**Paper Number: AIAA-2002-1761**

# FIELD-PROGRAMMABLE GATE ARRAY COMPUTER IN STRUCTURAL ANALYSIS: AN INITIAL EXPLORATION

Robert C. Singleterry Jr., NASA Langley Research Center, Hampton, Virginia[1]
Jaroslaw Sobieszczanski-Sobieski, AIAA F, NASA Langley Research Center, Hampton, Virginia
Samuel Brown, Star Bridge Systems Inc., Midvale Utah

## Abstract

This paper reports on an initial assessment of using a Field-Programmable Gate Array (FPGA) computational device as a new tool for solving structural mechanics problems. A FPGA is an assemblage of binary gates arranged in logical blocks that are interconnected via software in a manner dependent on the algorithm being implemented and can be reprogrammed thousands of times per second. In effect, this creates a computer specialized for the problem that automatically exploits all the potential for parallel computing intrinsic in an algorithm. This inherent parallelism is the most important feature of the FPGA computational environment. It is therefore important that if a problem offers a choice of different solution algorithms, an algorithm of a higher degree of inherent parallelism should be selected.

It is found that in structural analysis, an "analog computer" style of programming, which solves problems by direct simulation of the terms in the governing differential equations, yields a more favorable solution algorithm than current solution methods. This style of programming is facilitated by a "drag-and-drop" graphic programming language that is supplied with the particular type of FPGA computer reported in this paper. Simple examples in structural dynamics and statics illustrate the solution approach used. The FPGA system also allows linear scalability in computing capability. As the problem grows, the number of FPGA chips can be increased with no loss of computing efficiency due to data flow or algorithmic latency that occurs when a single problem is distributed among many conventional processors that operate in parallel.

This initial assessment finds the FPGA hardware and software to be in their infancy in regard to the user conveniences; however, they have enormous potential for shrinking the elapsed time of structural analysis solutions if programmed with algorithms that exhibit inherent parallelism and linear scalability. This potential warrants further development of FPGA-tailored algorithms for structural analysis.

## 1. Introduction

As in other areas of applied mechanics, structural analysis poses a persistent demand for faster computing. This is particularly true in applications involving nonlinear behavior and in design optimization because of the repetitive analysis. Until recently, the computer hardware technology has responded to this demand via a two-prong development of faster processors and concurrent processing with an increasing number of processors packaged either in a single machine or distributed in a network.

The Field Programmable Gate Array (FPGA) is a relatively new addition to the above technology of Massively Concurrent and Distributed Processing. Relatively few engineers and scientists have had an opportunity to explore the FPGA computers. This paper reports on an initial assessment of the utility of the FPGA machine and its graphic programming language illustrated by a few simple examples in structures and dynamics. A variety of commercially available FPGA systems now exist. They are supplied either as separate boards or as complete, turnkey systems. The information reported herein was obtained at the NASA Langley Research Center by using one such turnkey system containing 20 FPGA's called HAL-30 Hypercomputer™ (Reference [3]), programmed with a graphic language VIVA® (Reference [2]), both from Star Bridge Systems®. The assessment includes a simplified description of the salient features that distinguish the FPGA computer from single or multiple processor computers. The current merits and demerits of the FPGA computer and its future potential are discussed.

## 2. Conventional CPU vs FPGA

The kernel of most current computer architectures is the general purpose Central Processing Unit (CPU), programmed individually or in networked groups. The CPU itself consists of a large number of gates that work as on-off switches and are permanently wired in a variety of fixed circuits that implement, in a binary-logic framework, the whole gamut of functions needed

[1] Corresponding author: Robert C. Singleterry, Jr.; NASA Langley Research Center, MS 188B, Hampton VA 23681; 757-864-1437; r.c.singleterry@larc.nasa.gov

for its operation. At any particular time, only a small fraction of the total number of gates may be gainfully employed, while the idle gates consume power and generate heat. For engineers, the ability to generate floating-point results is essential; therefore, some CPUs have multiple floating-point units. In a massively parallel environment that employs many CPUs, the number of floating-point results that can be generated simultaneously is limited by the number of the CPUs, the communication speed, and the memory capability, all combined with the solution algorithm characteristics.

In an FPGA, a command to execute a particular function is interpreted by the resident operating system to interconnect "by software" as many gates as necessary to perform that particular function and only that function. The particular interconnection scheme exists only as long as the job executes and can be reconfigured to execute other functions many times per second. In effect, a "specialized CPU" is created for the job at hand. It is reminiscent of using an UV light source to burn-in the inter-gate conduits to implement a particular logic circuit, an operation known as programming by "firmware". However, the result of this firmware programming is irreversible, while the FPGA program can be changed by the user in the field -- hence "Field" in FPGA.

If the algorithm being implemented offers an opportunity to do M number of operations simultaneously, then M specialized CPUs are being created in the FPGA computer. In the extreme, the specialized CPUs might all be different in terms of the number of gates and their interconnection logic if the functions to be executed are different. Some of these specialized CPUs could operate simultaneously and some could operate sequentially as required by the logic of the algorithm being executed. Thus, the set of gates available in an FPGA computer is viewed as a resource for creating a computer tailored to the job at hand. In principle, if the job is not large enough to use all the gates available, another unrelated job might be executed simultaneously. Also, the hardware executes the entire algorithm at once unless a sequencing step is part of the algorithm, e.g., when the algorithm involves iterations.

To illustrate the computational parallelism afforded by a FPGA, consider the example of computing $\mathbf{a} = \mathbf{b} \cdot \mathbf{c} + \mathbf{d}$. A conventional CPU needs two steps for this operation: first to calculate the product $\mathbf{b} \cdot \mathbf{c}$ and then to perform the sum because the steps are logically sequential. The FPGA potentially can complete both operations in one step provided the gates are properly configured. Furthermore, if the example is extended to $\mathbf{a} = \mathbf{e} \cdot \mathbf{f} + \mathbf{b} \cdot \mathbf{c} + \mathbf{d}$, a FPGA still needs only one step because it can engages more gates. This example can be extended in this manner while holding the execution to a single step until the number of gates employed equals the number of gates supplied in the array. This "inherently parallel" attribute allows all M operations to be executed at the same time, so that the maximum M is only a function of the size and number of FPGAs available.

## 3. Graphic Programming of FPGA

To make the multitude of gates available to the user without performing machine-language level programming for each application, high-level languages exist, such as, HANDEL-C™ (Reference [1]), and VIVA® (Reference [2]), the latter having been used as a basis for this report.

The authors' experiences to date indicate that the FPGA-based computers can benefit engineers in at least two ways:

1. They exploit inherent parallelism in the algorithm being executed, hence they have a potential to radically reduce the computation elapsed time.
2. They may be programmed in an analog computer style that bypasses the need to develop solution algorithms to the equations that govern the problem.

These two points are illustrated by an introduction to the VIVA language using simple examples.

The VIVA language hides the tailoring of a specialized CPU from the amorphous supply of gates specifically for the job at hand under a veneer of a graphic language that enables coding by drawing what looks like a wiring diagram. In this regard, VIVA extends to the FPGA machines the same approach that languages such as MATLAB Simulink (Reference [4]) implemented on a conventional, single or multiple processor computers.

The essence of the VIVA graphic program is illustrated in Figure 1 by the simple example of computing $\boldsymbol{a}^2$ for a given value of $\boldsymbol{a}$.

VIVA has a library of processing tools available in a display on right the margin of the screen. The tools needed to execute the example problem are

1. Input $\boldsymbol{a}$ from the user, illustrated by a horn on the left
2. Multiply ($\boldsymbol{a} * \boldsymbol{a}$), illustrated by the box with ✕
3. Output $\boldsymbol{a}^2$ to the user, illustrated by the horn on the right

These objects are dragged and dropped onto the main screen area and the inputs and outputs are connected to the multiply object. The resulting diagram is "*compiled*" into an executable code that can accept any value of $\boldsymbol{a}$ as input. The entire diagram may be named, saved, turned into another icon in the library of tools, and displayed. In this manner, one can develop a complex code by building blocks forming many hierarchically related levels. These building blocks work best if the algorithm can be made recursive in nature.

## 3.1 Equation Solution

Now consider two algebraic equations with two unknowns *y* and *z* and six constants *a*, *b*, *d*, *e*, *P*, and *M*

$$\begin{cases} \mathbf{ay + bz = P} \\ \mathbf{dy + ez = M} \end{cases} \qquad (1)$$

One may program a VIVA solution using Cramer's rule:

$$\begin{cases} \mathbf{y = \dfrac{Pe - Mb}{ae - db}} \\ \mathbf{z = \dfrac{Ma - Pd}{ae - db}} \end{cases} \qquad (2)$$

and as in the previous example execute it in one step. However, expansion of the system of simultaneous equations to larger number of variables causes nonlinear explosion in the number of terms and operations that would quickly exhaust the FPGA capacity and make the VIVA program exceedingly complex. Therefore, a better method in VIVA is a fixed-point iteration solution, as in Figure 2, corresponding to

$$\begin{cases} \mathbf{y_i = \dfrac{P - bz_{i-1}}{a}} \\ \mathbf{z_i = \dfrac{M - dy_{i-1}}{e}} \end{cases} \qquad (3)$$

Upon initialization, the system represented by the diagram in Figure 2 converges to the fixed-point $\mathbf{(y_\infty, z_\infty)}$ solution as the iterations are stepped through. The need to iterate (with a rate of convergence that depends on the properties of the matrix of coefficients) is the penalty for the solution simplicity. The flow of execution within one cycle is: all the "divide" operators first, all the "multiply" operators next, followed by all the "subtraction" operators.

## 3.2 Analog programming style

The diagram in Figure 2 is strikingly similar to a wiring diagram one would have developed for this problem when using an analog computer. The similarity is more than superficial. It implies a style of programming based entirely on the now nearly forgotten analog computer methods. In fact, for a case of a spring-mass vibrating system with damping shown in Figure 3, one may revert to an analog computer diagram from a more than 30-year-old reference [5, Figure 2-9] rendered in Figure 4 and derive from it the VIVA diagram shown in Figure 5. The acceleration, velocity, and translation as functions of time computed by VIVA for an impulse force excitation are presented in Figure 6 for a subcritical value of the damping constant. Should the damping be set to a negative value, a divergent behavior reminiscent of wing flutter results.

## 3.3 Inclusion of non-linearity

A primary benefit of the analog computer programming style is that the handling of non-linearity becomes almost trivial. For example, should the right hand side of the first equation in Eq. (1) depend nonlinearly on the solution for *y*, Eq. (1) becomes

$$\begin{cases} \mathbf{ay + bz = P - ky^2} \\ \mathbf{dy + ez = M} \end{cases} \qquad (4)$$

and the diagram in Figure 2 changes by the simple addition of the "Two DOF NL term" object shown in Figure 7. The numerical solution is obtained by recording the output values of *y* and *z* as the system converges to a steady state. In effect the usual approach of devising a solution to the equations and then programming the solution is replaced by programming an iterative solution by directly simulating the terms of the governing equations and their connectivity. Table 1 shows the results of the above iterations along with results from a double precision FORTRAN program, which agrees to six significant figures.

## 3.4 Concurrent computing

To see the concurrent computing capability in the above application, consider a linear equation with only one unknown whose iterative solution may be written as

$$\mathbf{x_i = \frac{(x_{i-1})^2 - 1}{3}} \qquad (5)$$

and the corresponding VIVA diagram in Figure 8. The reduced diagram amounts to one half of the diagram for the system of two equations as in Eq. (1) shown in Figure 2 (top left inset) and, yet, the execution time per iteration for the full system would be the same as for the reduced system because of the concurrent processing.

## 4. Initial Assessment of Experience with Statics and Dynamics

The next step is the application of a larger number of degrees of freedom. Figure 9 depicts a system of two vibrating masses with three springs, three dampers, and two governing, coupled differential equations. The corresponding analog computer wiring diagram is illustrated in Figure 10. A VIVA program based on that diagram is shown in Figure 11 with the program's output shown in Figure 12. Again, if there were a non-linearity in the system, for example, the $k_2$ spring stiffness were proportional to the spring elongation ***x2-x1***, a simple modification shown by the inset in Figure 10 would implement that non-linearity.

A cantilever beam in Figure 13 provides an example of a static analysis case with multiple degrees of freedom. This simple case suggests a conclusion of general significance. The conclusion is that to use FPGA's efficiently, one needs to select among the solution algorithms available, one that affords the largest degree of inherent parallelism. For the case at hand, three possible algorithms to choose from are:

1. Forming the Load-Deflection equations $\mathbf{[K]\{u\} = \{P\}}$ and solving them by one of the direct solution methods, e.g., the Cholesky decomposition algorithm.
2. Solving the Load-Deflection equations by an iterative method as in Figure 2.
3. Treating the beam as a quasi-dynamic system that asymptotically tends to a solution corresponding to the static state.

The above alternative algorithms have the following merits and demerits.

## 4.1 Direct solution methods

The direct solution approach is the least advantageous in FPGA programming because it is a procedure, which consists mostly of the array bookkeeping that exploits the $\mathbf{[K]}$ matrix sparseness. Considering that in a large problem, only a few percent of the $\mathbf{[K]}$ matrix entries may be nonzero, the exploitation of sparseness is a necessity in problems of any significance because processing of all the elements including the null ones requires the number of arithmetical operations that is of order $N^3$. On the other hand, that number reduces radically to order $Nb^2$ if one exploits a banded structure of the matrix with the bandwidth $b$. Exploitation of an irregular sparsity, measured by $S$ equal to the ratio of the number of the nonzero elements in the matrix to $N^2$, generates comparable or greater reduction. For example, for an acoustic analysis reported in Reference [8] involving millions of equations, it was observed that the growth in the number of arithmetical operations was reduced to $N^{4/3}$ owing to the sparsity of the equation system. However, that reduction is too problem dependent to be expressed by a simple formula in terms of $N$ and $S$. Elaborate schemes have been created to exploit efficiently the bandedness or irregular sparsity on single and multiprocessor computers (Reference [6]). These schemes although efficient and effective on today's computers are also complex as they employ sophisticated bookkeeping and elaborate if-trees to find a compromise between the minimum of the numerical labor and the amount of memory required to hold the intermediate results. This complexity stems from the phenomenon of the sparse matrix filling-in with nonzero terms in the direct solution process. The number of the new, fill-in entries created in the process of a sparse $\mathbf{[K]}$ matrix solution is, typically, of the order of 10 to 20 times the original number of nonzero entries.

In general, the filling-in pattern cannot be precisely predicted but reordering the degrees of freedom and the associated reshuffling of the rows and columns in the matrix can mitigate its detrimental effect on both the amount of numerical labor and memory utilization. The graphic programming in VIVA does not lend itself easily to the large, sparse matrix reordering. Even if it did, that approach would still be unattractive because the experience shows that when the algorithms based on that approach are implemented on a multiprocessor computer, the interprocessor communication time grows relative to the computing time. The resulting loss of scalability brings in the law of diminishing returns that limits the number of processors that may be used efficiently to well below 100 (Reference [7]). For these reasons, Alternative 1 does not appear to be the best choice for an FPGA programmed by a pictorial language considering the currently available information.

## 4.2 Iterative solution methods

An iterative scheme has an obvious advantage of avoiding the filling-in phenomenon completely. In principle, one could assign a separate processor to each equation, or even to each $\ldots + a_{ij} * u_i \ldots$ term in the equations so that they all would be evaluated concurrently. That leaves the sparsity intact but does generate data transfers that consume significant time. Furthermore, a separate and complete fixed-point iterative process needs to be converged for each loading case whose number in practical applications may be in thousands. Focusing on the positive, one iteration of a $\mathbf{[K]}$ matrix, $N \times N$, with sparseness $S$ requires that the number of floating point multiplications be proportional to $SN^2$ and all of them could be performed in the time required by a single multiplication if a sufficient number of processors is available.

On the other hand, if the number of iterations to convergence is $M$ and the number of loading cases is $L$, the total number of multiplications grows to $SN^2ML$ and the total elapsed time to execute them in an ideally concurrent processing grows in proportion to $ML$. It follows that the advantage of the iterative solution in terms of the elapsed time decreases with the increase of the product $ML$ and may vanish beyond a certain critical threshold of that product. The value of $M$ is problem-dependent but should be expected to be large because the $\mathbf{[K]}$ matrix conditioning that governs $M$ is known to be poor. For example, Reference [6] cites a case with $\boldsymbol{N}$ = 16000 that required $\boldsymbol{M}$ = 50000 for $\boldsymbol{L}$ =1.

It appears that implementation of an iterative solution a large system of equations with exploitation of irregular sparsity is possible on a FPGA using VIVA. The language supports matrix operations by treating a matrix as one long vector and keeping track of the element location indices. Therefore, the foregoing assessment of the iterative solution, Alternative 2, implemented on a multiprocessor computer applies to FPGA as well. Therefore, one should consider Alternative 2, an iterative solution, as a viable one for FPGA up to a limit imposed by the product $\boldsymbol{ML}$.

## 4.3 Quasi-dynamic, step-by-step solution method

In Alternative 3, a structure such as the example of a cantilever beam represented by two finite element portrayed in Figure 13 is treated as a dynamic system solved by a step-by-step algorithm in time domain the same as the one used for the dynamic cases depicted in Figures 3 and 9. To use that approach, one converts the structure to a dynamic system by adding inertia and damping. In case of the beam in Figure 13, that is accomplished by assuming fictitious mass and moment of inertia associated with each of the unsupported degrees of freedom. Similarly, fictitious translational and rotational dampers are attached to these degrees of freedom. The magnitudes of the fictitious quantities are not important, except that for asymptotic convergence the damping coefficient values should be close to the critical ones.

The dynamic equation for the system so modified reads

$$[\mathbf{M}]\{\ddot{\mathbf{u}}\}+[\mathbf{D}]\{\dot{\mathbf{u}}\}+[\mathbf{K}]\{\mathbf{u}\}=\{\mathbf{P(t)}\} \qquad (6)$$

where matrices $[\mathbf{M}]$, $[\mathbf{D}]$, and $[\mathbf{K}]$ represent inertia, damping, and stiffness.

Using a matrix-vector capability in VIVA the solution diagram for Eq. (6) shown in Figure 14 looks the same as one for a single degree of freedom system except that the matrix operations replace the scalar ones. When the loading $\{\mathbf{P(t)}\}$ advances in small increments from zero to $\{\mathbf{P(t)}\}$ in fictitious time the system converges to the static state of the displacement and stress under $\{\mathbf{P(t)}\}$.

Further experimentation is needed to determine whether the number of increments for $\{\mathbf{P(t)}\}$ does not have to grow with $\boldsymbol{N}$ for good convergence. If it does not, then the solution elapsed time would grow only with the number of loading cases $\boldsymbol{L}$ and Alternative 3 would become better than Alternative 2 up to a limit governed by $\boldsymbol{L}$ and by the size of the FPGA.

Furthermore, the $\boldsymbol{L}$ limit can also be overcome if the FPGA is large enough to carry the solution of Eq. (6) for many loading cases simultaneously. Assuming that the FPGA hardware capacity develops to that level, programming the FPGA using Alternative 3 has a potential for near perfect linear scalability of the equation solution stage in static and dynamic structural analysis. As a bonus, this alternative is amenable to the simple implementation of a non-linearity as illustrated by inset in Figure 14.

## 4.4 Computing time for operations other than solving the load-deflection equations

Regardless of the load-deflection equation solution method, the total elapsed time of a structural analysis in any particular case is a sum of the equation solution time discussed above and the times consumed by the other stages in the total analysis process:

1. development of a finite element model,
2. assembly of the [**K**] matrix, and
3. interpretation of results.

The time of stage 1 is primarily the human labor. To make a radical reduction using an FPGA based system, the conventional finite element analysis would have to be rewritten, starting from the very fundamentals and tailoring the solution to the FPGA architecture. Regarding stage 3, the FPGA is being used for visualization of voluminous data in non-engineering applications, so it is conceivable that it may assist at that stage. For stage 2, the now-routine technique of assembling the [**K**] matrix consists of selective summing and placing the entries of the elemental stiffness matrices in the framework of [**K**]. That technique does not lend itself to efficient implementation in VIVA for the same reasons that were previously discussed in conjunction with Alternative 1. That leaves an option of assembling [**K**] by the textbook operation.

$$[\mathbf{K}] = [\mathbf{A}]^T\,[\mathbf{k_{diag}}]\,[\mathbf{A}] \qquad (7)$$

where [**A**] stands for the 0-1 connectivity matrix and $[\mathbf{k_{diag}}]$ is the block-diagonal matrix of the unassembled structure. VIVA supports matrix operations so it could execute Eq. (7); however, because the matrices involved are mostly zeros, it would be mandatory to implement a zero-bypass capability in the VIVA language.

Eventually, the FPGA capability may prove to be a strong enough motivator for development of non-conventional solution methods. One example of such emerging methods whose intrinsic parallelism makes them a natural match for FPGA is the cellular automata method that has been demonstrated effective in structural analysis (Reference [9]) and in both analysis and optimization in Reference [10].

The pictorial programming in VIVA has the obvious advantage of being very intuitive. In computer science community, there is no consensus whether that advantage can be sustained when the volume of detail to be seen begins to exceed of what a single screen may display. The authors experience to date reinforced with the growing use of graphical systems such as Reference [4], indicates that the ability to collapse a single screen diagram into a single icon to be used as a building block in another screen is effective in keeping the visual code complexity under control. It is roughly equivalent to the generally accepted good practice principle of structured programming to keep a subroutine length to one page.

## 5. Concluding Remarks

Initial exploratory experience to date with a particular FPGA system, the VIVA/HAL-30 software/hardware combination, has been reported and its merits and demerits assessed. Tentatively, on the negative side one should point to the lack of informative error messages, an effective debugger, and absence of textual or graphic output rendering to a disk. Considering that this FPGA technology is now in its infancy, one may expect that these shortcomings will be addressed as the VIVA product matures. On the positive side, the FPGA hardware/software system fully exploits the solution algorithm's inherent parallelism, and no engineering time needs to be spent on devising solutions to the governing equations. The problems of scaling, precision, and stability faced by the analog computer user in the past are absent owing to the digital accuracy and repeatability. However, the careful choice of the time step in dynamic applications to stay within the problem's time scale is still required.

The analog programming style realizes best the FPGA potential in structural analysis. It offers a potential to reduce the equation solving elapsed time, and an easy way to implement non-linearities. Extension of the FPGA utility to all the phases of a complete structural analysis process, will require redevelopment from the ground up tailored to the FPGA architecture.

There is no doubt that computing technology has embarked on a path of inherent parallelism in processing that in due course will likely supplant many of the engineering physics solutions that were developed in the past for the single processor, sequential computing. The FPGA-based system is one type of a computer on that path, and the methods tailored to its architecture now will grow in their effectiveness as the FPGA technology develops. The first exploratory step reported herein has but scratched the surface of the potential for this class of computers and solution methods.

## 6. References

1. HANDEL-C at www.celoxica.com
2. VIVA at www.starbridgesystems.com and are marks of Star Bridge Systems, Copyright 1998-2001 by Star Bridge Systems, Inc.
3. HAL-15 at www.starbridgesystems.com and are marks of Star Bridge Systems, Copyright 1998-2001 by Star Bridge Systems, Inc.
4. SIMULINK at www.mathworks.com
5. Michael G. Rekoff, Jr., Analog Computer Programming, Charles E. Merrill Books, Inc, Columbus, Ohio, 1967.
6. Nguyen, D. T.: Parallel-vector Equation Solvers for Finite Element Engineering Applications; Kluwer Academic Publ., 2002.
7. Storaasli, O.O.; Nguyen, D.T.; Baddourah, M.A.; and Qin, J.: Computational Mechanics Analysis Tools for Parallel Vector Supercomputers; Computing Systems in Engineering, Vol.4, N0:4-6, pp.349-354, Elsevier Publ. 1993.
8. Watson, W.R.; and Storaasli, O.O.: Application of NASA general purpose solver to large-scale computations in aeroacoustics; Advances in Engineering Software, No.31, pp.555-561.
9. Kim, P.; and Hajela, P.: Elastic Element Based Cellular Automata Models in Structural Design; 4th World Congress on Structural and Multidisciplinary Optimization, Dalian, China, June 2001.
10. Abdalla, M.; and Gurdal, Z.: Structural Design Using Optimality Based Cellular Automata; Paper AIAA-2002-1676; 43rd AIAA Structures, Dynamics, and Materials Conference; Denver, CO, April 2002.

**Table 1: Results of Fixed Point Iterations with $Z_0$=3, $Y_0$=2, a=P=1, b=d=0.5, M=0.0001**

| FORTRAN (double precision – $10^{-13}$) | | | VIVA | | K (non-linearity) |
|---|---|---|---|---|---|
| $Y_\infty$ | $Z_\infty$ | Iterations | $Y_\infty$ | $Z_\infty$ | |
| 1.3332667 | -0.66653333 | 47 | 1.333267 | -0.6665334 | 0.0 |
| 1.1553033 | -0.57755163 | 69 | 1.155303 | -0.5775517 | 0.1 |

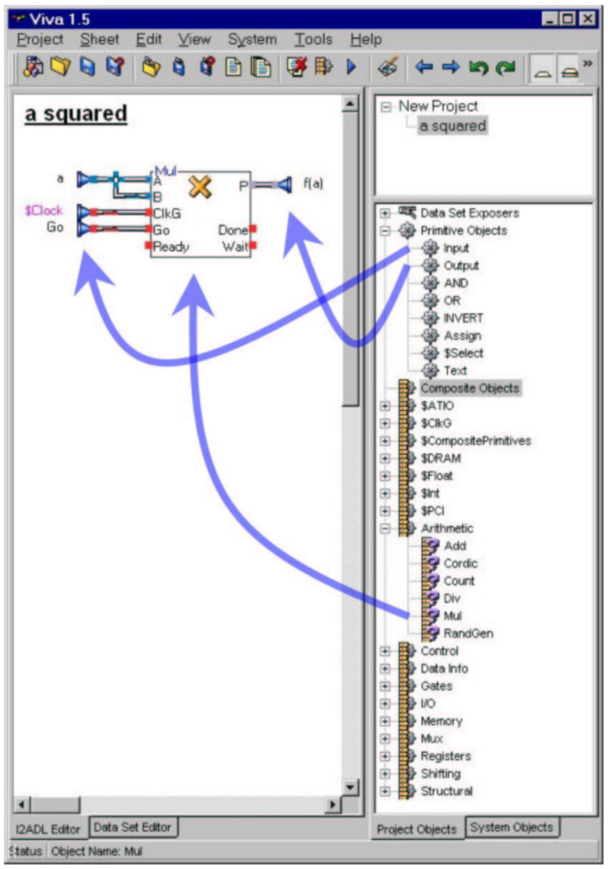


**Figure 1: Viva Algorithm to Calculate f(a) = a$^2$ (Features left not explained are internal VIVA housekeeping details beyond the scope of this report)**

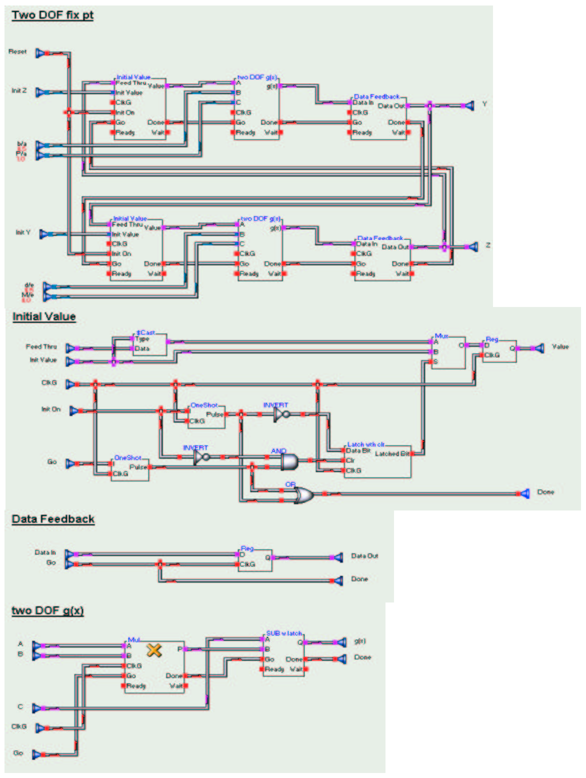
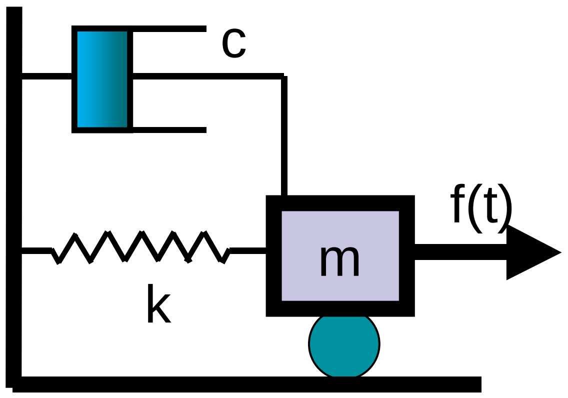


**Figure 2: Viva Algorithm to Solve Equation (3) Via the Fixed Point Algorithm for Two Degrees of Freedom**



$$\ddot{x} = \frac{1}{m}\left[c\dot{x} + kx + f(t)\right]$$

**Figure 3: Frictionless Mass and Spring with Damper**

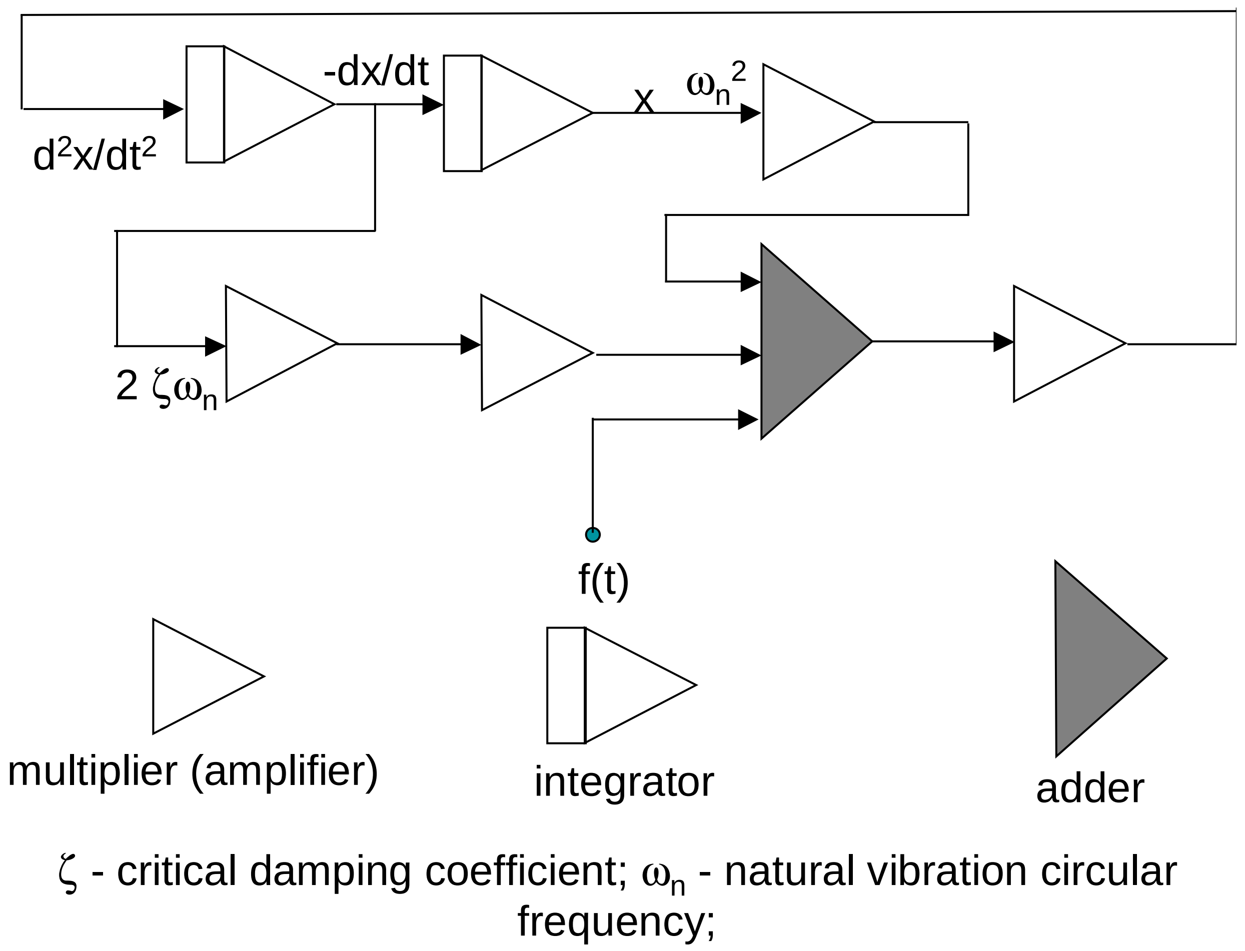

-dx/dt
$d^2x/dt^2$
x
$\omega_n^2$
$2\,\zeta\omega_n$
f(t)
multiplier (amplifier)
integrator
adder
$\zeta$ - critical damping coefficient; $\omega_n$ - natural vibration circular frequency;

**Figure 4: Analog Computer Diagram corresponding to the System in Figure 3; It is Directly Translatable into the Viva Algorithm in Figure 5**

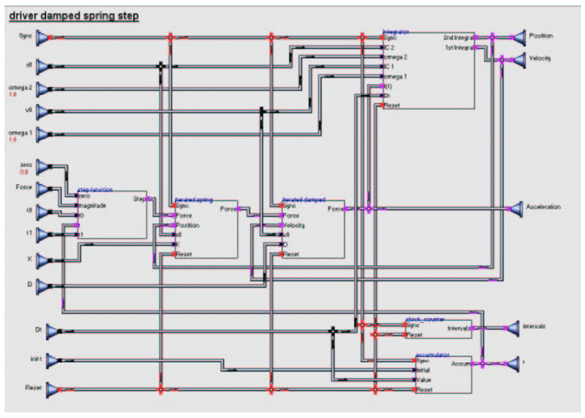


**Figure 5: Viva Algorithm to Solve the Mass and Spring with Dampers from Figure 3**

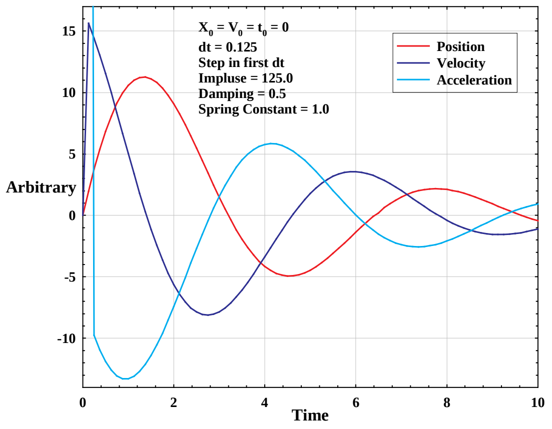


**Figure 6: Plotted Output from the Viva Algorithm in Figure 5**

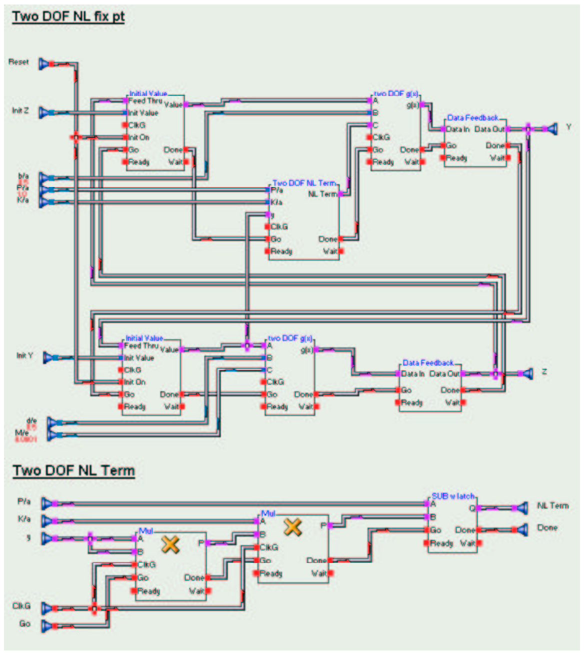


Figure 7: Viva Algorithm to Solve Equation (4) Via the Fixed Point Algorithm with a Non-linearity

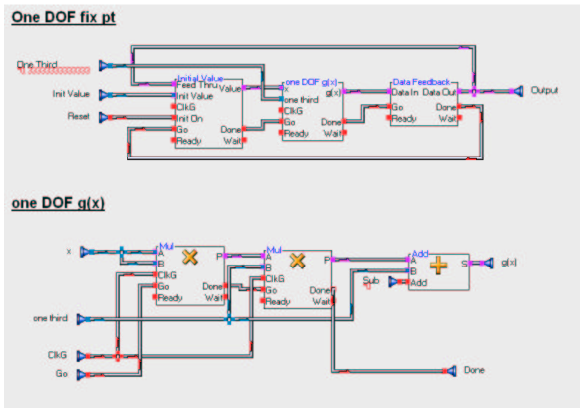
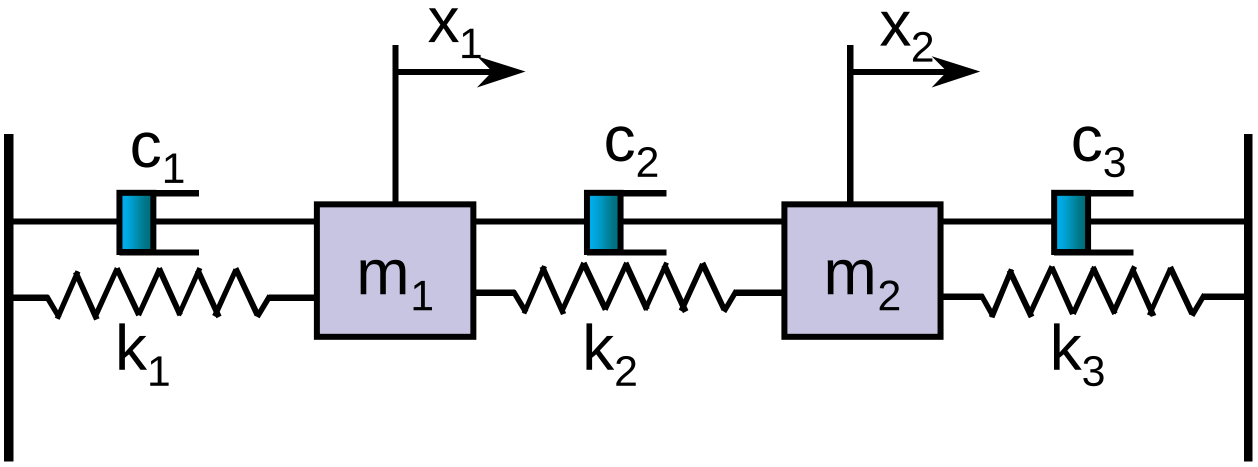


Figure 8: Viva Algorithm to Solve Equation (5) Via the Fixed Point Algorithm for One Degree of Freedom



$$\ddot{\mathbf{x}}_1 = \frac{\mathbf{1}}{\mathbf{m}_1}\left[\mathbf{c}_2\dot{\mathbf{x}}_2 + \mathbf{k}_2\mathbf{x}_2 - (\mathbf{c}_1 + \mathbf{c}_2)\dot{\mathbf{x}}_1 - (\mathbf{k}_1 + \mathbf{k}_2)\mathbf{x}_1\right]$$

$$\ddot{\mathbf{x}}_2 = \frac{\mathbf{1}}{\mathbf{m}_2}\left[\mathbf{c}_2\dot{\mathbf{x}}_1 + \mathbf{k}_2\mathbf{x}_1 - (\mathbf{c}_2 + \mathbf{c}_3)\dot{\mathbf{x}}_2 - (\mathbf{k}_2 + \mathbf{k}_3)\mathbf{x}_2\right]$$

Figure 9: Two Masses Connected by Three Springs with Three Dampers on a Frictionless Surface

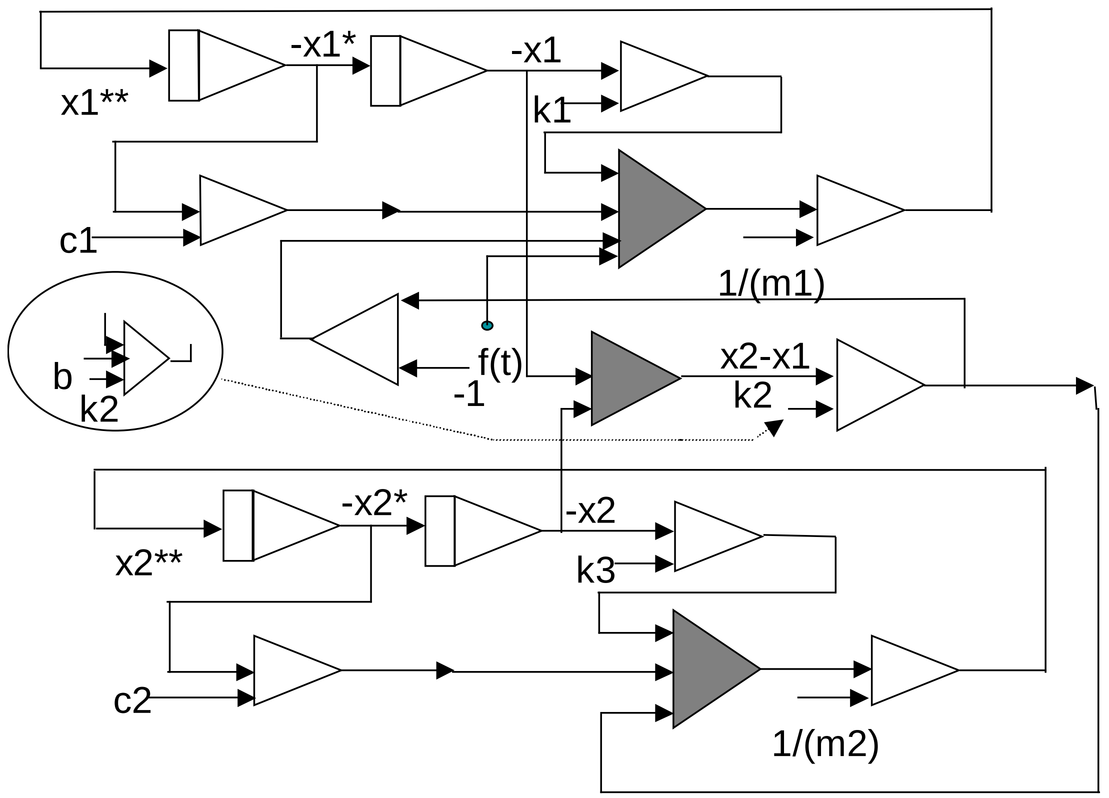


Figure 10: Analog Equivalent Wiring Diagram to Solve System in Figure 9

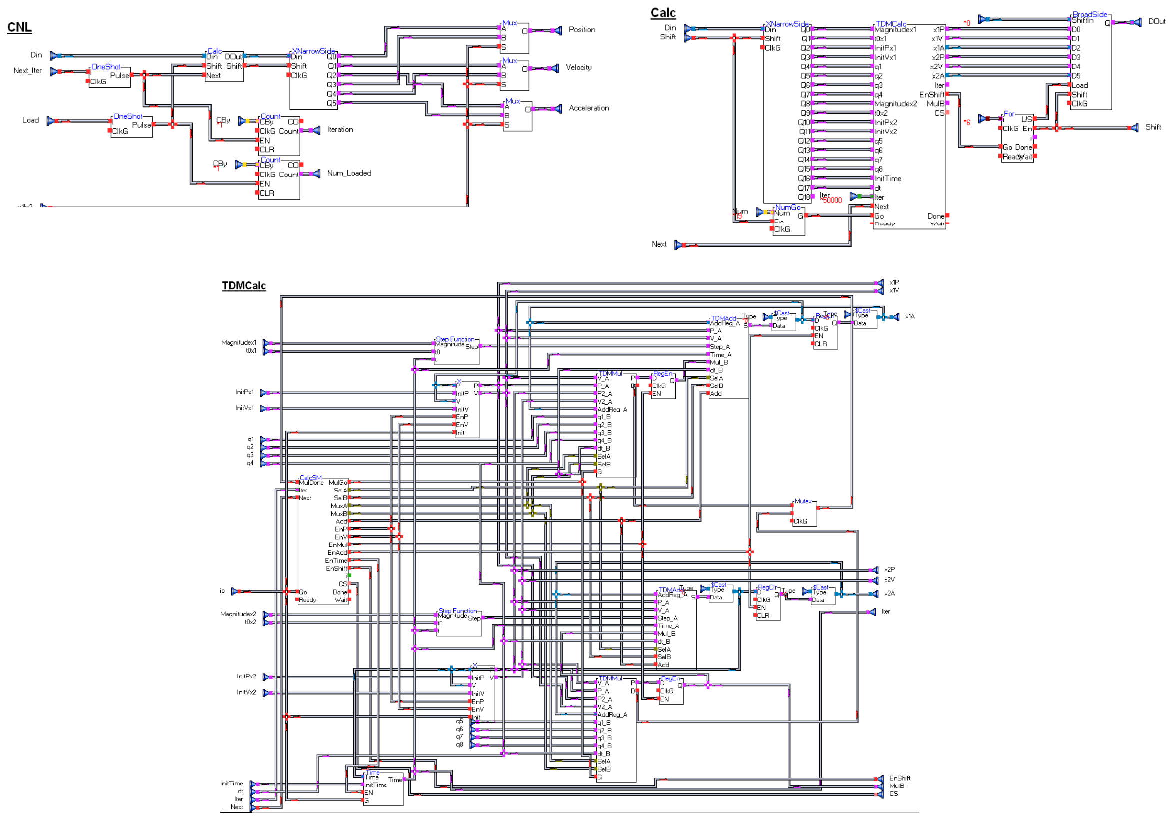


Figure 11: Viva Algorithm to Implement Figure 10

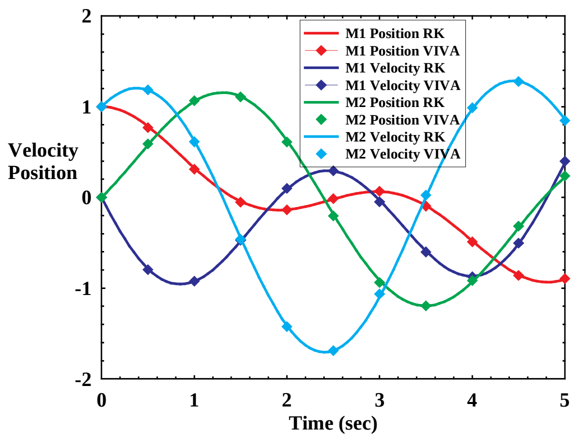


**Figure 12: VIVA and Runge-Kutta (MathCAD 2000) Output for the Model in Figure 9**

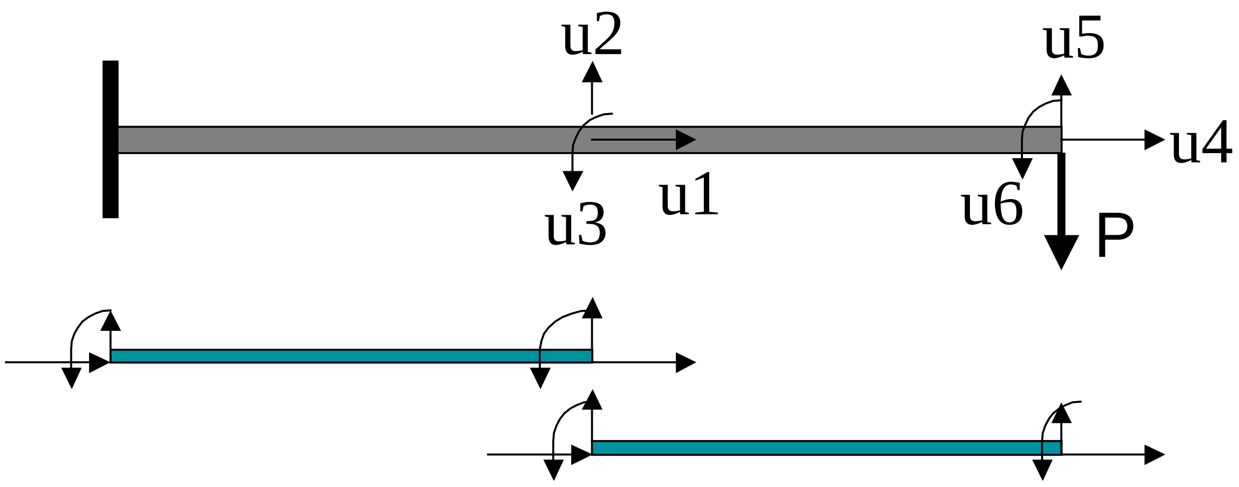


**Figure 13: A Cantilever Beam Represented By A Finite Element Model Of Two Elements; u - displacements of the structure assembled and supported; displacements of unassembled structure left not labeled.**

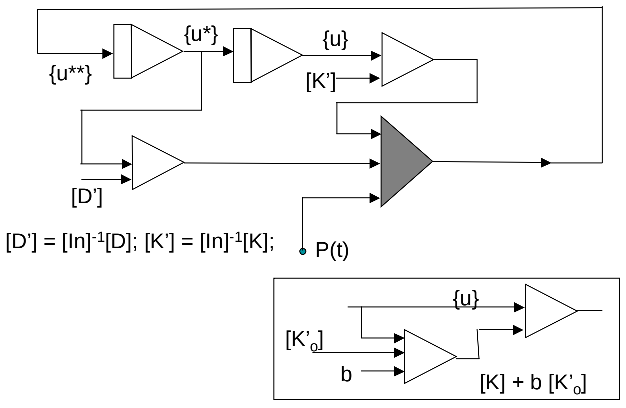


**Figure 14: Analog Solution Diagram For A Structure With N Degrees Of Freedom. Inset Shows a Modification To Simulate a Non-Linear Case In Which Stiffness [K] Depends On Displacement u**