**CHAPTER-1**

**INTRODUCTION**

**1.1 Motivation**

World has recently witnessed a steep growth in the number of components that can be fabricated on a single chip. Components having sizes as small as few nanometer are being grown on ICs. According to Moore’s Law the number of [transistors](http://en.wikipedia.org/wiki/Transistor) that can be placed inexpensively on an [integrated circuit](http://en.wikipedia.org/wiki/Integrated_circuit) doubles approximately every two years[9].

Scaling down of the size poses some other serious threats to the performance of the circuits. As we try to reduce the width to length ratio of the transistors, the delay at the output may increase substantially.

We have tried to utilize the Genetic algorithm to find us the smallest possible dimension of the transistor which adheres to all the parameters which need to be taken care off.

**1.2 Literature review**

Yousef Mortazavi and Amir Amirabadi tried to optimize the sizing of digital adder circuits. The circuit is implemented on .35u m technology. The simulated annealing is employed to optimizing the size of full adder. The MATLAB code is used to implement the simulated annealing algorithm. So as to implement the suggested changes, the MATLAB code changes the parameters in the net list itself and thus reducing the number of steps involved in optimization. Out of numerous simulated circuits it may be possible that some circuit may result in incorrect function, these circuits may be attributed a large cost.[8]

K. Navi and B. Mazloomniza proposed a full adder circuit with six transistors. They utilized multi valued logic for the operation of the circuit. The multi valued logic helps reduce the number of mos transistors and wiring requirement. In order to implement multi valued logic they made use of CML in this current is define to have logical levels that are integer multiple of reference current. The reduction in the transistor counts ultimately leads to the reduction of chip area. They have been able to achieve delays of as low as 150 pico seconds.

Andew R. Conn and Paula K. Coulman tried to implement a circuit optimization tool that used augmented lagrangian formulation. This tool enables us too target delay, rise time fall time and area. Each of these can either be a constraint or an objective function. The w/l ratio may be changed and similar structure may be grouped together. Some of the circuits designed using this tool may not work according to the desired logic that is when a transition is to be measured it doesn’t occur.[1]

Volker Schnecke and Oliver vorn berger proposed a variant of genetic algorithm fo physical design of vlsi chip. This algorithm tries to simultaneously optimize the both the placement and routing. Unlike previous works which serially optimize the cells and routing. This algorithm does it all in parallel. The work done by Oliver and Volker is appreciable but lacks a visualization of placement and wiring.[4]

K. GlasMacher and Zimmerman proposed a genetic algorithm for the improvements of macro cell layout. The complete layout can be considered to be made of numerous macro cells. The proposed method utilizes the concept of channel density by introducing a distance function to represent channel width. This width may be altered by shifting adjacent macro cells. Since the method is implemented serially the speed of the algorithm is rather low.[10]

**1.3 Objective of the project**

In an effort to reduce the area occupied by the circuits, researchers have implemented several techniques of routing and placement they have tried to optimized area by rearranging the modules and achieving better placement similarly they have tried to rearrange the interconnection paths for better routing.

Where as we have tried to optimize the area by transistors resizing. The transistors have been reduced to smallest possible dimensions for which the transistor posses acceptable performance parameters. A rough estimate of the size of the circuit can be calculated by the product of the dimension and the number of transistors. When we increase the w/l ratio, the delay increase and vice versa. The challenge is to reduce the dimensions and simultaneously maintain a permissible delay. Little work has been done to incorporate the use of genetic algorithm in area optimization. A lot of new things need to be explored in this field.

**CHAPTER-2**

**OPTIMIZATION**

**2.1 Introduction**

Optimization is the process of making something better. An engineer or scientistconjures up a new idea and optimization improves on that idea. Optimization consists in trying variations on an initial concept and using the information gained to improve on the idea. A computer is the perfect tool for optimization as long as the idea or variable influencing the idea can be input in electronic format. Feed the computer some data and out comes the solution Optimization is the math tool that we rely on to get these answers. A simple example reveals many shortfalls of the typical minimum seekers. Since the local optimizers of the past are limited, people have turned to more global methods based upon biological processes.

**2.2 Finding the Best solution**

The terminology “best” solution implies that there is more than one solution and the solutions are not of equal value. The definition of best is relative to the problem at hand, its method of solution, and the tolerances allowed. Thus the optimal solution depends on the person formulating the problem. Education, opinions, bribes, and amount of sleep are factors influencing the definition of best. Some problems have exact answers or roots, and best has a specific definition. Other problems have various minimum or maximum solutions known as optimal points or extreme, and best may be a relative definition.

**2.3 What is Optimization?**

Our lives confront us with many opportunities for optimization. What time do we get up in the morning so that we maximize the amount of sleep yet still make it to work on time? What is the best route to work? Which project do we tackle first? When designing something, we shorten the length of this or reduce the weight of that, as we want to minimize the cost or maximize the appeal of a product. Optimization is the process of adjusting the inputs to or characteristics of a device, mathematical process, or experiment to find the minimum or maximum output or result. The input consists of variables; the process or function is known as the cost function, objective

function, or fitness function; and the output is the cost or fitness. If the process is an experiment, then the variables are physical inputs to the experiment. For most of the examples, we define the output from the process or function as the cost. Since cost is something to be minimized, optimization becomes minimization. Life is interesting due to the many decisions and seemingly random events that take place. Quantum theory suggests there are an infinite number of dimensions, and each dimension corresponds to a decision made.

Life is also highly nonlinear, so chaos plays an important role too. A small perturbation in the initial condition may result in a very different and unpredictable solution. These theories suggest a high degree of complexity encountered when studying nature or designing products. Science developed simple models to represent certain limited aspects of nature. Most of these simple (and usually linear) models have been optimized. In the future, scientists and engineers must tackle the unsolvable problems of the past, and optimization is a primary tool needed in the intellectual toolbox



Figure 2.1Diagram of a function or process that is to be optimized. Optimization varies the input to achieve a desired output.

**2.4 Categories of Optimization**

The given divides optimization algorithms into six categories None of these six views or their branches are necessarily mutually exclusive. For instance, a dynamic optimization problem could be either constrained or unconstrained. In addition some of the variables



Figure 2.2 Six categories of optimization algorithms

may be discrete and others continuous. Let’s begin at the top left of Figure 1.2 and work our way around clockwise.

**2.4.1 Trial-and-error optimization**

It refers to the process of adjusting variables that affect the output without knowing much about the process that produces the output. Various mathematical manipulations of the function lead to the optimal solution. Theoreticians love this theoretical approach.

**2.4.2 Single variable and multiple variable optimizations**

If there is only one variable, the optimization is one-dimensional. A problem having more than one variable requires multidimensional optimization. Optimization becomes increasingly difficult as the number of dimensions increases. Many multidimensional optimization approaches generalize to a series of one-dimensional approaches.

**2.4.3 Static and Dynamic optimization**

Dynamic optimization means that the output is a function of time, while static means that the output is independent of time. When living in the suburbs of Boston, there were several ways to drive back and forth to work. What was the best route? From a distance point of view, the problem is static, and the solution can be found using a map or the odometer of a car. The shortest route isn’t necessarily the fastest route. Finding the fastest route is a dynamic problem whose solution depends on the time of day, the weather, accidents, and so on. The static problem is difficult to solve for the best solution, but the added dimension of time increases the challenge of solving the dynamic problem.

**2.4.4 Discrete or Continuous variables Optimization**

Optimization can also be distinguished by either discrete or continuous variables. Discrete variables have only a finite number of possible values, whereas continuous variables have an infinite number of possible values. Discrete variable optimization is also known as combinatorial optimization, because the optimum solution consists of a certain combination of variables from the finite pool of all possible variables.

**2.4.5 Constrained and Unconstrained Optimization**

Variables often have limits or constraints. Constrained optimization incorporates variable equalities and inequalities into the cost function. Unconstrained optimization allows the variables to take any value. A constrained variable often converts into an unconstrained variable through a transformation of variables. When constrained optimization formulates variables in terms of linear equations and linear constraints, it is called a linear program. When the cost equations or constraints are nonlinear, the problem becomes a nonlinear programming problem.

**2.4.6 Random and Minimum seeking Optimization**

Some algorithms try to minimize the cost by starting from an initial set of variable values. These minimum seekers easily get stuck in local minima but tend to be fast. They are the traditional optimization algorithms and are generally based on calculus methods. On the other hand, random methods use some probabilistic calculations to find variable sets. They tend to be slower but have greater success at finding the global minimum.[10]

**CHAPTER-3**

**FULL ADDER**

**3.1 What is full adder?**

A 1‐bit adder takes two 1‐bit inputs and adds them together. To make it a full adder, it also needs to consider a carry in and carry out flag. Thus a 1‐bit full adder takes three 1‐bit inputs and contains two 1‐bit outputs. The first two inputs are the two bits that are to be added together, respectively A and B. The third input is a carry in flag. This flag specifies whether or not a previous addition has occurred which contained a carry out. The first output is the 1‐bit result of the addition. The second output is the carry out flag which specifies if the result of the addition was larger than the 1‐bit result.

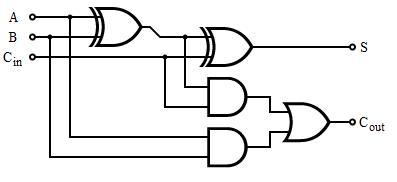


Fig 3.1(Full Adder Circuit using logic gates)

3.2 **How does a full adder works?**

The Boolean expression for full adder circuit is given as:



The truth table for full adder is as:

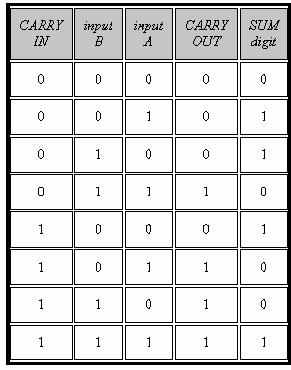


Fig 3.2 (Truth table of Full Adder)

**3.3 CMOS Full Adder**

For translating the gate-level design into a transistor-level circuit description, we note that both the sum..out and the carryout functions are represented by nested ANDOR- NOR structures in Fig. 3.3 Each such combined structure (complex logic gate) can be realized in CMOS as follows: the AND terms are implemented by *series-connected* nMOS transistors, and the OR terms are implemented by parallel-connectednMOS transistors. The input variables are applied to the gates of the nMOS (and the complementary pMOS) transistors. Thus, the nMOS net may consist of nested series-parallel, connections of nMOS transistors between the output node and the ground. Once the nMOS part of a complex CMOS logic gate is realized, the corresponding pMOS net, which is connected between the output node and the power supply, is obtained as the *dual* *network* of the nMOS net.

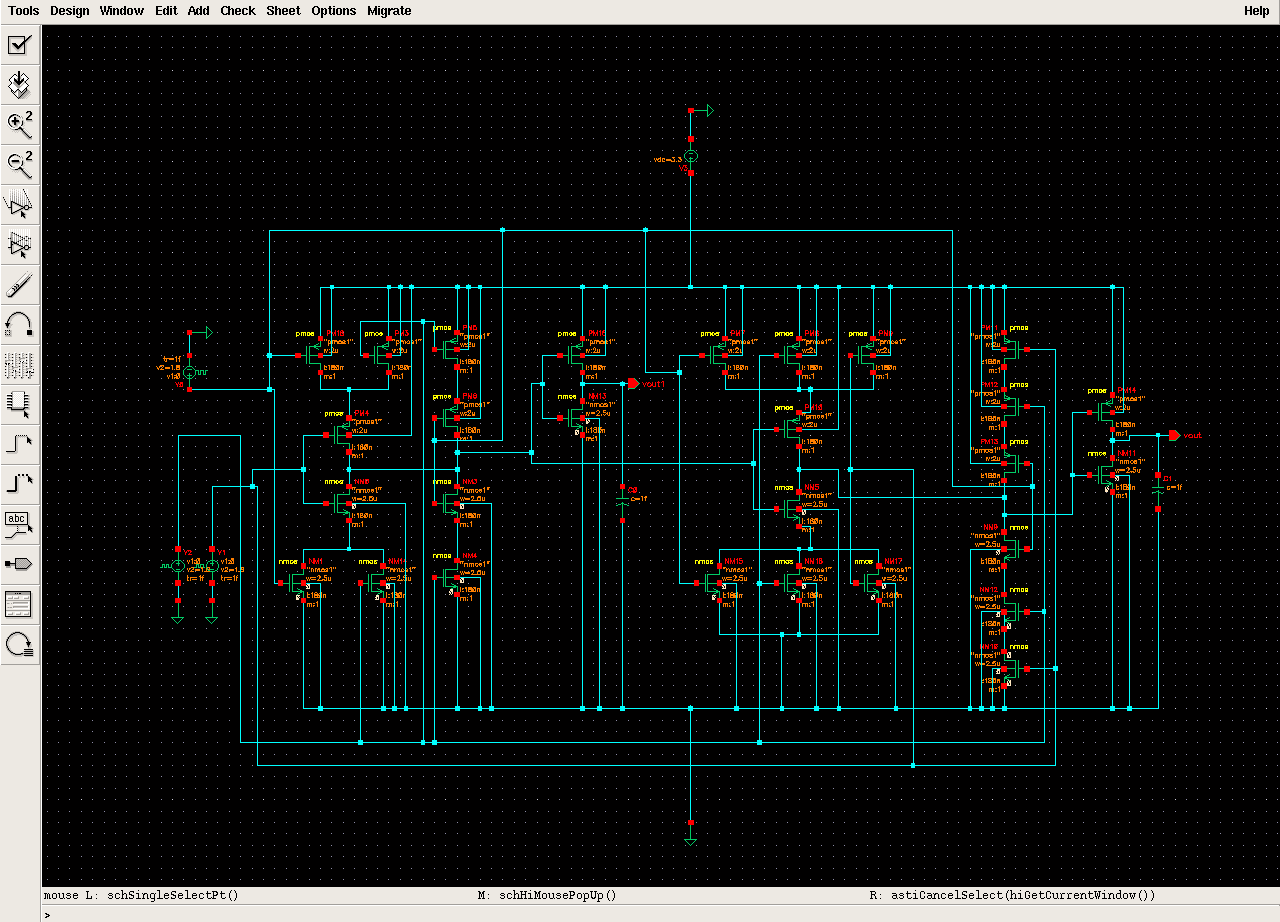


Fig 3.3 (1-Bit CMOS Full Adder Implemented On Cadence Virtuso-4 )

Fig 2.4 shows a basic CMOS full adder that was implemented using the cadence virtuso -4 tool. The Full adder circuit utilized 28 transistors. Fourteen of them are NMOS and the rest of them PMOS so as to complete the CMOS inverter structure. Our Full adder circuit is shown in the adjoining figure. This circuit is working at a supply voltage of 3.3V. The three inputs of circuit have been given a pulsating input. In order compensate the overall capacitance, Sum and Carry outputs have been connected with load capacitors. The logic table of a full adder is shown fig 3.2 .The following wave form depicts the input and output wave when the load capacitance in negligible (1f F), The input A has a pulse width of 5ns and a time period of 10ns, The input B has a pulse width of 10ns and a time period of 20ns, The input C has a pulse width of 25ns and a time period of 40ns. The pulsating inputs given to A, B & C cover almost all the input combinations. The results prove that the full adder is working as it should.

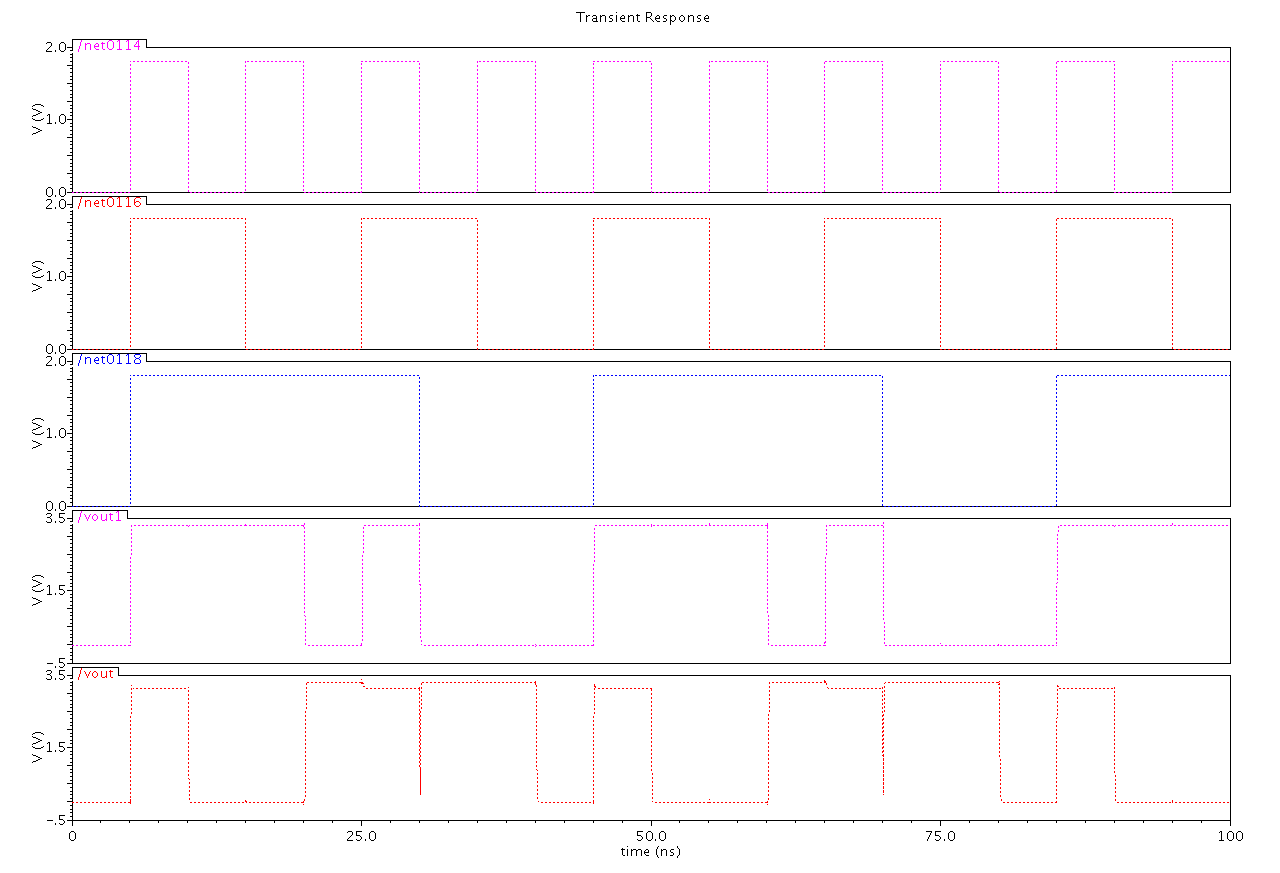


Fig 3.4 (Waveforms of CMOS Full Adder)

**CHAPTER-4**

**GENETIC ALGORITHM**

**4.1 Introduction to GA**

Genetic algorithm was developed in the year 1975 at University of Milligan by Prof. John Holland. Genetic algorithm and its variants are basically computational procedures which try to ape the natural phenomenon of evolution. Evolution is the phenomenon by which species may refine themselves over a number of generations in order to adapt to environment. Evolution was first studied by Darwin. He analyzed that only the fittest of every generation would survive, eventually adapting to the environment. Two parents reproduce to give new off springs. These offspring have chromosomes from both parents i.e have characteristics of both the parents. This is the general way that Genetic algorithm and its variants work to solve problems of adaptive systems and optimum solutions. Since Genetic algorithm is heuristic in nature they do not give us ideal solutions yet the solution given by them are reasonably good.

In Genetic algorithm each individual is assumed to have a fitness function associated with it, in order for an individual to be eligible for the next generation it must poses some minimum fitness value. For each generation individuals are selected from the existing population and crossed among each other to form the new and refined set of population. Here we can categorize Genetic algorithm in to two types:

1. Simple Genetic algorithm.
2. Steady state Genetic algorithm.

Simple Genetic algorithm is the one in which new generation may completely replace the old individuals whereas in Steady State Genetic algorithm the new population may consist of a combination of old and new individuals. Decision for which one to use may differ from application to application. In either of the case, fitness value increases from one generation to another.



Fig4.1 (Explanation of Genetic Algorithm)

**4.2 Two Basic Type of Genetic Algorithms**

**4.2.1 Simple Genetic algorithm**

The simple genetic algorithm utilizes three evolutionary operators viz: Selection, Crossover and Mutation. Each chromosome is an encoding of a solution to the problem at hand and each individual has an associated fitness which depends on the application. The fitness of the population increases with every evolving generation by selecting two individuals, crossing the parents and mutating the characteristics. In Genetic algorithm selection is done probabilistically but more towards highly fit individuals. Distinct generations are evolved and the process of selection crossover and mutation are repeated until all entries in a new generation are filled[2].

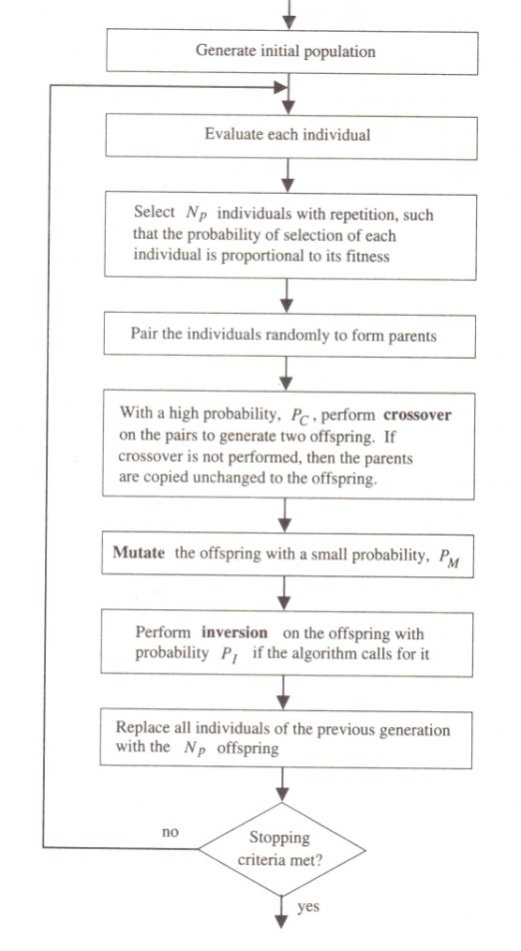


Fig 4.2 (Flow Chart of Simple GA )

**4.2.2 Steady State Genetic Algorithm**

In a Genetic algorithm having overlapping generations only a fraction of the individuals are replaced in each generation. In each generation, two different individuals are selected as parents according to their fitness. Crossover is performed with a high probability to form crossover. The offspring are mutated with a low probability and may be inverted if necessary. A duplicate check may be performed to discard the duplicates if. The offspring are then evaluated and survive only if they are better than the worst member of the current population. In steady state genetic algorithm the generation gap is minimal, Since only two offspring are produced in each generation [2].

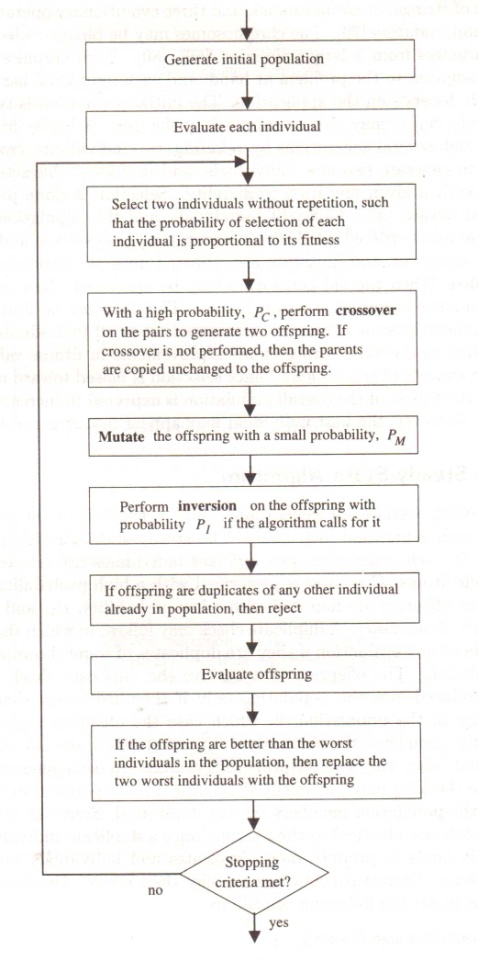


Fig 4.3 (Flow Chart of Static GA)

**4.3 Basic Terminology of Genetic Algorithm**

**4.3.1 Chromosomes**

Central to all genetic algorithms is the concept of the chromosome. The chromosome contains all information necessary to describe an individual. In nature, chromosomes are composed of DNA. In a computer, a long binary or character string is used. Chromosomes are composed of genes for the various characteristics to be optimized and can be any length depending on the number of parameters to be optimized.

**4.3.2 Encoding**

Encoding defines the way genes are stored in the chromosome and translated to actual problem parameters. A possible encoding scheme for a hypothetical circuit using a 16 bit binary chromosome is shown below:



**4.3.3 Fitness**

Fitness is a single numerical quantity describing how well an individual meets predefined design objectives and constraints. Fitness can be computed based on the outputs of multiple analyses using a weighted sum.

**4.3.4 Crossover**

Crossover is a method of exchanging genetic material between two parents to produce offspring. The operation is simple, a cross-over point is chosen at random and the genetic information to the left of the cross-over point in parent A is combined with the genetic information to the right of the cross-over point in parent B to produce new offspring. A second offspring can be created by using the information from the right of parent A and the left of parent B. The operation is illustrated with the example below:



**4.3.5 Mutation**

Mutation in Genetic algorithm is somewhat similar to the genetic mutation. It is used to maintain genetic diversity between subsequent generations. Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution or may be different only by a single bit. Mutation may be of the following types:[12]

* Bit String mutation.
* Flip-Bit Mutation.
* Uniform Mutation.
* Non – Uniform Mutation.
* Gaussian Mutation.

**Chapter-5**

**Layout Area**

Area is important for two reasons. First, a larger area means a chip can have more computing elements and do more work. Also, more area means a chip can have more I/O ports to facilitate data movement on and off the chip.

Therefore many tools and techniques have been developed to address the complexity of chip layout. Typically these tools and techniques use abstraction; i.e. they decompose a problem into successively lower level units of increasing complexity. At each level the number of units involved in a design is kept small so that the design can be understand easily.

The design of a VLSI chip begins with the specification of its functionality at the **functional** or **algorithmic level**. Either a function or an algorithm is given as the starting point.An algorithm is then produced and translated into a specification at the **architectural level**. At this level a chip is specified in terms of large units such as a CPU, random-access memory, bus, floating-point unit, and I/O devices. After an architectural specification is produced, design commences at the **logical level**. Here particular methods for realizing architectural units are chosen. For example, an adder could be realized either as a ripple or a carry-look ahead adder depending on the stated speed and cost objectives.

At the **gate level**, the next level in the design process, a technology, such as NMOS and CMOS, is chosen in which to realize the transistors and wires. This involves specifications of widths for wires, the number of layers of metal, and other things. If new transistor layouts are used, their physics is often simulated to determine their electrical properties.

At the next level, the **layout level**, a gate-level design is translated into physical positions for modules, gates, and wires. Often at this level a rough layout is produced manually, after which automatic routing and compaction algorithms are invoked to route wires between modules and squeeze out the unnecessary area. Space must be reserved on each layout for **I/O pads**, rectangular regions large enough to connect external wires. They serve as **ports** through which data is read and written. Because these wires and pads are very large by comparison with the wires on the chip, there is a practical limit on the number of I/O ports on a chip. A port can be both an input and an output port. Once a layout is complete it is usually simulated logically, that is, at the level of Boolean gates.

The physical design describes the transformation of a circuit description which is the result of the preceding circuit design process into the layout of a chip. The layout includes the geometrical description of the building blocks and the information for the routes of the interconnections between them and a schematic representation of a layout. The main concern in the physical design f VLSI chips is to find a layout with minimal area.

we began with a logic diagram along with design specifications. The logic circuit is first translated into a CMOS circuit and the initial layout is done. From the layout, the entire important parasitic are calculated by using a circuit extraction program. Once a full circuit description was obtained from the initial layout, we analyzed the circuit transient performance by using the circuit-level simulation program on cadence virtuoso4.

For translating the gate-level design into a transistor-level circuit description, we noted that both the sum..out and the carryout functions were represented by nested ANDOR- NOR. Each such combined structure (complex logic gate) can be realized in CMOS as follows: the AND terms were implemented by *series-connected* nMOS transistors, and the OR terms were implemented by *parallel-connected* nMOS transistors. The input variables were applied to the gates of the nMOS (and the complementary pMOS) transistors. Thus, the nMOS net may consist of nested series-parallel , connections of nMOS transistors between the output node and the ground. Once the nMOS part of a complex CMOS logic gate was realized, the corresponding pMOS net, which was connected between the output node and the power supply, was obtained as the *dual* *network* of the nMOS net. The resulting transistor level design of CMOS full adder circuit. Note that the circuit contains a total of 14 nMOS and 14 pMOS transistors, together with the two CMOS inverters which were used to generate the outputs. In this, it can also be shown that the dual (pMOS) network is actually *equivalent* to the nMOS network for both the sum\_out and the carry-out functions, which leads to a fully *symmetric* circuit topology.

The approach to increase switching speed, and thus, to reduce delay times, would be to increase the *(WIL)* ratios of all transistors in the circuit. However, increasing the transistor *(WIL)* ratios also increases the gate, source, and drain areas and, consequently, increases the parasitic capacitances loading the logic gates.

**CHAPTER – 6**

**TOOLS USED**

**6.1 Cadence**

Cadence is an Electronic Design Automation (EDA) environment that allows integrating in a single framework different applications and tools (both proprietary and from other vendors), allowing to support all the stages of IC design and verification from a single environment. These tools are completely general, supporting different fabrication technologies. When a particular technology is selected, a set of configuration and technology-related files are employed for customizing the Cadence environment. The Fig 6.1 shows the design flow in Cadence development tool. We first make a schematic of the circuit according to the specifications, then the circuit is simulated to check the correctness of working then the layout of the circuit is made and verified. Finally the layout is simulated again to check for last moment errors and the design is then sent for fabrication.[11]

##### **6.1.1 Features of Cadence Design Environment**

* Reduced learning curve with a simulator-independent environment
* Maximum efficiency in the script-driven mode
* Accelerated debug process using a variety of built-in analog analysis tools
* Facilitated design correction via easy comparison of pre- and post-parasitic extracted designs
* Quick detection of circuit problems via a clear visualization cockpit

****

Fig 6.1( Design flow in Cadence Design Environment )

**6.1.2 Cadence Virtuoso**

Cadence Virtuoso is integrated design platform targeted for both digital and analog applications. It includes multiple physical verification tools, such as Design Rules Check (DRC), Layout vs. Schematic (LVS), Electrical Rules Check (ERC), Layout Parasitic Extraction (LPE) and Abstract Generation. In order to invoke Virtuoso, additional files are required. New technology file (tech65n.tf) with process dependent layout information is used to create new library. The graphical display template (display.drf) is used to display the different layer color and stipple pattern.[12]

**6.2 MATLAB**

MATLAB is a programming environment for algorithm development, data analysis, visualization, and numerical computation. Using MATLAB, you can solve technical computing problems faster than with traditional programming languages, such as C, C++, and Fortran.

You can use MATLAB in a wide range of applications, including signal and image processing, communications, control design, test and measurement, financial modeling and analysis, and computational biology. For a million engineers and scientists in industry and academia, MATLAB is the language of technical computing. [7]

**6.2.1 Key Features of MATLAB:**

* High-level language for technical computing
* Development environment for managing code, files, and data
* Interactive tools for iterative exploration, design, and problem solving
* Mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, and numerical integration
* 2-D and 3-D graphics functions for visualizing data
* Tools for building custom graphical user interfaces
* Functions for integrating MATLAB based algorithms with external applications and languages, such as C, C++, Fortran, Java™, COM, and Microsoft® Excel®

## 6.2.2 Developing Algorithms using MATLAB

MATLAB provides a high-level language and development tools that let you quickly develop and analyze your algorithms and applications. The MATLAB language supports the vector and matrix operations that are fundamental to engineering and scientific problems. It enables fast development and execution. With the MATLAB language, you can program and develop algorithms faster than with traditional languages because you do not need to perform low-level administrative tasks, such as declaring variables, specifying data types, and allocating memory. In many cases, MATLAB eliminates the need for ‘for’ loops. As a result, one line of MATLAB code can often replace several lines of C or C++ code. At the same time, MATLAB provides all the features of a traditional programming language, including arithmetic operators, flow control, data structures, data types, [object-oriented programming](http://www.mathworks.in/products/matlab/object_oriented_programming.html) (OOP), and debugging features.[7]

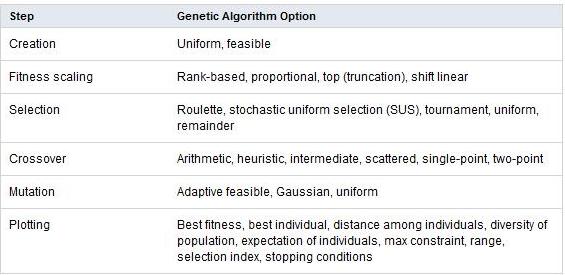
# 6.2.3 Global Optimization Toolbox

The MATLAB language supports the vector and matrix operations that are fundamental to engineering and scientific problems. It enables fast development and execution. With the MATLAB language, you can program and develop algorithms faster than with traditional languages because you do not need to perform low-level administrative tasks, such as declaring variables, specifying data types, and allocating memory. In many cases, MATLAB eliminates the need for ‘for’ loops. As a result, one line of MATLAB code can often replace several lines of C or C++ code.[7]

At the same time, MATLAB provides all the features of a traditional programming language, including arithmetic operators, flow control, data structures, data types, [object-oriented programming](http://www.mathworks.in/products/matlab/object_oriented_programming.html) (OOP), and debugging features.

## 6.2.4 Genetic Algorithm Solver

The [genetic algorithm](http://www.mathworks.in/discovery/genetic-algorithm.html) solves optimization problems by mimicking the principles of biological evolution, repeatedly modifying a population of individual points using rules modeled on gene combinations in biological reproduction. Due to its random nature, the genetic algorithm improves your chances of finding a global solution. It enables the user to solve unconstrained, bound-constrained, and general optimization problems, and it does not require the functions to be differentiable or continuous.[7]



Global Optimization Toolbox also lets you specify:

* Population size
* Number of elite children
* Crossover fraction
* Migration among subpopulations (using ring topology)
* Bounds, linear, and nonlinear constraints for an optimization problem

The following figure shows empty genetic algorithm toolbox:

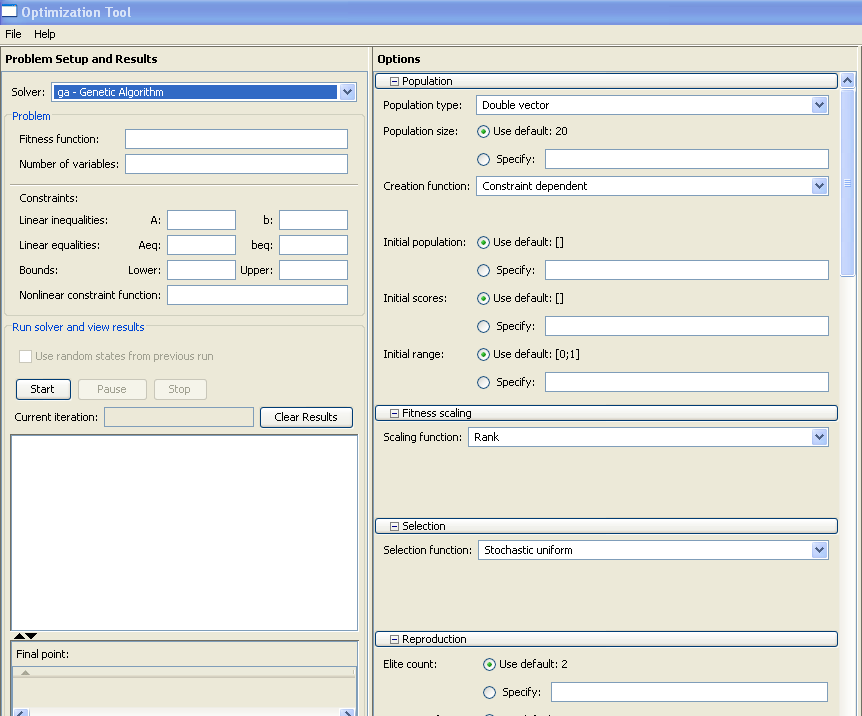


Fig 6.2 ( MATLAB Genetic toolbox )

The user can customize these algorithm options by providing user-defined functions and represent the problem in a variety of data formats, for example by defining variables that are integers, mixed integers, categorical, or complex.The user can base the stopping criteria for the algorithm on time, stalling, fitness limit, or number of generations. And you can vectorize your fitness function to improve execution speed or execute the objective and constraint functions in parallel (using Parallel Computing Toolbox).

**CHAPTER – 7**

**AREA OPTIMIZATION**

**7.1 Introduction to work**

Every aspect of our life may have a margin for optimization. every situation that confront us may not be ideal and may require some changes. For example In the morning we try to maximize our time to sleep, In the market we try to maximize our purchase with the minimum amount of money. There is a scope of optimization in every daily life problem so as to maximize the results by utilizing our limited resources. Optimization may be applied manually so as to solve to daily life problems but we may face problem in solving complex technical issues manually. Automated techniques such as ant colony, simulated annealing, bacterial foraging and lagrangian multiplication have already being developed and are being implemented to solve complex optimization problems.

Genetic algorithm is one such technique that is based on the natural process of evolution. Darwin studied that a species may adapt itself one step closer to the environment with every upcoming generation. Only the fittest member of a particular generation will survive and reproduce to pass their characteristics to the next generation. While using genetic algorithm to solve complex search problems an initial population of chromosomes may be considered, each chromosome has a different fitness value. Only a few having the required fitness value are selected to act as parents. Two parents may cross among themselves to produce an offspring. All the offspring together form the new population.

Here we are using the genetic algorithm to optimize the area of the VLSI circuits. As we know that area on the whole is dependent on the size of the components inside the circuits like transistors size (in terms of width and length), wire length or routing etc. The circuit under consideration has been developed on .18um technology using CMOS. When we vary the width and length of the CMOS transistors the area of the circuit starts changing accordingly. Reducing the width and length may give birth to some additional problems such as increase delay and it will decrease the speed of the device. There is a tradeoff between the W/L ratio and the delay. If we increase the W/L to reduce the delay or to increase the speed of the device then the area of gate, source and drain will increase consequently; moreover the parasitic capacitance of the circuit will increase. Here the genetic algorithm is used to find the appropriate width and length of the transistors.

The circuit under consideration is a simple one bit cmos full adder. It has three input (A, B & Cin ) and two output sum and carry\_out. In the three inputs two inputs A & B are the input variables (addend bits) and the third input Cin is the carry\_in. It performs binary addition between three inputs and generates corresponding sum and carry. Here we have used transistor level one bit full adder. For translating the gate level design into a transistor level circuit description, we note that both the sum and carry\_out functions are represented by nested AND-OR-NOR structure. Each such combined structure can be realized in CMOS as follows: the AND terms are implemented by series connected nMOS transistors, and the OR terms are implemented by parallel connected nMOS transistors. The input variables are applied to the gates of the nMOS (and the complimentary pMOS) transistors. Thus, the nMOS net may consist of nested series- parallel connections of nMOS part of a complex CMOS logic gate is realized, the corresponding pMOS net, which is connected between the output node and the ground. Then the corresponding pMOS, which is connected between the output node and the power supply, is obtained as the dual network of the nMOS net. The circuit contains a total of 14 nMOS and 14 pMOS transistors, together with the two CMOS inverters which are used to generate the outputs.[3]

**7.2 Procedure Followed**

Circuit schematic was implemented using cadence virtuoso-4 the full adder circuit has 28 transistors of which 14 are pmos and 14 are nmos. By default the value of length and width are 2u m and 180n m.

the width of the pmos was varied and corresponding output levels of sum and carry were recorded for logic one. A range of minimum and maximum values was found (values for which logic ‘1’ had acceptable voltage level)

After finding the range of the width of the pmos circuit then we varied the length of the nmos transistors and find the minimum and maximum value of the length for which the output level for sum and carry are evaluated as logic one.

Once a suitable range of values was found for both width and length, the corresponding values of the TPHL and TPLH were found on the extreme values of this range. By using these (TPHL and TPLH) we calculated the delay of the overall circuit. The delays thus found correspond to the widths and lengths for which the logic ‘1’ has acceptable voltage levels.

The values of delay found in the previous step are used as a constraint for the genetic algorithm to search a suitable solution for our problem. The genetic algorithm toolbox of MATLAB is utilized to implement genetic algorithm for area optimization. The genetic algorithm toolbox basically requires three inputs from the user viz. fitness function, constraint, lower and upper bounds of the variables.

The relation between width, length and delay acts as the constraint, the extreme values of width and length found above define the upper and lower bound of the variables and area is our fitness function, which is to be minimized.

So as to calculate the area covered by transistors, we have multiplied the number of transistors with the product of width and length of corresponding pmos and nmos.

General formula is

A= 14\*(W1\*L1) + 14\*(W1\*L1);

Formula used in genetic algorithm

function a =area(x)

a(1)=14\*x(1)\*x(2);

a(2)=14\*x(3)\*x(4);

a=(a(1)+a(2));

where x(1) is width of the pmos

x(2) is length of the pmos

x(3) is width of the pmos

x(4) is length of the pmos

To calculate the delay of full adder we used this approximated formula, which is applicable for the small geometry devices.

Tphl=(CL\*(Vdd/2)) / k Wn (Vdd - VT) …..for nmos

Tplh=(CL\*(Vdd/2)) / k Wp (Vdd - VT) …..for pmos

T=(Tphl + Tplh) / 2

For genetic algorithm we used the formula

function Tphl =Timeperiod(x)

C= 1;

Vdd=3.3;

Vtn=0.6;

Tphl=((C\*(Vdd/2))/(Vdd-Vtn))\*(x(3)/x(4));

function Tphl =Timeperiod(x)

C= 1;

Vdd=3.3;

Vtn=0.7;

Tphl=((C\*(Vdd/2))/(Vdd-Vtp))\*(x(2)/x(1));

Where value of c is in pico farad.

**CHAPTER-8**

**SIMULATION RESULTS**

**8.1 one bit CMOS Full Adder**

The schematic of CMOS full adder was implemented using cadence design environment. Fig 8.1 shows a basic CMOS full adder that was implemented using the cadence. The Full adder circuit utilized 28 transistors. Fourteen of them are NMOS and the rest of them PMOS so as to complete the CMOS inverter structure. Our Full adder circuit is shown in the adjoining figure. This circuit is working at a supply voltage of 3.3V. The three inputs of circuit have been given a pulsating input. In order compensate the overall capacitance, Sum and Carry outputs have been connected with load capacitors. The logic table of a full adder is shown fig 3.2 .

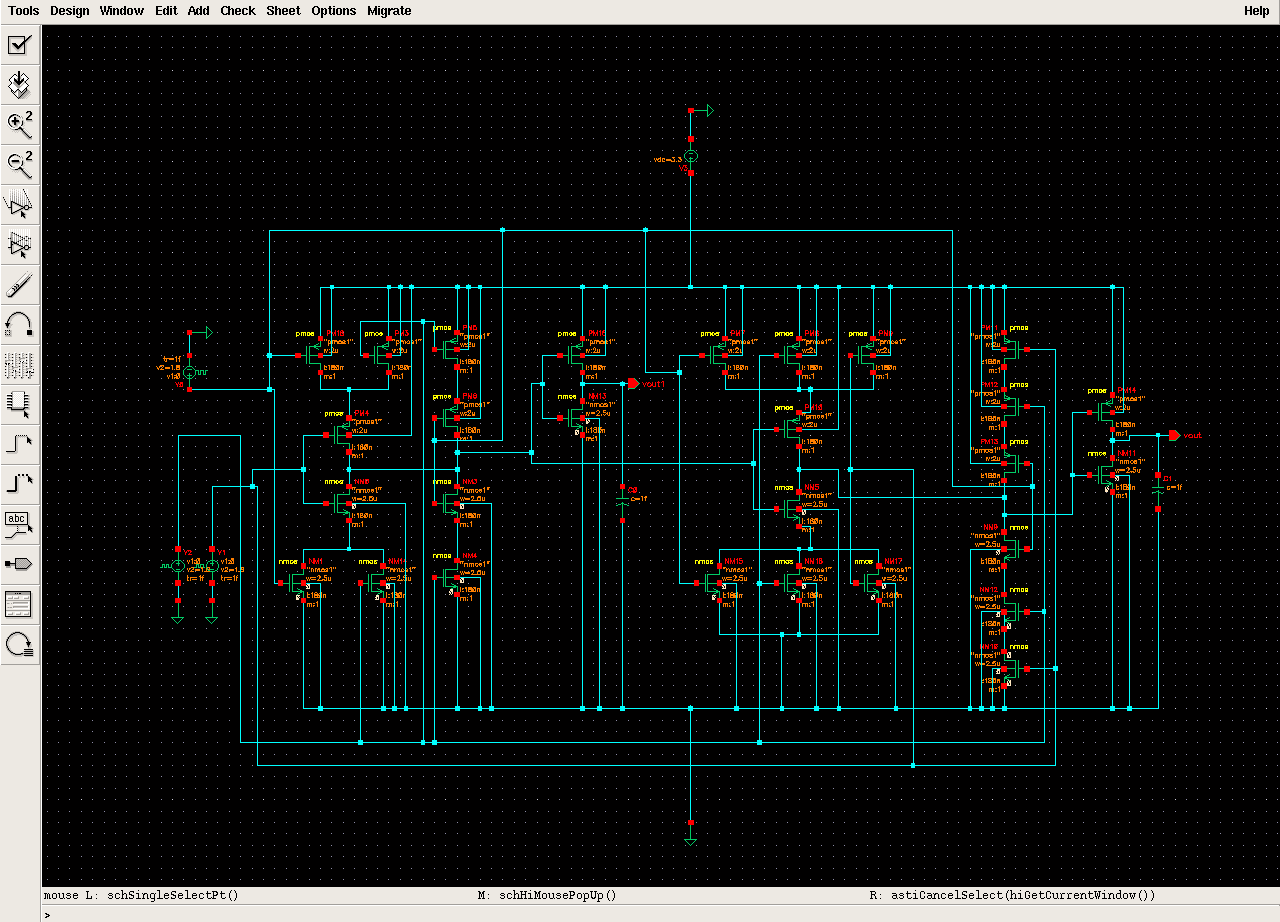


Fig 8.1 (1-Bit CMOS Full Adder Implemented On Cadence Virtuso-4 )

The Fig 8.2 form depicts the input and output wave when the load capacitance in negligible (1f F), The input A has a pulse width of 5ns and a time period of 10ns, The input B has a pulse width of 10ns and a time period of 20ns, The input C has a pulse width of 25ns and a time period of 40ns. The pulsating input given to A, B & C cover almost all the input combinations. The results prove that the full adder is working as it should.

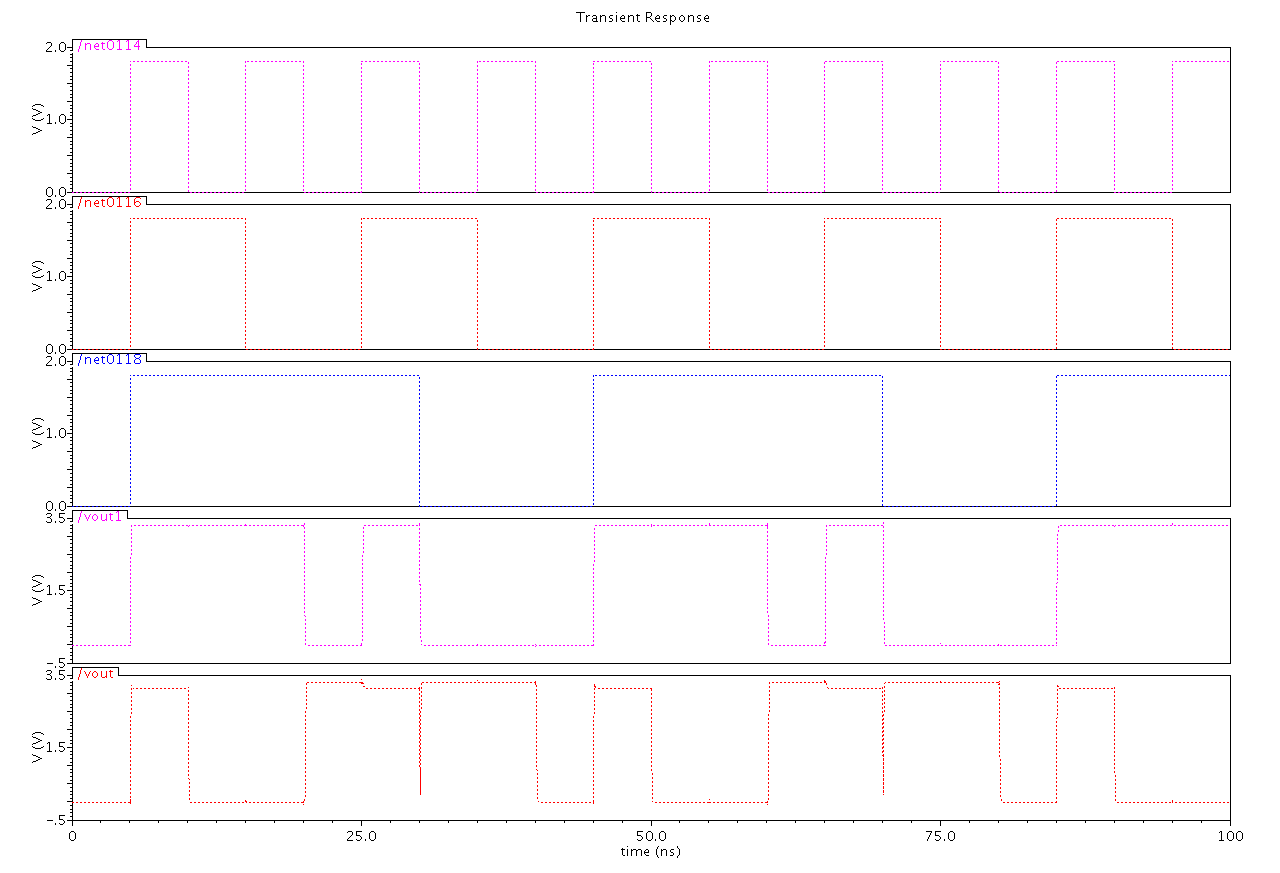


Fig 8.2 (Waveforms of CMOS Full Adder)

**8.2 Output waveform, when w=2.24u m and l= 390n m**

When we increase the width of the PMOS transistors from 2u m to 2.24u m the w/l ratio also increases. As we already know that delay and w/l ratio are inversely proportional to each other. The delay to be calculated in this case is TPLH for sum and carry are .0546n sec and .0806n sec respectively and TPHL for sum and carry are .0765n sec and .1642n sec respectively Hence the delay in this case decreases. The change in length has very small or negligible. As we increase the width of the PMOS transistors above the 2.24u m the output level falls down below 50% of the required output level.

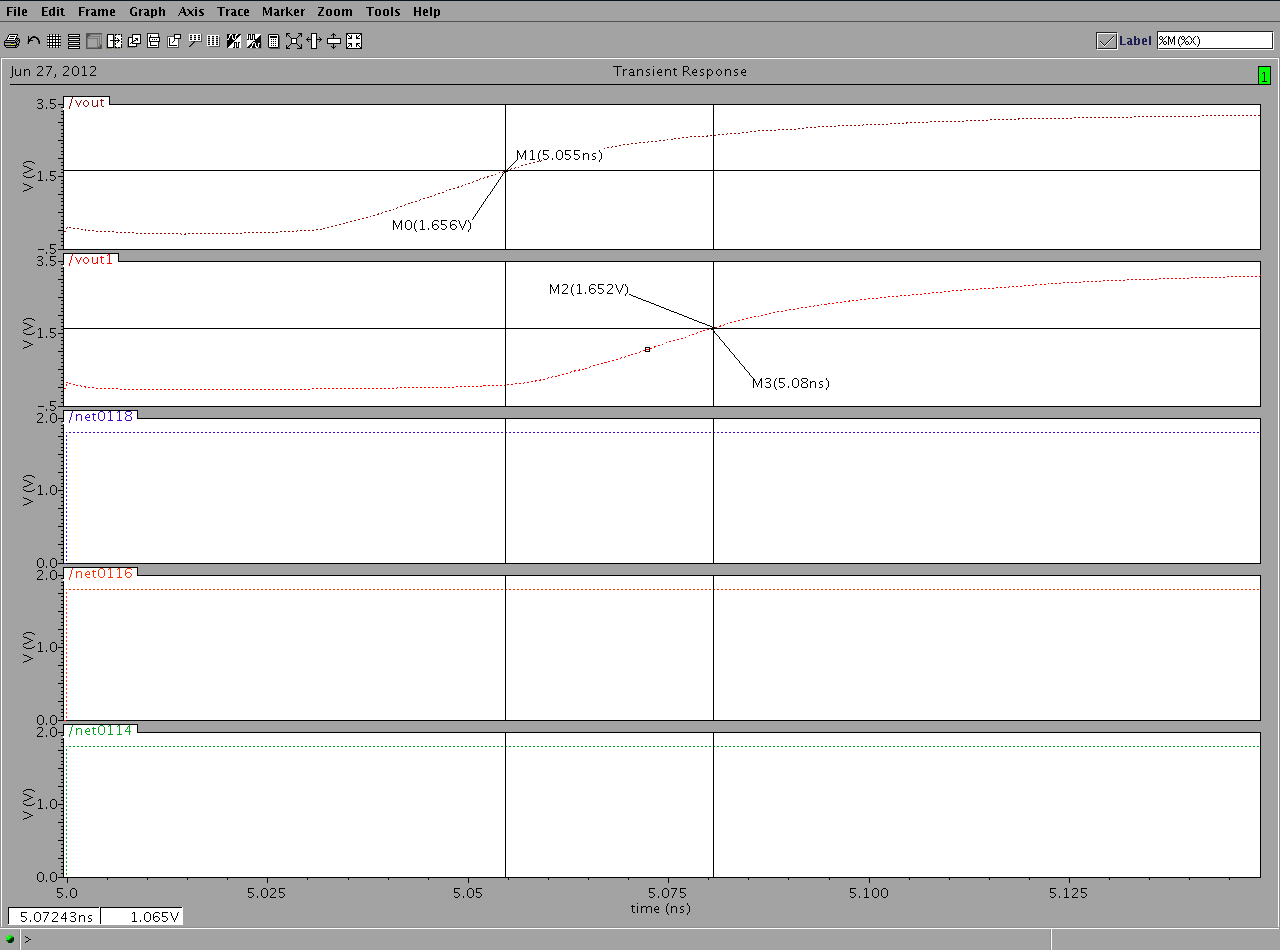


Fig 8.3 (Tplh, when w=2.24u m and l=390n m of PMOS for sum and carry)

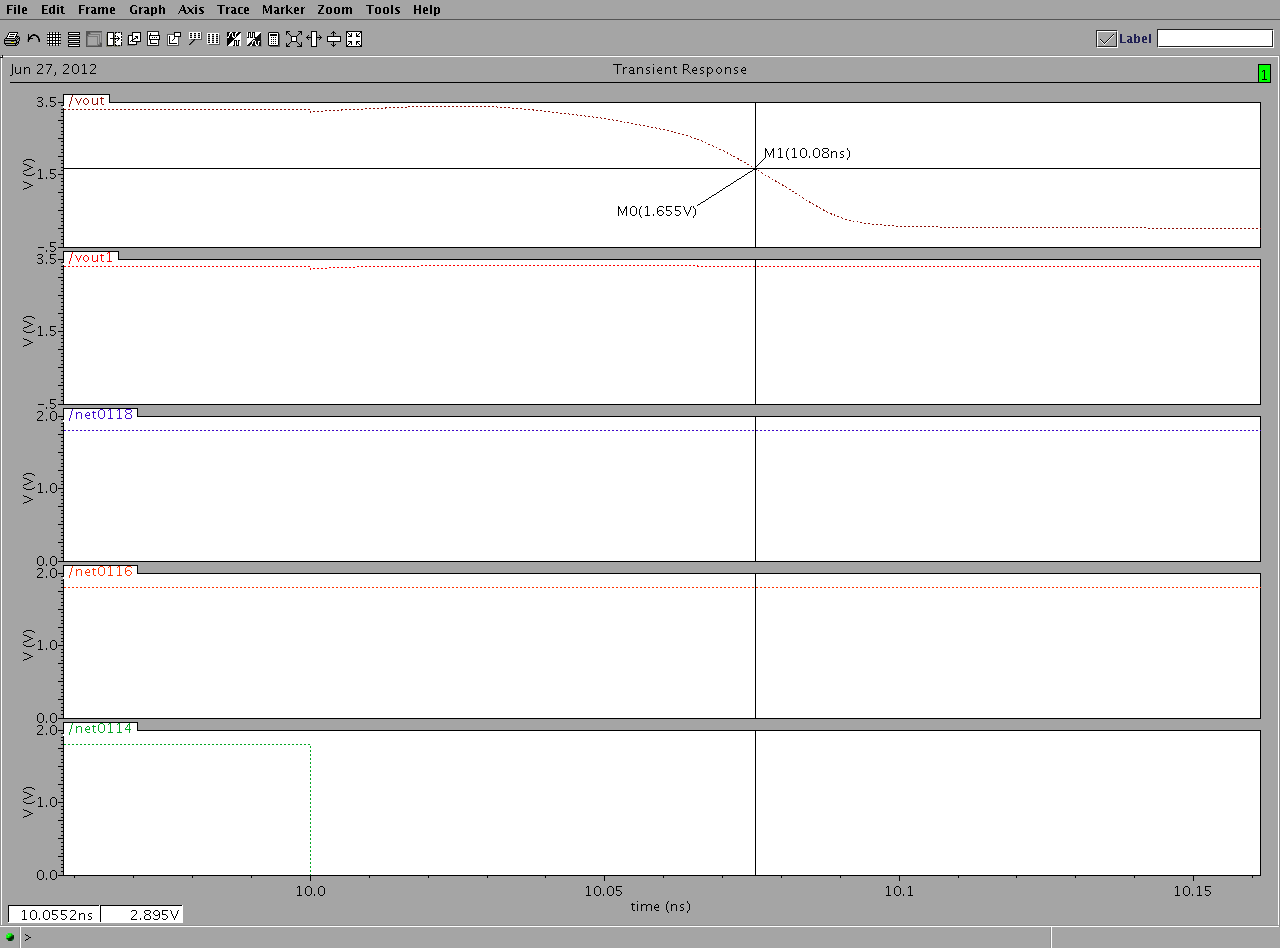


Fig 8.4 (Tphl, when w=2.24u m and l=390n m of PMOS for sum)

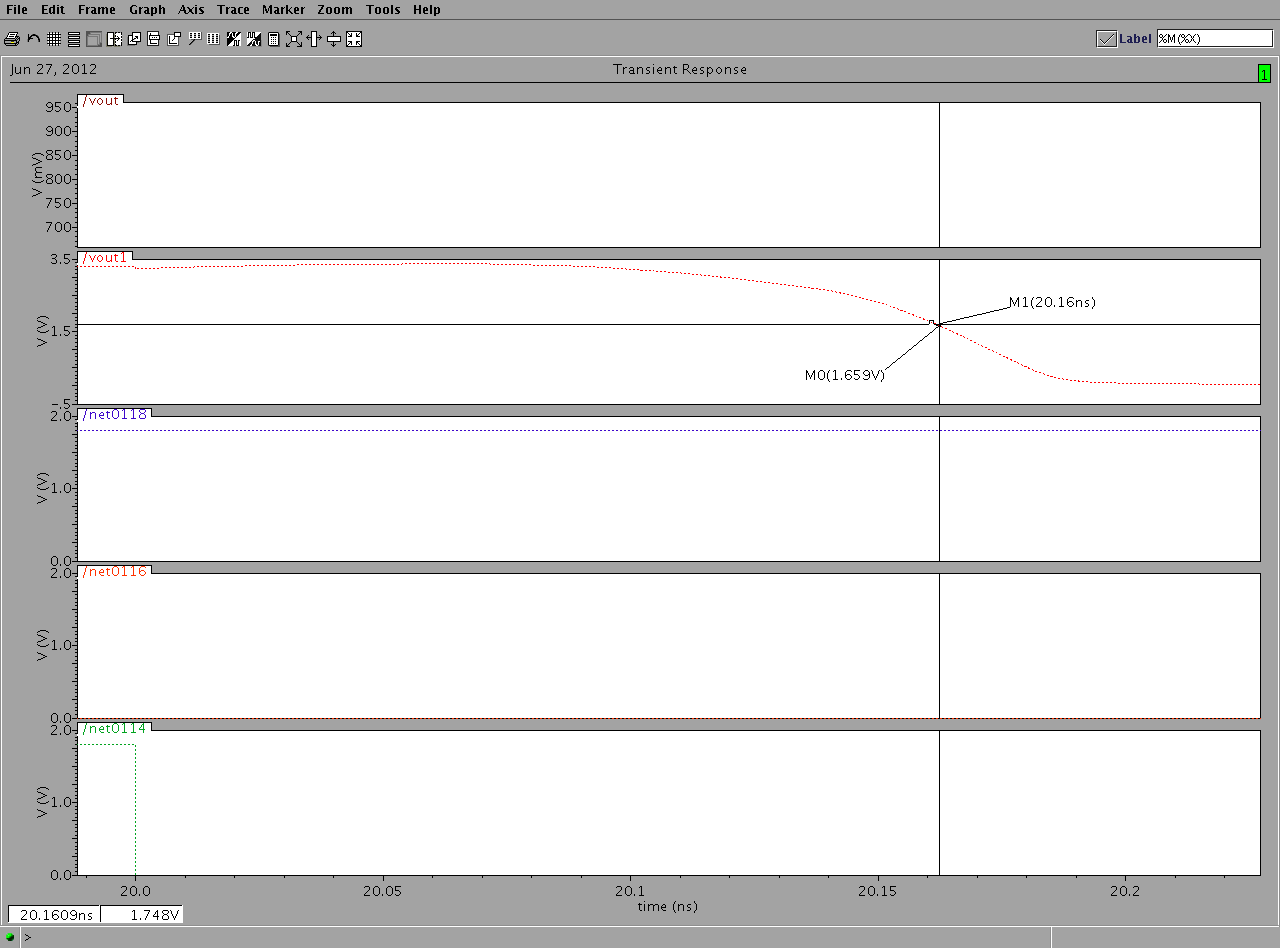


Fig 8.5 (Tphl, when w=2.24u m and l=390n m of PMOS for carry)

**8.3 Output waveform, when w=2u m and l= 180n m**

PMOS transistors is switched on when the input is zero. As a result it is responsible for the output to go from low to high. The width and length for figures 8.6, 8.7 & 8.8 is 2u m and 180n m respectively. The delay to be calculated in this case is TPLH for sum and carry are .0386n sec and .0535n sec respectively and TPHL for sum and carry are .0255n sec and .0642n sec respectively. It pulls output to high level. The w/l ratio comes out to be 11.11. A width of 2u m and the length of 180n m correspond to the minimum length and width which can be fabricated by using .18u m technology. Here the area occupied by the structure is rather small but the delay is somewhat higher than other dimensions.

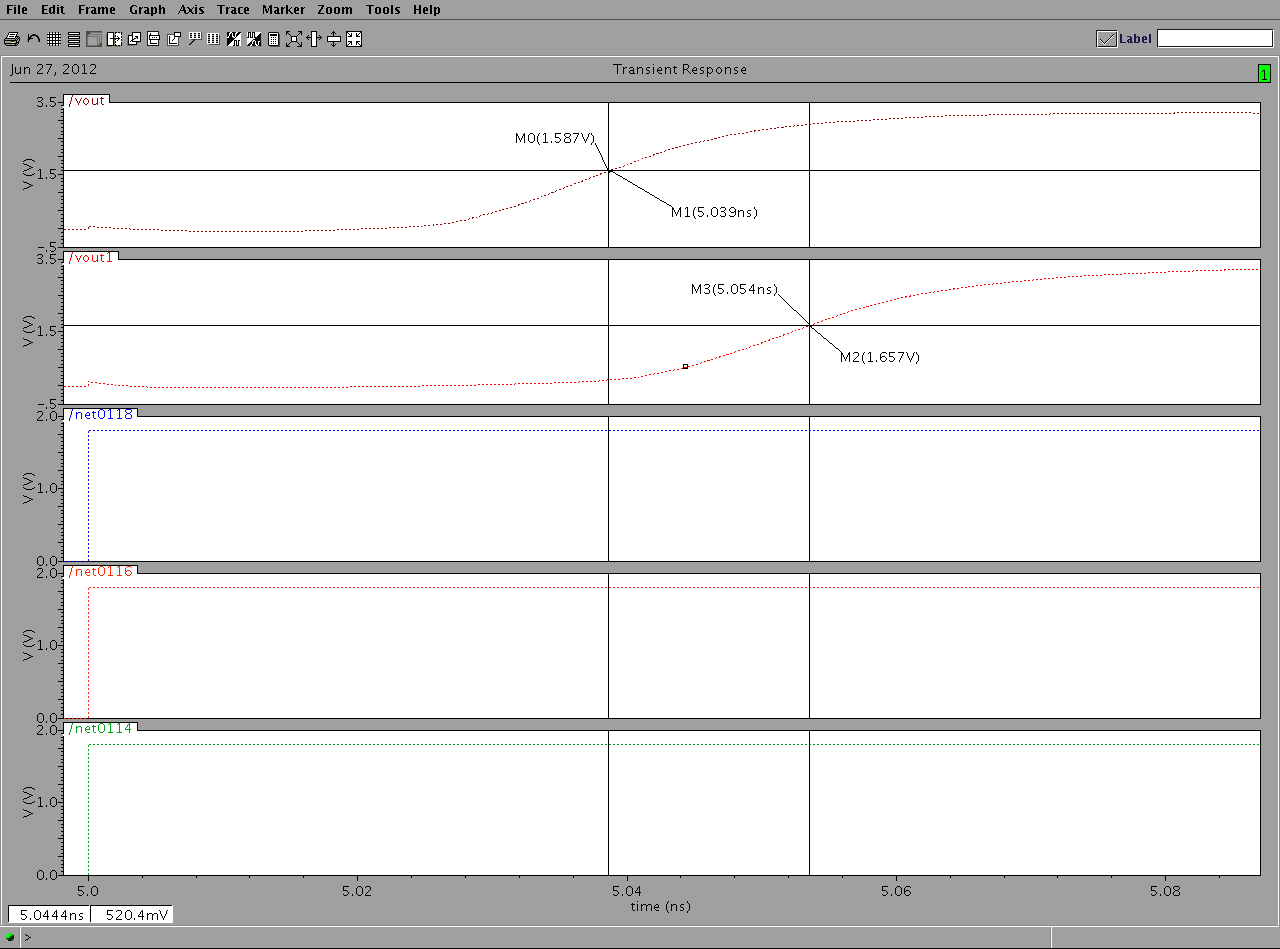


Fig 8.6 (Tplh, when w=2u m and l=180n m of PMOS for sum and carry)

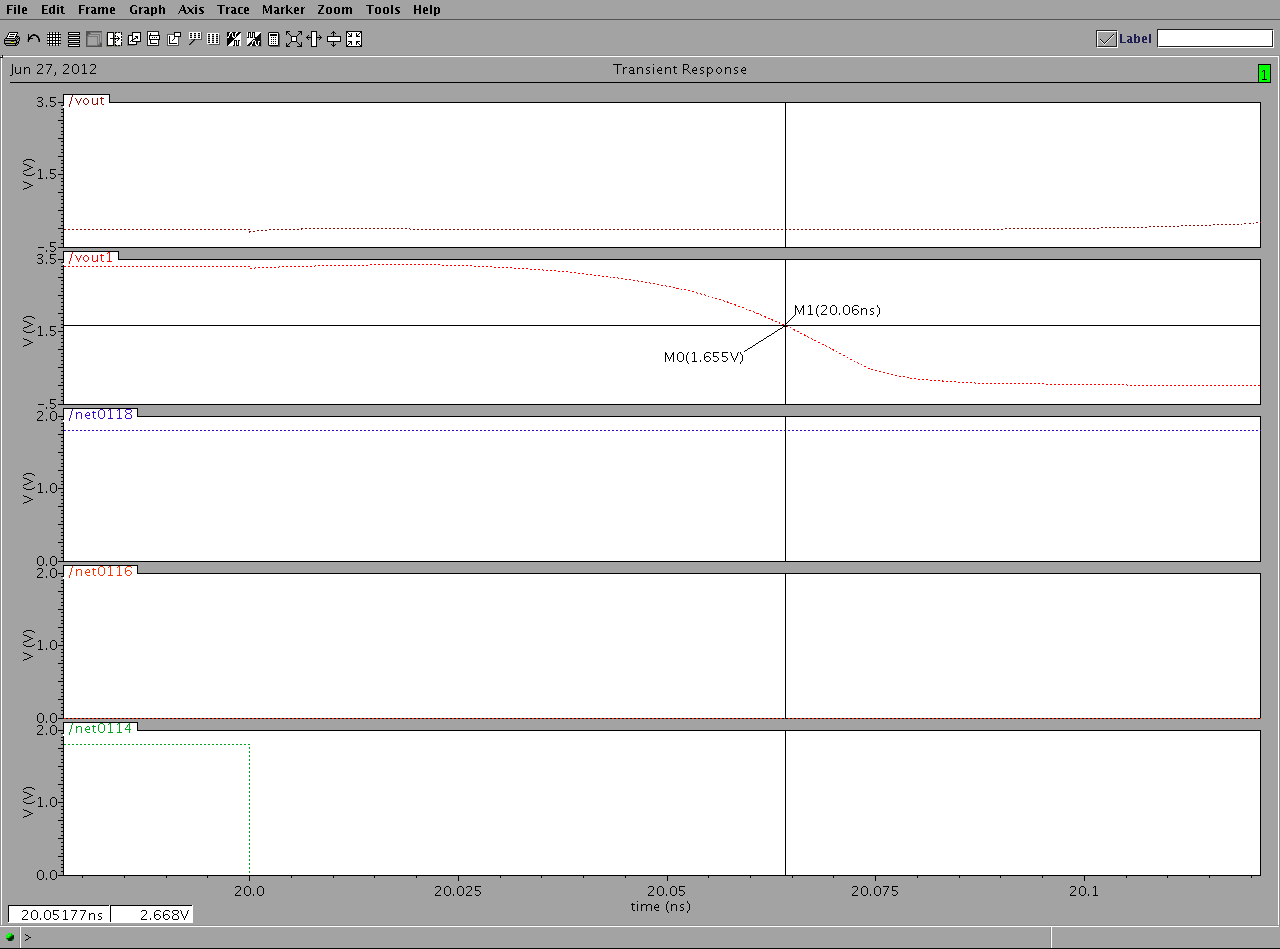
****

Fig 8.7 (Tphl, when w=2u m and l=180n m of PMOS for carry)

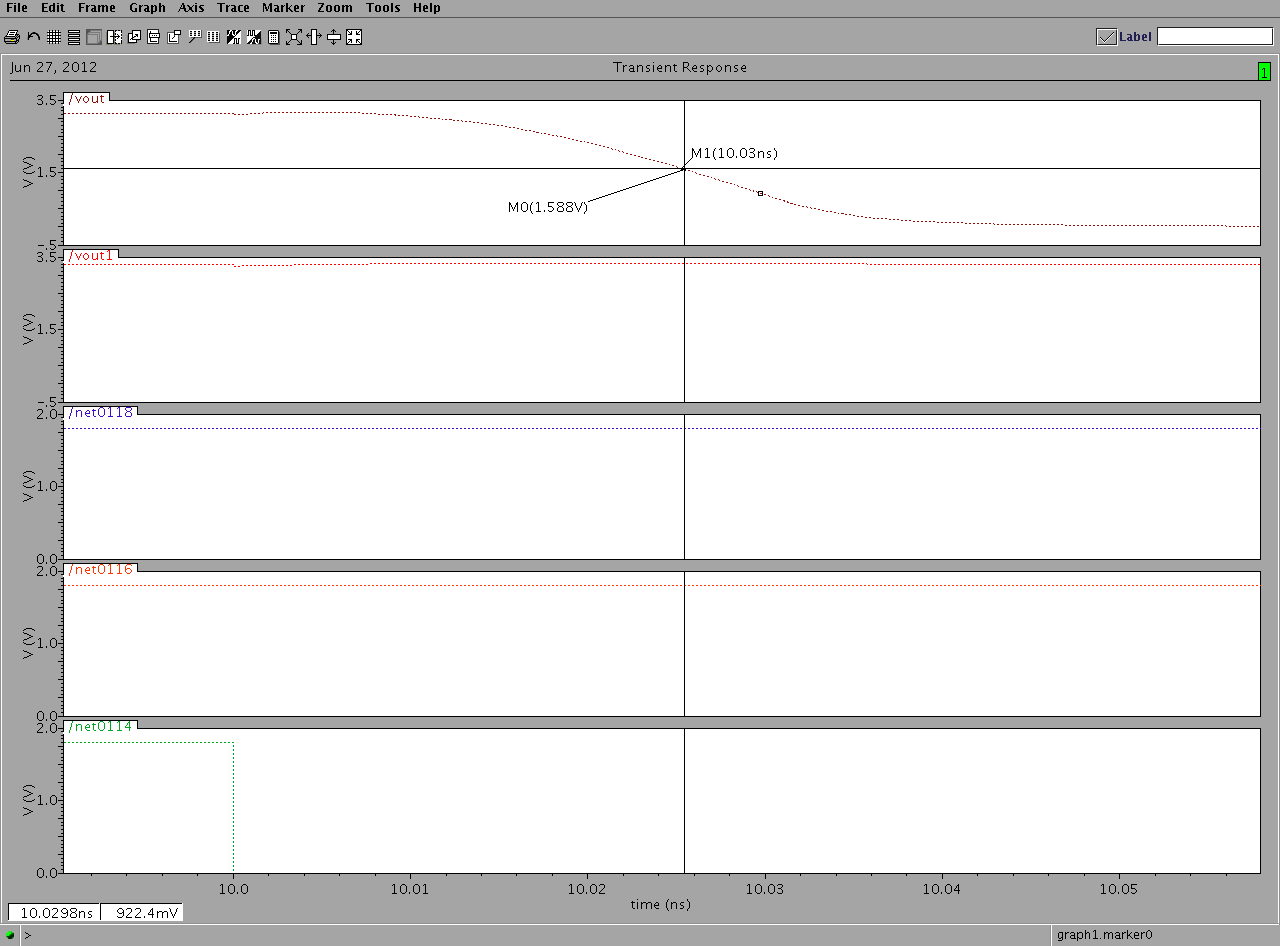


Fig 8.8 (Tphl, when w=2u m and l=180n m of PMOS for sum)

**8.4 Output waveform, when w=2.23u and l=180nm**

Figure no 8.9 show the delay at sum and carry terminal when width and length of nmos transistors is 2.23u m and 180n m respectively. The TPHL for the sum is .0153 n sec and for that of carry is .0604n sec and the TPLH for the sum is .0341n sec and fo r carry is .0559n sec. At this we can calculate the w/l ratio to be 12.38.we know that delay is inversely proportional to w/l ratio, we expect the value of delay to increase when we try to reduce the w/l ratio in further readings. When we try to reduce the width below 2.23u m the voltage at the output falls below the 50% of the required voltage at the output our circuit might not be able to recognize it as logic ‘1’.

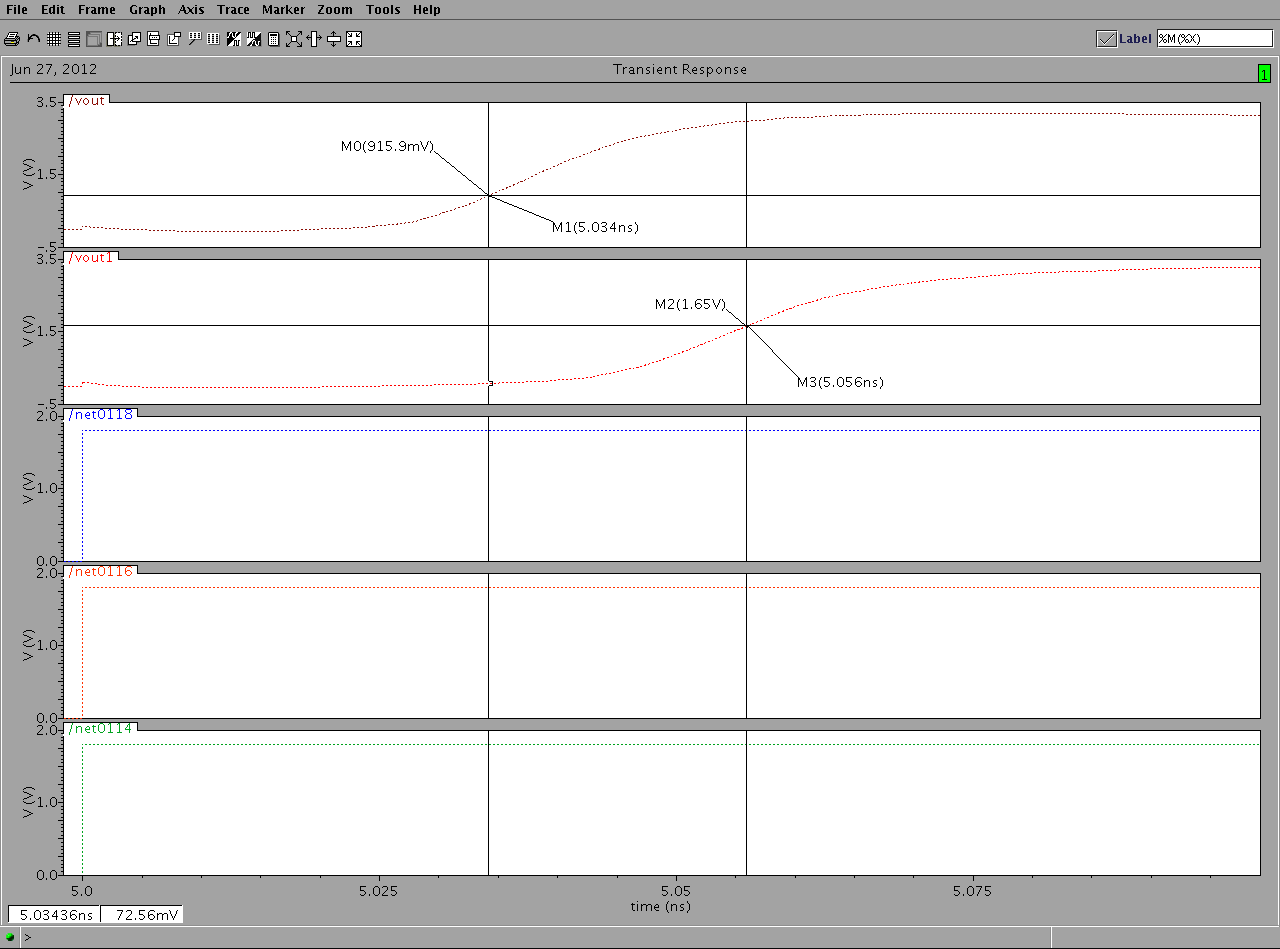


Fig 8.9 (Tplh, when w=2.23u m and l=180n m of NMOS for sum and carry)

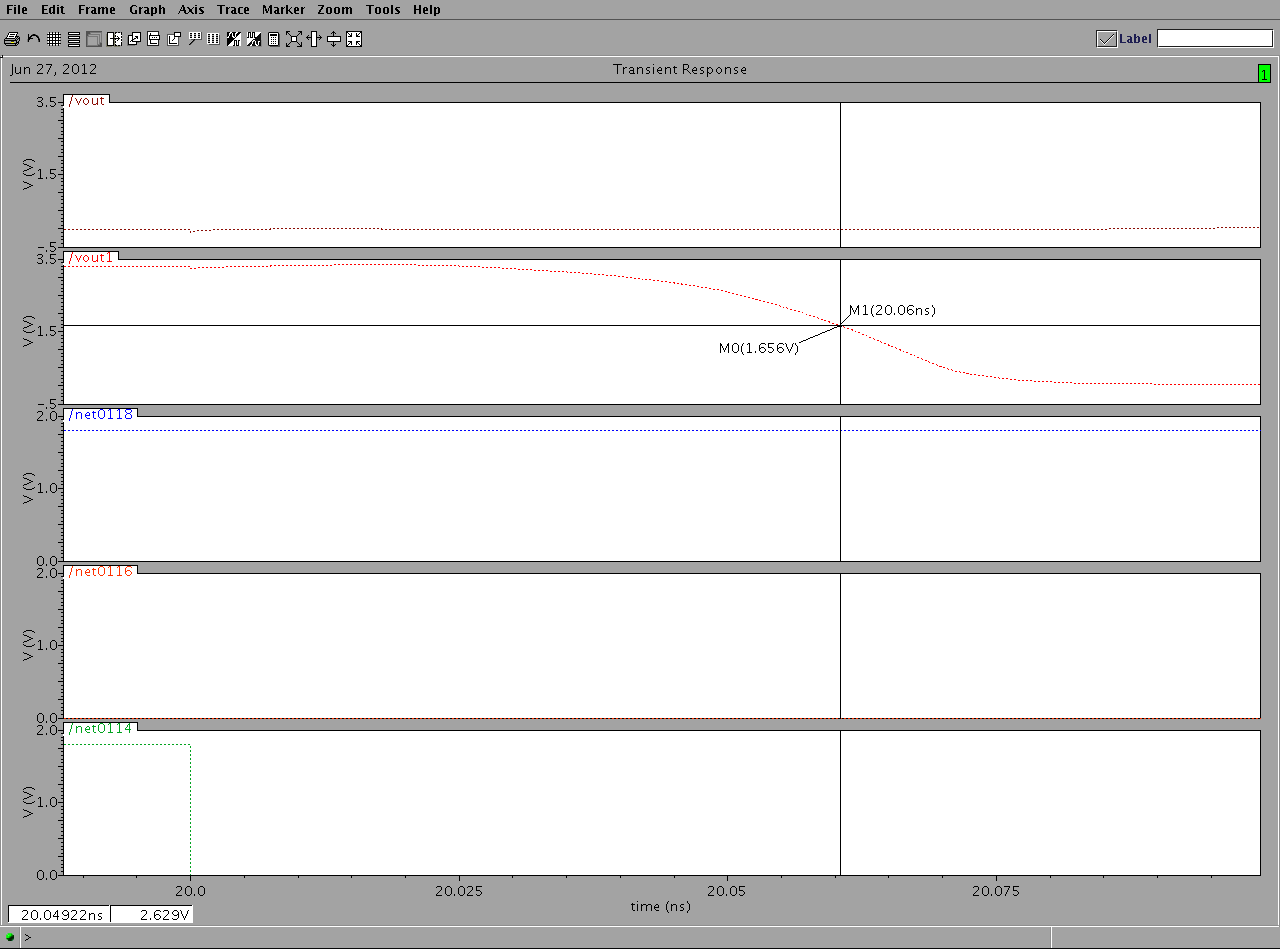


Fig 8.10 (Tphl, when w=2.23u m and l=180n m of NMOS for carry)

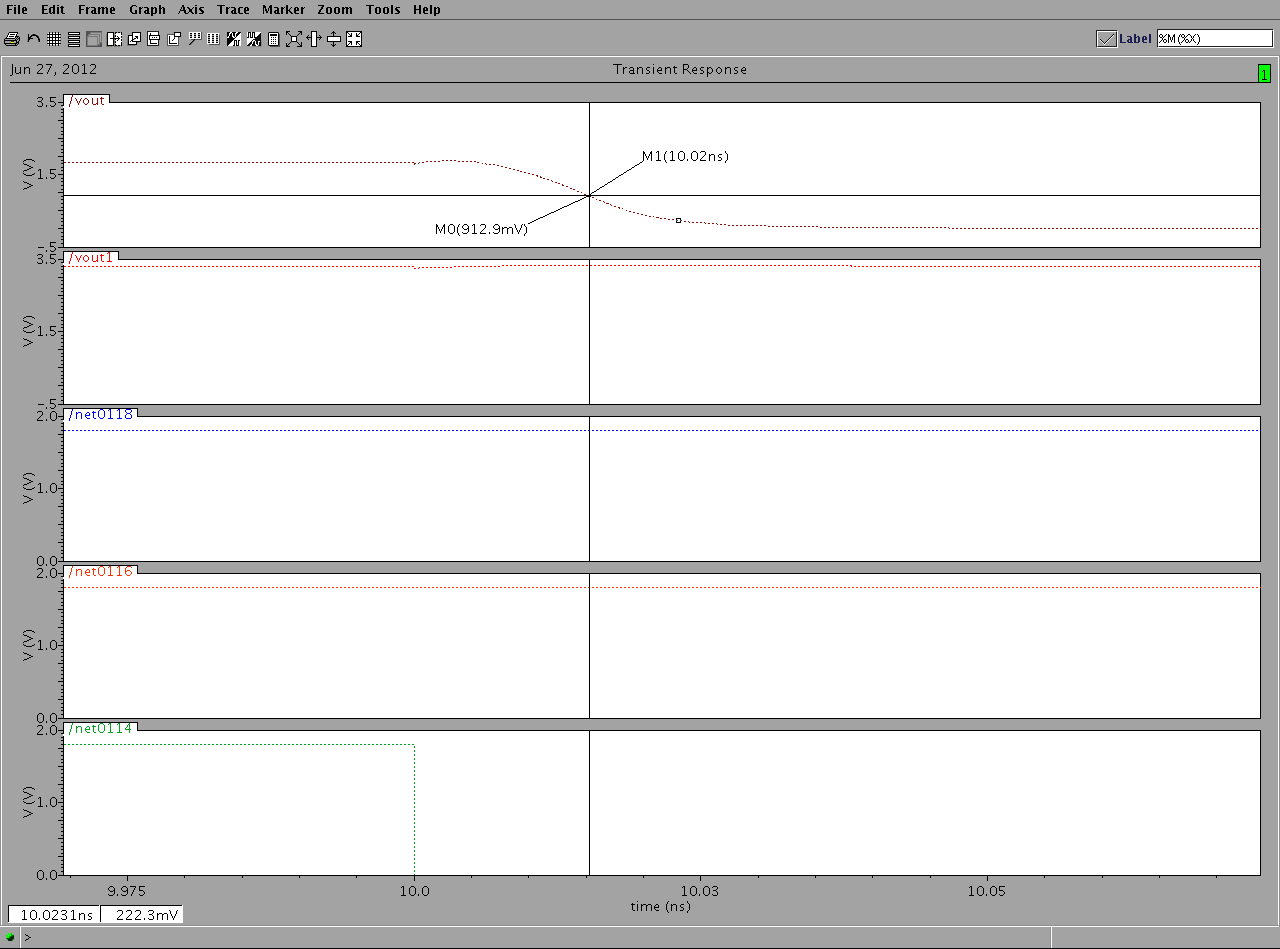


Fig 8.11 (Tphl, when w=2.23u m and l=180n m of NMOS for sum)

**8.5 Output wave form, when w=2.5u m and l=210n m.**

For figures 8.12, 8.13 & 8.14 The width is equals to w=2.5u m and length is equals to l=210n m. the w/l ratio can be calculated to be 11.90. this w/l ratio is lesser than that calculated for w=2.23u m and length is equals to l=180n m. as expected the delay is increased. The TPHL for sum is .018n sec and for carry is .05068n sec and the TPLH for the sum is .0393n sec and for carry is .05562n sec. If we increase length of nmos transistor any further, the output voltage for logic ‘1’ falls below the 50% level of the required output. This situation is unacceptable and the dimensions can’t be altered further. The delay corresponding to this is taken to be the upper bound to be used in the genetic algorithm.

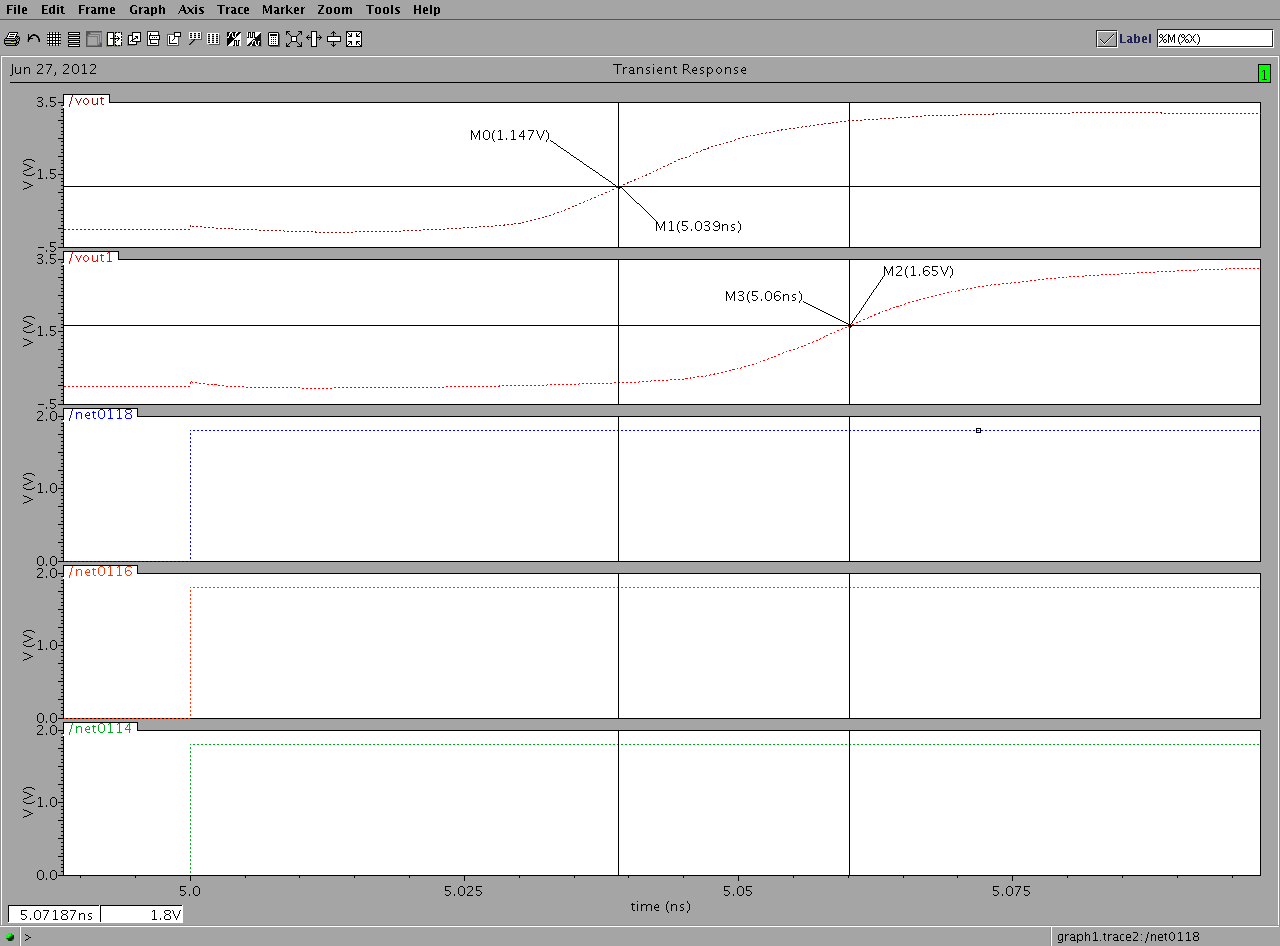


Fig 8.12 (Tplh, when w=2.5u m and l=210n m of NMOS for sum and carry)

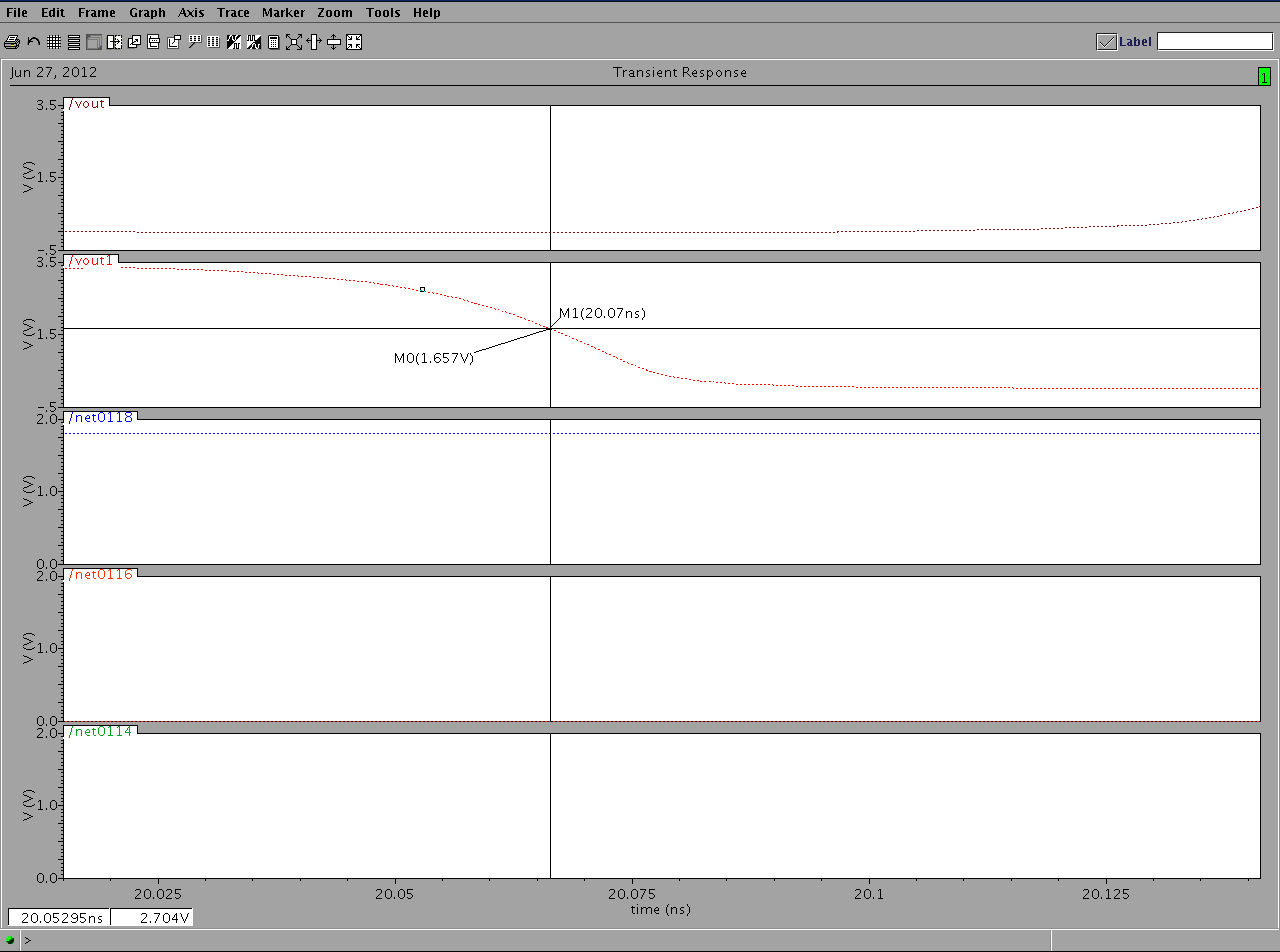


Fig 8.13 (Tphl, when w=2.5u m and l=210n m of NMOS for carry)

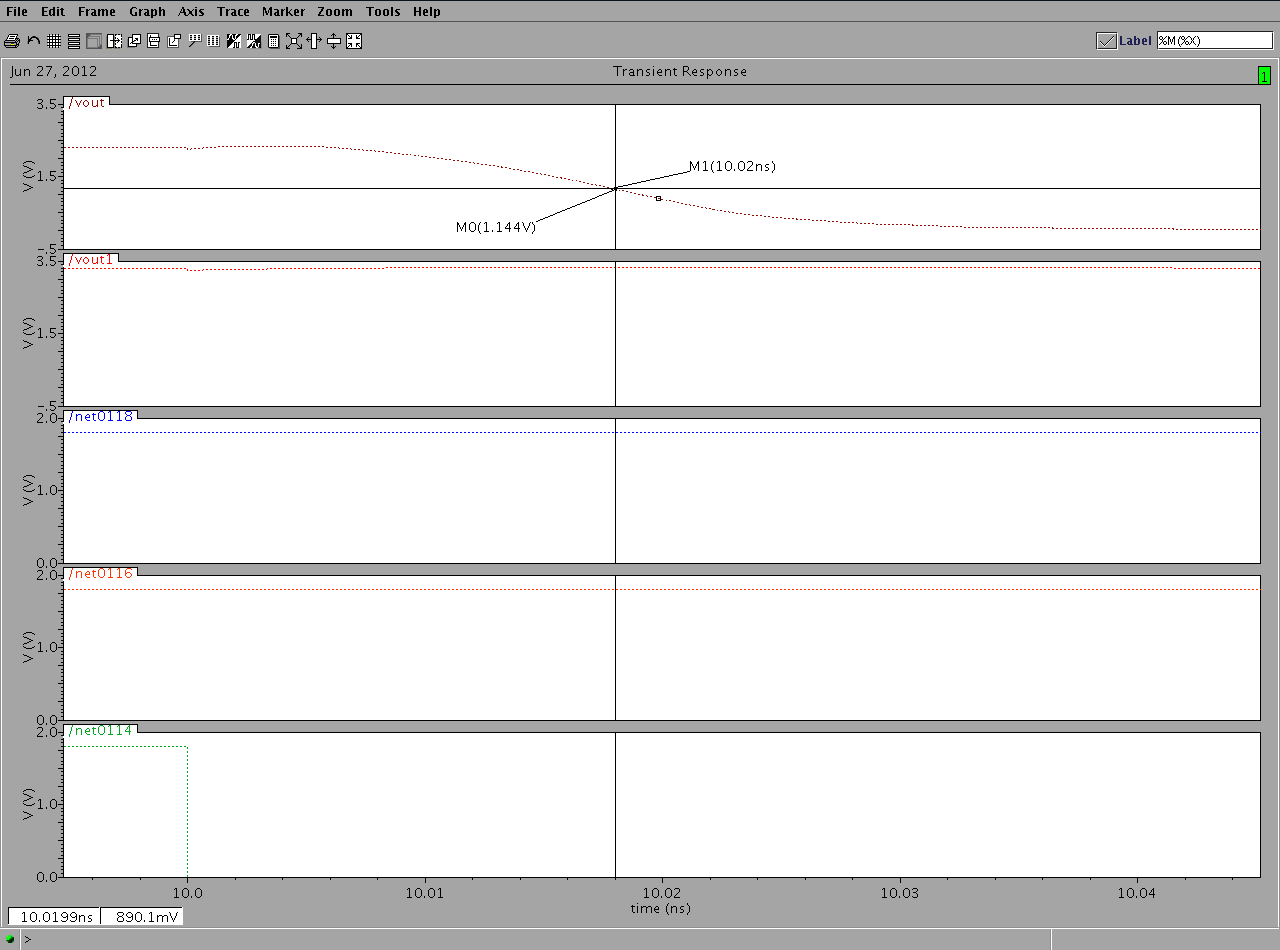


Fig 8.14 (Tphl, when w=2.5u m and l=210n m of NMOS for sum)

**8.6 Parametric analysis for PMOS, when w=2u m to 2.24u m and l=180n m and w=2u m to 2.24 and l= 180n m to 390n m**

The figure 8.15 depicts the parametric analysis for width. In this we keep the length constant at 180n m and vary the width in the range of 2u m to 2.24u m. The results show that when we increases the value of width(w) above 2.24 the output gets disturbed and the voltage level falls below the 50% of voltage mark. As the width is varied from 2u m to 2.24u m we see that the voltage level goes on decreasing at every step. The value of w/l ratio is 11.11 when the w=2u m and l=180n m and the value of w/l ratio is 12.44 when w=2.24u m and l=180n m.

Figure 8.16 shows the parametric analysis of length. In this we keep the width vary from 2u m to 2.24u m and vary the length in the range of 180n m to 390n m. The results shows that when we increase the value of width and length(l) above 2u m and 180n m respectively, the output gets distorted and the voltage level falls below the 50% of the required voltage level. As the length is varied from 180n m to 390n m and the width is varied from 2u m to 2.24u m the output level starts decreasing at every step. The value of w/l ratio is 11.11 when the w=2u m and l=180n m and the value of w/l ratio is 5.74 when the w=2.24u m and l=390n m.

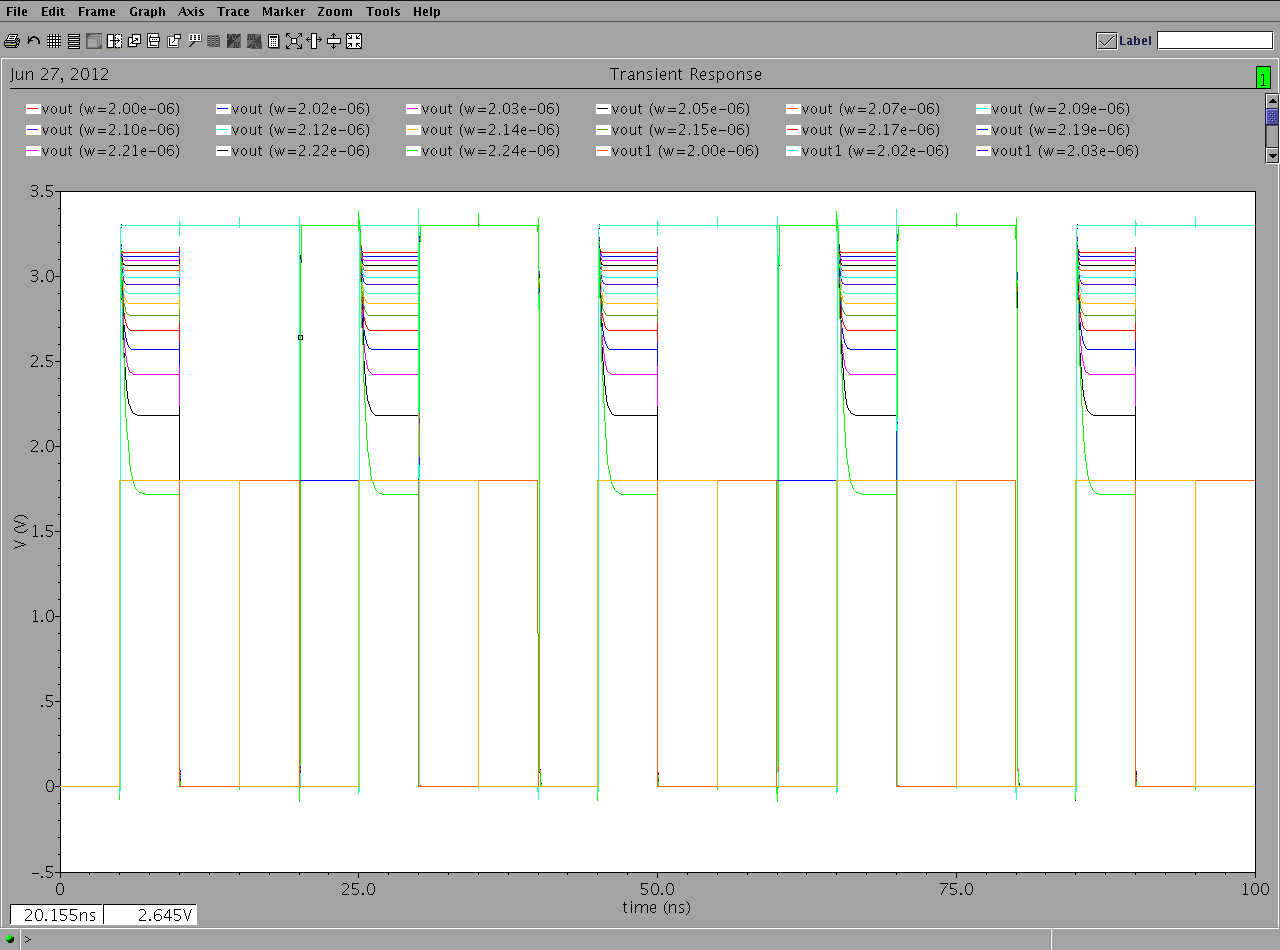


Fig 8.15 (parametric analysis, when w=2u m to 2.24u m and l=180n m of PMOS)

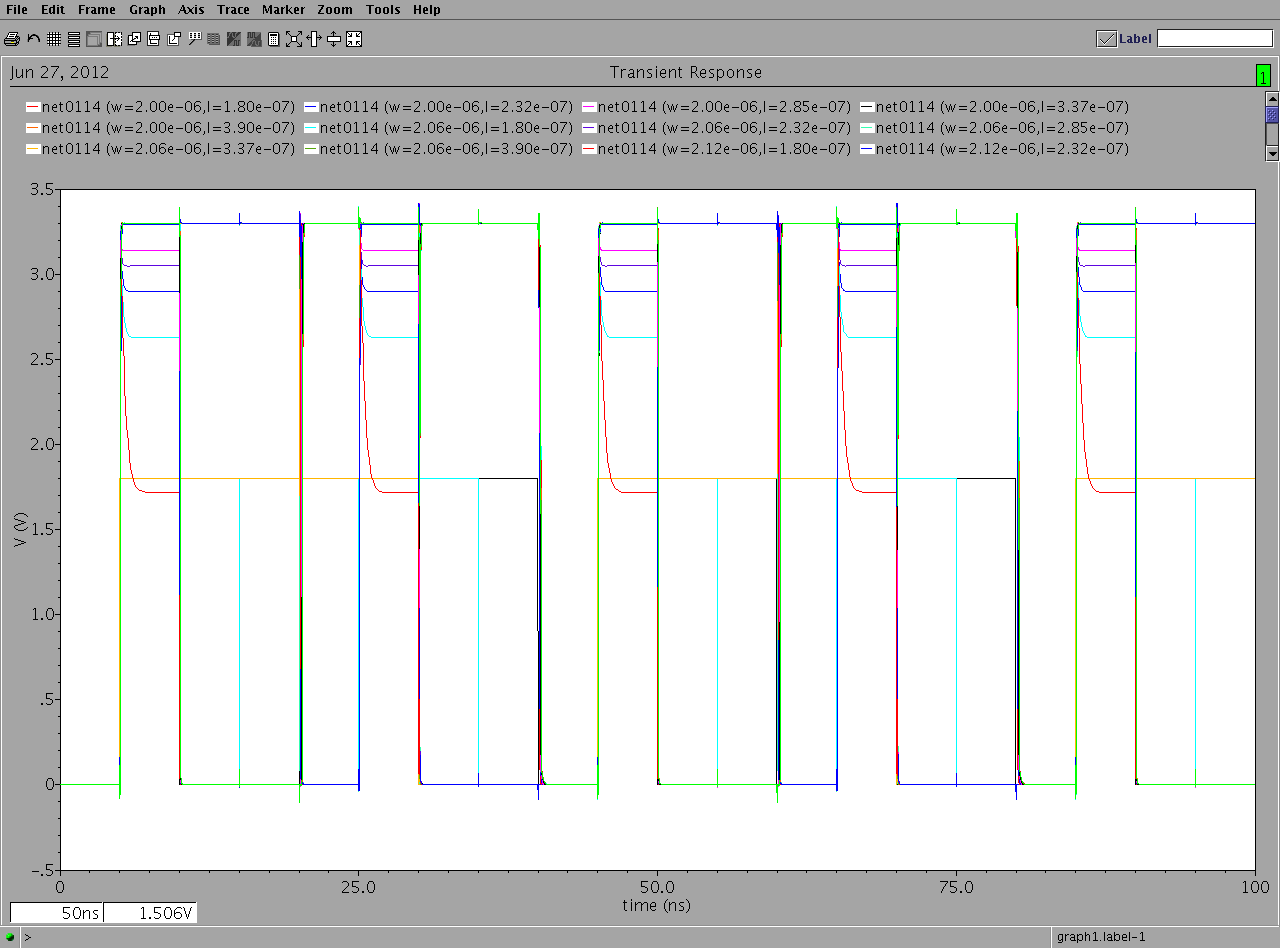


Fig 8.16 (parametric analysis, when w=2u m to 2.24u m and l=180n m to 390n m of PMOS)

**8.6 Parametric analysis for NMOS, when w=2.23 to 2.5u and l=180n m and when width w=2.5 and length l=180n m to 210n m.**

The figure 8.17 depicts the parametric analysis for width. In this we keep the length constant at 180n m and vary the width in the range of 2.23u m to 2.5u m. The results show that when we reduce the value of width(w) below 2.23 the output gets disturbed and the voltage level falls below the 50% of voltage mark. As the width is varied from 2.5u m to 2.23u m we see that the voltage level goes on decreasing at every step. The value of w/l ratio is 12.38 when the w=2.23u m and l=180n m and the value of w/l ratio is 13.88 when w=2.5u m and l=180n m.

Figure 8.18 shows the parametric analysis of length. In this we keep the width constant at 2.5u m and vary the length in the range of 180n m to 210n m. The results shows that when we increase the value of length(l) above 180n m the output gets distorted and the voltage level falls below the 50% of the required voltage level. As the length is varied from 180n m to 210n m the output level starts decreasing at every step. The value of w/l ratio is 13.88 when the w=2.5u m and l=180n m and the value of w/l ratio is 11.90 when the w=2.5u m and l=210n m.

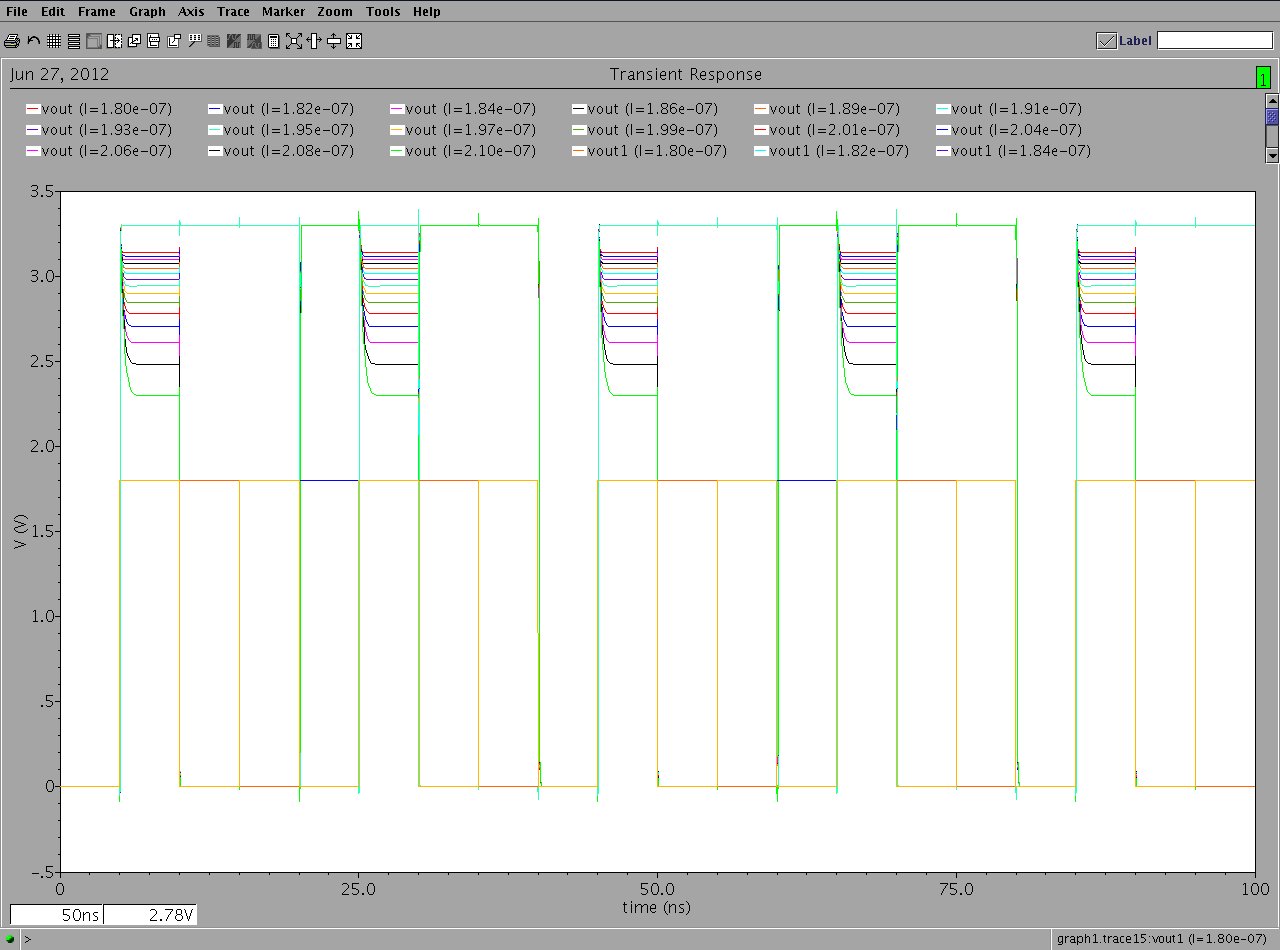
****

Fig 8.17 (parametric analysis, when w=2.5u m and l=180n m to 210n m of NMOS)

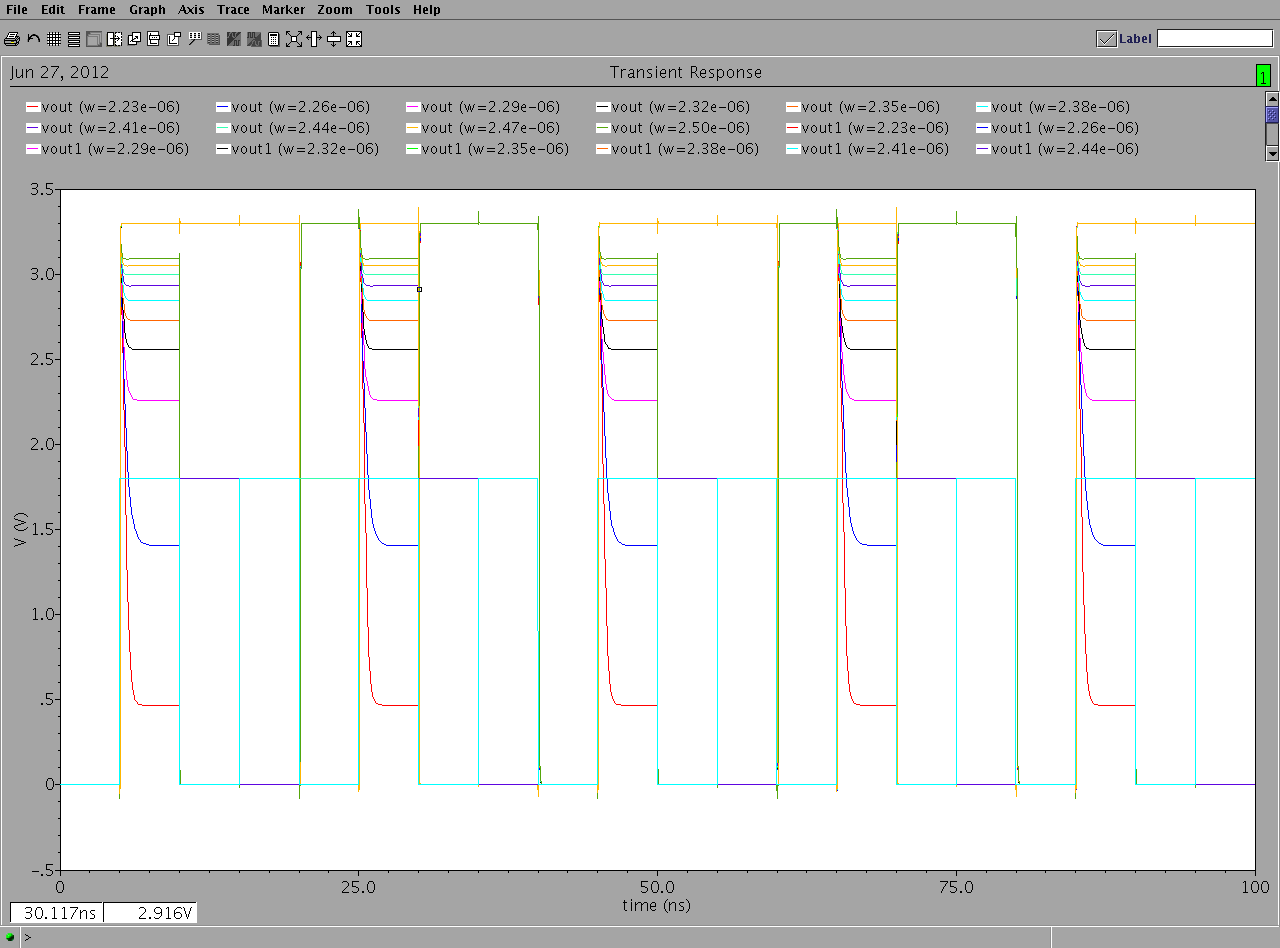


Fig 8.18 (parametric analysis, when w=2.23u m to 2.5u m and l=180n m of NMOS)

**CHAPTER-9**

**FINAL RESULTS**

**9.1 Results from Genetic Algorithm**

The layout area can be basically considered to be consisting of two parts. Area covered by the logic modules and the area occupied by interconnections between the modules. Many researchers utilized various methods to reduce the overall area. These methods may try to reduce the area covered by the logic modules by changing the components or logic blocks used to create the circuit. Some other methods try to reduce the areas used by the interconnections.

Here we have tried to reduce the area covered by logic blocks by reducing the size of the transistors and not by altering the structure of the circuit. Care has been taken to maintain acceptable voltage levels and corresponding delays. It has been ensured that the final output is in accordance with the logical working of the full adder circuit.

The adjoining figure shows the results obtained through the genetic algorithm toolbox. When the delay was taken to be the fitness function, the upper and lower values of the delay in case of pmos and nmos are used as constraint. We can see that the GA toolbox applies the genetic algorithm and calculates the suitable value of w and l, for pmos w=2.224u m and l=180n m and for nmos w=2.5u m and l=180n m. The value so found will be further utilized to find the final width and length for the completely optimized area.

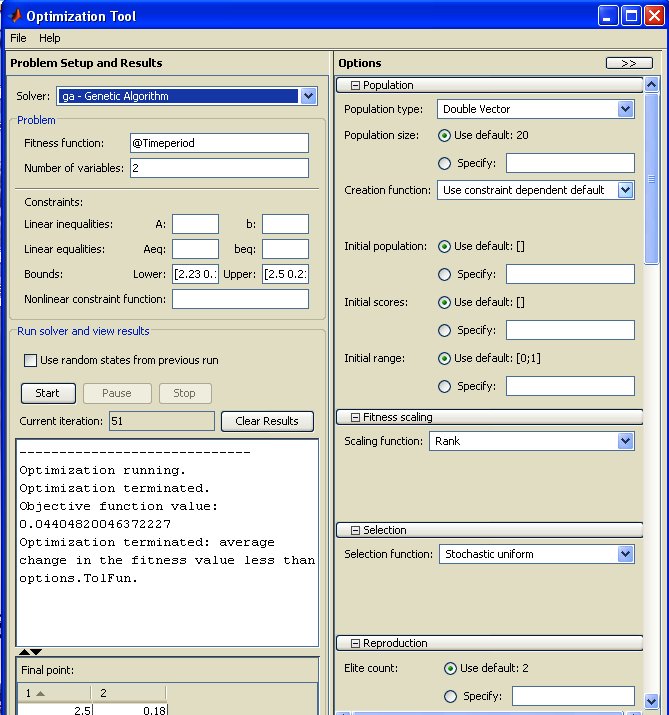


Fig 9.1 (GA toolbox results of optimized delay for NMOS)

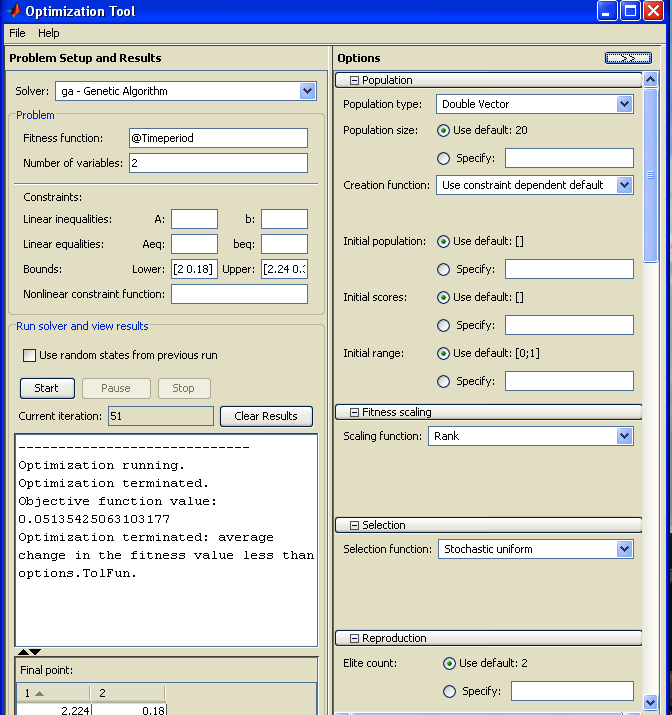


Fig 9.2 (GA toolbox results of optimized delay for PMOS)



Fig 9.3 (Execution of GA for optimized delay for NMOS)



Fig 9.4 (Execution of GA for optimized delay for PMOS)

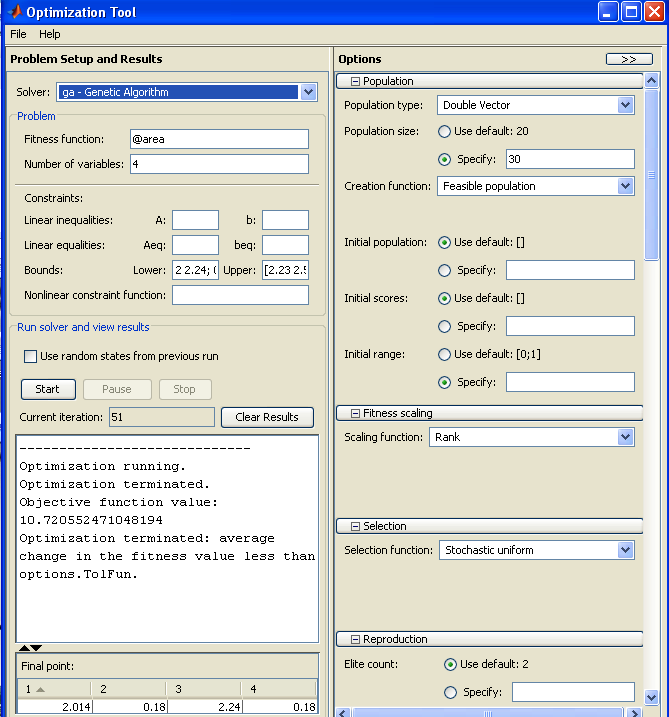


Fig 9.5 (GA toolbox results of optimized area covered by the transistors)



Fig 9.6 (Execution of GA for optimized Area covered by the transistors )

The values found for pmos and nmos are now used to find the fully optimized value of area covered by transistors. In this step the area is used as fitness function and delay is used as constraint for which the permissible values of w and l were obtained and these w and l values put in the GA toolbox to calculate the final optimized area.

The result of this shows that the optimized values of area covered by the transistors is 10.72055247u m2.

**9.2 Comparison between Results**

The optimum value of transistor dimensions calculated by the genetic algorithm tool for the permissible values of the w and l are far better than the maximum values w and l. The area calculated by using optimized values is not only less than that calculated by maximum w and l but also has the delay which is lesser than normal.

The figure 9.7 corresponds to the layout created by utilizing the optimized values, where as fig 9.8 shows the layout made with maximum values of w and l. The comparison shows that the layout area developed with optimized values is lesser (fig 9.8) than the other (fig 9.7).

When we calculate the area by multiplying the number of transistors using genetic algorithm, the area is calculated to be 10.7205524u m2 (for optimized values of w and l ),while the overall layout area as calculated from cadence (area of transistors + additional area) is 699.86u m2.

When we calculate manually the area by multiplying the number of transistors, the area is calculated to be 19.5804u m2 (for optimized values of w and l ), while the overall layout area as calculated from cadence (area of transistors + additional area) is 794.534u m2, which is larger than the previous.

Figure 9.9 & 9.10 shows the complete full adder layout in which none of the physical design rules are violated. This is checked by DRC (design rule checker), present in the tool itself.



Fig 9.7 ( Initial layout image with larger area)

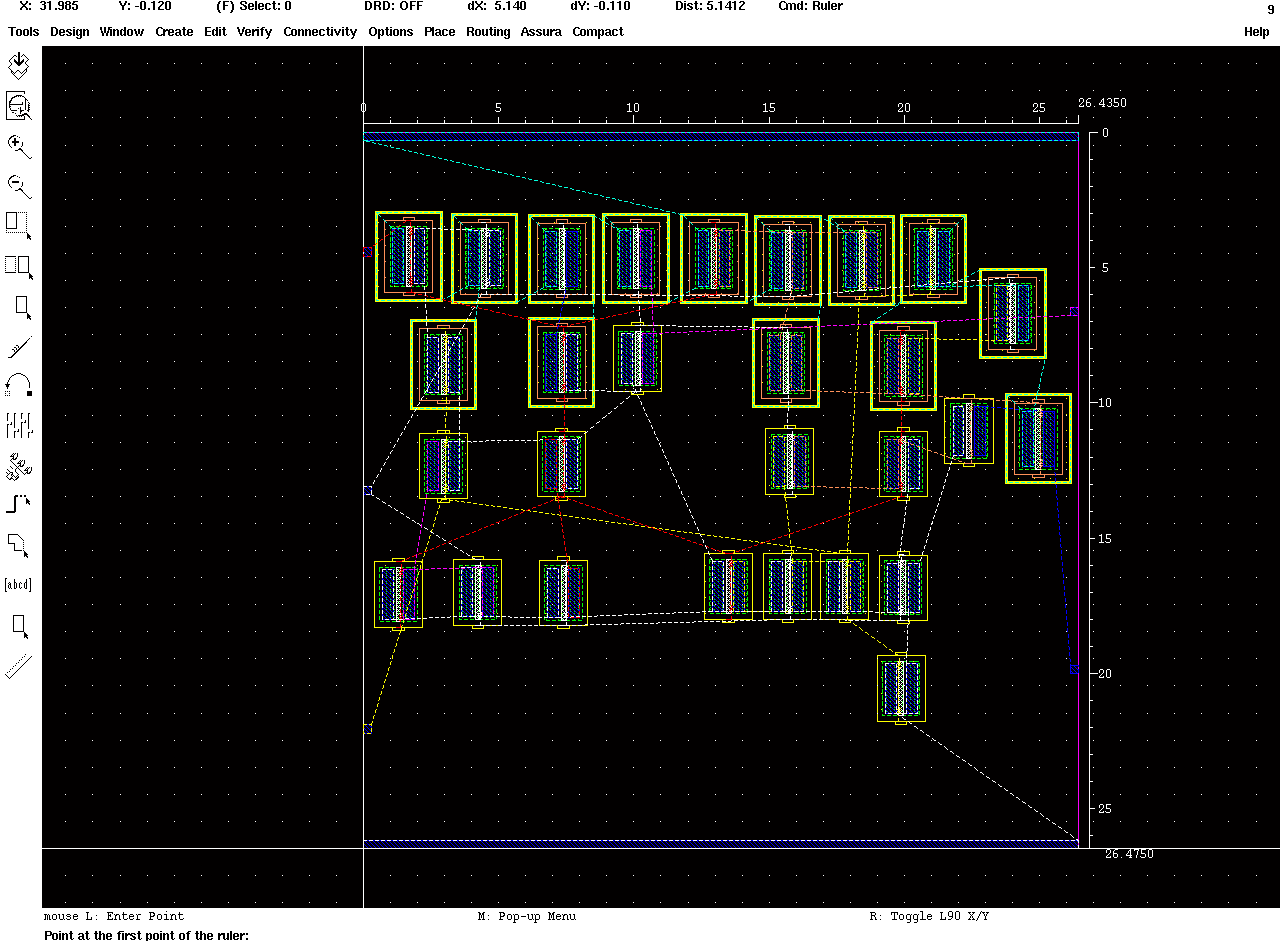


Fig 9.8 (Initial layout image layout with smaller area)

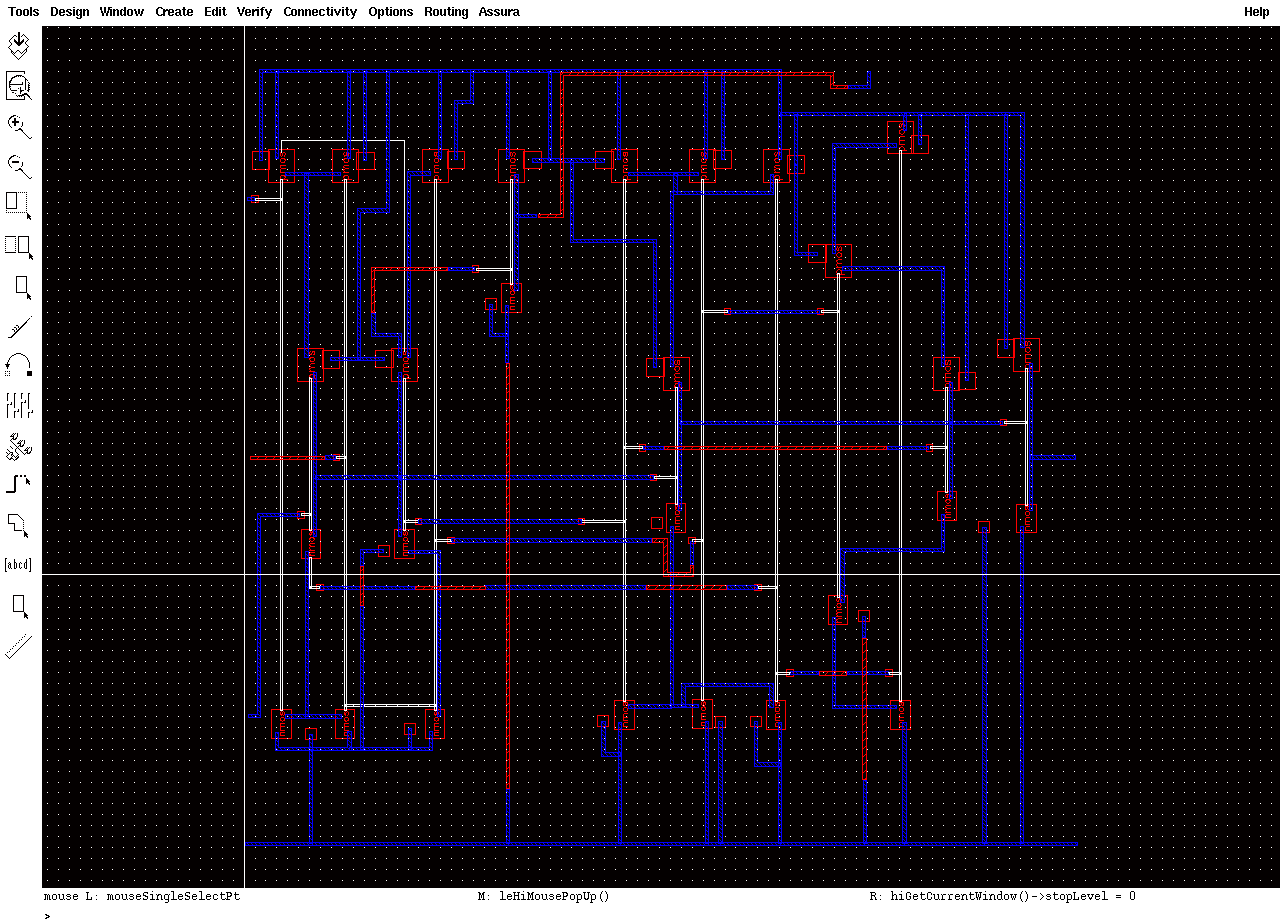


Figure9.9 (final layout of full adder circuit)

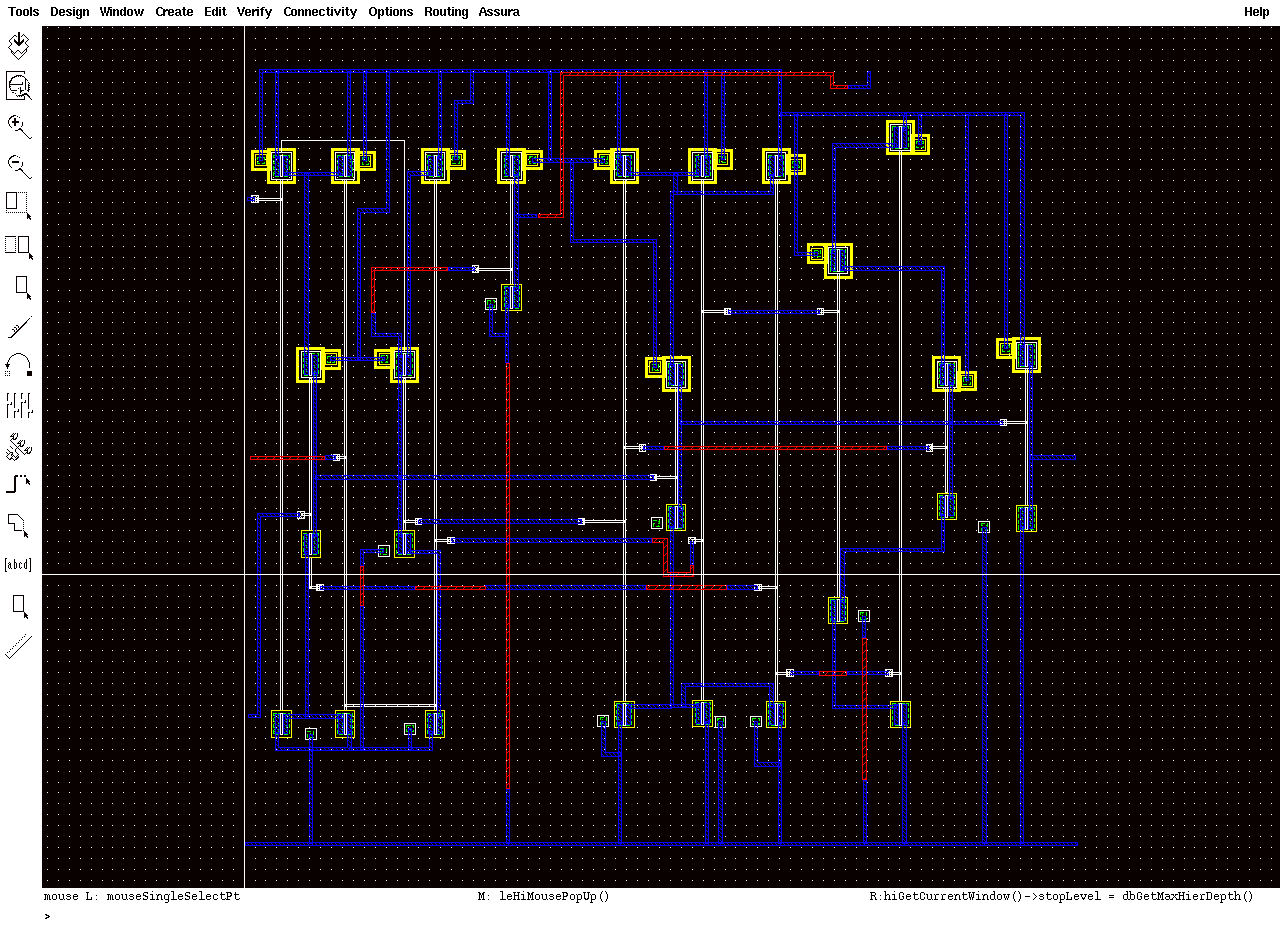


Figure9.10 (final layout of full adder circuit)

**CHAPTER-10**

**CONCLUSION AND FUTURE SCOPE**

**10.1 CONCLUSION**

The demand for portable devices has increased many folds in the recent past. There is a need to reduce the size of the circuits embedded in these devices. In an effort to do the same we have implemented Genetic algorithm to resize the cmos transistors. Transistors have been reduced in size, but not at the cost of other parameters. Care has been taken that the delay and the output voltage levels are well within the acceptable range.

The implementation of genetic algorithm to our current problem can be thought to be divided in three parts. In the first two parts we use the delay as the fitness function and find out optimum values of the delay for pmos and nmos respectively. The values of width and length for optimum delay found in step one and two are then utilized in step three, here the area is the fitness function and the delay as the constraint. The suitable values for width and length of pmos in step 1 are: 2.224 & 1.8 , The suitable values for width and length of nmos in step 2 are: 2.5, .18 .The final set of values for the optimized area as obtained by the genetic algorithm are: 2.014, .18, 2.24, 1.8 and the value of area covered by the transistors in the circuit is 10.7205u m2. The graphical results obtained from genetic algorithm are shown in figure 9.6

**10.2 Future Scope**

The basic methodology adopted by us was to reduce the layout area by transistor sizing, whereas correct routing and placement are other crucial aspect of area optimization, we may utilize advanced search algorithm like GA for this purpose.

We may try to further improve the results by comparing the results of various heuristic techniques such as PSO, simulated annealing etc.

Smaller areas may be achieved by reducing the number of transistors and rearranging the transistors to form a new circuit which works as a full adder.

We have made use of the genetic algorithm tool box of MATLAB to implement the genetic algorithm. A MATLAB code may be developed to implement the algorithm. A completely automated system may be created to reduce the manual effort.

.

**References**

[1] Andrew R. Conn, Paula K. Coulman, Ruud A. Haring, Gregory L. Morrill, ,Optimization of Custom MOS Circuits by Transistor Sizing, 1996

[2] Pinaki Mazumder and Elizabeth M. Rudnick, The Complete guide to VLSI design with genetic algorithms, Pearson Education, 2006.

[3] Sung-Mo Kang, Yusuf Leblebici, CMOS Digital Integrated Circuit Analysis and Design, 2003.

[4] Volker Schnecke, Oliver Vorn Berger, Genetic Design of VLSI Layouts, IEEE, 1996.

[5] Deepchand Patel ME, WCE, Sangli, Prvinkumar G. Parate ME, WCE, Sangli, ASIC Implementation of 1 Bit Full Adder, First International Conference on Emerging Trends in Engineering and Technology

[6] CMOS power consumption and Cpd calculation, Texas Instruments, SCAA03B, 1997.

[7] Official website of Math Works.

[8] Amir Amirabadi, Yousef Mortazavi, Ali Afzali Kusha, Optimizing Low power, High Speed Full Adder With Simulated Annealing, IEEE 2004 .

[9] Randy L. Haupt & Sue L. Haupt, Second Edition, Practical Genetic Algorithms, 2004.

[10] K. Glasmacher & G. Zimmermann, A Genetic Algorithm For Global Improvement of Macro Cell Layou, IEEE.

[11] Jens Lienig, Physical Design of VLSI Circuits and the application of Genetic Algorithm, Springer 1997.

[12] Kenneth V Noren & John E. Ross, Analog Circuit Design Using Genetic Algorithm, University of Idaho.

[13] Levent Aksoy & Eduardo Costa Optimization of area and Delay at Gate Level in Multiple Constant Multiplication, IEEE 2010.

[14] Arvind Kumar and Anil Kumar Goel, Study of Various Full Adder Using EDA Tanner Tool, IJCST 2012.

[15] Rs Jyurcsik, D.W Thomas & S.E Kerns Timing and area optimization of CMOS Combinational Logic Circuit Accounting for Total Dose Radiation Effect.

[16] Sachin Sapatnekar & Vasant B Rao, An Exact Solution To the transistor Sizing Problem For CMOS Circuits using Convex Optimization, IEEE