bachmann, M. Chang, M. Haldar, P. Joisha, A. Jones, A. Kanhare, A. Nayak, S. Periyacheri, and M. Walkden. Match: A matlab compiler for configurable computing systems. *IEEE Computer Magazine*, 1999.

[15] Spark. http://mesl.ucsd.edu/spark/.

[16] T. Saidani , D. Dia, W. Elhamzi, M. Atri, and R. Tourki. Hardware co-simulation for video processing using xilinx system generator. *Proceedings of the World Congress on Engineering (WCE) 2009*, I:295–311, July 2009.

[17] W. C. Y. Lo, K. Redmond, J. Luu, P. Chow, J. Rose and L. Lilge. Hardware acceleration of a monte carlo simulation for photodynamic therapy treatment planning. *Journal of Biomedical Optics*, 14(1), January/February 2009.

[18] Xilinx Corp. Xilinx system generator for simulink. *Version 2.1*, 29:411–419, April 2001.

[19] Z. Guo, B. Buyukkurt, W. Najjar and K. Vissers. Optimized generation of data-path from c codes for fpgas. *ACM/IEEE Design Automation and Test Europe (DATE)*, pages 112 – 117, 2005.