

THE THEORETICAL DEVELOPMENT OF A NEW HIGH SPEED  
SOLUTION FOR MONTE CARLO RADIATION TRANSPORT COMPUTATIONS

A Thesis

by

ALEXANDER SAMUEL PASCIAK

Submitted to the Office of Graduate Studies of  
Texas A&M University  
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2005

Major Subject: Health Physics

THE THEORETICAL DEVELOPMENT OF A NEW HIGH SPEED  
SOLUTION FOR MONTE CARLO RADIATION TRANSPORT COMPUTATIONS

A Thesis

by

ALEXANDER SAMUEL PASCIAK

Submitted to the Office of Graduate Studies of  
Texas A&M University  
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,  
Committee Members,

Head of Department,

John R. Ford  
Leslie A. Braby  
John W. Poston, Sr.  
Raytcho Lazarov  
William E. Burchill

December 2005

Major Subject: Health Physics

## ABSTRACT

The Theoretical Development of a New High Speed  
Solution for Monte Carlo Radiation Transport Computations. (December 2005)

Alexander Samuel Pasciak, B.S., University of Washington

Chair of Advisory Committee: Dr. John R. Ford

Advancements in parallel and cluster computing have made many complex Monte Carlo simulations possible in the past several years. Unfortunately, cluster computers are large, expensive, and still not fast enough to make the Monte Carlo technique useful for calculations requiring a near real-time evaluation period. For Monte Carlo simulations, a small computational unit called a Field Programmable Gate Array (FPGA) is capable of bringing the power of a large cluster computer into any personal computer (PC). Because an FPGA is capable of executing Monte Carlo simulations with a high degree of parallelism, a simulation run on a large FPGA can be executed at a much higher rate than an equivalent simulation on a modern single-processor desktop PC. In this thesis, a simple radiation transport problem involving moderate energy photons incident on a three-dimensional target is discussed. By comparing the theoretical evaluation speed of this transport problem on a large FPGA to the evaluation speed of the same transport problem using standard computing techniques, it is shown that it is possible to accelerate Monte Carlo computations significantly using FPGAs. In fact, we have found that our simple photon transport test case can be evaluated in excess of 650 times faster on a large FPGA than on a 3.2 GHz Pentium-4 desktop PC running

MCNP5—an acceleration factor that we predict will be largely preserved for most Monte Carlo simulations.

## ACKNOWLEDGEMENTS

I would like to thank my graduate advisor, Dr. John Ford, for his support of this project. Without his support, this research would never have started. I would also like to thank several other members of the Texas A&M Nuclear Engineering staff for their support including: Dr. Daniel Reece, Dr. Leslie Braby and Dr. John W. Poston, Sr.

Funding for this research was provided by the Nuclear Engineering Education Research (NEER) Grant Program. U. S. Department of Energy Grant No. DE-FG07-021D14329

## TABLE OF CONTENTS

	Page
ABSTRACT .....	iii
ACKNOWLEDGEMENTS .....	v
TABLE OF CONTENTS .....	vi
LIST OF FIGURES .....	viii
LIST OF TABLES .....	ix
CHAPTER	
I    INTRODUCTION .....	1
II   BACKGROUND .....	3
FPGA Background .....	3
Development Overview .....	6
III  TEST CASE ASSUMPTIONS .....	8
IV  IMPLEMENTATION METHODS .....	11
Random Number Generation .....	11
Cross Section Retrieval .....	14
Logarithms .....	21
Exponentials .....	25
Scattering Distributions .....	27
Overall Layout .....	29
V    RESULTS .....	33
Testing Methods .....	33
Monte Carlo Results .....	37
VI   CONCLUSIONS .....	43
VII  FUTURE WORK .....	44
REFERENCES .....	45

	Page
VITA .....	47

## LIST OF FIGURES

FIGURE		Page
1	A schematic showing the internals of an FPGA .....	5
2	A circuit schematic describing the Mersenne Twister FPGA implementation .....	15
3	The total macroscopic cross sections for aluminum, as it is stored on the FPGA .....	17
4	The exact EPDL97 cross sections and associated errors (red) compared with the cross sections as stored on the FPGA (blue) .....	18
5	A diagram of the interpolation scheme used for cross section retrieval .....	20
6	A schematic depicting the log preprocessing stage .....	24
7	A schematic depicting the log <sub>2</sub> evaluation stage .....	26
8	Dataflow of the overall algorithm, showing the ideal amount of ‘work’ completed per clock cycle .....	32
9	A flowchart describing the simulation process .....	36
10	The programmed FPGA with one Monte Carlo transport module .....	39

## LIST OF TABLES

TABLE		Page
1	The resources consumed on the Xilinx Virtex-II Pro 100 FPGA by the Monte Carlo radiation transport hardware design .....	38
2	The flux tallies for MCNP-5 vs. the flux tallies reported for the FPGA hardware design .....	41

## CHAPTER I

### INTRODUCTION

Currently, there are two widely used general computing methods for the expedited execution of highly complex scientific computations: standard personal computer (PC) based cluster computing and special purpose supercomputing. Twenty years ago, the idea of using home PCs for scientific computations would have seemed absurd, as home PCs (even in large numbers) could not match the speed of proprietary special purpose supercomputers. Today, quite the opposite is true. The demand for home PC's drives the computing industry and so cluster computing using off the shelf PC hardware has become the standard solution for scientific computation. This can be attributed largely to the high availability and low price associated with standard PC hardware. It is still true that special purpose machines can outperform standard PC hardware, but the extremely limited availability and high cost is usually a sufficient deterrent to their wide use.

Ignoring cost and availability constraints, the most efficient evaluation method for a computationally-intensive problem, which is based on a relatively fixed algorithm, is to utilize the power of a custom fabricated Application Specific Integrated Circuit (ASIC). Unlike a standard microprocessor, ASICs are not driven by software; instead, they are manufactured to perform one specific calculation. The advantage of using ASICs for high speed computation is that they can have a much higher work rate than a

---

This thesis follows the style of the Health Physics Journal.

standard microprocessor. The disadvantage of using an ASIC is that, once manufactured, the computation that it performs can never be altered. Since the algorithms used in Monte Carlo radiation transport computations are problem specific, it is unlikely that ASICs are flexible enough to be used as an aide to accelerate the speed of Monte Carlo computations.

A Field Programmable Gate Array (FPGA) is an integrated circuit which is capable of performing computations with nearly all of the same speed benefits of an ASIC. However, FPGAs have one important advantage over ASICs—they are reprogrammable. The computation that an FPGA performs can be completely changed in a fraction of a second by reprogramming the device. Since the ability to reprogram an FPGA makes it very flexible, it is likely that the same type of FPGA can be used in a wide range of industries, from cellular telephones to automobiles. In turn, this makes it an off-the-shelf, high-availability, and cost-effective device. FPGA technology has been around for a long time; however, the size and complexity of available devices is only just reaching a point in which they can be utilized to accelerate algorithms as complex as Monte Carlo radiation transport.

## CHAPTER II

### BACKGROUND

#### *FPGA Background*

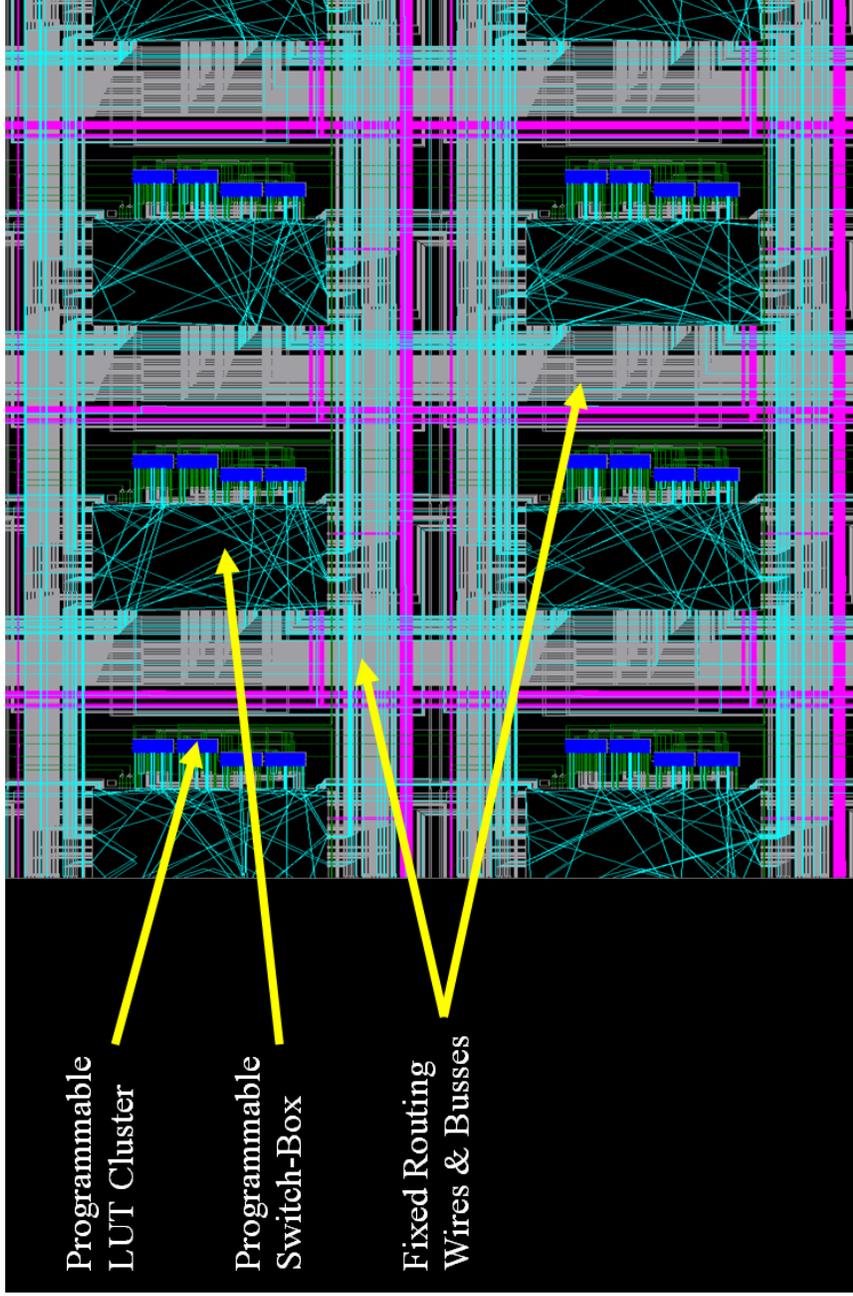
A field programmable gate array (FPGA) is a specialized computer chip composed of an array of small memory elements which can be reprogrammed to mimic the behavior of different elementary math and logic functions. Connections between the small memory elements in the array can be altered using pass transistor switches allowing many memory elements to work together to compute complex mathematical functions. A large FPGA has more than 50,000 of these reprogrammable memory elements, several hundred dedicated multiplier blocks, a large amount of onboard data storage elements, and the capability of multi-tera operation performance. FPGAs can be programmed to execute almost any algorithm, but they are not programmed from standard computer code. Instead, complete hardware designs (generally at the gate level) are used as logic patterns to program the FPGA. Once the FPGA has been programmed, it behaves in exactly the same way as an ASIC which has been manufactured with a particular algorithm in mind.

Classically, there have been two major types of FPGAs available in industry. The first is based on Static Random Access Memory (SRAM), the second is based on anti-fuse technology. Anti-fuse technology utilizes fuse like elements which are electrically “blown” when the FPGA is programmed. The fuses that are not “blown” during programming connect specific logic units within the FPGA to perform a given operation. FPGAs using anti-fuse technology are quite different than the more popular

SRAM based FPGAs of today. Anti-fuse FPGAs can be programmed only once, where SRAM based FPGAs can be programmed and reprogrammed in indefinite number of times. Look-up tables (LUTs) form the core of an SRAM based FPGAs logic reproduction ability. Standard LUTs are capable of mimicking any logic function which has the same number of inputs as the LUT. For instance, a 4-input LUT stores truth table-like data corresponding to the operation of any 4-input logic function. Multiple LUTs can be combined to perform more complex computations.

SRAM based FPGAs have a large number of LUTs, each of which can be programmed individually to perform a specific function. Most complex algorithms, however, will require the combined use of thousands or more LUTs to perform an evaluation. Methods for efficient and yet programmable routing of signals between LUTs is a highly critical component of modern SRAM based FPGAs. Often small clusters of LUTs are arranged in a square lattice formation with horizontal and vertical routing wires separating each LUT cluster within the lattice. Each LUT cluster is paired with a programmable SRAM based switchbox, capable of changing the direction of signals running on the fixed routing wires, as well as connecting signals carried by specific routing wires to appropriate LUTs. See Figure 1 for an illustration of these components.

Typically, FPGAs can be found mounted on computer interface boards, allowing a standard PC to stand as a host to an FPGA. While mounted on a computer interface board, the FPGA can act as an extremely powerful co-processor. Unprocessed data will be fed to the FPGA from the host PC, and processed data will be fed from the FPGA



**Fig. 1.** A Schematic showing the internals of an FPGA.

back to the host PC for hard-drive storage. Unlike standard computers, FPGAs are capable of parallelizing even the most serial of algorithms. The secret to the speed advantage of a single FPGA over a standard computer can be reduced to one important fact: an FPGA is capable of performing orders of magnitude more work per clock cycle than a standard computer. A useful and in-depth look at FPGAs and reconfigurable computing is given by Compton and Hauck (Compton and Hauck 2002 ; Hauck 1998).

### *Development Overview*

The knowledge of software programming skills has spread to the point where it is currently being taught in high schools. Unfortunately, the same cannot be said for hardware development skills. With today's high-level programming languages, it is unlikely that all but the most experienced of programmers will be able to apply their programming skills to advanced, FPGA-based hardware development. While there are a small number of commercial programs on the market capable of converting certain C and/or Java codes into FPGA compatible hardware designs, these programs are limited to simple state machines and combinational logic, and are unlikely to be useful to a programmer designing hardware for complex operations such as Monte Carlo transport. Lower-level implementation and description of hardware designs using a hardware definition language (HDL) is likely to be the only method versatile enough to implement these kinds of algorithms.

There are essentially two main HDLs widely available to programmers, Verilog HDL and VHDL. A detailed overview of Verilog is given by Palnitkar (Palnitkar 2003).

Similarly, the VHDL language is detailed by Yalamanchili (Yalamanchili 2000). They are similar languages, using only slightly different syntax. These languages do not behave like a typical computer programming language—instead they simply serve to describe in a “text” form the various items from a typical circuit schematic. For instance, instead of variables such as an ‘int’ in the C computer language, variable-like items called ‘wires’ are used to connect different functional blocks. When the code is synthesized for FPGA implementation, each definition of a wire will be manifested in exactly that way (a physical wired connection) on the FPGA. Learning these languages is typically easy, provided the user has sufficient understanding of how the underlying hardware is represented by the HDL.

## CHAPTER III

### TEST CASE ASSUMPTIONS

To date, Monte Carlo radiation transport computations have never been attempted using FPGA-based reconfigurable computing techniques. The development of the initial hardware algorithms is difficult and tedious since very few of the principles that are currently applied to software-based Monte Carlo code development can be applied to the hardware-based counterpart. The objective of this research is to verify and categorize the possible degree of speed increase that can be achieved by using reconfigurable, FPGA-based computing techniques to evaluate Monte Carlo radiation transport problems. An assumption is made that if the evaluation of a relatively simple transport problem on an FPGA is significantly faster than the evaluation of the same problem using a standard PC, then that speed increase will likely be preserved for more complex Monte Carlo evaluations. Therefore, to reduce the complexity of the initial hardware designs necessary, a Monte Carlo photon transport simulation on a target with simple geometric and material properties has been the subject of this thesis research.

The simple test situation consisted of an isotropic 250 keV photon point source in an infinite medium of aluminum (Al). The tallies that were used were spherical flux tallies (number of photons crossing a boundary) at intervals 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20 and 25 cm away from the source. The test algorithm uses a completely internalized design, meaning that the FPGA does all of the work internally, leaving the host computer to wait only for the requested number of histories to be completed and tallies

to be output. When the FPGA is programmed, all cross section data, scattering data, random number generator initial states, tally points, and geometrical data are included as part of the hardware design downloaded onto the FPGA.

Monte Carlo based photon transport computations can be extremely complex if no approximations are made and if all secondary particles are tracked. Since the purpose of this preliminary research was only to show the viability of FPGA-based Monte Carlo particle transport, secondary particles were ignored for simplicity. Ignoring secondary particles for low-Z test materials still allows a fairly accurate simulation. It should be noted that it would be possible to introduce Bremsstrahlung and fluorescence effects into our existing methods, so long as approximations can be used to eliminate full electron transport computations.

For our current methods, interaction cross sections and scattering modifying factors are those given by Lawrence Livermore National Laboratory's evaluated photon data library EPDL97 by Cullen (Cullen 1997a). Incoherent scattering distribution functions are given by the product of the standard differential Klein-Nishina formula and the incoherent scattering function as described by Hubbell (Klein and Nishina 1929 ; Hubbell et al. 1975). Coherent scattering distribution functions are given by the product of the Thompson scattering formula and the square of the coherent scattering form factor which is also described by Hubbell (Hubbell et al. 1975). Equations 1 and 2 show the general relationships for incoherent and coherent scattering, respectively, where the scattering functions and form factors are functions of the momentum transfer ( $x$ ) and the atomic number ( $Z$ ).

$$\frac{d\sigma_{INCOHERENT}(E, \theta, Z)}{d\Omega} = \left[ \frac{d\sigma_{KN}(E, \theta)}{d\Omega} \right] \bullet S(x, Z), \quad (1)$$

$$\frac{d\sigma_{COHERENT}(\theta)}{d\Omega} = \left[ \frac{d\sigma_T(\theta)}{d\Omega} \right] \bullet (F(x, Z))^2, \quad (2)$$

where,

$$\frac{d\sigma_{kn}(\theta)}{d\Omega} = \frac{r_e^2}{2} [1 + k(1 - \cos \theta)]^{-2} \left[ 1 + \cos^2 \theta + \frac{k^2(1 - \cos \theta)^2}{1 + k(1 - \cos \theta)} \right], \quad (3)$$

$$\frac{d\sigma_t(\theta)}{d\Omega} = \frac{r_e^2}{2} (1 + \cos^2 \theta), \quad (4)$$

$$x = \frac{\sin\left(\frac{\theta}{2}\right)}{\lambda(A)}. \quad (5)$$

Equation 3 is the general differential Klein-Nishina cross section, where  $r_e$  is the classical electron radius and  $k$  is the incident photon energy in units of electron rest mass. Equation 4 is the differential Thompson scattering cross section. Equation 5 is the momentum transfer as a function of scattering angle and photon wavelength in units of angstroms.

In addition to ignoring secondary particles, we have attempted to simplify our test case further by assuming that the source produces photons which have energies less than 1.022 MeV, insuring that the pair and triplet production cross sections for our energy range are not needed. Therefore, our total macroscopic interaction cross section is given by:  $\mu_{total} = \mu_{incoherent} + \mu_{coherent} + \mu_{photoelectric}$ .

## CHAPTER IV

### IMPLEMENTATION METHODS

#### *Random Number Generation*

Random number generation is the core of any Monte Carlo simulation, and the efficient generation of high quality random numbers in hardware is imperative to the success of this project. Classically, even the most advanced Monte Carlo radiation transport codes have relied on one of the least sophisticated pseudo-random number generation algorithms, the Linear Congruential Generator (LCG). The general form of the LCG is:

$$X_k = (A * X_{k-1} + C) \text{ mod } M \quad (6)$$

Where,  $X_k$  is the current random number in the series,  $X_{k-1}$  is the previous random number in the series and  $A, C$ , and  $M$  are constants. Careful selection of constants  $A$  and  $C$  is crucial to guarantee acceptable performance of the LCG as outlined by Park (Park and Miller 1988). Even with optimal selection of constants  $A$  and  $C$ , the period of the LCG will be limited by the value of  $M$  used for the algorithm. In fact, it is possible that the poor selection of constants  $A$  and  $C$  will result in a generator period which is much less than  $M$ .

For Monte Carlo simulations executed on standard PCs, or even on standard PC-based cluster computers, the linear congruential generator may be an acceptable option. However, when we begin to consider high-speed, Monte Carlo simulation using a

hardware based solution, the requirements become quite different. A simulation executed on a 1000-node cluster computer will essentially use 1000 carefully-seeded and independently-running random number generators to generate a particular set of random numbers. Using an FPGA-based solution, the same number of random numbers will be generated using only a few independently-running random number generators, since the work rate of each generator is significantly higher on an FPGA than on a standard PC. Therefore, the random number generation algorithms used for an FPGA based Monte Carlo simulation must generate random number streams with a much larger period than is typically required by Monte Carlo codes running on standard PCs.

In addition to identifying a pseudo-random number generation algorithm with a large period, one must also be found which can be efficiently implemented in hardware. L'Ecuyer has compiled a review of current methods in pseudo-random number generation and examines several types of generators which are based on a feedback shift register, as opposed to multiplication, to generate random streams (L'Ecuyer 1997). Chu and Jones have successfully implemented several forms of simple feedback shift register (FSR) random number generator algorithms using FPGAs (Chu and Jones 1999). While Chu and Jones' work only examines the simplest generators of this type (namely 1 bit linear feedback shift registers and simple lagged Fibonacci generators), the implementation ease of an FSR-based generator in hardware is shown. For our purposes, we have taken the next step and custom designed a new and efficient FPGA implementation for one of the most sophisticated FSR type generators, the Mersenne Twister.

Matsumoto and Nishimura are responsible for the development of the Mersenne Twister (a pseudo-random number generation algorithm), which has passed the most stringent of statistical tests for randomness and has an incredibly large period of  $(2^{19937} - 1)$  (Matsumoto and Nishimura 1998). A period of  $(2^{19937} - 1)$  implies that a virtually unlimited amount of random numbers can be generated from a single seed with no chance of a repeated sequence, which is essential when performing complex Monte-Carlo analysis with a large number of iterations. The methodology behind the Mersenne Twister (MT) is based on the equation:

$$X_{k+n} := X_{k+m} \otimes (X_k^u | X_{k+1}^l)A \quad (7)$$

where the  $\otimes$  symbol denotes the exclusive or operation (XOR),  $N = 624$ ,  $M = 397$ , and  $X_k$  represents the  $k^{\text{th}}$  32-bit random number in a sequence and  $k$  ranges from 1 to  $N$ .  $(X_k^u | X_{k+1}^l)$  is the most significant bit of the  $X_k$  random number concatenated with the lower 31 bits of the  $X_{k+1}$  random number. This concatenation is multiplied by a constant matrix  $A$ . As shown by Matsumoto and Nishimura, the matrix  $A$  can be selected such that the multiplication is reduced to a binary shift and another XOR.

The algorithm is simple and has equally simple hardware requirements. A small 1024-element, 32-bit wide Virtex II<sup>1</sup> blockram unit onboard the FPGA is used to store the previously generated 624 random numbers. Every time the MT module generates a new random number, it accesses two previously generated random numbers from the

---

<sup>1</sup> The Virtex II FPGA is manufactured by Xilinx, Inc. Virtex II and Virtex II-pro devices are the largest FPGAs available on the market circa 2004, and are the target test devices of this research.

memory element to create the next random number. In addition to the memory element, the only hardware operations necessary for the complete implementation of this algorithm are several counters, some small registers (flip-flops), and the bitwise exclusive or function. All of these functions are easily and efficiently implemented on an FPGA. When we implemented the MT on a large Virtex II FPGA, we found that each MT module requires 2 Virtex II blockram units and negligible (less than 1%) of reprogrammable logic slices. A circuit schematic describing our FPGA implementation of the MT algorithm is shown in Figure 2.

### *Cross Section Retrieval*

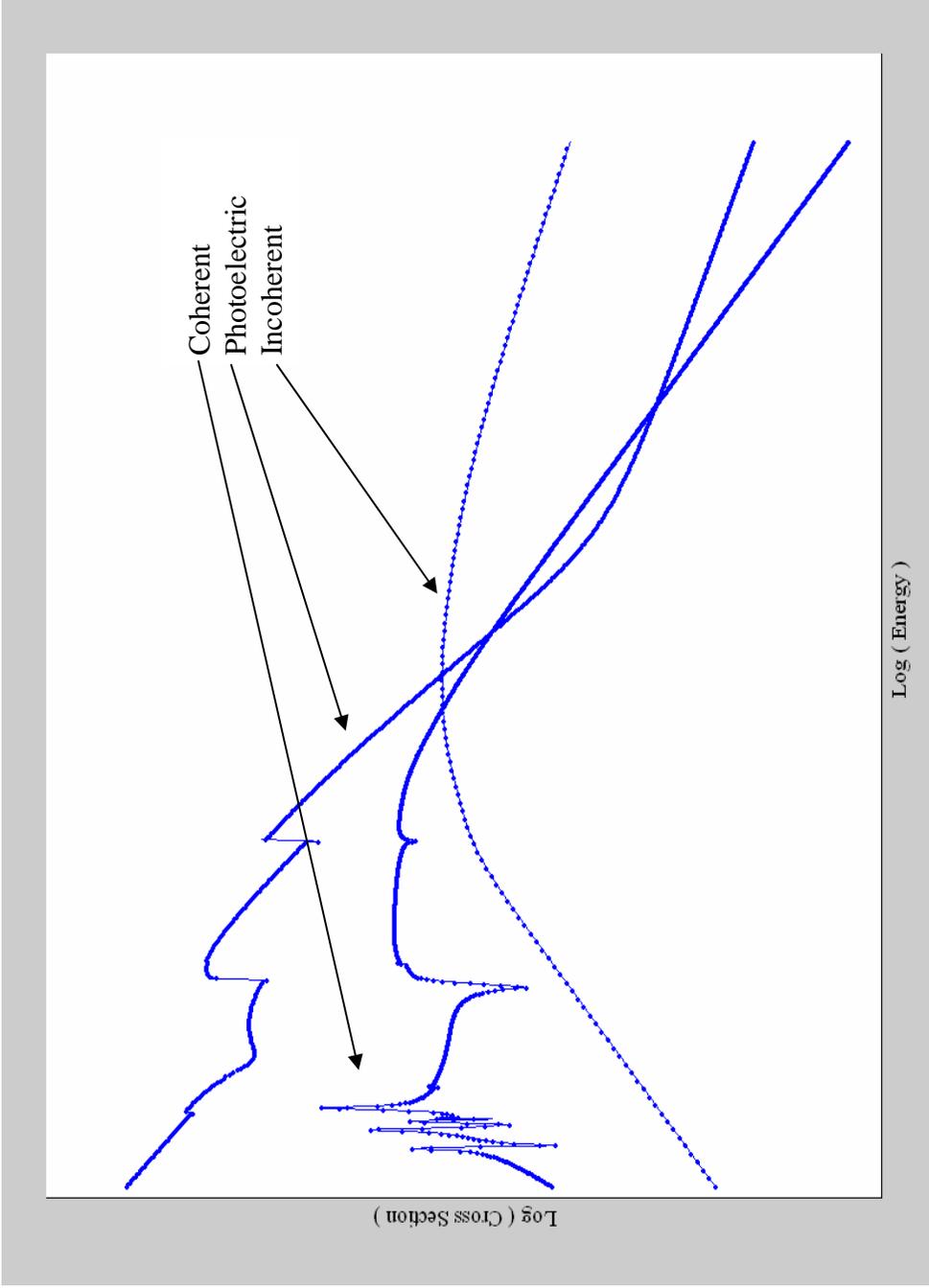
For photon transport with a maximum energy of 1.022 MeV, the photon interaction cross sections of concern are incoherent scattering, coherent scattering and the photoelectric effect. Our implementation uses analytical relationships to describe the differential coherent and incoherent scattering cross sections, as described by equations 1-5. The total interaction cross sections describing all 3 effects are based on the EPDL97 library by Cullen (Cullen 1997a).

Like many cross sections describing particle interactions, the EPDL97 cross sections describing total coherent, incoherent and photoelectric effects are only visually decipherable on a log/log plot. Assuming methods are available for efficient FPGA implementation of logarithms and exponentials (discussed in the next section), regeneration of the log/log cross sections can be performed on an FPGA. Onboard FPGA blockram modules are used to store three individual cross section tables for each

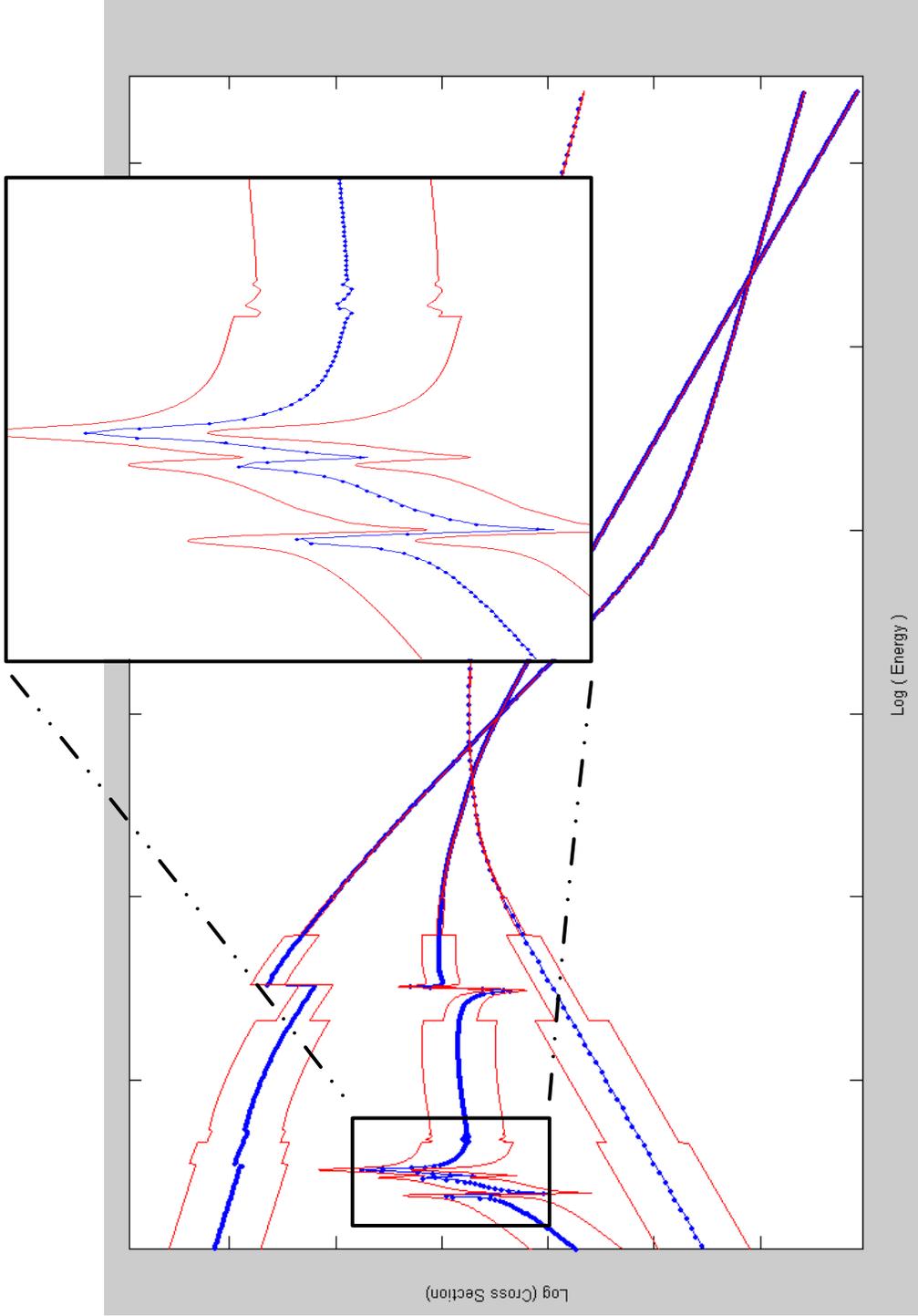


material used in the Monte Carlo simulation. A point-wise polynomial interpolation method is used to discretize each cross section so that the macroscopic cross section for a photon of any energy can be found using just one lookup attempt (no searching necessary). While there is a small amount of approximation to this method, the introduced error is minimal. In addition, we have found that the cross sectional values that are returned by our point-wise polynomial interpolation method are well within the energy specific error bars as defined by Cullen's EPDL 97 documentation (Cullen 1997b).

Figure 3 shows the interpolated discretization method used to store cross sectional data for non-differential photon interaction cross sections. As can be seen in the figure, a different number of interpolation points is used for the cross sections describing each interaction type. For incoherent scattering, the total interaction cross section is a very smooth curve as a function of energy. As a result, only a small number of interpolation points (128 points) are necessary to reproduce the data. The photoelectric total cross section is more complex, with K and L shell absorption edges visible in the cross section. To accurately reproduce this data, 512 interpolation points are used. The coherent scattering cross section is even more complex than the photoelectric, with detailed resonances in the low energy regions. These resonances are very important to low-energy photon transport and cannot be ignored. As a result, 1024 discrete interpolation points are used to reproduce the coherent scattering cross section. This number of interpolation points is capable of accurately reproducing the cross section data, even in resonance regions. Figure 4 illustrates the exact cross sections



**Fig. 3.** The total macroscopic cross sections for aluminum, as it is stored on the FPGA.



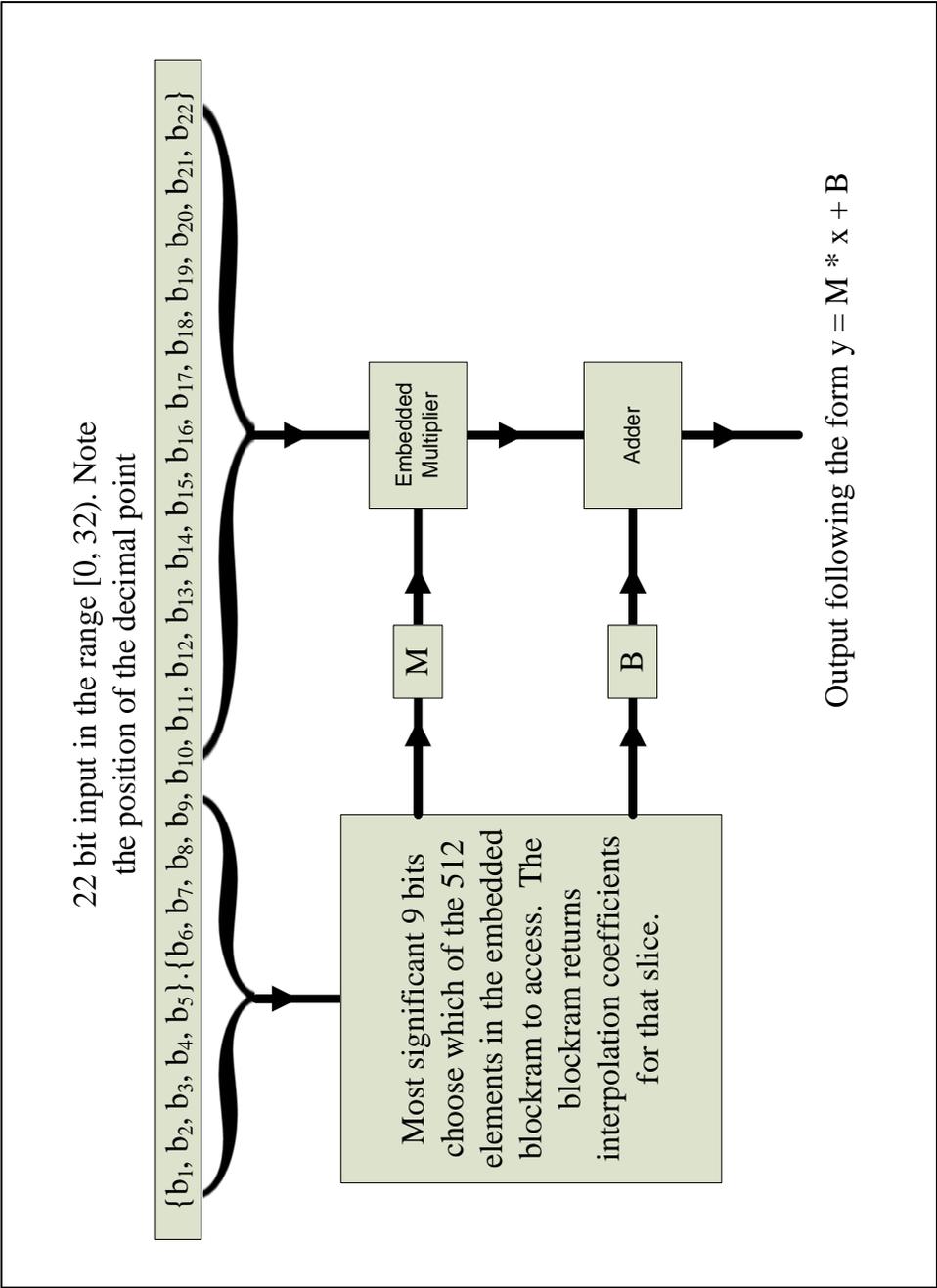
**Fig. 4.** The exact EPDL97 cross sections and associated errors (red) compared with the cross sections as stored on the FPGA (blue).

and their published error range in red, compared with our cross sections as stored on the FPGA shown in blue. The error range depicted in red is given by the exact cross sections plus/minus the energy specific error bars described by Cullen's EPDL97 documentation (Cullen 1997b).

The algorithm of choice used to interpolate between discrete points in the cross section retrieval design is variable, however, the easiest and most effective form is polynomial interpolation. The optimal order of the polynomial used depends on the shape of the cross section, the number of discrete points used, and desired multiplier versus blockram usage on the FPGA. To properly capture photoelectric edges and coherent scattering resonance regions, we have found that the optimal interpolation scheme is linear interpolation with a large number of discrete interpolation slices. Figure 5 illustrates the hardware interpolation scheme used to reproduce cross section data that has been pre-divided into 512 slices. Linear interpolation was used in this example, where 512 individual interpolation coefficients are stored in onboard FPGA blockram. In Figure 5, the most significant 9 bits of the input value are used as the memory address selector. After the memory unit returns the interpolation coefficients corresponding to a particular discrete slice, the remaining bits from the input are assumed to be a decimal value in the range [0,1) and are used as the operand in the equation:

$$\log(\text{Cross} - \text{Section}) = M * x[10: \text{end}] + B \quad (8)$$

where M and B are stored interpolation constants and x[10:end] are the least significant remaining bits of the input.



**Fig. 5.** A diagram of the interpolation scheme used for cross section retrieval.

It should be emphasized that performing the cross-section lookup in the manner described above will only require a small amount of FPGA resources. For each material, storing interpolation coefficients describing all three interaction types will take only 3-5 blockram slices (out of hundreds available) on a Xilinx Virtex-II FPGA. In addition to the blockram slices and regardless of the number of materials stored, one embedded multiplier is necessary to perform the interpolation. The cross-section lookup is fast using the described techniques, and can be completed with a work rate of 1 evaluation per clock cycle.

### *Logarithms*

Logarithm evaluations of different bases are used through the Monte Carlo hardware radiation transport schemes described in this thesis. Regardless of the base required, however, all initial evaluations are computed using  $\log_2$ —because it is most naturally implemented in a binary number system. Once  $\log_2(x)$  is evaluated, the change of base formula was used to convert  $\log_2$  to  $\log_b$  for any base  $b$  by simply multiplying  $\log_2(x)$  by a constant  $k$ , where  $k$  follows the formula:

$$k = \frac{1}{\log_2(b)}. \quad (9)$$

With a conversion mechanism to transform  $\log_2(x)$  into  $\log_b(x)$  efficiently, great detail must be paid to the accurate and fast evaluation of  $\log_2(x)$ . As opposed to cross-section retrieval, where some sort of lookup table implementation is the natural solution, the obvious solution to the hardware-based evaluation of a logarithm is typically a series

expansion. Series expansions for the evaluation of log fit the general form (Arfken 1985):

$$\log_e(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \dots \quad (10)$$

A calculator or a computer which uses a series expansion like that in equation 10 to evaluate a logarithm will cycle through each term in the expansion, evaluating one term at a time. Our designs, on the other hand, will require a much higher work rate than this, since the overall goal of this project is speed. With speed in mind, the expansion in equation 10 can be evaluated with a work rate of 1 evaluation per clock cycle in hardware using  $(2n - 2)$  multipliers, where  $n$  is the number of expansion terms to be evaluated. Large Xilinx Virtex-II FPGAs have hundreds of embedded multipliers, and so a requirement for  $(2n - 2)$  multipliers may not be an issue provided that only a small number of series terms needs to be evaluated to obtain convergence. Unfortunately, we need to evaluate  $\log(x)$  for all values of  $x$ , even as  $x$  approaches zero. The series expansion for  $\log$  given in equation 10 converges very slowly as the operand approaches zero and requires the evaluation of many terms to obtain convergence. Due to a slow convergence as the operand approaches zero, evaluation of the series expansion on an FPGA will result in either the loss of our 1 evaluation per clock cycle work rate or the use of a vast amount of FPGA resources to perform the evaluation; either of which is unacceptable to our project goal.

If an expansion-based evaluation is not an option, the other logical solution is a lookup table-based evaluation. Unfortunately, for a lookup-table based evaluation to be

effective, there must be a finite region for which the evaluation is performed--in our case we must perform  $\log(x)$  for any  $x$  such that  $0 \leq x \leq \infty$ . To solve this problem, we have developed a transformation to force the operand of  $\log(x)$  into a specific region, namely  $1 \leq x < 2$ . The shape of the log curve in the region  $1 \leq x < 2$  is smooth and simple and can be easily regenerated using a point-wise polynomial interpolation method similar to the one used for cross-section retrieval. The transformation used becomes extremely simplistic if 2 is the logarithm base used. Equations 11 and 12 illustrate the form of the transformation.

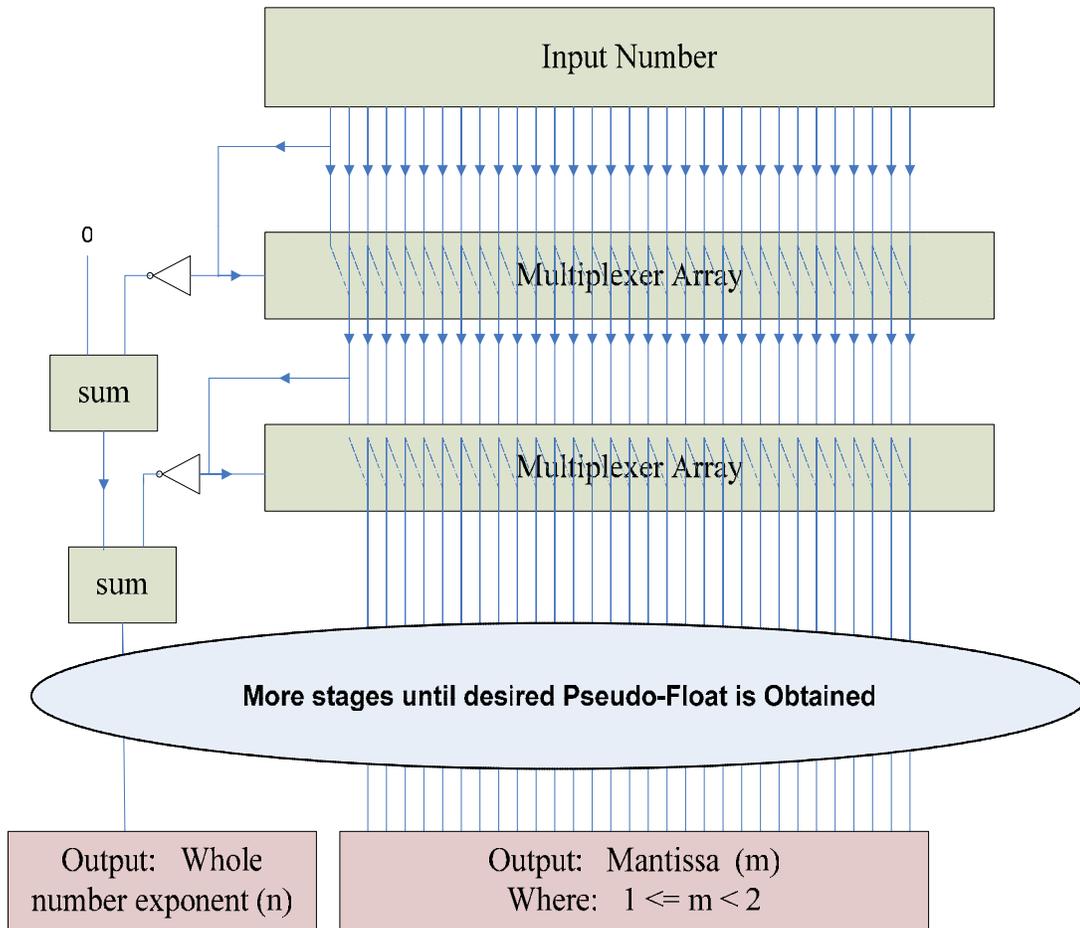
$$\log_2(x) = \log_2(m \cdot 2^n) = \log_2(m) + n \quad (11)$$

where,

$$x = m \cdot 2^n, \text{ and } 1 \leq m < 2. \quad (12)$$

For any operand  $x$ ,  $m$  and  $n$  can be found by placing  $x$  in a pseudo-floating-point form—an extremely efficient operation in hardware consisting mostly of binary shift operations. A preprocessing stage using pipelined multiplexer arrays handles the binary shifting such that an equivalent value for  $x$  is determined in the form of equation 12. Figure 6 shows this preprocessing stage. Once the preprocessing has been completed, the exponent  $n$  can be set aside until the  $\log_2(m)$  evaluation has been completed using lookup-table methods

With  $n$  and  $m$  in equation 11 determined by the preprocessing stage, the evaluation of  $\log_2(m)$  can be completed efficiently using lookup-table methods since  $m$  now has a fixed range. A polynomial-interpolation method similar to the one used for the cross-section retrieval is an excellent solution for the reproduction of  $\log_2(m)$ .



**Fig. 6.** A schematic depicting the log preprocessing stage. The pipelining stages have been omitted.

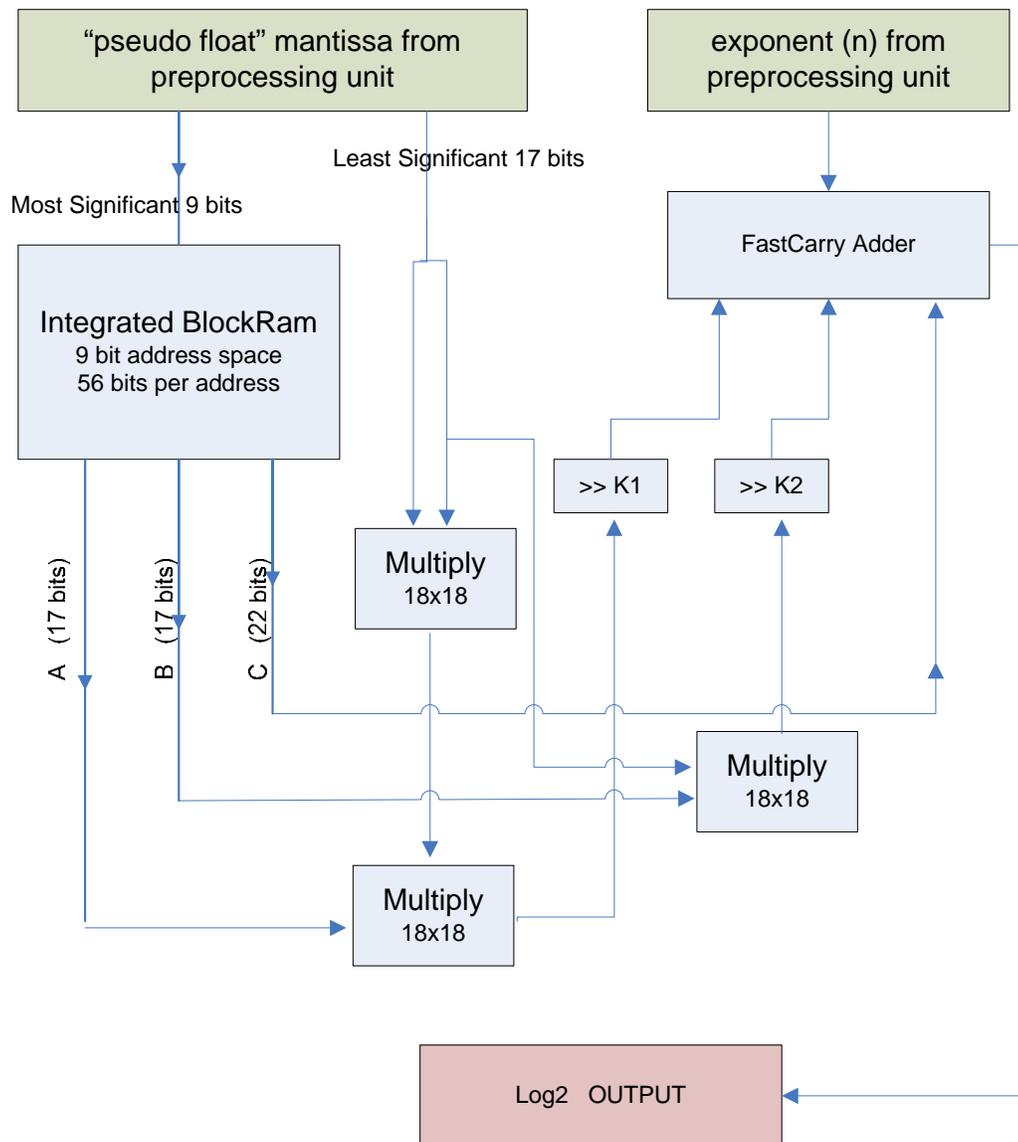
The  $\log_2$  curve is very smooth, unlike the curves representing cross-section data, and therefore will benefit from the use of a higher-order, polynomial-interpolation algorithm. Binomial-interpolation coefficients are stored for each  $\log_2$  slice for  $m$  in the range [1, 2). Using 512 slices and binomial interpolation between each slice,  $\log_2(m)$  can be evaluated with a maximum associated algorithm error of about  $10^{-10}$ . A diagram detailing this evaluation on the FPGA is shown in Figure 7. Evaluation of  $\log_2(x)$  can be performed for any input  $x$  with equivalent precision by adding the shift constant determined by the preprocessing stage to the resultant value of  $\log_2(m)$ .

### *Exponentials*

In much the same way that performing logarithm evaluations is necessary to cross-section retrieval, so is performing exponential evaluations. The cross-sectional data for each material are stored in a  $\log_2 / \log_2$  format, and a final evaluation of  $2^x$  will be necessary to obtain an un-transformed cross section value. As with logarithm, a series expansion based evaluation of  $2^x$  is certainly possible, although this will not be as efficient as an interpolated look-up table solution. Again, however, we must find a transformation that will allow us to perform the interpolated look-up table portion of the evaluation over a very fixed range. We have developed the transformation described in equations 13 and 14 below to allow for the efficient, hardware-based evaluation of  $2^x$  for all values of  $x$  in hardware:

$$2^x = 2^{i.d} \quad , \quad \text{where } i \text{ and } d \text{ represent the integer} \quad (13)$$

and decimal components of  $x$



**Fig. 7.** A schematic depicting the  $\log_2$  evaluation stage. The pipelining stages have been omitted as in Figure 6.

$$2^{i.d} = 2^{i+0.d} = 2^i \cdot 2^{0.d} = 2^{0.d} \ll i \quad (14)$$

A polynomial-interpolated, lookup table can now efficiently evaluate  $2^{0.d}$ , where  $d$  is a number in the range  $[0,1)$ . Then, a simple binary shift by the integer  $i$  will return the correct evaluation for  $2^x$ , for any value of  $x$ . The methods for the polynomial-interpolated look-up table evaluation of  $2^{0.d}$  are nearly identical to the methods used to evaluate  $\log_2(m)$  as described in the previous section. Just as in the case of  $\log_2$ , we have developed methods for implementing  $2^x$  very efficiently in hardware to a high degree of precision—methods which are unique to this project.

### *Scattering Distributions*

To determine the new scattering angle after a coherent or incoherent scattering event, the rejection technique must be used on the probability density functions described in equations 1-4. While the Klein-Nishina differential cross section is not directly invertible, Nelson has documented methods of efficiently sampling the Klein-Nishina distribution using a combined rejection-composition technique (Nelson et al. 1985). For simplicity, no combined rejection-composition techniques or even combined inversion-rejection techniques will be used for the initial tests described in this thesis. Nelson et al.'s methods are only mentioned for completeness and applicability to future research in this area.

One hardware module was designed to evaluate either a coherent or an incoherent scattering event. Based on the event type and the momentum transfer, either

the incoherent scattering function or the square of the coherent scattering form factor for the material is determined from a polynomial-interpolated lookup table—similar in design to the cross-section retrieval methods described in a previous section. The differential Klein-Nishina cross section given in equation 3 is used for both coherent and incoherent scattering by setting the incident photon energy to for coherent scattering, thereby reducing the Klein-Nishina cross section to the Thompson cross section in equation 4.

As stated previously, no combined inversion-rejection methods were used. Therefore, rejection technique attempts were made by sampling a uniformly distributed scattering angle between 0 and 180 degrees and also sampling uniformly between 0 and the maximum of the scattering function / form factor value for the incident photon energy of interest. As a result, rejection-technique efficiency was low for higher-energy incoherent scattering and lower still for high-energy coherent scattering. Fortunately, a high-energy, coherent scattering event is very rare, so the inefficiency is somewhat counterbalanced naturally by the cross sections. To further correct for the remaining inefficiency of the rejection technique, four complete hardware-based scattering modules were run in parallel to quadruple the chances of finding a non-rejected scattering angle for each attempt.

Standard software-based methods perform the rejection technique using a While loop, halting the progress of the entire program until a non-rejected value is identified. Unfortunately, it is not known how many iterations will be necessary to find a non-rejected value. This poses a problem for a hardware-based rejection technique

algorithm. Since our designs are highly pipelined and depend on the parallel execution of different portions of the calculations necessary for the simulation of each photon interaction, every operation must take a fixed number of clock cycles to complete. Because of this, a looping, rejection-technique algorithm cannot be compatible with our methodology. Therefore, if all four parallel rejection-technique attempts fail to find a non-rejected value, the transport of a particular particle will cease, and will be resumed from the previous successful interaction point in a future clock cycle. For an explanation of hardware pipelining techniques see Hennessy and Patterson (Hennessy and Patterson 1998).

Running four rejection algorithms in parallel makes the overall efficiency of the algorithm acceptable. If high-energy, coherent scattering is ignored, overall efficiency is about 85 percent. If high-energy, coherent scattering is included, the overall efficiency drops into the 60 percent range.

### *Overall Layout*

As stated earlier, the simple test transport problem consists of an isotropic 250 keV photon point source in an infinite, three-dimensional medium of aluminum with a number of spherical tallies measuring the number of photons crossing a boundary. Since speed is the most important goal of this project, the overall transport algorithm has been designed to optimally evaluate 1 complete photon interaction per clock cycle. The algorithm deviates from this optimal goal only in the event that all four of the parallel,

rejection-technique modules used to probe the scattering distributions return a rejected value. In this case, the interaction will be marked as incomplete and will be re-evaluated in a subsequent clock cycle. No useful work is done in a clock cycle where there is a 4-fold rejection.

After a scattering event is completed and a scattering angle found, transformations must be used to determine a Cartesian vector describing the new particle direction. The familiar Cashwell and Everett method was used to determine a unit vector representing the particle trajectory after scatter (Cashwell and Everett 1959). The Cashwell and Everett method is shown in equations 15, 16, and 17:

$$u' = u \cdot \cos(\mathcal{G}_s) + \frac{(u \cdot w \cdot \sin(\mathcal{G}_s) \cos(\Delta\psi) - v \cdot \sin(\mathcal{G}_s) \sin(\Delta\psi))}{\sqrt{1-w^2}} \quad (15)$$

$$v' = v \cdot \cos(\mathcal{G}_s) + \frac{(v \cdot w \cdot \sin(\mathcal{G}_s) \cos(\Delta\psi) + u \cdot \sin(\mathcal{G}_s) \sin(\Delta\psi))}{\sqrt{1-w^2}} \quad (16)$$

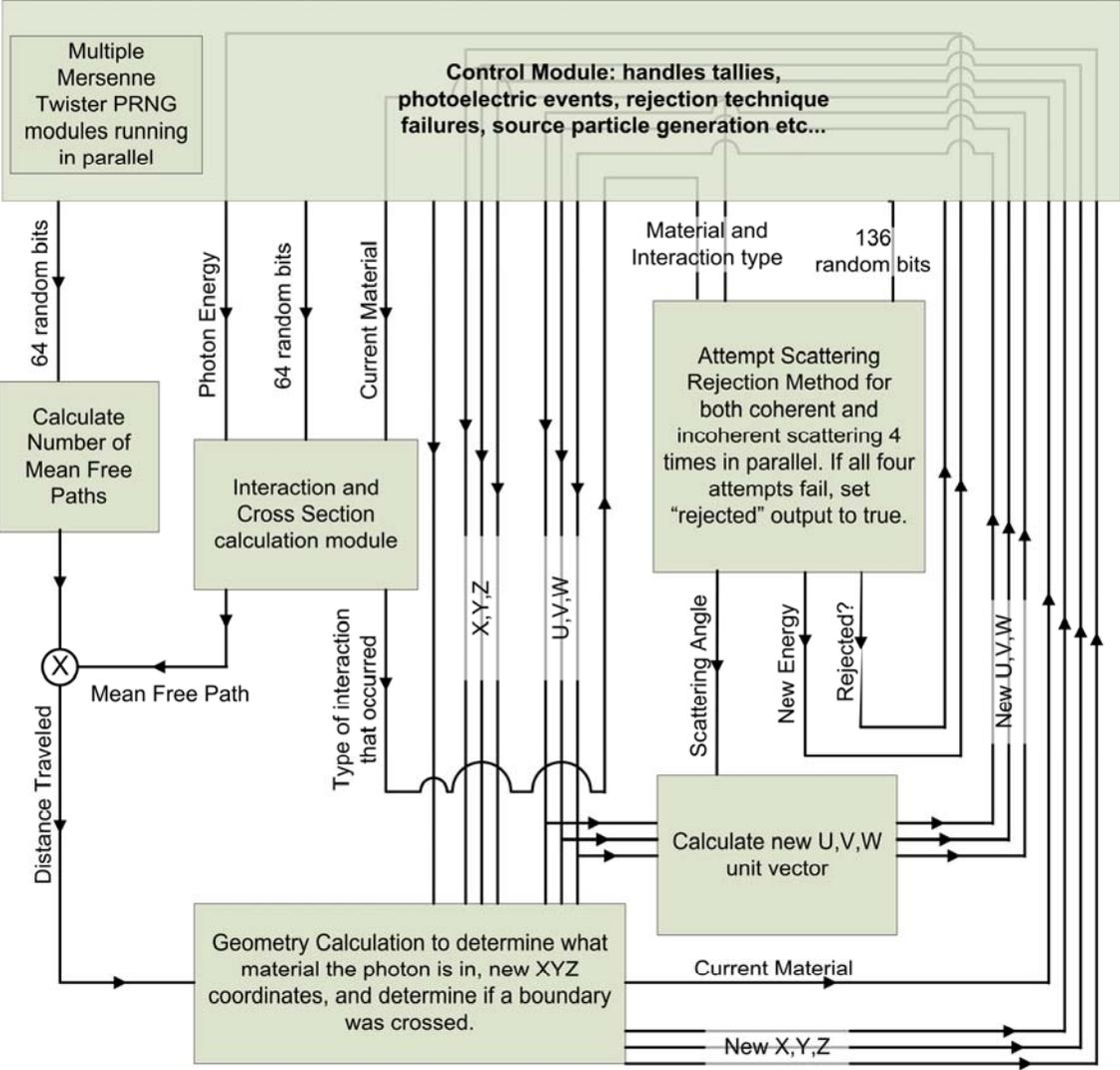
$$w' = w \cdot \cos(\mathcal{G}_s) - \sin(\mathcal{G}_s) \cos(\Delta\psi) \sqrt{1-w^2} \quad (17)$$

where,  $\Delta\psi$  is a uniformly distributed random number between 0 and  $2\pi$  radians representing the change in azimuthal angle,  $\mathcal{G}_s$  is the scattered angle obtained by probing the scattering distributions and  $u, v, w$  is a unit vector representing particle direction.

The Cashwell and Everett equations can be implemented efficiently onboard a Xilinx Virtex-II series FPGA due to the large number of embedded multipliers and blockram units available in the device. Implementation methods are similar to those described before, combining polynomial-interpolated lookup tables with, in this case,

several additional units of embedded multipliers to carry out the multiplications in equations 15-17.

A block schematic showing the entire algorithm operation is given in Figure 8. The algorithm is capable of running on a large Xilinx Virtex-II FPGA with negligible interaction from a host PC. All random numbers are internally generated, and all Monte Carlo transport operations are evaluated inside the FPGA. The design was heavily pipelined to increase the maximum clock speed at which the FPGA can operate. As a general design rule, pipeline stages were placed after no more than a single 32-bit fast carry adder, two 18x18 embedded multipliers or equivalent distributed logic—ensuring a clock speed surpassing 100 MHz.



**Fig. 8.** Dataflow of the overall algorithm, showing the ideal amount of 'work' completed per clock cycle.

## CHAPTER V

### RESULTS

#### *Testing Methods*

Modern FPGA technology is just now reaching a mature enough level so that implementation and execution of an algorithm of the complexity described in the preceding chapters are possible. Because of this, one of the largest FPGAs available on the market will be required for the actual implementation and testing of these algorithms. All of the hardware algorithms described previously were designed specifically for the Xilinx Virtex-II Pro 100 FPGA, one of the largest FPGAs available on the market circa 2005. Unfortunately, the price of the Xilinx Virtex-II Pro 100 FPGA is typically in the \$15,000 range when purchased in conjunction with a PCI interface board. Physical access to this device was not available at the time of this thesis research; however, this in no way precludes us from obtaining both accurate and useful results.

To obtain useful and accurate results without actually using the physical Xilinx Virtex-II Pro 100 FPGA, a two stage simulation has been performed. Both stages of the simulation use synthesizable HDL code describing the Monte Carlo hardware transport design as a basis for simulation. We briefly discussed Verilog and VHDL codes in Chapter II, however, what was not mentioned is that certain strict coding techniques must be followed for an FPGA place and route software package to translate the HDL code into a programming file that can be used to physically program the FPGA. When the HDL is written in such a way, we call a particular HDL code “synthesizable”.

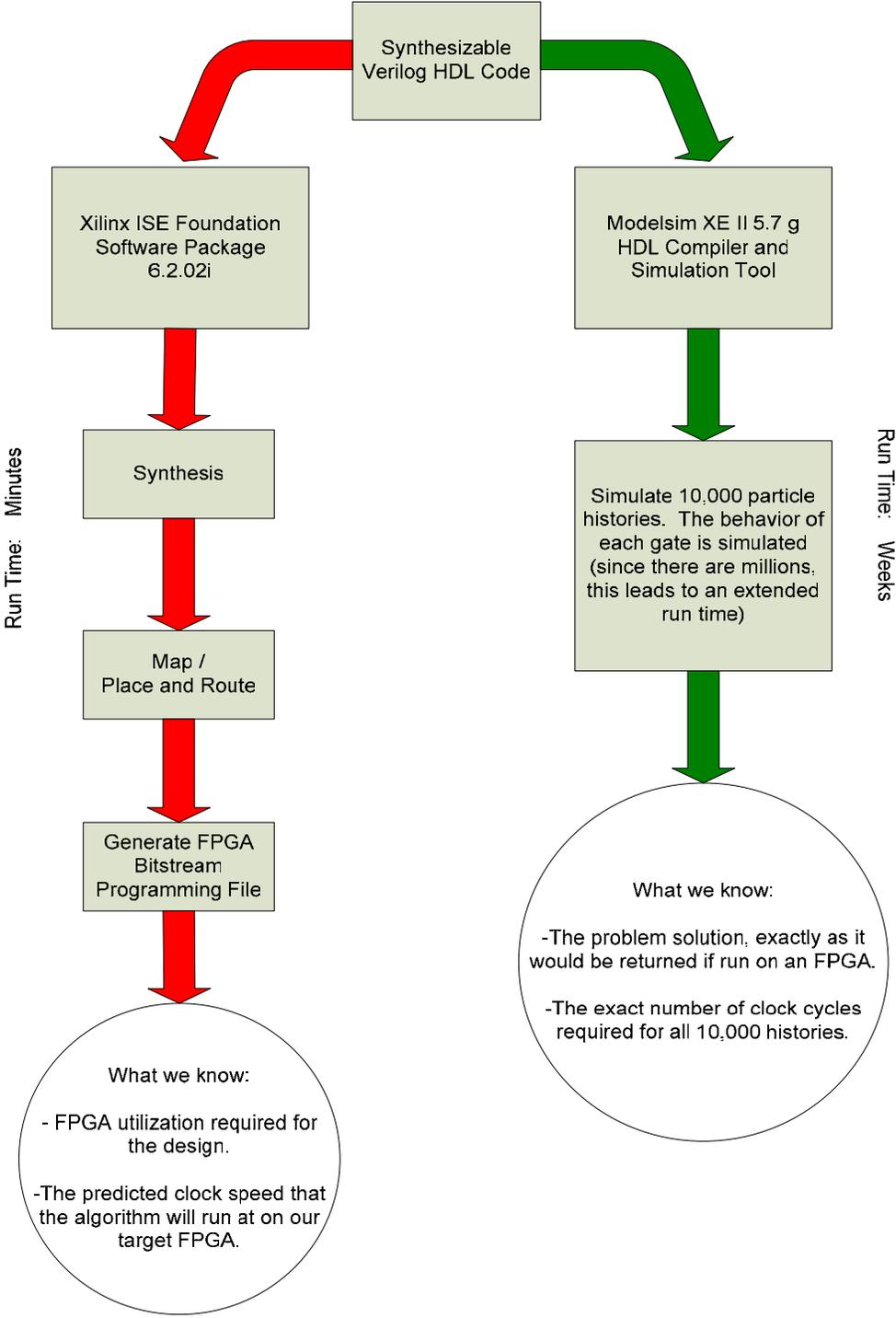
With synthesizable HDL code describing the design, the function of the code is verified using an HDL compiler/simulator. We used Modelsim XE II v 5.7 G to simulate the HDL code at the gate level, clock cycle by clock cycle, to obtain the exact design output that would be obtained from an FPGA programmed using the HDL code and run for the same number of clock cycles. This level of simulation is fairly slow since the computer simulates the response of each gate in the design (of which there are millions) to changing inputs. The simulation of our Monte Carlo HDL code using Modelsim XE took approximately 4 weeks to complete 10,000 photon particle histories. However, the information obtained by this simulation was invaluable. From the Modelsim simulation we were able to obtain the exact tally data for each tally (used for design verification), as well as the exact number of clock cycles necessary to complete the 10,000 photon histories. It is important to note that this process is not necessary if a physical device is in hand. Modelsim will still be used to debug the hardware, but this process will not be time consuming since a many particle simulation will not be performed.

The second portion of the simulation also utilized the synthesizable HDL code as a basis. The Xilinx ISE Foundation 6.2.02i software package was used to synthesize and implement the HDL design to a level such that a programming file was generated which can be used to physically program the Xilinx Virtex-II Pro 100 FPGA. The Xilinx ISE software breaks up the HDL code and uses sophisticated logic reduction algorithms to optimize the design as much as possible (the synthesis stage). After synthesis, the ISE software uses libraries detailing the exact structure of the Virtex-II Pro 100 FPGA to

optimally place different design components on the FPGA as well as to optimally route signals between interacting components on the FPGA. Place and route is performed by sophisticated algorithms which minimize the FPGA resources that the design consumes, as well as maximize the clock speed of the design (shorter routing distances between components translates to a higher overall clock speed). The place and route process returns an exact FPGA utilization report describing the consumption of different types of FPGA components when the device is programmed. In addition, the place and route process also returns a timing report, providing a conservative clock speed estimate at which the design can run when programmed on the Virtex-II Pro 100 FPGA. Part of the libraries built into the ISE software describing the structure of the Virtex-II Pro 100 FPGA include timing data for each part of the devices internal components. Using these data, and summing timing delays along components between pipelining stages, the ISE software determines an accurate clock speed estimation. A flowchart describing the simulation methods originating from the synthesizable HDL code is shown in Figure 9.

Once the FPGA clock speed and exact number of clock cycles required to compute 10,000 particle histories is determined, the theoretical throughput of the design when implemented on a Virtex-II Pro 100 FPGA can be found using:

$$Throughput \left[ \frac{\text{Particle Histories}}{\text{Second}} \right] = \frac{\text{Clock Speed (Hz)}}{(\text{Average Number of Clock Cycles Per } \cdot \text{ )}} \quad (18)$$



**Fig. 9.** A flowchart describing the simulation process.

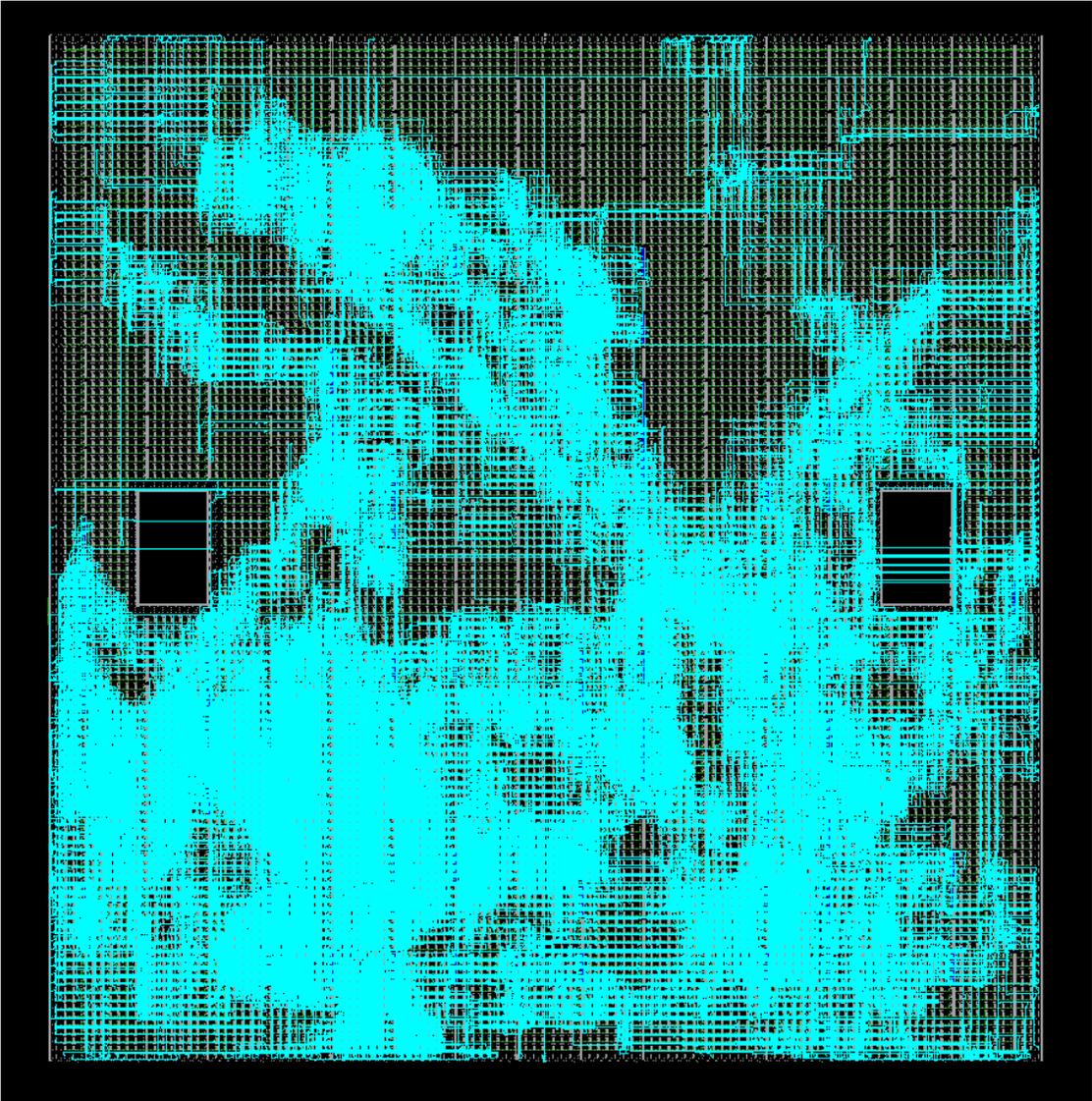
When a specific FPGA is not available, speed benchmarking methods similar to those described above are used widely in published works by researchers in the field of reconfigurable computing, and are considered to be acceptable forms of testing. Examples of some published articles which use similar methodologies to speed benchmark their algorithms on FPGAs are Jarvinen et al. 2003 and Shackleford et al. 2002.

### *Monte Carlo Results*

Using the Xilinx ISE Foundation 6.2.02i software package, the synthesizable hardware designs created for this research were analyzed. The software package returns FPGA usage data, timing analysis and a bit-stream FPGA programming file. A summary of the Virtex-II Pro 100 FPGA utilization and timing data can be found in Table 1. In Figure 10, a visual representation of the FPGA programmed with our Monte Carlo hardware design is shown, where the blue regions represent used portions of the FPGA. As can be seen by the data presented in Table 1, roughly 20% of the general logic portion of the device (LUTs) is used in this implementation. The device has plenty of room to implement a more complex Monte Carlo simulation. On the other hand, if we wish to utilize the entire processing power of this FPGA to evaluate this particular simulation, three independent implementations can be programmed to run in parallel on the device. In this case, the overall clock speed was reduced from 136.7 MHz to 111.5 Mhz due to less optimal signal routing incurred by the utilization of a more significant amount of the programmable logic within the FPGA.

**Table 1.** The resources consumed on the Xilinx Virtex-II Pro 100 FPGA by the Monte Carlo radiation transport hardware design.

	Used	Available	Percent Utilization
Number of Slices	11,944	44,096	27 %
Number of Slice Flip-Flops	12,648	88,192	14 %
Number of 4-Input LUTS	17,747	88,192	20 %
Number of Embedded Block-Rams	79	444	17 %
Number of Embedded 18x18 Multipliers	117	444	26 %
Estimated Clock Speed	136.733 MHz	N/A	N/A



**Fig. 10.** The programmed FPGA with one Monte Carlo transport module. Blue denotes the utilized portions of the FPGA.

Using the Modelsim XE II 5.7 g HDL compiler, a simulation of the hardware design was also performed. The flux tallies that we obtained from Modelsim XE were compared with the flux results generated using MCNP-5 (X-5 Monte Carlo Team, 2003). MCNP-5 was used as a comparison to our hardware method for two reasons. First, we used MCNP as a result comparison to ensure our methods are on target, and there are no significant bugs in our hardware design. More importantly, however, we obtained a speed comparison from MCNP-5. To closely compare to our hardware design, MCNP was programmed to ignore Bremsstrahlung radiation and all secondary electrons. However, MCNP did simulate 1<sup>st</sup> fluorescence x-rays, which we did not. This accounted for a 3.5 percent increase in the number of photons MCNP tracked that our hardware design did not track. Repercussions to be aware of are that MCNP reported slightly higher flux tallies and performed about 3.5 percent more computational work than our hardware design. Flux comparisons are shown in Table 2 for 10,000 histories. The flux tallies reported by our FPGA hardware design are very close to those reported by MCNP. The fluxes reported by MCNP were slightly higher in regions where photoelectric and Compton interactions (and thus fluorescence yield) were highest, but this was expected.

Simulation of 10,000 histories using Modelsim XE II 5.7 g running the hardware design revealed that the evaluation of 10,000 histories corresponded to 76,231 photon interactions—which agrees almost exactly with MCNP-5. The number of clock cycles required to evaluate 76,231 photon interactions was reported to be exactly 129,541. This deviation from our 1 clock cycle per photon interaction ideal work rate was caused by an

**Table 2.** The flux tallies for MCNP-5 versus the flux tallies reported by the FPGA hardware design.

Tally Distance <sup>a</sup>	MCNP-5	FPGA	% Difference	% Standard Error
2 cm	11,717	11,625	0.785	0.924
3 cm	11,680	11,521	1.361	0.925
4 cm	11,168	10,962	1.844	0.946
5 cm	10,388	10,133	2.455	0.981
6 cm	9,488	9,145	3.615	1.026
7 cm	8,412	8,122	3.447	1.090
8 cm	7,326	7,140	2.539	1.168
9 cm	6,160	6,083	1.250	1.274
10 cm	5,183	5,128	1.061	1.389

<sup>a</sup> Although tallies were performed at 15, 20, and 25 cm, the standard error resulting from running only 10,000 histories at those distances becomes high enough that the results are not reliable. Both the FPGA-based simulation and MCNP-5 suffer from this issue. Therefore, these flux tallies are not presented in this table.

efficiency decrease due to the use of the rejection technique to probe the differential scattering distributions. Using equation 18, we find that the theoretical work rate of the Xilinx Virtex-II Pro 100 FPGA is 633.15 million complete photon histories per minute when programmed with our Monte Carlo particle transport hardware designs. This is the work rate for only one instance of the design (where the FPGA is approximately 20% utilized) and running at 136.7 MHz. Utilization of the entire FPGA by running three transport simulations in parallel produced a higher work rate. At 111.5 MHz, three independently running transport simulations produced a theoretical work rate of 1.55 billion complete photon histories per minute on the Xilinx Virtex-II Pro 100 FPGA.

For speed comparison purposes, MCNP-5 was run on a modern 3.2 GHz Intel Pentium-IV desktop computer. MCNP-5 was run under the Microsoft Windows XP operating system with no other active processes running on the PC. MCNP-5 performed the same simulation, ignoring secondary electrons and Bremsstrahlung radiation. As was previously stated, the only difference in the simulation that MCNP-5 performed was that it simulated 1<sup>st</sup> fluorescence x rays which accounted for 3.5 % additional photon histories. Including the fluorescence x rays, MCNP-5 can compute this simulation at a work rate of 2.3704 million source particles per minute. Multiplying by 3.5% additional photons per source particle, we estimate that MCNP-5 running on a 3.2 GHz Pentium-4 PC can track about 2.4534 million complete photon histories per minute.

Comparing the two work rates, it seems that a single Xilinx Virtex-II Pro FPGA is more than 650 times faster than a 3.2 GHz Intel Pentium-IV desktop PC running MCNP-5 at evaluating this particular radiation transport problem.

## CHAPTER VI

### CONCLUSIONS

This thesis research has accomplished its key goals. First, a new method for Monte Carlo radiation transport has been developed and exercised on a simple transport problem. We have shown that the radiation transport problem described in this thesis can be evaluated in excess of 650 times faster on a large FPGA than it can be evaluated on a 3.2 GHz Pentium-IV desktop PC running MCNP-5. This is a substantial acceleration factor which we believe can be preserved when the techniques discussed in this thesis are expanded to evaluate more complex Monte Carlo simulations.

This is just a first step for FPGA and hardware based Monte Carlo radiation transport. The research in this thesis has shown the incredible potential of the application of FPGAs to Monte Carlo radiation transport problems, opening the door to further research in any of the unbounded number of applications that these techniques may have for accelerating radiation transport computations.

## CHAPTER VII

### FUTURE WORK

Plans to continue this work are well underway. Development will be done to support FPGA based coupled photon-electron Monte Carlo transport as well as possible extensions to include neutron transport. As these development steps are completed, published comparisons will be performed between commercial Monte Carlo transport codes and our methods to characterize the speed increase achieved. In addition, we have plans to support complex voxel geometries to accurately model biological organs and tissues. Combining the support of voxel geometry representation with ultra-high speed coupled photon-electron transport will be important for the evaluation of internal and external dosimetry calculations for both health physics and medical physics applications.

## REFERENCES

- Arfken G. Mathematical methods for physicists, 3rd ed. Orlando, FL: Academic Press; 1985.
- Cashwell E, Everett C. A practical manual on the Monte Carlo method for random walk problems. Elmsford, NY: Pergamon Press; 1959.
- Chu P, Jones R. Design techniques of FPGA based random number generator. Proceedings of the Military and Aerospace Applications of Programmable Devices and Technologies Conference. 1999:1-6.
- Compton K, Hauck S. Reconfigurable computing: A survey of systems and software. ACM 34:171-210; 2002.
- Cullen D. 1997a: the evaluated photon data library, '97 version [online]. Available at: <http://www.llnl.gov/cullen1/DOCUMENT/EPDL97/epdl97.htm>. Accessed 20 November 2004.
- Cullen D. 1997b: Report UCRL-50400, Vol.6, Rev.5, 1997 [online]. Available at: <http://www.llnl.gov/cullen1/document/epdl97/epdl97.pdf>. Accessed 20 November 2004.
- Hauck S. The future of reconfigurable systems. Proceedings of the Fifth Canadian Conference on Field Programmable Devices. 1998:1-8.
- Hennessy J, Patterson D. Computer organization and design, 2nd ed. San Francisco, CA: Morgan Kaufmann; 1998
- Hubbell J, Veigele W, Briggs E, Brown R, Cromer D, Howerton R. Atomic form factors, incoherent scattering functions, and photon scattering cross sections. J. Phys. Chem. Ref. Data 4:471-539; 1975.
- Jarvinen KU, Tommiska MT, Skytta JO. A fully pipelined memoryless 17.8 Gbps AES-128 encryptor. Proceedings of the ACM FPGA '03 Conference. 2003:1-9
- Klein O, Nishina Y. Über die streuung von strahlung durch freie elektronen nach der neuen relativistischen quantendynamik von dirac. Z. Physik 52:853; 1929.
- L'Ecuyer P. Uniform random number generators: A review. Proceedings of the 1997 Winter Simulation Conference. 1997:127-134.
- Matsumoto M, Nishimura T. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. ACM Transactions on Modeling and

Computer Simulations: Special Issue on Uniform Random Number Generation 8:3–30; 1998.

Nelson W, Hirayama J, Rogers W. The EGS4 code system. Stanford Linear Accelerator Center; SLAC-265, UC-32; 1985.

Palnitkar S. Verilog HDL, 2nd ed. Indianapolis, IN: Prentice Hall PTR; 2003.

Park SK, Miller KW. Random number generators: good ones are hard to find. ACM 31:1192-1201; 1988.

Shackleford B, Tanaka M, Carter RJ, Snider G. FPGA implementation of neighborhood-of-four cellular automata random number generators. Proceedings of the ACM FPGA '02 Conference. 2002:1-7

X-5 Monte Carlo Team. MCNP5-A general monte carlo n-particle transport code. Los Alamos National Laboratory; 2003.

Yalamanchili S. Introductory VHDL: From simulation to synthesis. Indianapolis, IN: Prentice Hall; 2000.

## VITA

Name: Alexander Samuel Pasciak

Address: 1328 Rathwood Ave.  
Richland, WA 99352

Email Address: [pasciak@cedar.ne.tamu.edu](mailto:pasciak@cedar.ne.tamu.edu)

Education: B.S., Electrical Engineering, The University of Washington, 2003