*The Monte Carlo Method: Versatility Unbounded In A Dynamic Computing World*
Chattanooga, Tennessee, April 17–21, 2005, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2005)

# ULTRA-HIGH SPEED MONTE CARLO COMPUTING TECHNIQUES USING FIELD PROGRAMMABLE GATE ARRAYS

**Alexander S. Pasciak and John R. Ford**
Department of Nuclear Engineering
Texas A&M University
3133 TAMU
College Station, TX 77843
pasciak@cedar.ne.tamu.edu; ford@ne.tamu.edu

## ABSTRACT

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 Monte Carlo 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 desktop 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 paper we discuss a simple radiation transport problem involving moderate energy photons incident on a 3 dimensional target. By comparing the evaluation speed of this transport problem on an FPGA to the evaluation speed of the same transport problem using standard computing techniques, we show that it is possible to accelerate Monte Carlo computations significantly using field programmable gate arrays. In fact, we have found that our simple photon transport test case can be evaluated about 650 times faster on a large FPGA than it can on a 3.2 Ghz Pentium-4 desktop PC running MCNP5—an acceleration factor that we predict will be largely preserved for many Monte Carlo simulations.

*Key Words*: FPGA; high speed; hardware

## 1 INTRODUCTION

Currently, there are two widely used general computing methods for the expedited execution of highly complex scientific computations: standard 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 it is quite the opposite. The demand for home PC's drives the computing industry and so cluster computing using off the shelf PC hardware has become the most standard solution for scientific computation. This can largely be attributed to the high availability and low price tag 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 that uses 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 standard microprocessor. The disadvantage of using ASICs is that, once manufactured, the computation that they perform 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 the same way that an ASIC can perform them; very quickly and without software. 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.

### 1.1 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 around 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.

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 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. [1, 2]

## 1.2 Design Scheme

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

For our current methods, interaction cross sections and scattering modifying factors are those given by Lawrence Livermore National Lab's evaluated photon data library (EPDL97). [3] Incoherent scattering distribution functions are given by the product of the standard differential Klein-Nishina formula [4] and the incoherent scattering function as described by Hubbell *et al*. 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 *et al*. [5] Equations 1 and 2 below 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).

$$\left[\frac{\partial\sigma_{KN}(E,\theta)}{\partial\Omega}\right]\bullet S(x,Z) \tag{1}$$

$$\left[\frac{\partial\sigma_{T}(\theta)}{\partial\Omega}\right]\bullet\left(F(x,Z)\right)^{2} \tag{2}$$

We also make the assumption in our current methods, that our photon 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 nonexistent. Therefore, our total macroscopic interaction cross section is given by: $\mu_{total} = \mu_{incoherent} + \mu_{coherent} + \mu_{photoelectric.}$

Our test algorithm, utilizing the assumptions described in the proceeding paragraphs, uses a completely internalized design. This means 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 sectional data, scattering data, random number generator initial states, tally points, and geometrical data is included as part of the hardware design downloaded onto the FPGA.

### 1.2.1 Algorithm

The design algorithm used is based off of a per photon event model, which will be capable of evaluating one complete photon interaction event per clock cycle, under ideal circumstances. In this case, ideal circumstances means no more than 4 rejection technique attempts fail during the computation of a scattering angle for either a Compton or Rayleigh scattering event. In our work, one photon event represents all of the computations required to move a photon from one

interaction to the next interaction in a given material including, but not limited to, determination of the mean free path, interaction type, new XYZ coordinates, new 3-D direction, and the generation of all required random numbers. Figure 1 below illustrates our hardware data flow for photon transport.

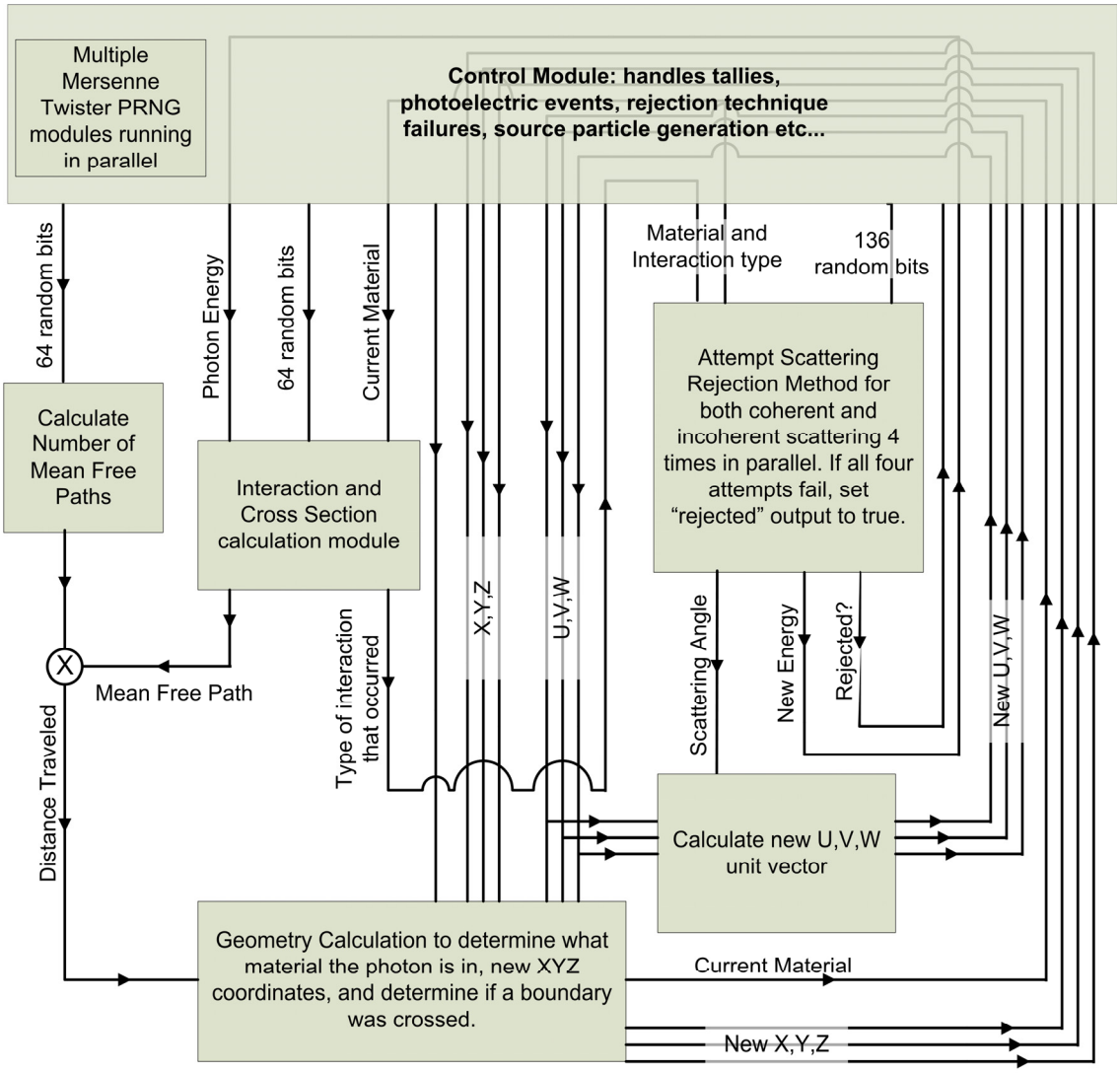


**Figure 1 : Hardware Data Flow for Photon Transport—work done per clock cycle**

### 1.2.2 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 should be discussed in some detail. As a basis for random number generation using an FPGA, we shall use what is widely considered to be one of the best modern pseudorandom number generation algorithms. Matsumoto and Nishimura [6] are responsible for the development of the Mersenne Twister (a pseudorandom number generation algorithm), which has passed the most stringent of statistical tests for

randomness and has an incredibly large period of $2^{19937}$ - 1. 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 following equation:

$$X_{k+n} := X_{k+m} \otimes (X_k^u \mid X_{k+1}^l) A \qquad (3)$$

Where the $\otimes$ symbol denotes the *exclusive or* operation (XOR), N = 624, M = 397, and $X_k$ represents the $k^{th}$ 32-bit random number in a sequence and k ranges from 1 to N. $(X^u_k \mid X^l_{k+1})$ 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 *exclusive or*. [6]

The algorithm is simple and has equally simple hardware requirements. A small 1024 element, 32-bit wide Virtex II* 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 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 implemented on a large Virtex II FPGA, each MT module requires 2 Virtex II blockram units and negligible (less than 1%) of reprogrammable logic slices.

Since our hardware based photon transport scheme is designed to evaluate one photon interaction per clock cycle, it will clearly require more than one 32-bit random number per clock cycle. This is accomplished by running several MT modules in parallel and independently seeding each module using pre-calculated *skip-ahead* seeds in order to avoid the well known parallel pseudo random number generator seed dependence issue.

### 1.2.3 Cross section retrieval

Onboard FPGA blockram modules are used to store three individual cross section tables for each material used in a particular Monte Carlo simulation. The individual cross sections stored for each material are macroscopic cross sections for that element or mixture and correspond to $\mu_{incoherent}$, $\mu_{coherent}$ and $\mu_{photoelectric}$. 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 within the energy specific error bars as described in the EPDL 97 documentation. [7]

* 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.

For the above mentioned polynomial interpolation method to be useful, the interpolation tables must be stored as a log/log function of energy and macroscopic cross section. Log base 2 is used through our Monte Carlo hardware implementation because it is the most naturally implemented logarithm in a binary number system. With an input photon energy *E* ranging from 0 eV to 1,000,000 eV, the $\log_2$ operation is performed on the value E' corresponding to the following simple relationship.

$$E'(E) = \begin{cases} 1\ (\text{eV}) & \text{... if } E < 1\ \text{eV} \\ E\ (\text{eV}) & \text{... if } E \geq 1\ \text{eV} \end{cases} \qquad (4)$$

It should be noted that E' is only used to evaluate cross sections, and only serves the purpose of preventing negative values of $\log_2(E)$. While the details of the implementation of the $\log_2$ function in hardware are discussed later, it should be clear that the range of $\log_2(E')$ is from 0 to ~20. However, since 20 is not a power of 2, we will build our polynomial interpolated cross section storage table to effectively compute cross sectional values of $\log_2(E')$ ranging from 0 to 32 (1 eV to about 4 GeV).

The point wise discretization of each cross section is done by even division in the range 0 to 32 such that the number of discretization points is a power of 2. Coefficients used in the polynomial interpolation equations are stored for each discretization point and describe the log/log behavior of the cross sectional data between points. The number of discretization points used for each cross section depends on the complexity of that cross sections log/log energy curve. Incoherent scattering cross sectional data, as a function of energy, is a fairly smooth curve, and only requires about 128 divisions if linear interpolation is used. We have found that if binomial or trinomial interpolation is used, the number of divisions required to achieve the same precision will be much less (by a factor of 4 or more). This will decrease the amount of onboard memory required to store the cross sectional data at the expense of requiring more multipliers to perform the interpolation. Because of photoelectric electron shell edges and coherent resonance regions, coherent and photoelectric scattering cross sectional data is typically more complex than incoherent scattering cross sections. In the case of photoelectric and coherent cross sections, 1024 or 2048 divisions will be necessary if linear interpolation is used. While the easiest way to perform the point-wise polynomial interpolation is with evenly spaced discretizations, we have found that variable discretization spacing is more efficient for cross sections that are very complex at low energies and very smooth at high energies (i.e. coherent scattering). This method is conceptually more complex, but it is efficiently implemented in hardware and reduces reported error for some cross sections.

### 1.2.4 $\text{Log}_2$(E) computation

$\text{Log}_2$ is a widely used operation in our FPGA based photon transport methods. Obviously it is used for cross section lookup, but it is also used to evaluate the scattering functions and form factors described in equations 1-2 above. In addition, the evaluation of the $\log_e$ computation is performed using $\log_2$ multiplied by a constant. The methodology we have employed to evaluate $\log_2$ is the same methodology used for cross section retrieval, which is based on a point-wise polynomial interpolated lookup table. Point-wise polynomial interpolation is only effective for functions which are defined with in a fixed interval, such as our cross sections which are defined where $\log_2(E')$ ranges from 0 to 32. Since we must be able to perform the $\log_2$ operation for

nearly any value of E', it is necessary to transform E' so that it has a more finite range in order for polynomial interpolation to be an effective method of evaluating $\log_2$. To this end we utilize a simple transform to force the $\log_2$ operand to be within the region [1, 2), where the value of the $\log_2$ function is a smooth, almost linear, curve.

The transform that is used in this case is very simple and can largely be thought of as transforming a fixed decimal-point binary number to a pseudo floating-point equivalent binary number. A pipelined multiplexer array in the FPGA sequentially shifts the fixed decimal-point number until it consists of a mantissa, which is between [1, 2), and a shift factor M corresponding to the number of shifts required to force the mantissa into that range. The number is now in the convenient form described in Equation 5 below.

$$(Mantissa) * 2^{M}, \quad \text{where } 1 \leq (\text{Mantissa}) < 2 \tag{5}$$

$$\log_2[(Mantissa) * 2^{M}] = \log_2(Mantissa) + \text{M} \tag{6}$$

As can be seen by the resultant of Equation 6, this simple transform allows us to use piecewise polynomial interpolation to evaluate $\log_2$ within a small, fixed range to obtain $\log_2$ values for nearly any operand. Using only 512 divisions in the range [1, 2), linear interpolation between divisions and combined with the transform described above, the $\log_2$ function can be evaluated for any input with a maximum associated deviation of about $5 \times 10^{-7}$. Using binomial interpolation with 512 divisions, the maximum associated deviation can be reduced to about $2 \times 10^{-10}$. For the purposes of performing the $\log_2$ operation for cross section lookup, linear interpolation is adequate due to the much more overwhelming error already associated with the cross sectional data. Linear interpolation constants for each of the 512 divisions can readily be stored in two Virtex-II onboard blockram units, where the evaluation of the linear interpolation equation requires just one multiplier block and one adder. The whole scheme is capable of efficiently evaluating one $\log_2$ computation per clock cycle.

Similar methods are used through our hardware designs for the evaluation of sine and cosine functions, square root functions, $2^x$ functions and others. Varying output precisions of each function are selected by: the order of the polynomial used for the interpolation, and the number of interpolation divisions in order to meet the individual precision requirements of each function.

### 1.2.5 Rejection technique

In order 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 by Equations 1 and 2. The Klein-Nishina differential photon scattering cross section can be implemented in hardware with a good deal of precision on a large FPGA and still obtain a one evaluation per clock cycle work rate. Multiple built-in Virtex II multiplier blocks are used, in conjunction with some well placed piecewise polynomial interpolation techniques, in order to evaluate the differential Klein-Nishina function. The hardware implementation details of the Klein-Nishina formula are quite complex, and they involve several tricks in order to maintain good evaluation precision while conserving a one evaluation per clock cycle work rate.

Standard software methods would 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 our hardware based rejection technique. Since our methods 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.

In order to force the probability of rejection to be low, our algorithm simply performs the rejection technique four times in parallel on four parallel scattering distribution functions. A rejection event occurring in all four evaluations is quite rare for medium to low energy Compton scattering, however, in the event it does occur, the interaction calculations for that clock cycle will be marked as incomplete. If a particular interaction is marked as incomplete, the photon interaction parameters stay the same and it is re-evaluated in the following clock cycle. This process is quite efficient if high energy coherent scattering is ignored, producing incomplete interactions less then 15 percent of the time. If high energy coherent scattering is not ignored, the average number of incomplete interactions is increased to about 40 percent.

### 1.3 Final Results

Our 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. We realize that this type of simple Monte Carlo transport problem may be reduced using variance reduction techniques; however, our hardware made no simplifications and performed the complete 3-D simulation. *The geometry and materials used for this simulation were not complex simply because the object of this preliminary research was only to determine if FPGA based Monte Carlo radiation transport is possible, and if so, what speed gains can be attained.*

MCNP-5 was used as a comparison to our hardware method for two reasons. While we could not make MCNP perform the exact simulation that we performed with all of the assumptions that we made, we can use it as a general result comparison to ensure our methods are on target, and there are no significant bugs in our hardware design. More importantly, however, we can obtain a speed comparison from MCNP-5. To closely compare to our hardware design, MCNP was run ignoring Bremsstrahlung radiation and all secondary electrons. However, MCNP did simulate 1$^{st}$ 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 include that MCNP reports slightly higher flux tallies and is doing about 3.5 percent more computational work than our hardware design. Flux comparisons are shown in Table I for 10,000 histories.

**Table I. Flux tallies: MCNP-5 vs. FPGA hardware design**

| Tally Distance | MCNP-5 | FPGA Hardware Design | % Difference | % Std. 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 |

The flux tallies reported by our FPGA hardware design are very close to those reported by MCNP. The fluxes were lower in regions where photoelectric and Compton ionizations (and thus fluorescence yield) were highest, but this was expected.

When our entire hardware design for this Monte Carlo photon transport problem is heavily pipelined and then placed and routed on a Xilinx Virtex-II pro 100 FPGA using Xilinx Foundation 6.3I, the FPGA resources that are consumed are listed in Table II.

**Table II. Virtex-II pro 100 FPGA utilization**

| | Used | Available | Percent Utilization |
|---|---|---|---|
| **Number of Slices** | 11944 | 44096 | 27% |
| **Number of Slice Flip-Flops** | 12648 | 88192 | 14% |
| **Number of 4-Input LUTs** | 17747 | 88192 | 20% |
| **Number of Block-Rams** | 79 | 444 | 17% |
| **Number of 18x18 Multipliers** | 117 | 444 | 26% |
| **Maximum Clock Speed** | 136.733 MHz | | |

The usage numbers from Table II that matter the most are the number of 4-input LUTs, the number of Block-Rams, and the number of 18x18 multipliers. Only about 1 quarter of the available logic space on this large FPGA is used by our Monte Carlo transport design. There is a large amount of room for adding complicated geometries and more material cross sections to the simulation while still conserving a 1 clock cycle per photon interaction ideal work rate. The hardware photon transport designs that we made had no energy cutoffs for any interaction type, including high energy coherent scattering. Therefore, a 10,000 history simulation which

consisted of 76,231 photon interactions actually took 129,541 clock cycles due to lost efficiency in the rejection technique for higher energy scattering. At 136.7 Mhz, one instance of our hardware photon transport design on a Xilinx Virtex-II pro 100 FPGA can complete a 10,000 history simulation (129,541 clock cycles) in exactly $9.476298 \times 10^{-4}$ seconds. *This corresponds to a rate of about 633.15 million complete photon histories per minute.*

The photon history work rate last stated utilizes about ¼ of the available logic space on the Xilinx Virtex-II pro 100 FPGA. For accurate comparison to MCNP, which we have simulated at 100% CPU time on a 3.2 GHz Pentium-4 PC, an effort must be made to put the resources of the entire FPGA to work on this transport problem. Therefore, three parallel instances of our photon transport hardware design will be run. *This corresponds to a maximum work rate of about 1.55 BILLION complete photon histories per minute.*

As we stated earlier, MCNP-5 was run on a 3.2 GHz Pentium-4 PC, under Windows XP Professional with no other processes running. MCNP-5 performed the same simulation, ignoring secondary electrons and Bremsstrahlung radiation. The only difference in the simulation that MCNP-5 performed is that it simulated 1$^{st}$ 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.

Executing almost the identical simulation, the speed increase that we have found possible is overwhelming. The Xilinx Virtex-II pro 100 FPGA is capable of executing this particular photon transport problem **650 times faster** than it can be executed on a 3.2 GHz Pentium-4 PC running MCNP-5.

## 2 CONCLUSIONS

It is clear that some of the methods used in our hardware design are not identical to the methods that MCNP uses to perform photon transport. The methods cannot be completely identical because the platforms are so different. For instance, a desktop PC performs every computation to floating point precision (23 bits) because there are few alternatives. Hardware designs on FPGAs have unlimited versatility allowing computations to be custom tailored to operate with the exact output precision required. In some instances the computations in our design are executed with slightly less than floating point precision, in others computations are executed with much more than floating point precision. The output precision of each operation varies based on the error associated with the experimental data that these operations depend on (cross sections, form factors, etc…).

This is just a starting point for hardware based Monte Carlo particle transport. While this is a comparison based on one particular Monte Carlo simulation, we expect that a 2-3 order of magnitude speed increase can be realized for many forms of Monte Carlo radiation transport.

## 3 REFERENCES


1. K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computing Surveys*, **Vol 34, No. 2**, pp.171-210 (2002).
2. S. Hauck, "The Future of Reconfigurable Systems," *5th Canadian Conference on Field Programmable Devices*, Montreal, June (1998).
3. "EPDL97: the Evaluated Photon Data Library, '97 Version," http://www.llnl.gov/cullen1/DOCUMENT/EPDL97/epdl97.htm (1997).
4. O. Klein, Y. Nishina, *Zeits. für Physik*, **Vol 52**, pp.853 (1929).
5. J. Hubbell, Wm. Veigele, E. Briggs, R. Brown, D. Cromer, R. Howerton, "Atomic Form Factors, Incoherent Scattering Functions, and Photon Scattering Cross Sections," *J. Phys. Chem. Ref. Data*, **Vol 4**, pp.471-539 (1975).
6. M. Matsumoto, T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudorandom Number Generator," *ACM Transactions on Modeling and Computer Simulations: Special Issue on Uniform Random Number Generation*, **Vol 8**, pp.3-30 (1998).
7. "EPDL97 Documentation: Report UCRL-50400, Vol.6, Rev.5, 1997," http://www.llnl.gov/cullen1/document/epdl97/epdl97.pdf (1997).