

# Monte Carlo Simulation on Field Programmable Gate Arrays using VIVA<sup>®</sup>

Siddhartha Krishnamurthy  
Starbridge Systems, Inc.  
At NASA Langley Research Center  
Hampton, VA

## Introduction

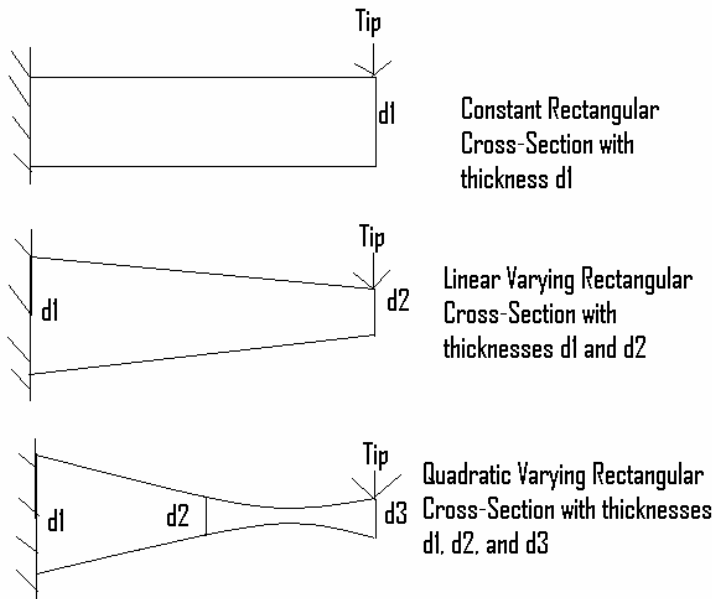
Monte Carlo simulation methods are among the most powerful and commonly used techniques for analyzing complex physical problems. Applications can be found in an array of areas including radiation transport, structure analysis, and river basin modeling. In addition to exploring more efficient simulation techniques, analysts performing Monte Carlo simulation usually seek advanced computer systems to perform more complex studies. Super computers based on Field Programmable Gate Arrays (FPGAs) can run operations in parallel to reduce computation time. In Monte Carlo simulation, parallelism can be very useful. In addition to running basic computations in parallel, random samples can be taken concurrently with a FPGA and reduce the computation time to fractions of the time taken for synchronous samplings. Furthermore, for integrals over four dimensions, Monte Carlo is the only method to evaluate these integrals. Other methods such as Runge-Kutta and numerical integration do not work or are much more resource intensive than Monte Carlo.

The computational power of FPGAs coupled with ease of designing a circuit on VIVA<sup>®</sup> gives Starbridge Systems's supercomputers much promise in being used to solve complex problems involving Monte Carlo simulations. This research explored how and sought to verify if Monte Carlo simulations can be implemented more efficiently on an FPGA with the help of VIVA<sup>®</sup>.

## Problem Statement

The Analytical and Computational Methods Branch (ACMB) at NASA Langley Research Center performs structure and mechanics simulations and uses Monte Carlo methods (MCMs) in many applications. Currently, it mostly uses standard and multi-processor computers for performing these computations. If a Monte Carlo simulation can be implemented easily and performed efficiently on an FPGA-based machine, these machines can increasingly be used over standard computers to execute MCMs. Starbridge Systems, Inc., has provided ACMB with an HC-38 FPGA-based machine as well as its VIVA<sup>®</sup> programming environment. The performance of the FPGA-based machine must be analyzed to verify that it can be a better alternative to standard processors, multi-processor computers, as well as dedicated non-reconfigurable processors for running MCMs.

Monte Carlo simulation can be used to optimize cantilever beams based on stress, weight, and displacement constraints. Analysis will be performed on three types of beams. All beams have rectangular cross-sections. They all have the same length, width, density, and Young's Modulus. The load applied at the tip of a beam is the same for all other beams. The thickness varies among the beams. The three different types of beams are the constant cross-section beam, linear varying cross-section beam, and quadratic varying cross section beam shown in Figure 1.



**Figure 1.** Three different types of beams being analyzed.

The values  $d_1$ ,  $d_2$ , and  $d_3$  produce the thickness.  $d_1$ ,  $d_2$ , and  $d_3$  are  $> 0$ , and  $d_1 > d_2 > d_3$ .  $d_1$ ,  $d_2$ , and  $d_3$  are randomly generated values between 0 and 1 inch.

The displacement of a cantilever beam at the tip can be found using the equation:

$$Displ = \frac{P}{E} \int_0^L \frac{x^2}{I} dx = \frac{P}{E} \int_0^L \frac{x^2}{\frac{wd^3}{12}} dx$$

$P$ = load applied at tip,  $E$ = Young's Modulus,  $I$ = Area Moment of Inertia,  $w$ = width,  $L$ = length,  $d$ = thickness,  $Displ$ = displacement, and  $x$  is a distance along the length of the beam in  $[0, L]$ .

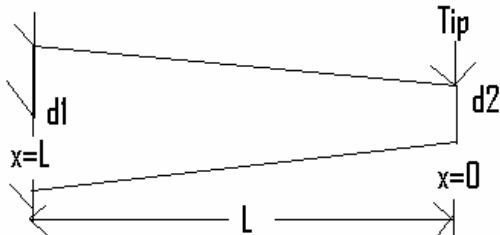
$I = \frac{wd^3}{12}$  for a rectangular cross section about the x-axis passing through the centroid.

To find the solution to the integral using Monte Carlo integration, the following equation is used:

$$Displ = \frac{P}{E} \sum_{i=0}^N \frac{x_i^2}{I} \frac{L}{N} = \frac{P}{E} \sum_{i=0}^N \frac{x_i^2}{\frac{wd^3}{12}} \frac{L}{N}$$

$X_i$  is a random value in the interval  $[0, L]$ , and  $N$  is the number of random values taken. The larger the value of  $N$ , the more accurate the displacement will be.

The thickness,  $d$ , varies from beam to beam. For a constant cross-section, the thickness=  $d_1$ . For a linear varying cross section, the thickness,  $d$ , is given by the equation,  $d = bx + c$ , or for Monte Carlo integration,  $d = bx_i + c$ .  $b$  and  $c$  are functions of  $d_1$  and  $d_2$ . Taking  $x=0$  at the tip and increasing  $x$  towards  $L$  from right to left as shown in Figure 2, it is found that  $b = \frac{d_1 - d_2}{L}$  and  $c = d_2$ .



**Figure 2.** How  $x$  varies along length of beam.

For a quadratic varying cross section, the thickness,  $d$ , is given by the equation,  $d = ax^2 + bx + c$ , or for Monte Carlo integration,  $d = ax_i^2 + bx_i + c$ .  $a$ ,  $b$ , and  $c$  are functions of  $d_1$ ,  $d_2$ , and  $d_3$ . Taking  $x=0$  at the tip again,  $a = \frac{2d_1 - 4d_2 + 2d_3}{L^2}$ ,  $b = \frac{-d_1 + 4d_2 - 3d_3}{L}$ ,  $c = d_3$ .

In summary, the Monte Carlo integration functions for the constant, linear, and quadratic beams are:

$$Displ = \frac{P}{E} \sum_{i=0}^N \frac{x_i^2}{wd_i^3} \frac{L}{N} \quad \text{Constant Beam}$$

$$Displ = \frac{P}{E} \sum_{i=0}^N \frac{x_i^2}{w(bx_i + c)^3} \frac{L}{N} \quad \text{Linear Beam}$$

$$Displ = \frac{P}{E} \sum_{i=0}^N \frac{x_i^2}{w(ax_i^2 + bx_i + c)^3} \frac{L}{N} \quad \text{Quadratic Beam}$$

### Finding the Optimum Beam Configuration

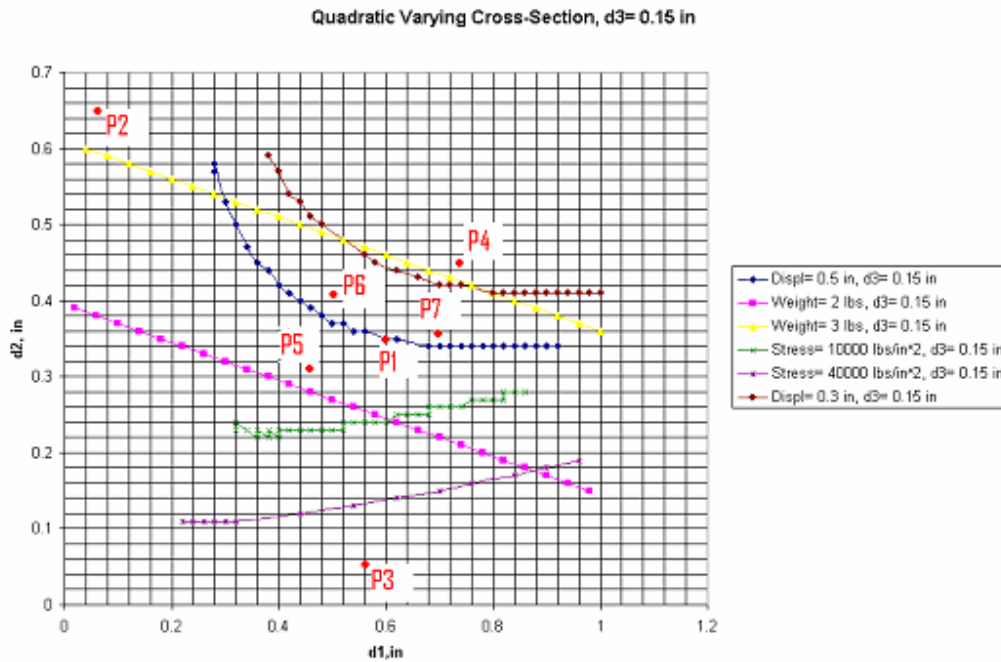
All the beams analyzed share the following parameters:

$P = 20$  lbs. // Load at tip  
 $L = 24$  inches // Length  
 $w = 3$  inches // Width  
 $E = 10e6$  lbs/in<sup>2</sup> // Young's Modulus  
 $p = 0.097$  lbs/in<sup>3</sup> // Density  
 $S = 40000$  lbs/in<sup>2</sup> // Maximum stress allowed before beam deforms or breaks  
 $DisplMax = 0.5$  in // Maximum displacement allowed at tip

$$d1 > d2 > d3$$

Monte Carlo simulation is performed to find the optimum configuration. Hundreds or thousands of different beams, each with its combination of  $d1$ ,  $d2$ , and  $d3$  are analyzed. Of course, for linear varying beams there is no  $d3$  and for constant beams there are no  $d2$  and  $d3$ .

The goal of the optimization procedure is to find the beam with the lowest weight that fits within stress and displacement constraints. The following graphs will be used to illustrate how optimization is achieved for quadratic varying beams. For now and for simplicity, let  $d3$  be constant at 0.15 in.  $d1$  and  $d2$  will be randomly sampled. The weight, stress, and displacement curves produced as  $d1$  and  $d2$  vary are shown in Graph 1.



**Graph 1.** Weight, stress, and displacement curves for  $d3=0.15$  in

From the graph, it can be seen that the stress decreases “up” the graph, the weight decreases “down” the graph, and the displacement decreases “up” the graph. Point P1 on Graph 1 is the optimum configuration point because it is lower than the maximum stress and has the lowest possible weight without having a displacement  $> 0.5$  in.

Let P2 be chosen as a configuration point.  $d1 < d2$  at this point, and P2 is discontinued from further analysis. P3 is the next point chosen. Here,  $d1 > d2$ , but the stress at this point is greater than 40000 lbs/in<sup>2</sup>, and this point is discontinued from further analysis. The next configuration generated is at P4. P4 is less than the maximum stress, but it is  $> 3$  lbs, which is initially set to be the lowest weight. Therefore, P4 is discontinued from further analysis. The next point, P5, is less than 3lbs and Monte Carlo integration is performed on it to find the displacement at the tip. The displacement is  $> 0.5$  inches and it is discontinued. P6 is generated and it is  $< 3$ lbs and  $> 0.5$  in. The weight at P6 becomes the new lowest weight. If another point is generated such that its weight is  $> P6$ 's weight but  $< 3$ lbs and  $> 0.5$  in it will still be discontinued. P6 is the current optimum configuration until P7 is generated with a lower weight. The process repeats for the total number of

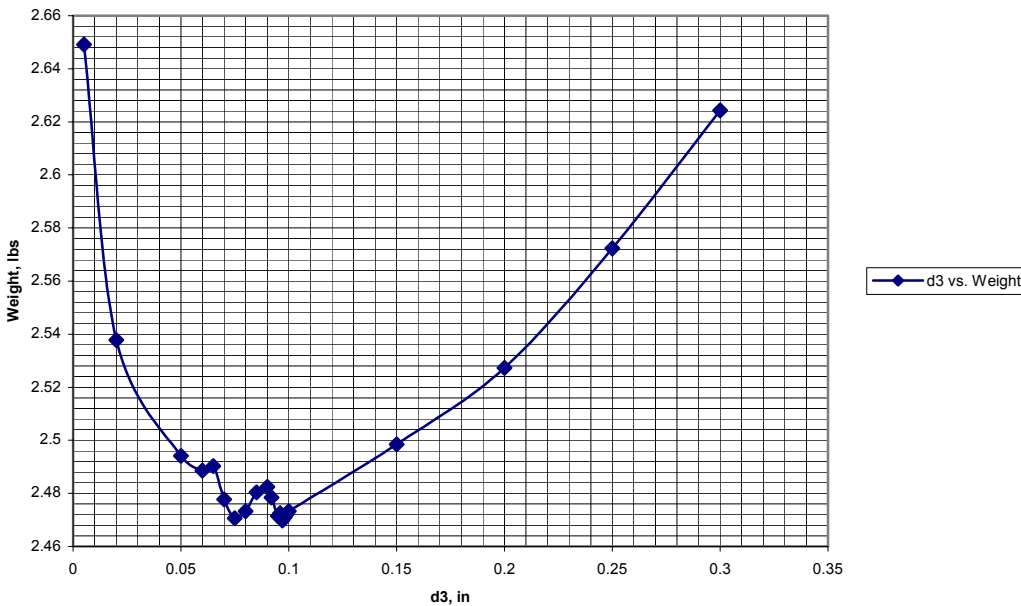
beam configurations desired and the only place where an optimum point can occur moves closer and closer to point P1.

In summary, the process followed is:

- Check if  $d1 > d2 > d3$
- If not → produce next configuration
- Else
- If stress  $> 40000 \text{ lbs/in}^2$  → produce next configuration
- Else
- If weight  $>$  current lowest weight → produce next configuration
- Else
- Perform Monte Carlo integration to find displacement
- If displacement  $> 0.5$  inches → produce next configuration
- Else
- current lowest weight = weight
- This configuration is the current optimum configuration → produce next configuration

However,  $d3$  varies and so the curves shift in the  $d1, d2$  plane. Therefore, it is important to know which value of  $d3$  has the optimum point with lowest weight possible. Clearly from Graph 1, the optimum configuration will have a displacement close to 0.5 inches. Graph 2 shows the optimum points found for different values of  $d3$  and their associated weights.

**d3 vs. Weight for Displ= 0.5 in**

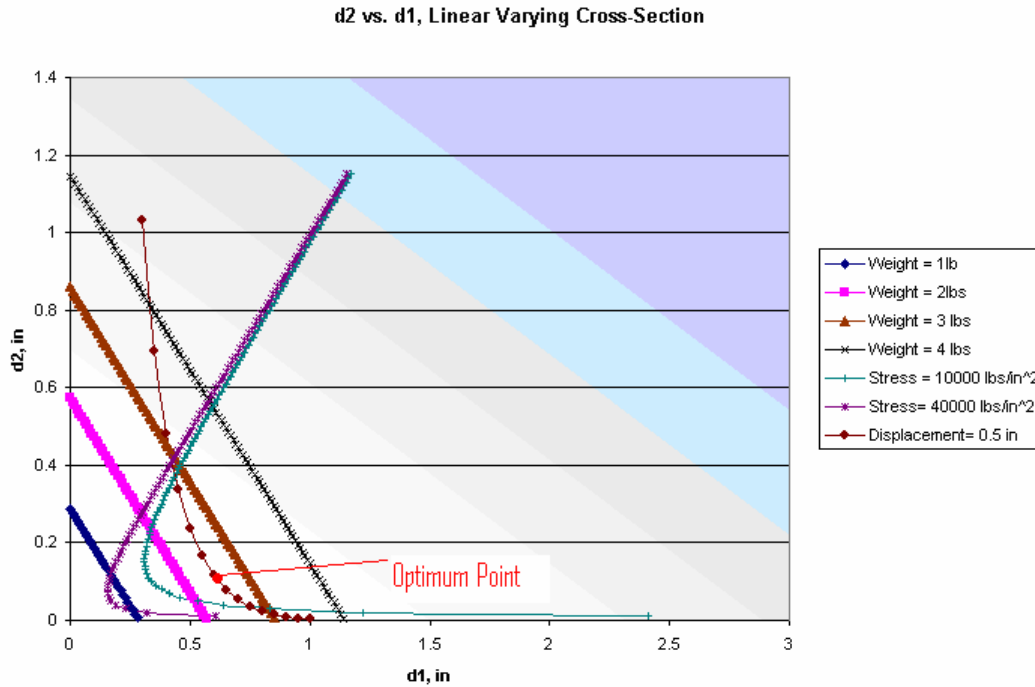


**Graph 2.**  $d3$  vs. Weight.

Graph 2 shows that the optimum configurations with the lowest weights occur for  $0.06 < d3 < 0.1$  inches.

Constraining d3 to be between 0.1 and 0.07 inch can produce an optimum configuration point faster and closer to the true optimum point. The more constrained around the optimum point d1, d2, and d3 are, the more accurate the optimum configuration.

For linear varying beams, the graph of d1 vs. d2 is shown in Graph 3. The way in which the optimum point is reached is similar to that for the quadratic varying beams, but there is one less variable.



**Graph 3.** Optimization graph for linear varying cross-section beams

For constant cross-section beams, knowing what the optimum point should be is simpler and graphs are not needed. Just knowing that a decrease in weight means a larger displacement is enough.

After solving the equations,  $Displ = \frac{4PL^3}{Ewd_1^3}$  and  $Weight = pwLd_1$  for  $Displ = 0.5$  inches, the lowest

weight is found to be 2.93 lbs for  $d1 = 0.4193$  inches. The optimum point is found the same way as for the quadratic and linear varying beams.

### Measuring Performance

In general, FPGAs can perform computations faster than general-purpose machines, especially because of their ability to run operations in parallel. As stated previously, this research explored how and sought to verify if this increase in efficiency can be obtained for Monte Carlo simulations on a FPGA with the help of Starbridge Systems’s VIVA<sup>®</sup> programming environment. The algorithm to find the optimum beam configuration was developed to run on both standard CPUs using C++ and on the FPGA using VIVA<sup>®</sup>. Performance of the different implementations and

machines was measured in terms of the time it took to run the Monte Carlo simulation algorithm. The precision of answers for the computations performed was not used to determine performance. The reason for not including precision is that the pseudo random number generators used are different between the VIVA and C++ implementations. Answers can be made more precise by using certain sequences of numbers. The computations did need to produce accurate answers before they were analyzed for run time. It is highly desirable to compare a FPGA implementation to a cluster or multi-processor implementation. However, this comparison could not be accomplished during the research period, but must be done in the immediate future.

### Running Algorithm with C++ using Standard CPUs

Cantilever beam optimization algorithms for the constant, linear, and quadratic beams were implemented in C++ and run.

The times taken for optimization analysis on various processors are given in Table 1 for the three types of beams. 1000 beam configurations were run and 100000 samples were taken for Monte Carlo integration. C++ was the only user program running on the Windows™ operating system when these times were taken.

**Table 1.** Times taken to run three types of beam analysis on various processors with C++. 1000 beam configurations were run and 100000 samples were taken for Monte Carlo integration.

Processor	Constant Beam Analysis Time (seconds)	Linear Beam Analysis Time (seconds)	Quadratic Beam Analysis Time (seconds)
Intel Pentium III, 550 MHz	29.622	10.035	12.418
Intel Pentium 4, 1600 MHz	15.713	5.698	6.029
Intel Pentium 4, 2400 MHz	10.716	3.064	4.066
AMD Athlon, 1250 MHz	10.465	3.595	4.166

The Monte Carlo integration times to find displacement for the three types of beams on various processors are given in Table 2. 96000 samples were taken for each run.

**Table 2.** Monte Carlo integration Times to find displacement

Processor	Constant Beam Analysis Time (seconds)	Linear Beam Analysis Time (seconds)	Quadratic Beam Analysis Time (seconds)
Intel Pentium III, 550 MHz	0.091	0.09	0.09
Intel Pentium 4, 1600 MHz	0.041	0.05	0.06
Intel Pentium 4, 2400 MHz	0.03	0.03	0.05
AMD Athlon, 1250 MHz	0.03	0.03	0.03

Constant and linear cantilever beam optimization analyses based on only stress and weight constraints were run on the 2400 MHz Intel Pentium processor with 10000 samples. The runtimes are shown in Table 3.

**Table 3.** Stress-weight optimization runtimes for Intel Pentium 4, 2400MHz

Processor	Constant Beam Stress-Weight Optimization Run Time, 10000 samples (seconds)	Linear Beam Stress-Weight Optimization Run Time, 10000 samples (seconds)
Intel Pentium 4, 2400 MHz	0.013	0.013

### VIVA<sup>®</sup> Implementation

Since VIVA<sup>®</sup> is still being improved by Starbridge Systems, it cannot be expected that all operations will perform satisfactorily or work properly. It must still be seen whether or not VIVA<sup>®</sup> has the potential to allow users to implement Monte Carlo simulations easily on a FPGA to obtain faster computations than from traditional machines.

Due to software problems with VIVA<sup>®</sup>, a full cantilever beam analysis of all three types of beams could not be performed on a FPGA. Despite this difficulty, results were obtained that indicate that a Monte Carlo simulation can easily be implemented with VIVA<sup>®</sup> on a FPGA and have better computational performance than when implemented on current CPUs. VIVA<sup>®</sup> is continually being improved by Starbridge Systems and the number of software problems is expected to decrease.

The following parts of the overall optimization algorithm were able to be successfully developed on VIVA<sup>®</sup> and executed on the FPGA:

1. Constant beam optimization based on stress and weight constraints
2. Linear beam optimization based on stress and weight constraints
3. Constant beam displacement finder using up to two Monte Carlo integrators running in parallel

These next two parts were able to be developed on VIVA<sup>®</sup> and run on the emulator, but were unable to run on the FPGA:

1. Entire constant beam optimization algorithm using stress, weight, and displacement constraints
2. Linear beam displacement finder using up to two Monte Carlo integrators running in parallel

In designing the VIVA<sup>®</sup> implementation of the cantilever beam analysis, running as many operations in parallel as possible was a major goal. The steps to be followed and the computations necessary to perform the analysis were already developed in the C++ version of the algorithm. Unlike the synchronous sequence of operations in C++, operations that did not rely on each other to perform their computations were run in parallel. The following operations were made to run in parallel:

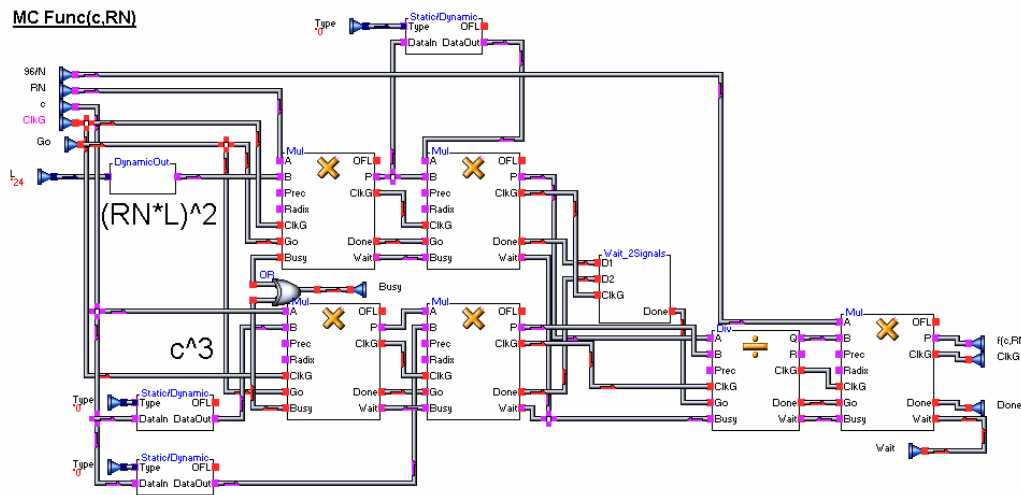
For the linear and quadratic beam analyses, d1, d2, and d3 could be randomly chosen in parallel.

For the quadratic beam analysis, finding the values of a and b as a function of d1, d2, and d3 could be performed in parallel.

Both the stress and weight analyses were performed in parallel. It was necessary to wait for both analyses to finish their computations before continuing to the displacement analysis or continuing to the next configuration; otherwise, signals telling the next operation to execute would be outputted twice in the same iteration. The displacement analysis required the most computations. Therefore, if the thickness constraints ( $d1 > d2 > d3$ ), stress constraints, or weight constraints were not met, the displacement analysis did not have to run, thus saving run time. For this reason, the displacement analysis was not run in parallel with the stress and weight analyses even though its computations were independent of them.

In the displacement analysis, multiple Monte Carlo (MC) integrators could be performed in parallel. If 96000 samples were to be taken, 96000 iterations needed to be performed on a standard CPU. With the FPGA, it is possible to take multiple samples in parallel. Therefore, if MC integration is being performed twice in parallel, then only 48000 iterations need to be run and still have 96000 samples. If three MC integrators are running in parallel, then 32000 iterations need to be run, and so forth.

Within the MC integrator, certain mathematical computations could be performed in parallel. For example, in performing MC integration on a constant cantilever beam, the following operation was performed:  $\frac{(RN \times L)^2}{c^3} \times \frac{96}{N}$  where RN is a randomly chosen value, and L and 96/N are constants with N being the total number of samples to be taken. Since  $(RN \times L)^2$  and  $c^3$  do not rely on each other, they could be performed in parallel, as shown in the VIVA<sup>®</sup> implementation in Figure 3.



Operation:  $\frac{((RN \times L)^2)}{c^3} \times \frac{96}{N}$   
 $(RN \times L)^2$  and  $c^3$  are performed separately and in parallel

**Figure 3.** VIVA<sup>®</sup> implementation of  $\frac{(RN \times L)^2}{c^3} \times \frac{96}{N}$

The linear and quadratic beam integrators have similar mathematical operations that can be performed in parallel.

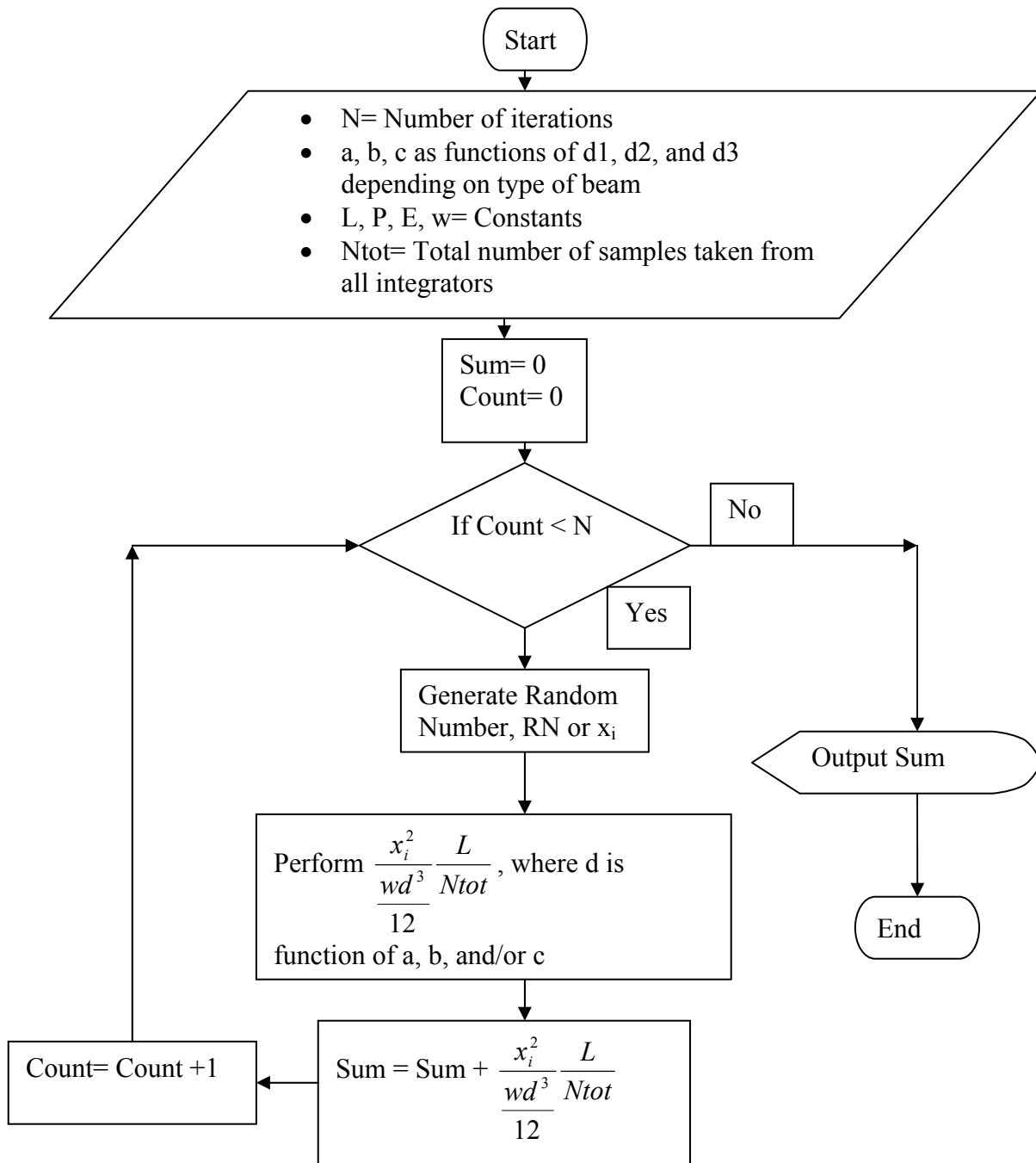
The procedure followed by the MC integrator will now be described and the flowchart in Figure 4 will be used to illustrate the design.

An accumulator object starts the integration by setting Sum= 0 and signaling a loop object to initiate the first random sample. The random number,  $x_i$ , is evaluated in the function  $\frac{x_i^2}{12} \frac{L}{wd^3 Ntot}$ , where d

is a function of a, b, and c depending on the type of beam. Ntot is the total number of samples taken from all integrators. Thus, N, the number of iterations, = Ntot/number of integrators. The

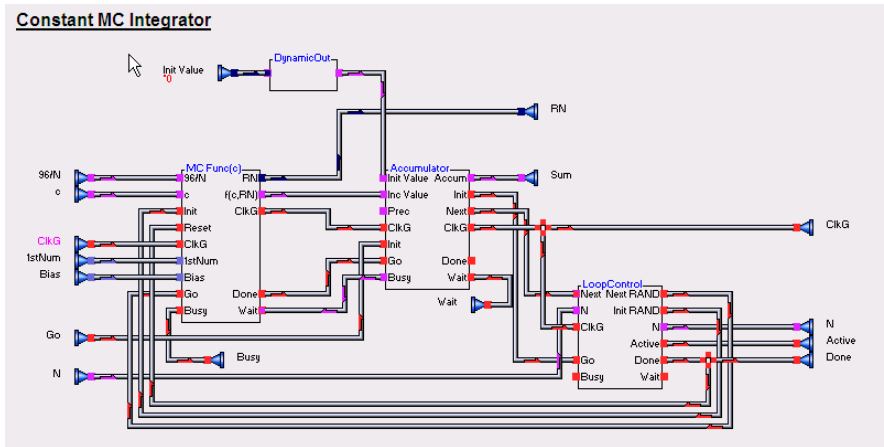
accumulator object performs  $Sum = Sum + \frac{x_i^2}{12} \frac{L}{wd^3 Ntot}$  and the process is continued N times. The

final sum is then outputted. It is eventually multiplied by P/E to give the displacement.

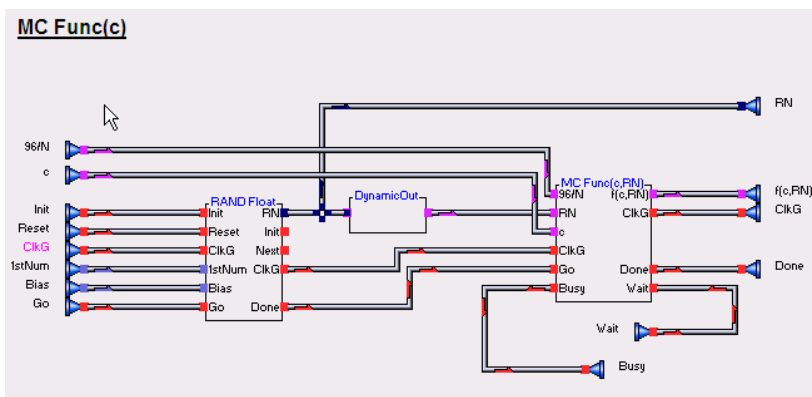


**Figure 4.** Flowchart describing how Monte Carlo integrator works

The VIVA<sup>®</sup> implementation of the MC integrator for a constant beam is shown in Figure 5. Figure 6 shows the inside of the MC Func(c) object which contains the random number generator.

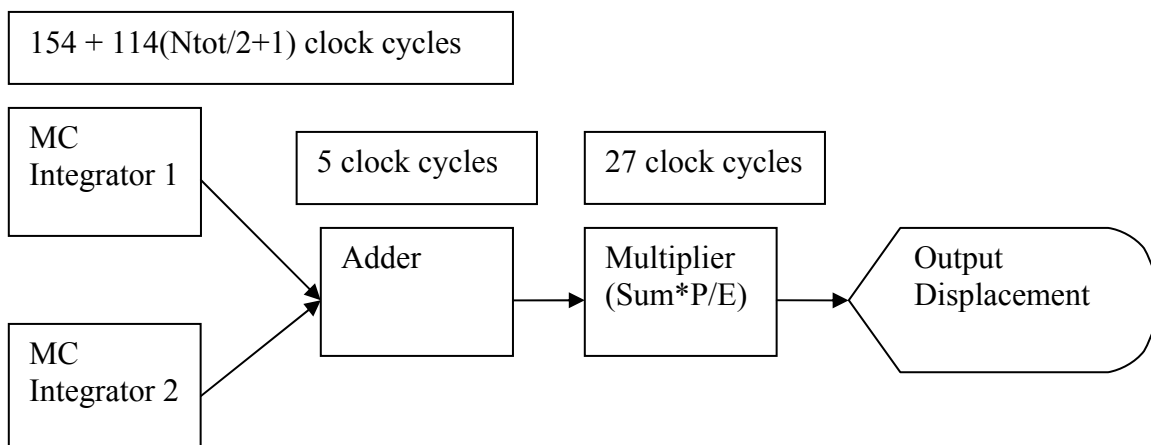


**Figure 5.** Constant beam Monte Carlo integrator



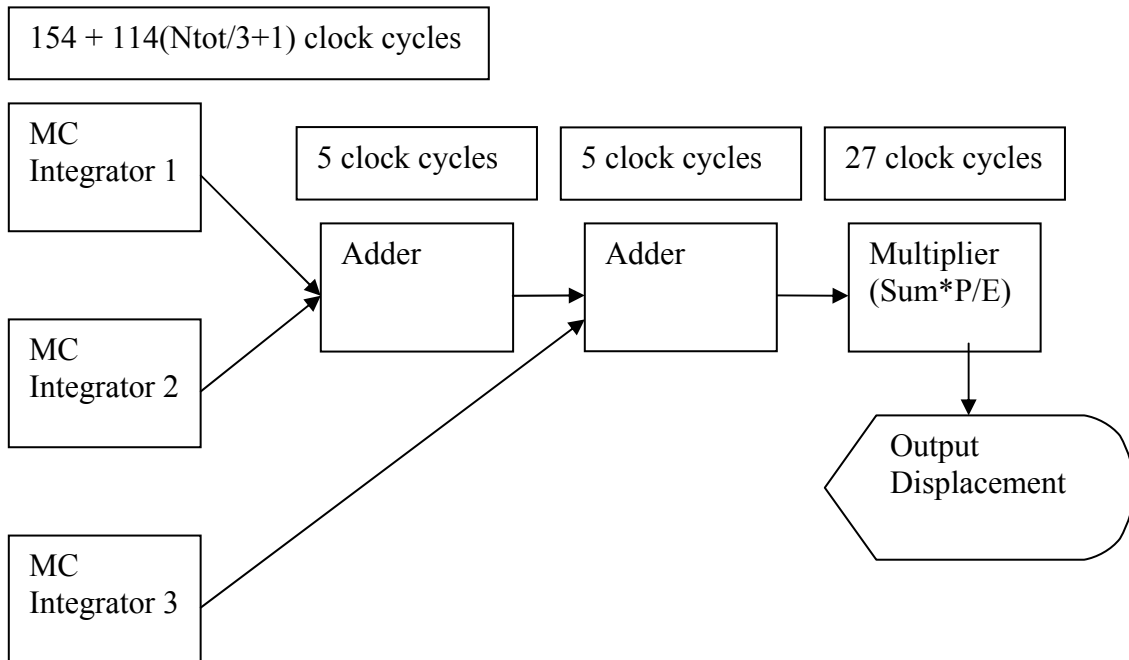
**Figure 6.** MC Func(c) object. Contains random number generator, RAND Float

As mentioned before, multiple MC integrators can be performed in parallel. Figure 7 shows a simplified diagram of a beam displacement finder with two MC integrators being performed in parallel. In VIVA 2.2<sup>®</sup> using Corelib\_b10\_01, it has been found that, for a constant beam integrator, the number of clock cycles for the MC integrators in Figure 7 is  $154 + 114(N_{tot}/2+1)$ , where  $N_{tot}$  is the total number of samples taken. 5 clock cycles are taken for the Adder and 27 for the Multiplier.



**Figure 7.** Beam displacement finder with 2 Monte Carlo integrators

If there are three MC integrators an extra Adder is used, and the clock cycles taken for the integrators changes as shown in Figure 8.



**Figure 8.** Beam displacement finder with 3 Mont Carlo integrators

With 4 MC integrators, the integrator clock cycles will change to  $154 + 114(N_{tot}/4+1)$  and an extra Adder will run in parallel to the first Adder in Figure 8. In summary, the number of clock cycles taken for a constant cantilever beam displacement finder is given by the equation:

$$154 + 114[N_{tot}/S+1] + 5(F(S)) + 27 \text{ clock cycles,}$$

where  $S$  is the number of MC integrators and  $N_{tot}$  is divisible by  $S$ ,

$$F(S) = S/2 \text{ when } S \text{ is even}$$

$$F(S) = (S+1)/2 \text{ when } S \text{ is odd,}$$

The values of 154 and 114 in the above equation will change for linear and quadratic beam integrators.

Different random number generators were used between the VIVA<sup>®</sup> and C++ implementations of the displacement finder. This difference can be allowed because this research sought to explore how and to verify if increased efficiency for the displacement finder can be obtained on a FPGA through the running of integrators in parallel using VIVA<sup>®</sup>. It did not seek to explore whether or not running a single integrator on VIVA<sup>®</sup> is more efficient than a C++ implementation. If the type of random number generator used in VIVA<sup>®</sup> is changed, the values of 154 and 114 in the clock cycle equation must also change.

## Results

In testing the displacement finder in VIVA<sup>®</sup>, 96000 samples were taken as was done in C++. A circuit to calculate displacement of a constant beam with 2 MC integrators was successfully implemented and run on Starbridge Systems's HC-38 FPGA-based machine. According to the equation above, the number of clock cycles that theoretically should be taken is 5472268 clock cycles. The frequency of the FPGAs on the HC-38 is 66MHz, and it takes 15ns for one clock cycle. Therefore, the theoretical time that should be taken for the displacement finder using 2 MC integrators is 0.08208 seconds. The number of clock cycles taken for the actual implementation was 5456157 clock cycles, or 0.08184 seconds, which is an error of 0.3% from the theoretical. It is hypothesized that the reason for the error is that the arithmetic objects take up more or less clock cycles than was calculated to produce the clock cycle equation because floating point numbers are being used. For example, the MC Func(c, RN) object in Figure 3, which performed the bulk of the integration arithmetic, took an average of 107 clock cycles to produce an output based on specific inputs. Some floating point numbers take up less clock cycles to perform arithmetic on. During the actual run, many floating point numbers computations may have taken less than 107 clock cycles.

Circuits with 3 and 8 MC integrators were developed, but not run. The main reason for not being able to run them is that the synthesize time in adding more integrators increases significantly. Running these circuits on VIVA 2.3<sup>®</sup>, which has a synthesize time orders of magnitudes less than VIVA 2.2<sup>®</sup>, should enable these larger circuits to be synthesized faster and run on the FPGA. However, based on the results with 2 MC integrators, the time taken using 3 and 8 integrator circuits to run should be close to the theoretical 0.0547 and 0.0205 seconds, for the 3 and 8 MC integrators respectively.

The time taken for the 2 MC integrator circuit to run on a 66MHz FPGA is faster than the time taken for the C++ algorithm to execute on a 550MHz Pentium III processor. An 8 MC integrator circuit running on a 66MHz FPGA machine can perform better than a 2400MHz Pentium 4 processor and a 1250MHz AMD Athlon processor. The time taken should be even lower for FPGAs with higher frequencies. Note that with only 1 MC integrator, the clock cycle equation predicts that the run time will take 0.16 seconds for 96000 samples, which is slower than the speeds taken by the CPUs tested. Thus, parallelism must be exploited whenever possible on an FPGA to make it run faster than a standard CPU.

The ease of implementing algorithms in VIVA<sup>®</sup> gives an added advantage. For example, a group of 8 MC integrators and their Adders can be formed into one object. This object implemented 8 more times can produce 64 integrators running in parallel. Using 8 of these objects with Adders to form another object and implementing it 8 times, 512 integrators could easily be made to run in parallel, and so on. Thus, it is possible to have 96000 integrators running in parallel to take 96000 samples. If such a circuit is implemented, it would take 0.003606 seconds to perform MC integration on a 66MHz FPGA to find the displacement of a constant cantilever beam.

The constant and linear cantilever beam analyses using just stress and weight constraints were implemented in VIVA<sup>®</sup> and performed on the FPGA. Again, the stress and weight analyses were run in parallel. 10000 samples were taken during these Monte Carlo simulations. The weight and stress of a constant cantilever beam are given by:

$$Weight = \rho \times L \times w \times d$$

$$Stress = \frac{6PL}{wd^2} \leq Stress_{allowed}$$

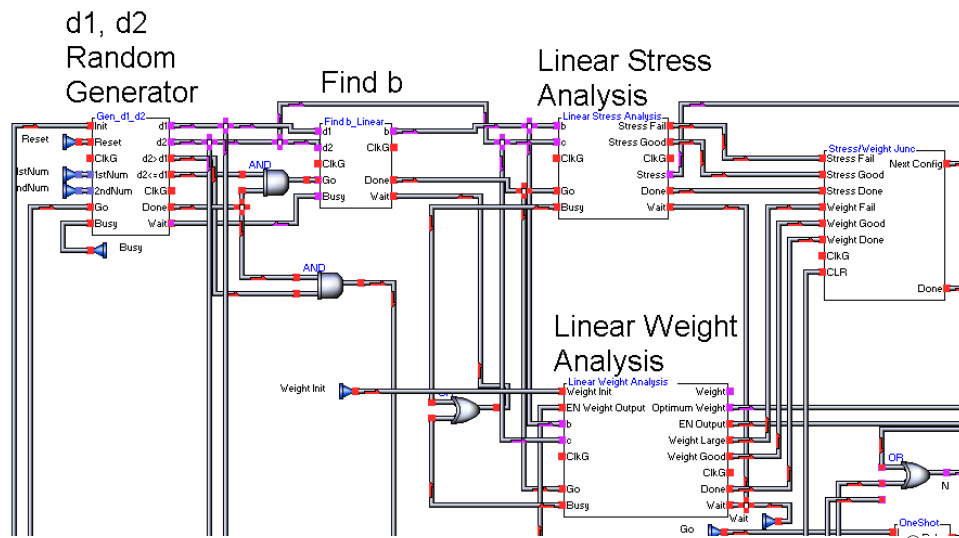
When  $Stress_{allowed} = 40000 \text{ lbs/in}^2$ , the theoretical optimum value is  $d = 0.155$  inches to give a minimum weight of 1.082 lbs. The VIVA<sup>®</sup> implementation produced  $d = 0.1549530$  inches, 0.03% from the theoretical, with optimum weight at 1.082191 lbs. The time taken to run the simulation was 0.0057 seconds. For the linear varying beam, the time taken for optimization was 0.01266 seconds. The optimum  $d_1$  and  $d_2$  values were 0.209 inches and 0.096 inches, respectively. The optimum weight was 1.066489 lbs, close to what the theoretical optimum point should be without the displacement constraint when  $Stress_{allowed} = 40000 \text{ lbs/in}^2$  as shown in Graph 3. The C++ version of the stress-weight optimization implementation was run on only the Intel Pentium 4, 2400MHz processor. The runtime was 0.013 seconds for both the constant and linear beam stress-weight analyses. However, it cannot be stated that C++ implementations perform slower than VIVA<sup>®</sup> implementations of the stress-weight algorithms because of the argument that follows.

It is important to note that the sequence of random numbers has an impact on computation time for the stress-weight optimization algorithm, especially for the linear varying beam. If the constraint of  $d_2 < d_1$  is not met, the stress and weight analyses are not performed. The random number generator used in VIVA<sup>®</sup> for the linear varying beam was different than the one used for the constant beam. Furthermore, the C++ implementation had a different random number generator than the ones used in VIVA<sup>®</sup>. Also, storing or outputting the current optimum configuration during an analysis takes up more clock cycles. Therefore, if optimum configurations are encountered more frequently with a particular random number generator, the more time an algorithm will take to finish. If the optimum value is encountered sooner in an analysis, the rest of the combinations will not be outputted. This scenario most likely occurred with the linear varying beam running in C++ because of the data shown in Table 4. Table 4 shows at which configuration optimum values are reached in a sequence of 10000 configurations for both the C++ algorithm and the VIVA<sup>®</sup> algorithm. For the C++ version, the optimum value was reached at configuration 311 out of 10000, whereas VIVA<sup>®</sup> reached its optimum configuration very late in the analysis. Since the types random number generators used vary, the stress-weight optimization analyses cannot be used to test FPGA or standard processor performance based on run time. The displacement finder can be used to test performance because the arithmetic and integration are performed regardless of what random numbers are generated. The stress-weight optimization analyses do show that the optimization algorithms can be implemented easily in FPGAs with VIVA<sup>®</sup> and make use of parallelism. They also illustrate why future performance tests on the optimization algorithm must be done using a universal random number generator. Figure 9 shows the VIVA<sup>®</sup> implementation of the linear beam stress-weight optimization algorithm. The entire algorithm is not shown, but the stress analysis and weight analysis objects are shown running in parallel.

**Table 4.** Optimum Configuration Number out of a Sequence of 10000 for Constant Beam and Linear Beam Analyses

Programming environment	Constant Beam Optimum Configuration # out of 10000	Linear Beam Optimum Configuration # out of 10000
C++	6213	311
VIVA <sup>®</sup>	1482	9887

LinSWPara



**Figure 9.** Linear beam stress-weight optimization algorithm in VIVA<sup>®</sup>

In summary, Table 5 shows the times taken for constant cantilever beam displacement, constant beam stress-weight optimization, and linear beam stress-weight optimization algorithms performed on different machines.

**Table 5.** Times taken on different machines for three parts of overall Monte Carlo simulation optimization analysis that worked on FPGA

Machine	Time taken for constant cantilever beam optimization (seconds)	Time taken for Constant Stress-Weight Optimization (seconds)	Time taken for Linear Stress-Weight Optimization (seconds)
Intel Pentium III, 550 MHz	0.091	N/A	N/A
Intel Pentium 4, 1600 MHz	0.041	N/A	N/A
Intel Pentium 4, 2400 MHz	0.03	0.013	0.013
AMD Athlon, 1250 MHz	0.03	N/A	N/A
Starbridge Systems, HC-38 FPGA-based machine, 66MHz	Running with 2 MC Integrators: 0.08184	0.00570087	0.01265913

## Conclusion

The results obtained from this research indicate that Starbridge Systems's FPGA-based machines have promise in allowing users to easily implement Monte Carlo simulations and obtain results faster than with conventional computers. Starbridge Systems's HC-38 machine has shown to outperform conventional processors when running the constant cantilever beam displacement algorithm. The parallel capabilities of the FPGA allow runtime to be decreased further. The VIVA<sup>®</sup> programming environment allows this parallelism to be developed easily. Measuring the performance based on run time of a VIVA<sup>®</sup> implementation must involve a universal random number generator that is the same for VIVA<sup>®</sup>, C++, or another programming environment being tested against VIVA<sup>®</sup>, as the linear and constant stress-weight algorithms showed. Continuing improvements to VIVA<sup>®</sup> such as decreased synthesize time and decreased clock cycles for arithmetic objects will improve the speed at which Starbridge Systems's machines can perform computations. These improvements can lead to more complex problems requiring Monte Carlo simulation to be performed on Starbridge Systems's machines. Further tests and runs that need to be performed in the immediate future are as follows:

- With decreased synthesize time of VIVA 2.3<sup>®</sup>, running 3 and 8 MC integrators in parallel need to be performed.
- A universal random number generator that can be used in VIVA and in C++ needs to be produced.
- The times taken to run the quadratic and linear beam displacement finders on the FPGA need to be found.
- Once software problems preventing the implementation of the full constant, linear, and quadratic cantilever beam optimization algorithms are solved, these analyses can be run on the FPGA.
- Running the beam optimization on a cluster needs to be performed to test the Starbridge Systems FPGA-based computers against multi-processor computers in performing Monte Carlo simulation.

## Acknowledgements

I would like to thank my mentors, Dr. Robert Singleterry, Dr. Olaf Storaasli, and Dr. Jaroslaw Sobieszczanski-Sobieski for giving me their guidance, support, and the opportunity to perform this research. I would also like to thank William Fithian, Cris Kania, Shaun Foley, and Hoy Loper who worked with me this summer and who were always willing to help me and learn with me. Jonathan Ransom, Branch Head of the Analytical and Computational Methods Branch, was very supportive of my efforts during this summer and made sure I was provided with enough resources to accomplish my work. I would like to thank Starbridge Systems, Inc., for giving me the privilege and the opportunity to work for their company and helping to develop this amazing technology. The personnel of the Analytical and Computational Methods Branch at NASA Langley Research

Center provided me with a very supportive and friendly environment that enabled me to concentrate and focus on my work. Special thanks to Dr. Mark Jones at Virginia Tech who gave me guidance and support on how to tackle my research. David Rutishauser at NASA Langley also gave me input on how to go about my research. Brian Mason at NASA Langley Research Center helped teach me optimization analysis and made sure that the work I did was correct and thorough.