

NASA/CR-2002-000000



Development of an Integration Algorithm for Field Programmable Gate Arrays using VIVA

*Siddhartha Krishnamurthy
Star Bridge Systems
Langley Research Center, Hampton, Virginia*

*Olaf O. Storaasli and Robert C. Singleterry
Langley Research Center, Hampton, Virginia*

Work Performed at Analytical and Computational Methods Branch

National Aeronautics and
Space Administration

Langley Research Center
Center
Hampton, Virginia 23681-2199

Prepared for Langley Research
under Contract LAC16713

September 2002

Development of an Integration Algorithm for Field Programmable Gate Arrays using VIVA

Olaf O. Storaasli
Robert C. Singleterry
Siddhartha Krishnamurthy

Abstract

In engineering, there is a need to develop algorithms that can perform computations simultaneously. Developing the ability to solve differential equations simultaneously will increase the speed at which engineering problems can be solved. Field-Programmable Gate Arrays (FPGAs) are new computational tools that can run operations in parallel. In traditional CPUs, gates are hardwired to the processor. Many jobs must execute sequentially and gates that are not being used waste power and generate heat. FPGAs interconnect only the gates needed to perform an operation at hand. They reemploy gates when another operation needs to be run. Many circuits can be built, destroyed, and rebuilt to perform jobs in parallel on a single FPGA. The graphical programming language, VIVA, has been developed and tailored for programming FPGAs. An integration algorithm needs to be developed in VIVA so solutions to differential equations can be found using FPGAs.

The numerical method of finding the sum of areas of rectangles under a function will be used to integrate the function. This method requires iterations to be used. Assuming the iterative process is not too large, the parallelism capability of FPGAs allows the processes to complete instantly. Two methods of integrating with rectangles can be used. The first method is of the form $\sum \Delta X * f(i+1) \Delta X$ where i goes from 0 to $n-1$ where n is the number of rectangles. The second method is of the form $\sum \Delta X * f(((i+1)-0.5) * \Delta X)$ where i goes from 0 to $n-1$ where n is the number of rectangles.

Two integration algorithms were brainstormed. One algorithm was implemented. It shows that an increment value of 0.01 produces the least error for most functions. For linear functions, it was found that error increases with more iterations while for other functions it decreases. For integrating functions that involve square root, the algorithm implemented does not produce accurate results. A C++ version of the algorithm was written and produces more accurate results for integrating square root functions. Due to this problem, it is hypothesized that a bug in either the VIVA algorithm or in VIVA itself needs to be found and repaired. For most cases, the algorithm has shown that the second method of integration with rectangles is more

National Aeronautics and
Space Administration

Langley Research Center
Center
Hampton, Virginia 23681-2199

Prepared for Langley Research
under Contract LAC16713

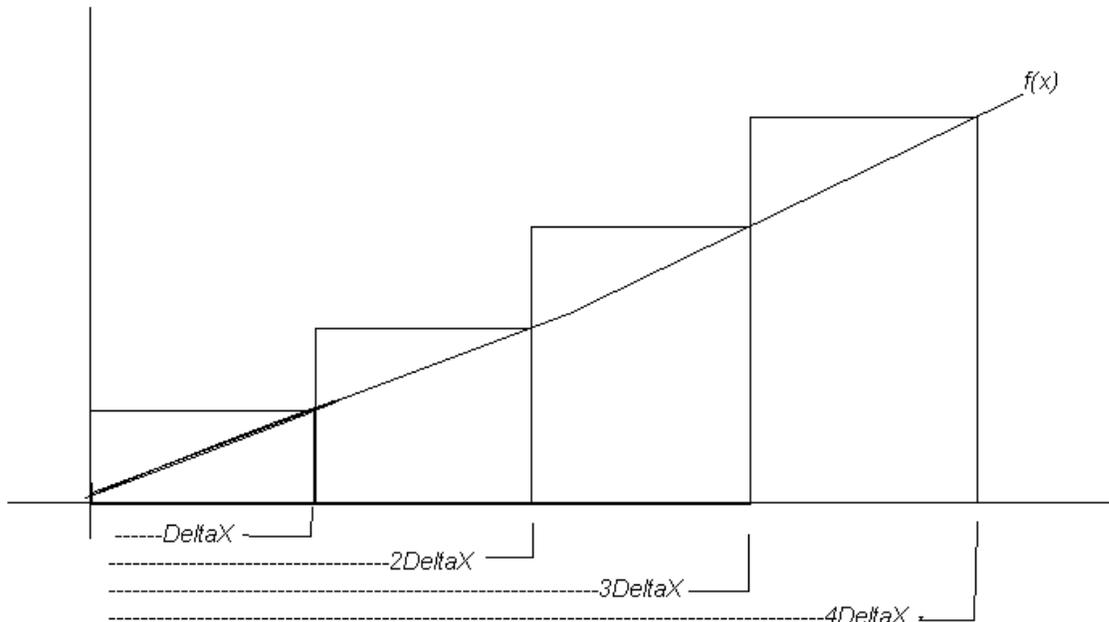
The definite integral of a function, $f(x)$, can be found by finding the sum of the areas of the rectangles underneath it from a lower limit X_0 to an upper limit X . The area of each rectangle is given by $\Delta x * f(X_i)$ where Δx is small. This numerical method will be used to integrate functions on VIVA. Other ways of integrating such as using Simpson's Rule and the Trapezoidal Rule require a higher degree of complexity for VIVA to perform at this time. They can be implemented later or as further improvements to VIVA arrive.

Two Methods of Integrating with Rectangles

Two methods of integrating using rectangles can be implemented in VIVA. They are shown under 1stMethod and 2ndMethod.

1st Method:

The graph below demonstrates how a function can be integrated using rectangles.



Graph1

Thus, the area can be written as:

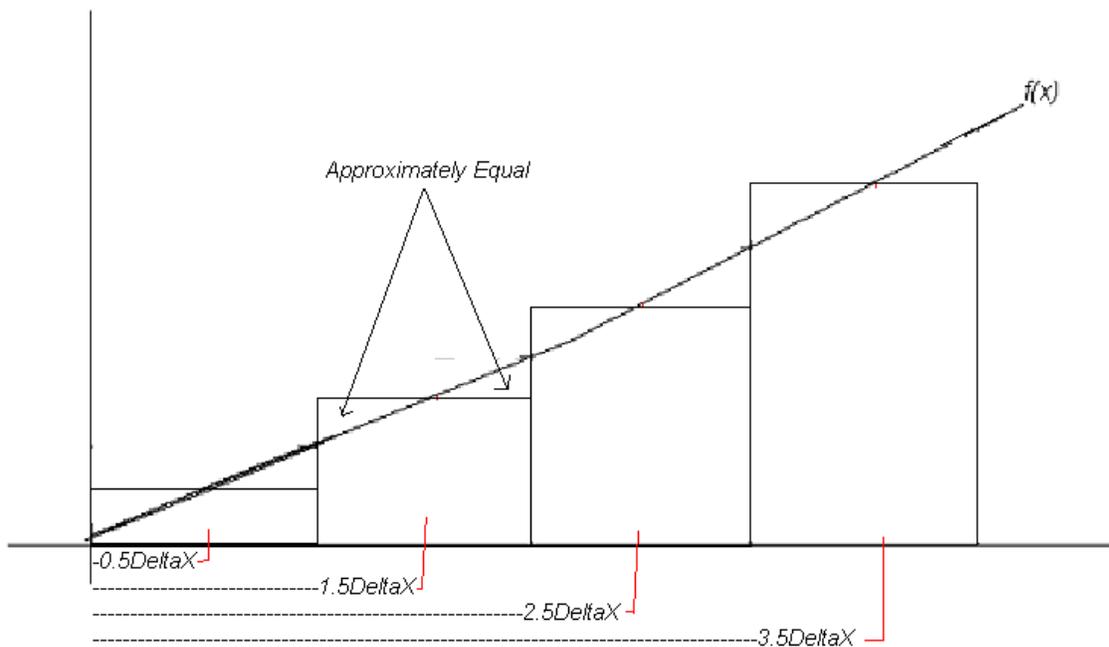
$$\text{Area} = \Delta X * f(\Delta X) + \Delta X * f(2\Delta X) + \Delta X * f(3\Delta X) + \dots + \Delta X * f((i+1)\Delta X),$$

where i goes from 0 to $n-1$ where n is the number of rectangles.

The problem with this method is that the portions of the rectangles above $f(x)$ cause error. ΔX needs to be small to give more accurate results, but the smaller it is the more iterations VIVA needs to perform.

2nd Method:

The graph below demonstrates a different approach to integrate functions using rectangles.



Graph2

Thus, the area can be written as:

$$\text{Area} = \Delta X * f(0.5\Delta X) + \Delta X * f(1.5\Delta X) + \Delta X * f(2.5\Delta X) + \dots + \Delta X * f((i+1)-0.5)\Delta X,$$

where i goes from 0 to $n-1$ where n is the number of rectangles.

Using Method 2 reduces the error caused by the area of the rectangles above $f(x)$. Since the height of each rectangle is a function of half its base, portions of the area under $f(x)$ do not get calculated. Their error is nullified by the area of the rectangles above $f(x)$. As seen in graph2, the area under $f(x)$ that does not get calculated is nearly equal to the area of the rectangles above $f(x)$. This method will be able to use larger values of ΔX .

Developing the Integration Algorithm

An integration algorithm has already been developed that can integrate X^2 using the 1st method of adding rectangles. The integration algorithm needs to be expanded to integrate any function and use methods 1 and 2 for adding rectangles.

The integration algorithm needs to answer the following questions:

1. What value of ΔX gives a reasonable result?
2. How much does error increase or decrease as the number of iterations increase?
3. Which integration method, 1 or 2, is the best?

Establishing Constraints

It is hypothesized that integration method 2 will be more efficient. However, both methods need to be run to show just how efficient they are with respect to each other. An integration program must be developed that can use both methods 1 and 2. It is also necessary to allow any function to be integrated. Thus, a function box that can hold any function needs to be developed. The user should be able to change ΔX when the program has either slowed down or temporarily stopped. Starting the integration at an initial value other than 0 is also desired.

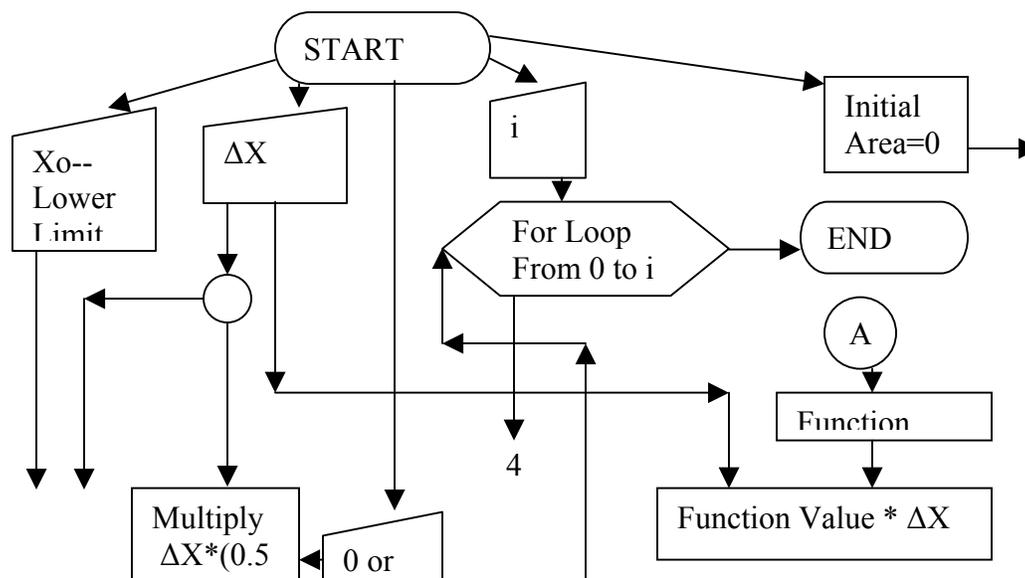
The constraints for the best integration algorithm are summarized as follows:

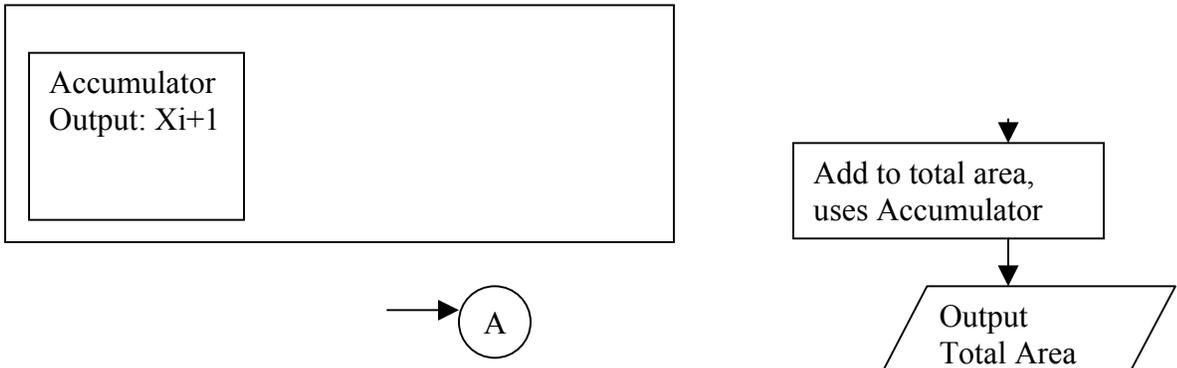
1. Need to use Methods 1 and 2.
2. Allow any function to be integrated (provide a function box).
3. Allow ΔX to be changed when the program has either slowed down or temporarily stopped.
4. Allows a lower limit and upper limit for definite integral.

The Two Solutions

Two algorithms were brainstormed. Their pseudo codes are shown in the following flowcharts. Each algorithm will be analyzed to see which one can best meet the constraints.

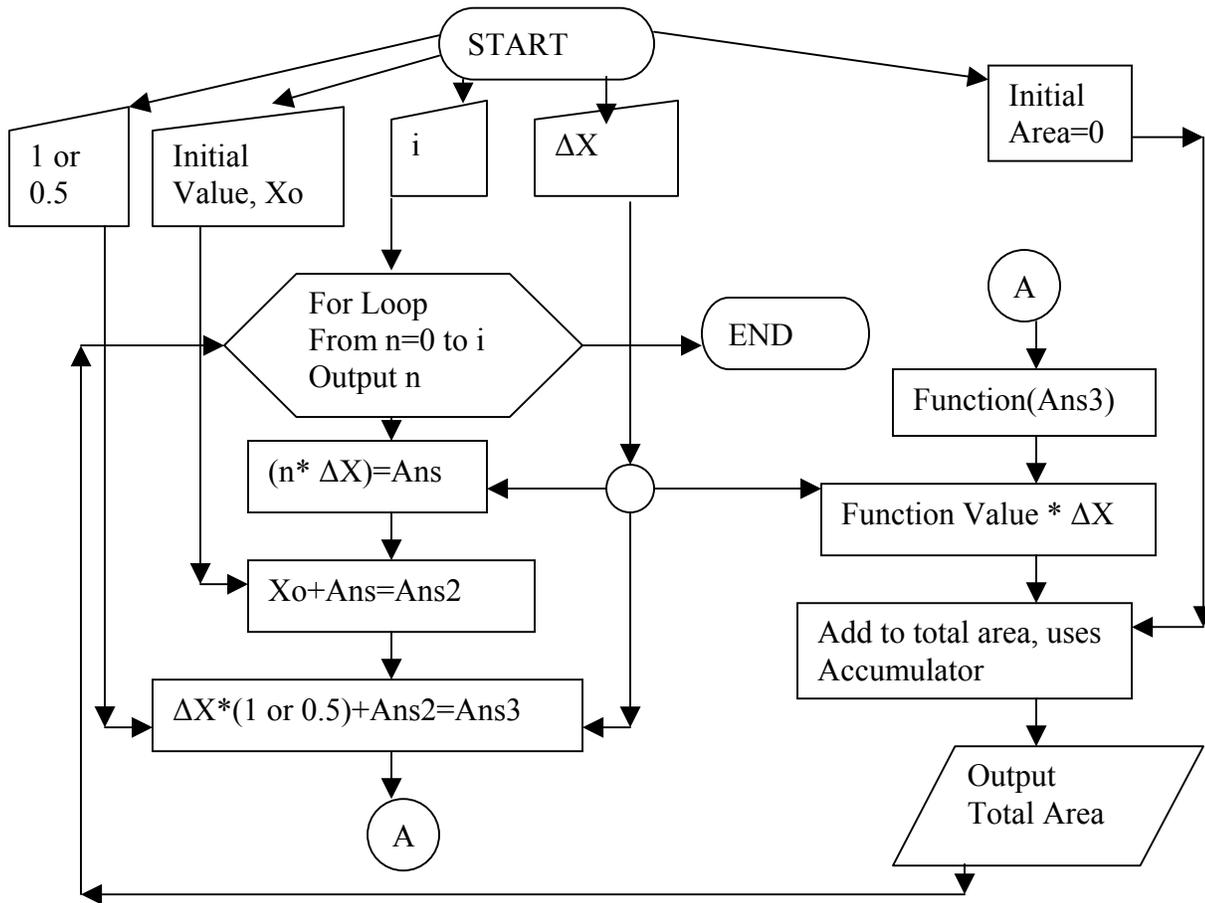
Idea1:





The Accumulator object is used to add ΔX to X_i , outputting X_{i+1} . When the process is exited, a feedback mechanism causes X_{i+1} to equal X_i and be used in the next iteration. The same mechanism is used to keep track of the total area.

Idea2:



Analysis of each Solution

Each algorithm is evaluated to see how well it meets the constraints.

Each idea is given an integer score in the range of 1-4 to rate how it satisfies each constraint.

The idea with the most points will be implemented.

Rating Scale:

1—Poor

2—Fair

3—Good

4—Excellent

Idea1:

1. Allows either method 1 or method 2 to be used. In this idea, the multiplier object runs at the same time as Accumulator. Running these objects parallel to each other utilizes the parallelism capability of FPGAs, thus decreasing running time. **Score: 4**
2. The function object allows any function to be integrated. The function object calls on a function developed in the function object's sheet. A score of 4 cannot be given because the program needs to be recompiled for each new function. **Score: 3**
3. Allows ΔX to be changed. However, slowing down or temporarily stopping the program can only change ΔX . **Score: 3**
4. The input value X_0 allows a lower limit to be specified. **Score: 4**

Total Score: 14/16

Idea2:

1. Allows either integration method 1 and 2 to be used. Step-by-step approach. Does not use the parallelism found in Idea1. ΔX is to be multiplied by 1 instead of 0 to use method 1. **Score: 3**
2. The function object allows any function to be integrated. The function object calls on a function developed in the function object's sheet. A score of 4 cannot be given because the program needs to be recompiled for each new function. **Score: 3**
3. Allows ΔX to be changed. However, slowing down or temporarily stopping the program can only change ΔX . **Score: 3**
4. This algorithm allows a lower limit to be specified. **Score: 4**

Total Score: 13/16

Decision

Since Idea1 received the highest score, it will be implemented.

Improvements to VIVA will continue to come. Unforeseen problems in implementing Idea1 can also arise.

Idea2 will be used if it is later felt that it is the better option.

Results

Idea1 was implemented.

The following modifications were made:

1. A bug was discovered when running the Accumulator object and multiplier object in parallel. The program was changed so that the Accumulator and the multiplier objects ran sequentially. Despite this bug, Idea1 remained the algorithm of choice because improvements to VIVA or the fixing of the bug will allow the Accumulator and multiplier objects to run parallel.
2. Idea1 was made into an object called Integral object. The function box inside the Integral object was taken out. Instead, the user defines the function outside of Integral. Integral calls on the function by passing to it a value, x , obtained after the 1st subtraction object. The value $f(x)$ is passed back to Integral from the function so that it can be multiplied by ΔX .

The integration results for functions $f(x)=x$, $f(x)=x^2$, $f(x)=x^5$, and $f(x)=x^8$, $f(x)=1/x$, $f(x)=x^4$, $f(x)=x^3$, and $f(x)=1/x + x^2 + 5$ are shown in the following tables.

Using a $|\Delta X|$ value that was a power of $10 < 1$ like 0.1 and 0.01 was preferred for the following functions. With $|\Delta X|$ as a power of $10 < 1$, any lower and upper limits could be specified provided that the number of digits to the right of the decimal in the upper and lower limits was not greater than the number in ΔX .

Function: $f(x)=x$ Integrating from $x=0$ to $x=2$

Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 1	0.1	0-19	2.099998	2	4.9999
Method 2	0.1	0-19	1.999998	2	0.0001
Method 1	0.01	0-199	2.009992	2	0.4996
Method 2	0.01	0-199	1.999992	2	0.0004

Function: $f(x)=x^2$ Integrating from $x=0$ to $x=2$

Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 1	0.1	0-19	2.869995	2.666667	7.624799047
Method 2	0.1	0-19	2.664995	2.666667	0.062699992
Method 1	0.01	0-199	2.686688	2.666667	0.750787406
Method 2	0.01	0-199	2.666637	2.666667	0.001125

Function: $f(x)=x^5$ Integrating from $x=0$ to $x=2$

Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
--------------------	--------	---	----------------	-------------------	---------

Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 1	0.1	0-19	12.33325	10.66667	15.62418262
Method 2	0.1	0-19	10.63332	10.66667	0.312656152
Method 1	0.01	0-199	10.82725	10.66667	1.50543703
Method 2	0.01	0-199	10.66625	10.66667	0.003937499

Function: $f(x)=x^8$ Integrating from $x=0$ to $x=3$ for $\Delta x=0.1$ and from $x=0$ to $x=2$ for $\Delta x=0.01$

Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 1	0.1	0-29	2529.595	2187	15.6650663
Method 2	0.1	0-29	2179.701	2187	0.333744856
Method 1	0.01	0-199	58.1768	56.889	2.263706516
Method 2	0.01	0-199	56.884	56.889	0.008789045

Function: $f(x)=1/x$ Integrating from $x=1$ to $x=2$ to find $\ln(2)$

Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 1	0.1	0-9	0.6687716	0.693147181	3.516652919
Method 2	0.1	0-9	0.6928357	0.693147181	0.044937146
Method 1	0.01	0-99	0.6906521	0.693147181	0.359964035
Method 2	0.01	0-99	0.69314429	0.693147181	0.00041702

Function: $f(x)=x^4$ Integrating from $x=1$ to $x=-2$

Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 1	-0.1	0-29	-7.379959	-6.6	11.81756061
Method 2	-0.1	0-29	-6.584982	-6.6	0.227545455
Method 1	-0.01	0-299	-6.675242	-6.6	1.140030303
Method 2	-0.01	0-299	-6.599793	-6.6	0.003136364

Function: $f(x)=x^3$ Integrating from $x=-3$ to $x=2$

Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 1	0.1	0-49	-14.51252	-16.25	10.69218462
Method 2	0.1	0-49	-16.24377	-16.25	0.038338462
Method 1	0.01	0-499	-16.07513	-16.25	1.076123077
Method 2	0.01	0-499	-16.24994	-16.25	0.000369231

Function: $f(x)=1/x + X^2 + 5$ Integrating from $x=1$ to $x=3$

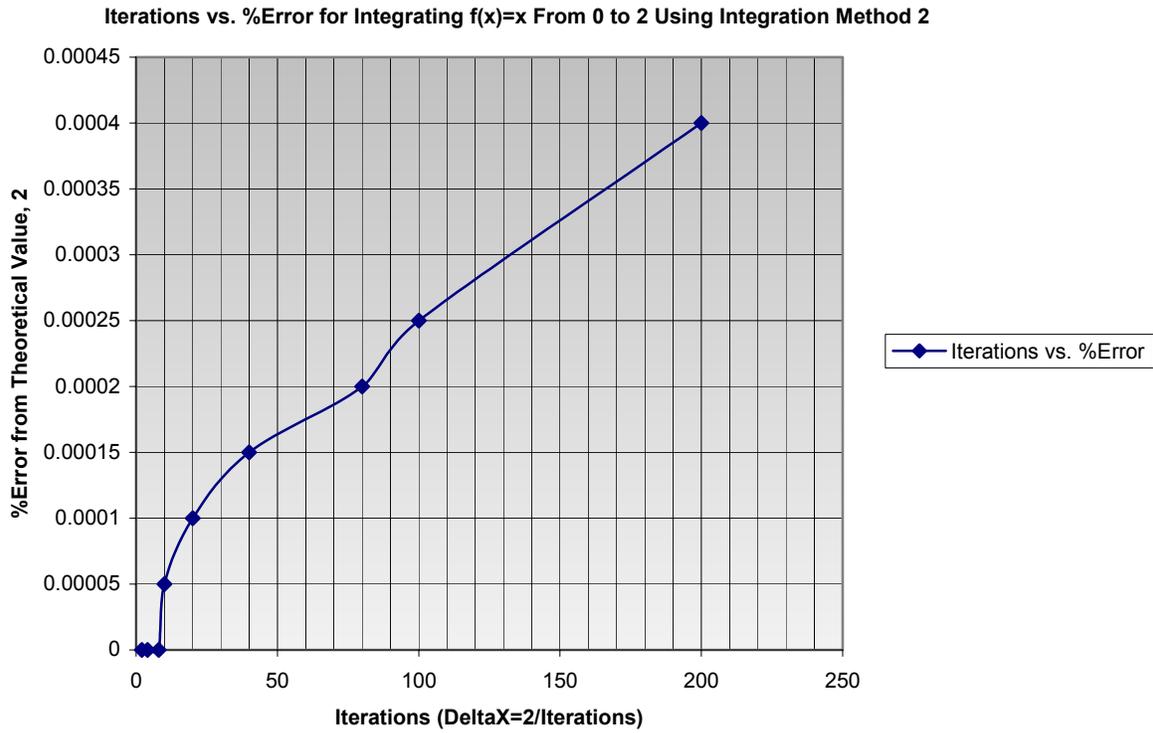
Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 1	0.1	0-29	16.33332	16.33333	0.000006122
Method 2	0.1	0-29	16.33332	16.33333	0.000006122
Method 1	0.01	0-299	16.33332	16.33333	0.000006122
Method 2	0.01	0-299	16.33332	16.33333	0.000006122

Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 1	0.1	0-19	20.13599	19.76527896	1.875566975
Method 2	0.1	0-19	19.76322	19.76527896	0.010417031
Method 1	0.01	0-199	19.8019	19.76527896	0.185279675
Method 2	0.01	0-199	19.76516	19.76527896	0.00060184

For most of the functions, a ΔX value of 0.01 can produce a result with a small percent error.

For the function $f(x)=x$, the error increases as the number of iterations increases. The following table and Graph 3 show this behavior.

Function: $f(x)=x$ Integrating from $x=0$ to $x=2$					
Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 2	1	0-1	2	2	0
Method 2	0.5	0-3	2	2	0
Method 2	0.25	0-7	2	2	0
Method 2	0.2	0-9	1.999999	2	5E-05
Method 2	0.1	0-19	1.999998	2	0.0001
Method 2	0.05	0-39	1.999997	2	0.00015
Method 2	0.025	0-79	1.999996	2	0.0002
Method 2	0.02	0-99	1.999995	2	0.00025
Method 2	0.01	0-199	1.999992	2	0.0004

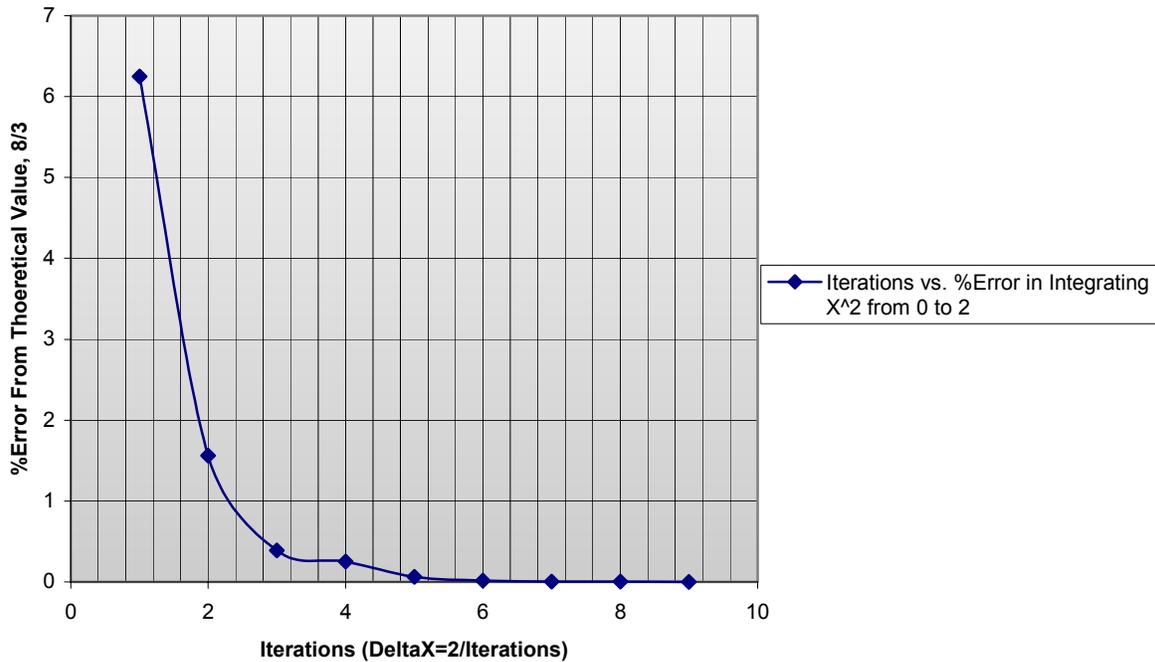


Graph 3

For $f(x)=x^2$, the percent error decreases as the number of iterations increases. The following table and Graph 4 show this behavior.

Function: $f(x)=x^2$ Integrating from $x=0$ to $x=2$					
Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 2	1	0-1	2.5	2.66666667	6.25
Method 2	0.5	0-3	2.625	2.66666667	1.5625
Method 2	0.25	0-7	2.65625	2.66666667	0.390625
Method 2	0.2	0-9	2.659997	2.66666667	0.2501125
Method 2	0.1	0-19	2.664995	2.66666667	0.0626875
Method 2	0.05	0-39	2.666244	2.66666667	0.01585
Method 2	0.025	0-79	2.666554	2.66666667	0.004225
Method 2	0.02	0-99	2.666591	2.66666667	0.0028375
Method 2	0.01	0-199	2.666637	2.66666667	0.0011125

Iterations vs. %Error in Integrating X² from 0 to 2 Using Integration Method 2



Graph 4

It is concluded that for functions that are more linear in nature, using a large value of ΔX can yield a smaller percent error.

Functions that involve a square root do not integrate well with the current VIVA algorithm. Using smaller values of ΔX reduces the error, but the error is still large. The following two tables show the results for integrating $f(x)=\sqrt{x}$.

Function: $f(x)=\sqrt{x}$ Integrating from $x=0$ to $x=3$

Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 1	0.1	0-29	3.677477	3.464101615	6.159616794
Method 2	0.1	0-29	3.763621	3.464101615	8.646379874
Method 1	0.01	0-299	3.672074	3.464101615	6.003645619
Method 2	0.01	0-299	3.649745	3.464101615	5.359062911

Function: $f(x)=\sqrt{x}$ Integrating from $x=1$ to $x=3$

Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 1	0.1	0-19	3.03544	2.797434948	8.507974481

Method 2	0.1	0-19	3.090086	2.797434948	10.46140686
Method 1	0.01	0-199	3.014357	2.797434948	7.75431978
Method 2	0.01	0-199	2.999727	2.797434948	7.231340684

At first, it may seem that the method of integration needs to be changed. However, a C++ version of the algorithm was written. It produced the following results for integrating square root functions:

Function: $f(x)=\sqrt{x}$. Integrating from $x=0$ to $x=3$ in C++

Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 1	0.1	0-29	3.5443709	3.464101615	2.317174661
Method 2	0.1	0-29	3.4659069	3.464101615	0.052114085
Method 1	0.01	0-299	3.4725544	3.464101615	0.244010881
Method 2	0.01	0-299	3.4641602	3.464101615	0.001691199

Function: $f(x)=\sqrt{x}$. Integrating from $x=1$ to $x=3$ in C++

Integration Method	DeltaX	i	Computer Value	Theoretical Value	% Error
Method 1	0.1	0-19	2.8338614	2.797434948	1.30213759
Method 2	0.1	0-19	2.797523	2.797434948	0.003147598
Method 1	0.01	0-199	2.8010919	2.797434948	0.130725185
Method 2	0.01	0-199	2.7974355	2.797434948	1.97324E-05

The C++ program gives more accurate results for the same inputs as the VIVA program. Thus, it is concluded that the problem with integrating square root functions is not due to the integration method. It is hypothesized that there is a bug within the algorithm or within VIVA itself. This bug needs to be found and corrected.

Integral object can be stopped or slowed down in a calculation to allow the increment value to be changed. This ability is useful when integrating functions that have intervals where the slope is small. This ability was used to integrate the $f(x)=1/X^2$.

Function: $f(x)=1/X^2$ Integrating from $x=0.5$ to $x=2$ by $\Delta x=0.01$,
from $x=2$ to $x=4$ by $\Delta x=1$

Integration Method	DeltaX	Computer Value	Theoretical Value	% Error
Method 1	0.01 for $i=0$ to 149, 1 for $i=149$ to 151	1.654988	1.75	5.42925714
Method 2	0.01 for $i=0$ to 149, 1 for $i=149$ to 151	1.741563	1.75	0.48211429

Conclusion

Ideal was chosen as the algorithm to be implemented to perform integration. Ideal allows integration methods 1 and 2 to be implemented, allows any function to be integrated, allows ΔX to be changed, and allows a lower limit to be specified.

The integration algorithm was run. Ideal was made into an object called Integral object. For linear functions, using large values of ΔX and less iterations produce smaller errors. For other functions, using more iterations increases their accuracy. A ΔX value of 0.01 produces a reasonable result with a small percent error. The functions used have shown that Method 2 is more accurate than Method 1. However, for functions that involve square root, the Integral object does not produce accurate results. It is hypothesized that there is a bug within the VIVA algorithm or within VIVA itself that needs to be found and corrected.

It is now possible to perform integration on any function using FPGAs. The integration algorithm needs some fine-tuning. However, the groundwork has been done to make the development of more complex algorithms possible. The algorithm that has been developed can be implemented to solve differential equations. It is hoped that the new capability given by the integration algorithm can significantly speed up engineering problem solving.

Acknowledgments

I would like to thank my mentors Dr. Robert C. Singleterry and Dr. Olaf Storaasli of the Analytical and Computational Methods Branch (ACMB) at NASA Langley Research Center for giving me the task of developing the integration algorithm and allowing me to work with Reconfigurable Computing. They taught and gave me the tools to learn the VIVA language. I thank William Fithian of Star Bridge for his help and support on VIVA. I thank Dr. Jaroslaw Sobieszczanski-Sobieski of ACMB for his support. The personnel of ACMB have all been helpful and supportive towards my efforts. NASA Langley's Volunteer Service Program has made my work with FPGAs possible. I thank the program for giving me the privilege of working at NASA. Finally, I thank the Bradley Department of Computer and Electrical Engineering at Virginia Tech for promoting my work at NASA Langley Research Center.

Works Cited

Singleterry Jr., Robert C, et al. Field-Programmable Gate Array Computer
In Structural Analysis: An Initial Exploration. Paper Number: 2002-1761.
Presented at 43rd AIAA Structures, Structural Dynamics and Materials
April 22-25, 2002, Denver, CO.

Star Bridge Systems, Inc. Hypercomputer® Technology and Products. Star Bridge
Systems, Inc.