Switching Activity in CMOS Digital Circuits

Sandeep P. Shenoy

McGill University, Montreal

August 1996

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfilment of the requirements of the degree of Master of Engineering.

ISBN 0-612-19881-2

Abstract

In [48, 47] a pattern-independent method to estimate the switching activity of a CMOS circuit was presented. The technique relies on the use of abstract waveforms, described down to the level of individual transitions, which are propagated through the circuit. In order to improve the switching activity estimate so obtained, case analysis is undertaken on nodes with large fanout.

The objective of this thesis is to develop and implement a method to further improve upon the switching activity estimate through consideration of reconvergent fanout regions in the circuit. The idea is to impose functional consistency upon the waveforms at the nodes of a subset of the circuit to obtain an exact count of the number of transitions and potentially the exact waveforms which give rise to that. The result is the same as if an exact simulation was performed, but the novelty here is in the technique. An exact simulation would have exponential complexity as all possible waveforms on the PIs to the sub-circuit would have to be enumerated. Branch and bound techniques are used here instead to execute a progressively limited analysis which avoids exponential complexity. Furthermore heuristics are used to speed up the algorithm.

In addition a simple greedy algorithm has been developed and implemented to identify the sub-circuits where application of the above described technique would have the best results. The greedy algorithm represents only a preliminary step, and further work needs to be done on a more comprehensive circuit partitioning technique.

Résumé

Une méthode non liée à la forme est présentée en [48, 47] pour estimer l'activité de commutation d'un circuit CMOS. La technique repose sur l'utilisation de formes d'ondes abstraites, décrites jusqu'au niveau des transitions individuelles qui se propagent dans le circuit. Pour améliorer l'estimation de l'activité de commutation ainsi obtenue, on a soumis des noyaux de large sortance à une analyse par cas.

Cette thèse a pour but de développer et de mettre en oeuvre une méthode permettant d'améliorer l'estimation de l'activité de commutation en tenant compte des régions de sortance reconvergente dans le circuit. La méthode consiste à imposer une cohérence fonctionnelle sur les formes d'ondes aux noyaux d'un sous-ensemble du circuit pour déterminer le nombre exact de transitions et éventuellement les formes d'ondes exactes permettant d'arriver à cette estimation. Le résultat est le même que si l'on procédait à une simulation exacte, mais la nouveauté tient ici à la technique. Une simulation exacte comporterait une complexité exponentielle, car il faudrait dénombrer toutes les formes d'ondes possibles sur les entrées primaires du sous-circuit. Dans le cas présent, on a plutôt recours à des techniques de dérivation et de limitation pour exécuter une simulation progressivement restreinte qui évite la complexité exponentielle. De plus, des heuristiques permettent d'accélérer l'exécution de l'algorithme.

On a aussi conçu et mis en oeuvre un algorithme glouton simple afin d'identifier les sous-circuits oú l'application de la technique décrite ci-dessus risque de donner les meilleurs résultats. L'algorithme glouton ne constitue qu'une étape préliminaire et il faudra consacrer d'autres travaux à la mise au point d'une technique de partitionnement de circuit plus exhaustive.

Acknowledgements

I would like to thank my supervisor, Dr. Nicholas Rumin, for his constant support, help and enthusiam throughout this research. Thanks also due to Dr. Eduard Černý at the Université de Montréal, for his numerous ideas and inexhaustible enthusiasm. I am deeply grateful to Jindrich Zejda, also at the Université de Montréal, for taking the time to explain the inner workings and implementation of the power bounding system and answering my numerous questions in full and always promptly. I would also like to acknowledge his help and work in developing the algorithm for counting transitions on a node.

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through a strategic grant awarded to Dr. Nicholas Rumin and Dr. Eduard Černý.

Contents

List of Figures

List of Tables

1 Introduction

There is a constant insatiable demand for faster, better and smaller computers. More and more powerful processors are being used in a myriad of applications. The range of applications of computers is continually increasing. More than any other industry, the computer industry has an ethos of innovation driven by frenzied urgency. Pushed as much by advances in technology as by the demands (performance-wise and economic) made upon it, chip densities and operating frequencies are increasing and feature sizes are shrinking. Of late, there has been a considerable demand for portable battery powered devices. One of the issues that has come to the fore as result of this is that of power consumption. The power profile of a chip is important not only in situations of limited power supplies but also in cases where the nature of operation demands high throughput. In the latter case chip overheating can lead to degradation of performance and has critical implications for packaging and heat-sink arrangements. Furthermore consistent overheating can actually reduce chip lifetime through physical deterioration. Higher levels of integration and shrinking line widths have resulted in circuits being increasingly susceptible to the effects of power dissipation.

The demand for low-power chips means designers have one more feature to integrate into CAD tools. There is a need to predict power consumption during the design process, as well to modify a design to take account of power-related considerations. In this regard, two problems that have garnered much interest are the reliability of metal interconnect lines and the voltage-drop problem. For both of these, it is necessary to obtain information about transient current waveforms. The problems can be approached at various levels of abstraction but the requirements are the same for all proposed solutions: a fast and accurate resolution. The dominant technology in use at present is CMOS. A widely accepted assumption is made about the power characteristics of CMOS digital circuits: significant power is consumed only during logic transitions when charging/discharging currents are drawn. This implies that the power consumed depends on the switching activity of a circuit and therein lies the crux of the whole problem: how to estimate switching activity. It is clear that it is the applied inputs and the functionality of the circuit that determine the switching activity. The problem is input pattern-dependent and the direct way to solve it is to perform exhaustive simulation. But this is clearly impractical for large circuits. Most other techniques attempt to work around

the pattern-dependent nature of the problem. One way would be to employ statistical means to apply a comparatively small number of inputs to obtain a result that is within certain error bounds. Another would be to use probability factors to represent the input patterns. Using abstract waveforms allows one to compute maximum switching activity estimates. Out of these techniques, only those that keep track of temporal relationships between transitions are able to compute transient current waveforms.

The work in this thesis falls into the pattern-independent class of solutions. It adds to the system presented in [47, 48], where part of the results of this thesis has been included. This system is itself based on ideas first developed in [41] dealing with timing verification. The underlying technique of analysis used in timing verification is the same as that used for estimating switching activity, thus allowing the synthesis of a single tool to handle both timing verification and maximum switching activity estimation. However, the waveform representations used are different. This thesis contributes algorithms to obtain improved switching activity estimates. Much of the remainder of this section will review in detail various approaches to the problem of power estimation reported in literature after which the motivation and outline of this thesis will be discussed.

1.1 Literature Review

Most techniques reviewed in this section compute either the power or transient current waveforms or both. Transient currents, of course, have a direct bearing on the reliability of metal interconnect lines which is ensured by designing the width, or more correctly the cross-sectional area, appropriately. The *median time to failure, MTF*, is given by the following equation[2],

$$MTF = \frac{Area}{\beta J_{rms}^{M}} e^{\phi/kT}$$

where ϕ and β are constants of the materials and crystal structure, M is constant (usually taken as 2) and J is the current density. To size metals lines against electromigration one needs the RMS current, I_{rms}, which takes into account both the peak value and the duration of different currents, since the MTF depends on the shape of the waveform and not just its time-average[4]. It is given by the following relationship,

$$I_{rms} = \sqrt{\int i^2(t) \partial t / T}$$

where $i(t)$ is the instantaneous current and T is the time period or inverse frequency. For a gate or small module, the current is usually approximated by a triangular waveform of some sort which depends on three parameters: the peak current value, the duration of the current and the frequency. The peak current depends on transistor size and supply voltage, and the current duration on the load capacitance. The frequency is in fact the *effective* frequency which is input-pattern dependent. It is this *effective* frequency that is the target of much research. The RMS value of the current may be computed from this current waveform allowing one to estimate the MTF. The other consideration in designing the width of metal lines is the problem of voltage drops in power and ground buses. Large drops in voltage adversely affect switching speed and can even result in incorrect logic operations. Voltage is a direct function of current and the width of the conductor. To design for the worst-case voltage drop, again one needs the peak value of the transient currents. Power analysis techniques generally focus on either obtaining the transient currents or computing the power directly.

Power analysis can be done at several levels of abstraction. The most accurate are simulators like SPICE which operate at the transistor level. They are in general also the slowest because of the detailed non-linear models and the complete pattern dependence. Techniques which can be classified as low-level attempt to speed up computation by approximating the underlying models trading off accuracy for speed. In order to obtain accurate data a large number of input patterns must be applied and the results of each simulation run recorded (cumulatively or otherwise). Probabilistic techniques dispense with the application of input patterns altogether substituting representative probability factors instead. These are then propagated through the circuit according to some model. Further computation may be performed on a global or local basis to take account of spatial and/or temporal correlation. The results are probability factors at individual nodes which indicate switching activity. This type of technique is limited by the accuracy of user-supplied information on the probabilities at the inputs. Statistical techniques attempt to combine the advantages of both simulation (accuracy) with probabilistic techniques (speed). Fairly simple models are used to represent the circuit allowing the application of a large number of randomly generated input patterns in a reasonable amount of time. The number of patterns to be applied is limited according to the desired level of confidence and error.

All the above mentioned techniques provide data on typical or average switching

activity or power consumption. The problem of worst-case power consumption is dealt with in pattern-independent techniques. By eschewing the pattern-dependent nature of the problem altogether, they attempt to provide only upper bounds on power consumption. Typically this is done by computing upper bounds on transient currents. Such information is of particular interest in deciding the line widths in chips. At present very conservative estimates of power consumption are used in industry and hence considerable area is wasted. The trend towards increasing chip densities, higher operating frequencies and shrinking feature sizes concomitant with the current emphasis on low-power design has increasingly focussed attention on this area.

An alternative approach to analysing a circuit for power consumption is to design for low power in the first place. The functionality of the circuit is considered, and the design altered accordingly. These high-level techniques are not strictly in the power verification category, but an understanding of them provides insight into where and how power analysis techniques may be useful.

The remainder of this subsection is given over to a review of the literature on power estimation. It is divided into the groupings discussed previously.

1.1.1 Low-Level Techniques

Methods in this area operate mainly at the transistor level. A good example is SPICE. It is accurate enough to be used as the benchmark for comparision but slow and most inappropriate for the problem at hand. In order to obtain an estimate for average power dissipation one would have to apply a large number of input patterns and keep track of the current waveforms. Direct circuit simulation of this kind is impractical because of the sheer size of current circuits. Even if it were possible, the question of what input patterns to apply would have to be considered. The obvious answer would be to apply typical input patterns. What exactly are typical patterns is a moot point.

Nonetheless there are a number of offerings in the literature of SPICE-like circuit simulators. Many attempt to speed up the simulation through simplification of the models

used, sacrificing some accuracy in the process. The priority accorded to characteristics of such simulators such as speed, accuracy, ease of use, etc. are the defining factors. An event-driven simulator which uses such low-level modeling is presented in [14]. Here a model is computed for each gate by collapsing the set of transistors. Four basic waveforms are taken to cover all possible capacitative currents and a single waveform is used for the short circuit current. The waveforms are piecewise-linear whose defining parameters are computed from the collapsed gate. Maple, a symbolic software package, is used for the computations. A timing simulation is carried out to obtain temporal information about switching activity. This is then combined with the current models obtained previously to estimate the total supply current. This approach is suitable mainly for fast input signals. An accuracy of 10% (maximum deviation) and a speed-up of 3-4 orders of magnitude over SPICE is claimed.

A technique derived from [15] is presented in [6]. Here MOS devices are modeled as current-limited switches, ie. a device can produce only two values of currents depending on the gate-to-source voltage. Reactive effects are dealt with by modeling them as capacitors connected between circuit nodes and ground. This modeling dispenses with the need for integration during time-domain response computation. The original scheme in [15] has been modified to allow increased accuracy in evaluating current peaks, to simulate non-fully-complimentary CMOS circuits as well as sequential circuits with positive feedback loops. Comparision of supply current waveforms produced by SPICE and the Power Estimator[6] indicate that the error for the average power consumption estimate is below 10%. The speedup over SPICE is about two orders of magnitude and the authors expect this advantage to increase with larger circuits.

A probabilistic low-level approach is proposed in [13]. In contrast to most probabilistic techniques which consider only the charging and discharging of output capacitances, this method takes into account the power consumption of gate internal nodes and the capacitance feedthrough effect as well. The authors claim that neglecting internal gate nodes could well result in underestimation of power consumption by 10% to 20%. Modified *State Transition Graphs for Power Estimation* (STGPE), with several different fanout loads, are constructed for every gate in the cell library as a model of its power consumption behaviour.

Each edge of the STGPE is accorded a triplet, $(i_{jk}, E_{jk}^{i_{jk}}, W_{jk}^{i_{jk}})$, where i_{jk} is the input

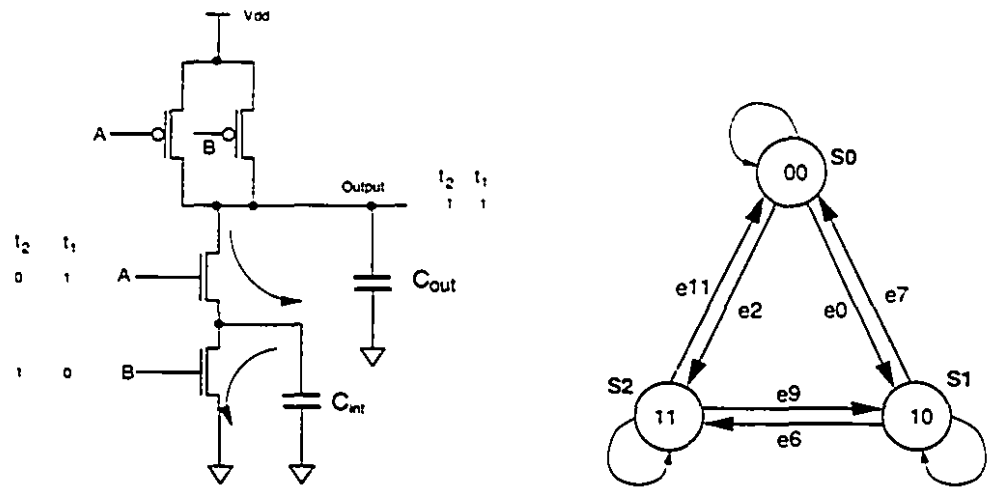


Figure 1: The STGPE of a 2-input Nand gate[13]

pattern which causes a state transition, $E_{jk}^{i_{jk}}$ is the *edge activity number* when the input is i_{jk} and N sequential patterns are fed into the gate and $W_{jk}^{i_{jk}}$ is the energy consumed. Briefly, the graph is used to capture information regarding state (output and internal nodes only) changes and the accompanying power consumption which is obtained through SPICE simulation. Given input signal probabilities and transition densities, a logic simulation is performed to obtain the same data for every node in the circuit. The edge-activity is then computed assuming temporal independence. The power consumption for each gate can then be computed. Results shown for some circuits were good (very close to SPICE results) while others contained an error of as much as 20%. In the absence of any detailed analysis of the results it is difficult to attribute the range of errors obtained to any specific factor.

A technique of collapsing CMOS logic gates into equivalent inverters[10] combined with a fast analytical method for computing the maximum current[11] is detailed in [12]. The main advantage of using this technique is that it avoids numerical integration and it does not impose any restrictions with respect to the number of switching inputs, relative delays of the inputs or the transition times. The collapsed circuit is full fledged in the sense that it can account for the effects of output load, input slope, short-circuit current and transistor size. However, glitches are ignored in the analysis. Estimates of peak supply currents are accurate to within 12% compared to SPICE with a speedup of over three orders of magnitude. Investigation of the model's performance when used for large circuits[7] determined that while

fast and accurate results are obtained for the peak current and its time of occurence, in general, there are certain instances where significant discrepancies in the waveforms occur. This can be directly attributed to the problem of glitches which has yet to be fully dealt with.

An empirical approach is proposed in PowerPlay[9] where the instantaneous power waveform is derived from a predefined empirical energy model. The model itself is derived from a detailed analog simulation of all the cell types in a design library by approximating the analog power waveform by a rectangular one. A digital simulation is performed upon the entire circuit which yields logic waveforms giving information about timing and transitions. The data-base of waveforms for each cell-type is now used to create an instantaneous power waveform for the entire circuit. The time required for power analysis is of an order comparable to that of logic simulation with good accuracy claimed.

Thus far there is a great body of work dealing with the problem of producing fast accurate simulators. A few of the varied approaches have been discussed above. However, issues arising from the way in which simulators should be used have been inadequately addressed. As mentioned earlier, one of the key questions is just what and how many patterns to apply to a circuit in attempting to obtain an estimate of the average power. The enormous size of modern circuits would seem to automatically preclude any practical application of simulators because of time and reliability considerations. Nonetheless these techniques remain useful for sub-system power analysis.

Recently a seminal step was taken in addressing these very matters in [8]. A definite quantitative relationship has been proposed between the number of patterns to be applied and the desired accuracy. The basic idea is to use Bernoulli random variables to model the occurence of a switching event, rising or falling, at each node. The authors make an assumption that a single pattern will at most generate one rising and one falling transition at any node. While this might seem to be too constricting at first, simulation data provided seems to surprisingly indicate that the assumption is well borne out. However, it is admitted that in specific cases this assumption will break down and that it does not deal fully with glitches. However, this does not detract from the importance of the overall analysis.

Using probability theory the relation $n = \frac{z^2}{4\epsilon^2}$ is proposed for the number of patterns,

n, required for an error ε. z is a variable related to the confidence level, $(1-\delta)$, desired. A tabulation of the ε and $(1-\delta)$ shows that for an error of 5% and a confidence level of 99.90% only 1084 patterns need be applied to a circuit. This result touts discrete simulation as an extremely viable method of obtaining accurate average switching activity estimates. Comparison of switching activity estimates obtained from using the number of input patterns predicted by the model against extensive simulations performed indicate that the number of nodes which have activity above the predictions is within the confidence level. Further work is underway in this area, particularly with respect to multiple transitions and bounding the possible error on individual nodes in very large circuits. The results may have significant implications.

1.1.2 Statistical Techniques

Statistical techniques rely upon repeated simulation of circuits to obtain measurements of power consumption. The idea here is basically to reduce the size of the problem, that of having to simulate for an impractically large number of possible inputs, to one where only a relatively small number of simulations need be carried out. Over time the power being measured will converge to the average power. Statistical analysis allows one to put a confidence and accuracy rating on the obtained result for a given number of simulations.

A Monte Carlo technique is proposed in McPOWER[16] which estimates the energy dissipated per clock cycle by applying a set of randomly generated input patterns. The number of patterns to be applied depends upon the desired accuracy and confidence level. The average power at each node during a given time interval T is,

$$\frac{1}{2}V_{dd}^2 C_i \frac{n_{x_i}(T)}{T}$$

where $n_{x_i}(T)$ is the number of transitions that occur at node n_{x_i} during the the time interval T and C_i is the total capacitance at i. There are two phases to the simulation: the *setup* and the *sample* phase. Two requirements must be met in deciding the length of both phases: one is that the signals be stationary processes, and the other is that the total average power computed from each sample phase be obtained from samples of independent random variables. The first requirement is met by ensuring that the setup phase lasts long enough to allow the switching activity due to an input, time to propagate through the circuit. For a combinational

circuit this implies the setup phase must last at least as long as the maximum possible delay of the circuit. For sequential circuits the use of unspecified heuristics is suggested since technically the maximum length of a path would be infinite. The second requirement is satisfied by restarting the simulation at the beginning of every setup phase. This is considered sufficient to guarantee independence between the simulations. The length of the sample time is harder to determine. Clearly the longer the sample time the smaller will be the sample standard deviation, which itself is dependent upon the circuit, but this dependency is not clear and hence the sample time is determined experimentally.

The input patterns are generated using random number generators available on computers which in reality are not truly random. It is unclear what effect this might have on the results. The results themselves are very good in terms of speed, the only drawback being that only an estimate of the total power is available and there is no information about individual gates or even groups of gates. In addition the stopping criterion used assumes normality with respect to power distributions but this may well be not the case quite frequently, especially if techniques proposed in [34] are used in designing circuits. The one advantage of this technique is the ability of the user to specify a desired accuracy beforehand.

A modification of the above technique is implemented in MED[20] which estimates individual node transition densities. Here the user is required to supply the transition density as well as the probability of the signal being high at every input node. Random logic waveforms based on the supplied information are generated and the circuit is simulated. As before, over time the transition density count at the nodes of the circuit should converge towards the mean. The simulation time then depends on the desired accuracy and confidence level. The caveat here is that nodes of a circuit do not all converge at the same rate – nodes which experience little switching activity in particular converge slowly thereby necessitating a larger number of input patterns. This problem is overcome by classifying such nodes as *low-density* and certifying them with absolute error rather than percentage error bounds. A drawback of this method is its slow speed.

Statistical methods are used again in [19] to estimate state line probabilities for sequential circuits. A synchronous sequential circuit is assumed and randomly generated input patterns are used to simulate it at a very high level – functional or zero-delay logic simulation.

This allows one to simulate for a very large number of cycles in the minimum of time, but glitching is thereby not taken into account. Given the accuracy and confidence level, the number of simulations that must be run is computed. These simulations are run in parallel, in the implementation, until the node probabilities are said to have converged. The run time of this method is rather slow.

Another way of dealing with sequential circuits using the Monte Carlo technique is presented in [17]. The primary objective was to deal with the intial transient problem which biases a Monte Carlo-based technique. A method to choose the intial states prior to simulation and the length of the "warmup" periods is proposed based on the Markov Chain theory. Results on ISCAS benchmark circuits indicate that the average absolute error is under 3%.

1.1.3 Probabilistic Techniques

The pattern-dependent nature of the power estimation problem can also be tackled by using a set of probabilities to represent signals. Information about typical input behaviour, assuming this is known, can then be supplied in this fashion. This is then propagated into the circuit according to some model enabling one to obtain an estimate of node activity at every node in the circuit. These type of techniques are potentially very powerful since they can provide information on switching activity for every node. They are limited, however, by the accuracy of the models used and the user-supplied probabilities.

The earliest approach to utilising probablities for power estimation, implemented as LTIME[23], relied on the concept of controllability. A signal flow direction is assigned to every transistor in order to identify signal paths. The controllabilities of a node is computed by considering the union of the controllabilities of the signal paths which culminate in that node. Capacitive loads on transistors vary depending on the input vectors; the probability of certain pathways being open and thereby adding to the capacitive load is considered. The power is expressed as,

$$P = \frac{1}{2} V_{DD}^2 \sum_i C_{L_i} T_i G_i$$

where C_{L_i} is the load capacitance, T_i is the transition probability and G_i is the probability

that the source of the switching transistor leads to a power or ground line. A zero-delay model was used which meant glitches were ignored, and temporal independence was also assumed. Any practical assessment of [23] is difficult as the only circuit it was run on was a four-input AND gate.

A more comprehensive approach was taken in CREST[28] where a real-delay model was used and temporal independence was not assumed. User-specified *probablity waveforms* are used to generate *expected current waveforms* at every node. CREST operates at the transistor level and propagates probability waveforms in an event-driven manner. Each gate current pulse is modeled as a triangular pulse that starts with a peak value, $E[I]$, and then decays linearly to zero over time τ. However, CREST cannot handle pass-transistor networks completely and only propagates events through them ignoring any power dissipation. This method requires that the probability waveforms at the inputs to a gate be independent, which obviously cannot be guaranteed if the circuit contains reconvergent fanout or feedback. *Supergates* are used to ensconce such regions which are then simulated with logical waveforms generated from the probability waveforms.

The results for this method were compared with SPICE yielding peak currents within 20% and average currents and timing estimates within 10%, as well a very large speed-up over SPICE. However, the largest circuit that CREST was run on consisted of only 1800 transistors.

The problem of correlation between nodes encountered in CREST was tackled in [32], a method, which uses the steady-state conditions of the transition waveforms, and approximates the correlation between nodes. Each transition waveform is *tagged* by its steady-state values upon the assumption that the correlation is purely between the steady-state values. An OBDD is then used to compute signal probabilities therefore limiting the application of this method to only small circuits.

Both spatial and temporal correlation are handled in a technique proposed in [24]. It uses symbolic simulation operating under a general or unit delay model using BDDs to compute Boolean functions at every node in terms of the primary inputs. The results exhibit great accuracy but as with any such symbolic simulation method, the intensive computation required makes it impractical for large circuits.

A similar approach to [32] is taken in [25] where spatiotemporal correlations are taken account of by building OBBDs. Correlation is approximated by allowing only pairwise correlation between signals based upon the lag-one Markov Chain: this makes the problem more tractable. Two possible approaches to building the OBBDs are outlined: *global* and *incremental*. The global method is similar to that used in [24] while the incremental method entails building an OBBD for each node in terms of its immediate fan-in only. This reduces accuracy but is less resource-intensive. The implementation uses a zero-delay model, thereby underestimating the power.

To get over the limitations of constructing BDDs, a Taylor expansion method is used in BAM[33]. Signal probabilities are approximated at each node using this technique. The signal probabilities are computed incrementally as are the cofactor probability terms, but with respect to the primary inputs. This ensures that correlation due to reconvergent fanout regions is taken into account at the same time ensuring that the procedure is not computationally expensive. Comparision with the results from [25] indicate improved accuracy in general with drastically reduced run times for the benchmark circuits.

A method of improving performance when using BDDs is proposed in [21] where the concept of *supergates* is used based on an algorithm given in [31]. The *supergate* of a node X is the minimal sub-circuit in X's transitive fan-in, feeding X, such that the sub-circuit's fan-ins are logically independent[21]. The idea here is to define regions of the circuit where a BDD analysis would be most effective. This is treated as a preprocessing step prior to the power estimation step in any method using BDDs. Certain *stubborn* nodes exist in circuits which have a large supergate and a correspondingly large potential BDD, necessitating further special consideration. In general the BDDs constructed will tend to be smaller as nodes will tend to be expressed in terms of the supergate inputs rather than the primary inputs. Results indicate that where previously a BDD-based technique may have run out of memory or taken excessive time, the same technique modified with the addition of supergate analysis is able to complete the analysis in reduced time and memory consumption.

Some of the techniques described above are based on zero gate-delay models. Thus *toggle power* due to glitches is effectively unaccounted for. Glitches can, however, form a considerable fraction of the total power[7]. One way of forming an accurate estimate of

switching activity is through *transition density*[29]. This is basically the average number of transitions on a node per unit time. The power would then be calculated by the following equation,

$$P = \frac{1}{2} V_{DD}^2 \sum_i C_{L_i} D(x_i)$$

where C_{L_i} is the load capacitance and $D(x_i)$ is the transition density. Further analysis of the transition density measurement reveals that some nodes are *extremely sensitive* to internal delays[30]. For these nodes slight delay variations result in large changes in the transition density. A technique to identify these nodes and compute an upper bound is implemented in MaDest[30]. Signal probabilities are propagated through gates using the interval delay model to form a *loose* upper bound on the switching activity which is then improved through a simple heuristic based on gate logic functions. MaDest is extremely fast posing an insignificant overhead and the upper bound computed is *robust* with respect to gate delay variations.

The assumption so far in the techniques computing transition density[29, 30] is that only a single gate input switches at any one time. This assumption is realistic for FSMs which follow the gray code sequence but not in general. Computing transition density in the case of simultaneous switching at the inputs to a logic gate is dealt with in PAS[22]. Signal probabilities at the primary inputs are propagated through the circuit using *symbolic probablity expressions*. A heuristic similar to that proposed in [18] is used for partitioning the circuit to improve results and retain speed. Gates are assumed to have zero-delays thus neglecting glitching.

A technique which takes into consideration both glitching and non-zero gate delays is presented in PSIM[27]. It is based upon computing boolean differences using BDDs. The results, for a number of non-standard circuits, are reported in terms of a *power factor* which is proportional to the actual power, although exactly how is not made clear so it is difficult to make any assessment.

A novel technique based upon the concepts of *conditional independence* and *almost isotropic signals* is presented in [26]. The details are rather involved but results presented for inputs with low correlation and for those with high correlation indicate that the correlation on primary inputs is a very significant factor in power estimation. These results are very relevant in view of the fact most techniques assume independence on the inputs streams on the PIs,

ignoring correlation between states of a synchronous sequential circuit.

1.1.4 High-Level Techniques

At this level the focus is on designing for low power. The problem can be approached at the system (modules not in use can be turned off), architectural (transforming logic networks for low power) or device level (reducing supply voltages where possible or resizing gates). At the logic level, one technique is to reduce the amount of switching activity by encoding states in FSMs appropriately. Another technique is to add extra logic to circuits specifically to avoid unnecessary switching activity where it is known beforehand that this would not serve any purpose. At the device level reducing the load capacitance faced by gates would result not only in lower power consumption but reduced area as well. To lower the load capacitance means to reduce gates sizes but this reduces the drive capacity of the now-smaller gates which may lead to increased rise/fall times. A thorough analysis is necessary before attempting resizing so as not to vitiate the functionality of the ciruit. A brief review of the work in this area follows.

The technique of gate-resizing requires that one has accurate timing information about a circuit. Since resizing with a view to lowering power consumption necessarily adds delays to a circuit, it is imperative that the accuracy of information on false paths and true circuit delays is of the higest order. A gate-resizing tool is presented in [35], an extension of the work in [36], which provides timing information of the necessary calibre using symbolic procedures based upon Algebgraic Decision Diagrams (ADDs). The concept is fairly simple. The objective is to calculate the *arrival time*, $AT(g, x)$, which is the time when the output of gate g settles to its value if input vector x is applied at time 0, and the *required time*, $RT(g, x)$, which is the time at which the output of g is required to be stable when input vector x is applied. The *slack time*, $ST(g, x)$ is then computed as the difference between the arrival time and the required time. This then gives the extent to which a delay can be added to the gate without compromising the functionality of the circuit and concurrently by how much the gate can be resized. The complete algorithm is more complicated since the effect of resizing one gate will mean recomputing the slack time for gates down the line. Results are given for a set of MCNC '91[5] benchmarks circuits and the power saved per circuit ranges from 0.5% to 39%.

No CPU times are given and the largest circuit consists of 306 gates.

A technique which re-encodes an existing circuit is presented in [37]. The objective is to find an encoding scheme such that the number of bit changes per state transition is minimized. Two methods are discussed; one using *recursive weighted non-bipartite matching* and one using *recursive mincut bi-partitioning*. Results are presented for a set of ISCAS '89 and MCNC circuits and show considerable reduction in power consumption as measured by [24]. However, it is not made clear which method was used. Furthermore this technique is not applicable for large circuits as it is too memory-intensive.

A method of minimizing power consumption in a boolean network through consideration of network "don't cares" is presented in [38]. A zero-delay network is assumed and the aim is to optimize each node for switching activity and its effect upon its fanout. A different approach is used in NOVA[40] where the problem is reduced to the solution of *face hypercube embedding* and *ordered face hypercube embedding*. These two problems arise in the course of optimizing a symbolic representation of the combinational part of the FSM where the states form a set of possible values for a single multiple-valued variable.

In a similar vein the problem of state encoding of an FSM is dealt with in [39]. Using a zero-delay model the total power is expressed as,

$$P_{reg} + P_{inputs} + P_{comb}$$

where the components are as shown in Figure 2. A simple approach is to minimize the Hamming distance between state pairs to reduce P_{reg} but this may increase the other two components. An improved power cost model is presented which takes into account not only the switching activity but the capacitive loading and the frequency of occurence of each state as well. The technique has been implemented for dynamic PLAs only and results for circuits from the MCNC '91[5] benchmarks compared to NOVA[40] show an average of 6.3% improvement for two-level logic. There is, however, no data on the CPU times for these experiments.

A novel approach that considers the problem from a logic level is to use add special *precomputation logic* to certain parts of a circuit to turn off those parts in order to minimize switching activity[34]. This involves computing the output values one clock cycle in advance

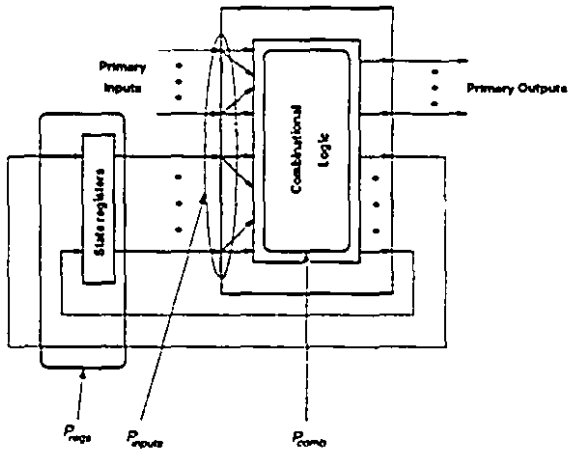


Figure 2: Power model for a Finite State Machine[39]

and using the results to either turn on or off the circuit under consideration. As the addition of this precomputation logic adds to the area and the clock period, the technique must be applied carefully only to non-critical regions of a circuit. A number of precomputation architectures are presented as well as procedures to automate the process. Results for MCNC '91 benchmark circuits indicate substantial power reductions with an average area penalty of only 3%. No results are given for any increase in clock period which may well have occurred.

1.1.5 Pattern-Independent Techniques

Most of the techniques described previously have to some extent or other been input-pattern dependent. They are only able to give estimates and, in general, there have been no guarantees on the error of those estimates. The exception in this regard are statistical techniques. Bounds on power consumption are useful in designing for worst-case situations in particular in designing power lines. The aim of the techniques in this section is to compute an upper bound on the total current or the power.

A technique to measure the maximum currents at every contact point in a circuit has been implemented in `iMax`[42]. The idea is based on the use of *uncertainity waveforms* which describe the presence of the following excitations at various time intervals: *high, low, high-low, low-high.* A typical waveform may look something like in Figure 3.

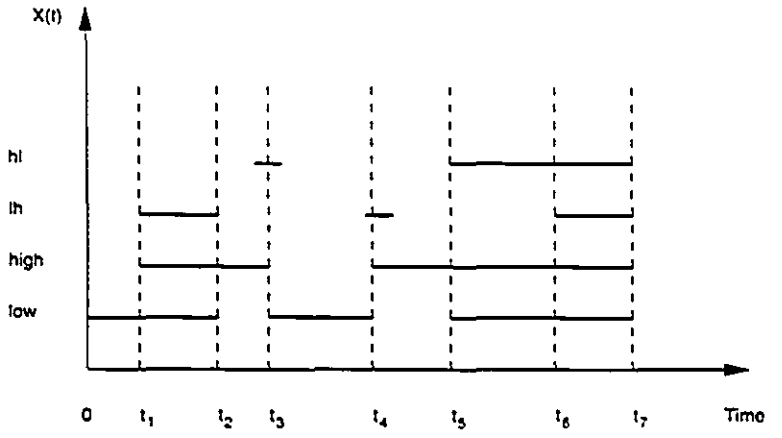


Figure 3: Uncertainty waveform[42]

The solid lines in Figure 3 indicate which of the four possible excitations are possible during specific time periods. So for instance between time 0 and t_1 the waveform is *low* as indicated by the single solid line at *low*. Between time t_1 and t_2 the waveform can be *low*, *high* or switching from *low* to *high* which means that the waveform is *low* at time t_1, switches from *low* to *high* somewhere between time t_1 and t_2 and is *high* at time t_2.

First the primary inputs are all assigned a waveform which contains the set of all possible transitions. This is propagated through the circuit to obtain an uncertainity waveform on every node. To reduce the complexity of the waveforms that are generated, some merging is carried out on the intervals depending on a user-specified parameter. Each *low-high* and *high-low* transition is assumed to draw a triangular pulse of current. Combining these currents at a node results in a *Maximum Envelope Current*, or *MEC*, which is basically the maximum current that can be drawn at that time. Signal correlations are taken into account at fan-out nodes by exhaustively enumerating the waveforms and running the `iMax` algorithm for all the gates that are contained within a *cone of influence* defined by the fanout. The results are compared to a logic simulation using randomly generated inputs which provide a lower bound on the switching activity. The ratios between the upper and lower bounds for a set of ISCAS circuits range from 1.23 to 2.23 with reasonable CPU times.

The `iMax` algorithm is further improved by the use of a *Partial Input Enumeration*, `PIE`, technique that better resolves signal correlation. It was first proposed in [43]; a more detailed exposition can be found in [44]. The idea is to enumerate waveforms on certain

primary inputs to resolve signal correlation and thereby improve the MEC estimate. The enumeration proceeds by conducting a *best first search*, or *BFS*, on the search space of all possible input patterns. One advantage of BFS is that the procedure can be terminated at any time since it effects a progressive improvement on the upper bound. Two heuristics are proposed for the decision-making process, whereby primary input nodes to be enumerated are chosen. The improved technique gives marginally better results for most ISCAS circuits, but certain circuits exhibit significant improvement in the upper bound to lower bound ratio at the expense of increased running times.

Another technique which is based on a similar uncertainty waveform representation is presented in [45]. A waveform is a set of four binary functions, $LH(t), HL(t), L(t), H(t)$, which describe when *low-high* switching and *high-low* switching may occur and when the logic state is at *low* or *high* respectively. A two-vector input stimulus is applied to all the primary inputs and then propagated through the circuit. A count of the transitions establishes an upper bound on the switching activity. This is the basic algorithm. Due to the exponential growth in the number of transition data points at internal nodes in a circuit, a merging strategy is resorted to whereby transitions of the same polarity with no intervening transitions of opposite polarity are merged to form a single interval in time where only a *single* transition may occur. The waveform representation is now in terms of intervals rather than individual transitions reducing significantly the amount of storage needed. Signal correlation is not taken into account in this technique. Results for ISCAS benchmarks circuits show that for most circuits, the ratio between the upper bound and lower bounds, provided by simulation, on individual nodes is 1.0 or less for about 75% of the nodes. CPU times are not provided but it is claimed that the technique is applicable to large circuits.

A technique which computes not only maximal currents but also maximal current derivatives has been implemented in PRITI[46]. A novel representation of switching activity, *activity waveforms*, is used to describe the *turning-on* and *turning-off* of transistors as shown in Figure 4 over all possible voltage waveforms.

The graph at the top in Figure 4 shows three possible waveforms, $V_1(t), V_2(t)$ and $V_3(t)$, that might occur at a node. These waveforms are represented by the activity waveforms shown in the bottom graph. The idea is to represent only the "activity" or the rising and

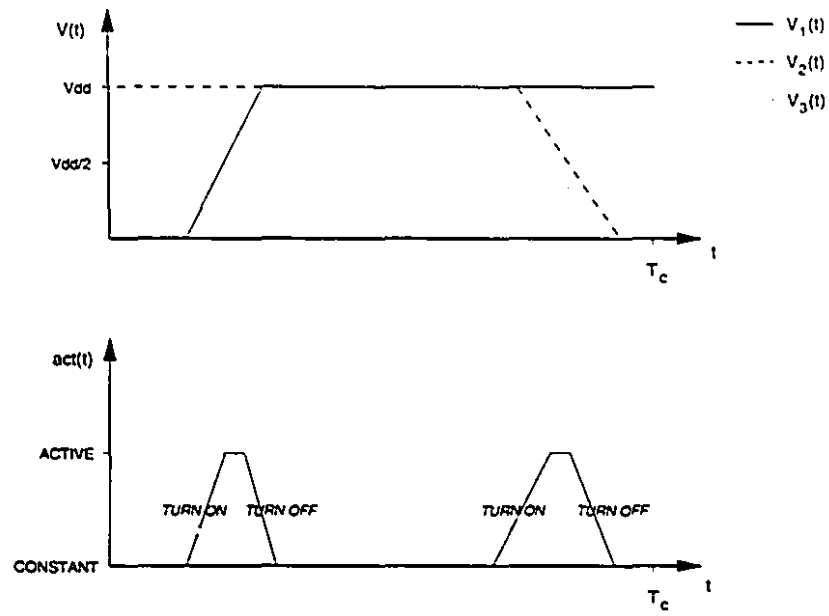


Figure 4: Activity waveform[46]

falling of the voltage waveforms. The first trapezoid in the activity waveform is formed by combining the earliest and latest rising voltage; and similarly for the second trapezoid. Such an activity waveform does not allow for any distinction between rising and falling activity so two separate activity waveforms are used per node: one for rising and one for falling transitions.

A transistor-level description of a circuit is used. The problem is simplified by splitting the circuit at certain nodes which form *additional inputs* like clocks whose behaviour is known beforehand. D flip-flops are also considered additional inputs since they influence their fanout only at certain points in time, ie. their effect on their fanout does not depend on the *exact* time at which their input nodes switch. It is assumed that only one input of a gate will switch at any point in time. Activity waveforms as described in Figure 4 are then propagated through the circuit from the primary and *additional* inputs.

The current and its derivative is modelled by a trapezoidal waveform and a rectangular waveform respectively. The duration of the current waveforms and their amplitudes are dependent on factors drawn from the activity waveforms on each node. However, it is not made clear what kind of waveforms are initially on the primary inputs. The authors claim results are 2 to 5 times better than existing commercial tools.

1.2 Motivation and Overview of Thesis

The work presented in this thesis falls into the category of pattern-independent techniques described in section 1.1.5. It represents a contribution to the power bounding method presented in [47, 48]. The technique reported there relies on the use of abstract waveforms, described down to the level of individual transitions, which are propagated through the circuit. In order to improve the switching activity estimate so obtained, *case analysis* is undertaken on nodes with large fanout. This is a global analytical technique which attempts to reduce the pessimism in the switching activity estimate over the entire circuit.

The alternative to a global analysis is local analysis. Developing and implementing a method of local analysis to further improve upon the switching activity estimate through consideration of sub-circuits is the objective of this thesis. The idea is to impose functional consistency upon the waveforms at the nodes of a subset of the circuit in order to obtain an exact count of the number of transitions and potentially the exact waveforms which give rise to that. If an exact simulation had been performed, the result would have been the same, but the novelty here is in the technique. An exact simulation would have exponential complexity as all possible waveforms on the PIs to the sub-circuit would have to be enumerated. Branch and bound techniques are used instead to execute a progressively limited analysis which avoids exponential complexity. Furthermore heuristics are used to speed up the algorithm.

In addition a simple greedy algorithm has been developed and implemented to identify the sub-circuits where application of the above described technique would have the best results. The local analysis is best applied to regions or sub-circuits which exhibit reconvergent fanout. The greedy algorithm is only meant to represent a first step, and further work needs to be done on a more comprehensive circuit partitioning technique.

The next section will describe in some detail the main concepts and operation of the overall power bounding system into which the results of the present work are incorporated. Sections 3 and 4 will then detail the algorithms developed for local analysis and the greedy algorithm to pick sub-circuits respectively. Experimental results and conclusions will be presented in the final two sections.

2 System Overview

In [47, 48] a power verification tool was presented which utilized a pattern-independent method to compute an upper bound on power consumption. This thesis presents further contributions to that tool. To put this work into context it is necessary to first understand the overall system described in [47, 48]. A brief discussion of the main concepts in the power estimation part of the tool follows, along with a definition of some of the terminology that will be used.

As mentioned previously, the problem of power estimation in CMOS circuits is an input pattern-dependent problem. Any solution technique must have a way of describing inputs. The input waveform representation used in this thesis was first proposed in [48]. It uses a set of four *transitions*: *rising*, *falling*, *stable 0*, and *stable 1* as shown in Figure 5.

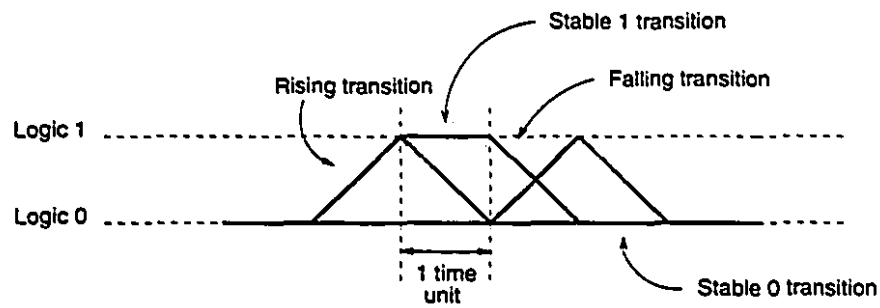


Figure 5: Waveform representation

These transitions will henceforth be referred to by the following symbols:

stable 0 transition: 0
stable 1 transition: 1
rising transition: r
falling transition: f

A real waveform would then be mapped onto the discretized version as shown in Figure 6. Simple waveforms which begin and end at the same logic levels can then be merged together to form a *waveform class*. So for instance all simple waveforms which begin at logic level 0

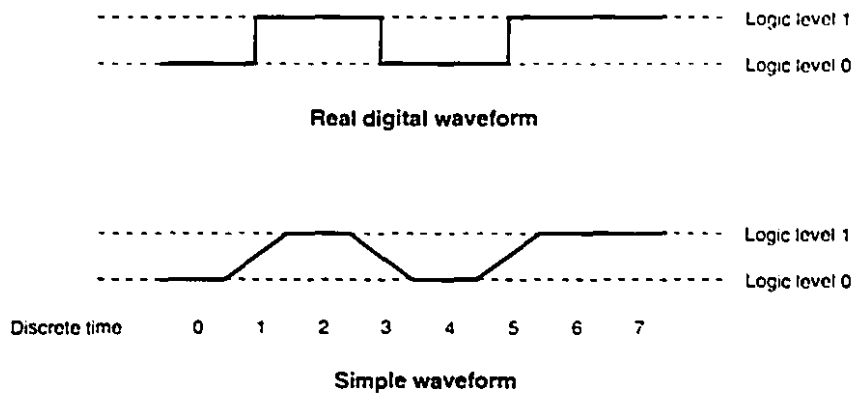


Figure 6: Mapping a real waveform onto a *simple waveform*

and end at logic level 1 could be merged together to form a class referred to as c01 as shown in Figure 7. There are four possible classes of *complex waveforms*: c00, c01, c11 and c10.

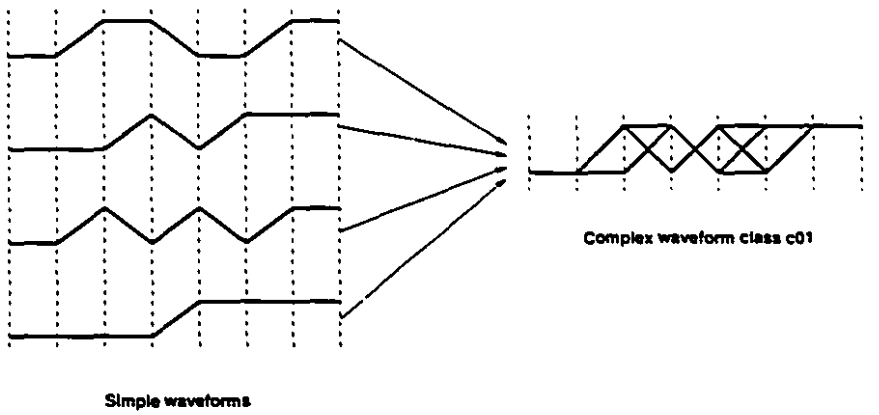


Figure 7: Waveform class c01

An *abstract waveform* is the set of all these four waveform classes.

The tool takes in a gate-level description (Verilog format) of a circuit and applies a two-vector input stimulus representing the four transitions to the primary inputs. The waveforms are then propagated through the circuit. Since every node has a set of abstract waveforms, evaluation at a gate consists of combining each of the classes of waveforms on the input nodes according to the functionality of the gate. For a two-input gate this would mean 16 possible input combinations to be evaluated which are then merged into four classes on the output node as illustrated in Figure 8. The AND gate has a delay of one time unit.

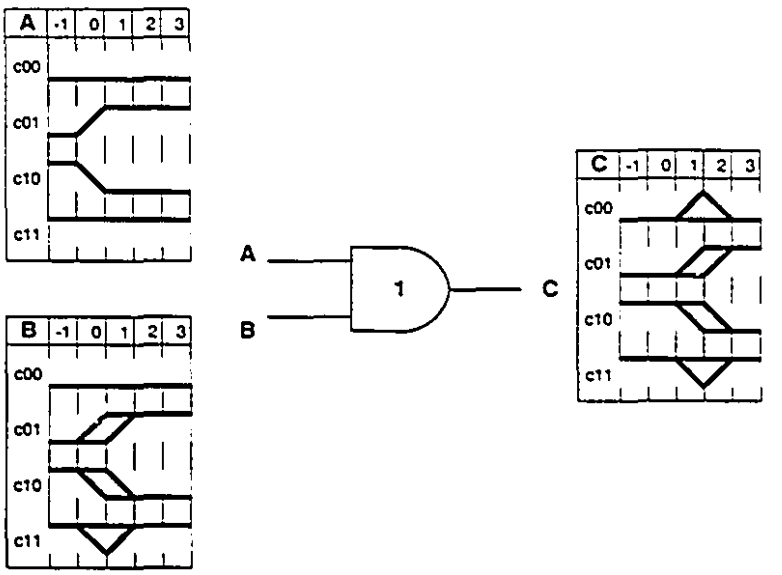


Figure 8: Abstract waveform evaluation at an AND gate

The output waveforms are therefore shifted forward by one time unit. The evaluation of the transitions is carried out according to the definitions shown in Figure 9. This type of table

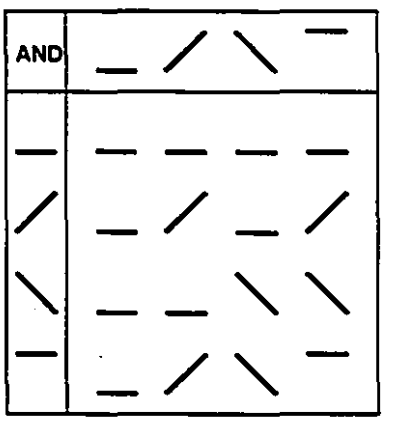


Figure 9: Transition evaluation for the two-input AND function

can be constructed for any function such as OR, XOR, NAND, etc., quite easily. It is simply a matter of examining the endpoint of each transition, as to whether it is a logic 0 or logic 1, and evaluating them separately according to the functionality of the gate under consideration.

Gates with more than two inputs are handled slightly differently. Instead of creating an n-dimensional, for n inputs, transition evaluation look-up table, the computation is carried

out using 2-dimensional tables such as one in Figure 9. Figure 10 illustrates the concept of

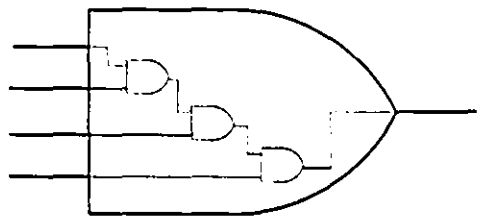


Figure 10: Functional evaluation of a three-input AND gate

using 2-dimensional tables for a multi-input gate. The four-input AND gate is broken down into three two-input AND gates. Each of these is evaluated individually using the table in Figure 9. Other gates are handled similarly differing only in the internal breakdown. For instance a three-input NAND gate might internally consist of three AND gates and a single NOT gate. The implementation of the functionality of multi-input gates is hidden from the user.

Two-vector input stimuli, as shown on node **A** in Figure 8, are applied to every primary input of a circuit. The waveforms are propagated through the circuit and an initial estimate of the switching activity in the circuit is obtained by determining the greatest number of transitions possible at each node. This involves implicitly obtaining the simple waveforms in each class at a node. The maximum transition count of these is then the transition count for that abstract waveform at that node. The transition count thus obtained is quite clearly very pessimistic mainly because of the merging that is performed with no regard to correlation such as that introduced at fanout nodes.

To improve this estimate *case analysis* is performed. This is a global analysis over the whole circuit that attempts to find the set of classes on each node which corresponds to the greatest number of transitions over the entire circuit. The idea is to impose constraints on various nodes and evaluate the circuit. A node may be constrained to a certain class and the effect of this is evaluated both forwards and backwards. For the backward case this means determining which classes of waveforms on the input of the gates could possibly give rise to that particular class. For the forward case, it is akin to simulation except that classes are being considered.

Backward propagation is carried out using *partial inverse functions*[3]. These describe the deduction of an abstract waveform set for one input as a function of the other inputs and the output. Basically all transitions on the input which could not possibly have contributed to the reduced waveform set on the output are eliminated; a necessarily pessimistic operation. The operation procedes one time unit at a time considering each transition on the output individually as illustrated in Figure 11. Given an AND gate with a set of waveforms as shown

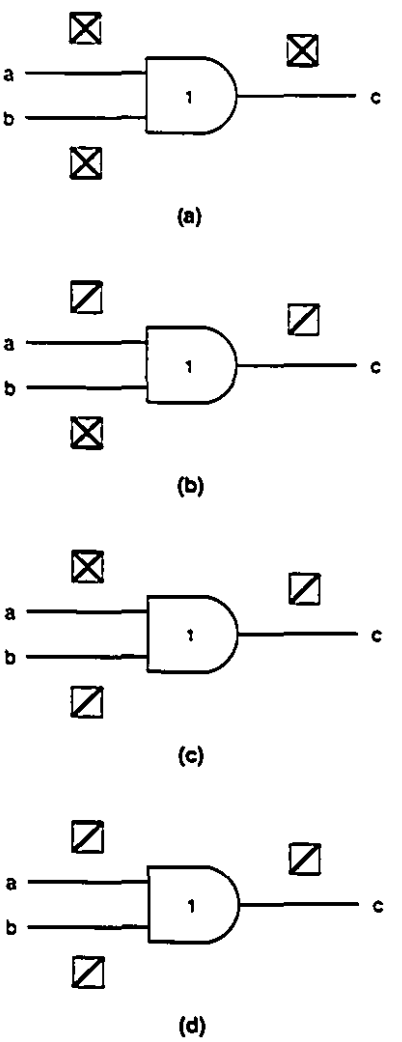


Figure 11: Backward propagation for a transition set on an AND gate

in Figure 11a, the output being at some time *t* and the inputs at time *t-1*, suppose that the output is restricted to r. If this is propagated backwards on **a** with respect to **b** and **c**, the result is shown in Figure 11b. Only r1 on **a** ANDed with 0rf on **b** can possibly result in r on

c. The 0 on **a** ANDed with 0rf on **b** cannot result in r on **c** so it can be eliminated. Now r1 ANDed with 0rf actually gives 0rf, more than just the r needed, but for the moment that is not of concern. By similar reasoning, propagating r on **c** backwards onto **b** with respect to **a** gives r on **b** as shown in Figure 11c. The final result is shown in Figure 11d, where both the new transitions on **a** and **b** have been computed. This computation is easily extended to multi-input gates.

The procedure in case analysis is to pick a node, impose a class constraint and perform a transition count. This is done for each of the four classes on a node in turn; the circuit at the end of each constraint evaluation is returned to its initial state. Based on a comparision of the four transition counts, for each class, the largest one is chosen and the corresponding constraint imposed. The process now repeats with a different node. In [48, 47] the criterion used to pick nodes was fanout; nodes with the largest fanout have constraints imposed first. The reasoning behind this was that such nodes would have a greater influence on the rest of the circuit.

Case analysis proceeds by constructing a *case analysis tree* using a *frontier* technique. A sample tree is shown in Figure 12 for a circuit which, after the initial propagation of

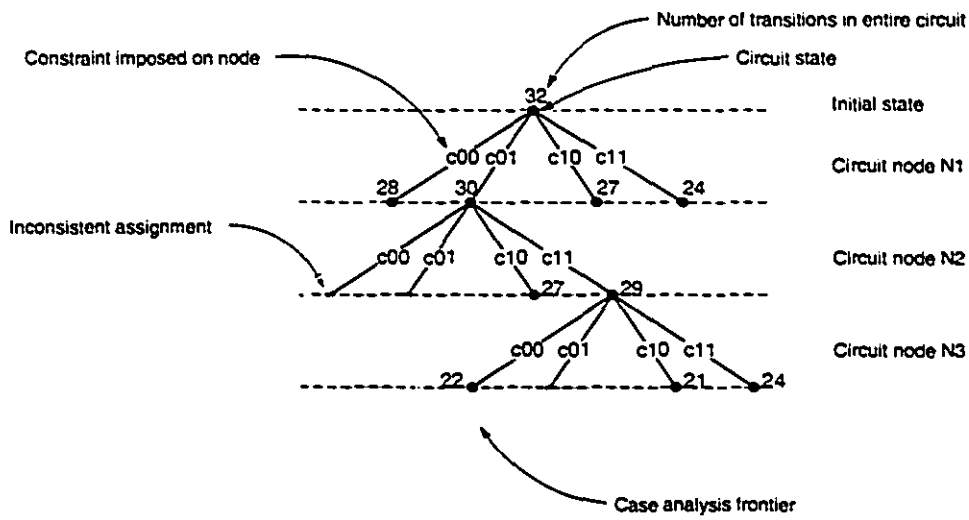


Figure 12: A sample case analysis tree

waveforms, has a transition count of 32 over the entire circuit. Imposing the constraints c00, c01, c10 and c11 in turn on node N1 results in a transition count of 28, 30, 27 and 24

respectively. Out of these, the worst-case, 30, is chosen as the point from which to further explore the tree. Constraints are now imposed on node N2. Two of those constraints, c00 and c01, result in an *inconsistent assignment* which means that restricting node N1 to the class c01 is incompatible with restricting node N2 to either class c00 or c01. The tree expansion then continues from the remaining nodes.

The success of case analysis depends to a large degree on the nodes chosen to be analysed. For small circuits it may be sufficient to simply apply it to the primary inputs as these very likely exert the most influence. In large circuits, however, the nodes which have the greatest influence must be identified according to a topological or functional consideration of their influence upon the circuit.

Case analysis is scalable in the sense that the analysis can be continued for as long as is desired or resources permit. Potentially one can explore the tree down to the bottom leaf and in effect realise the equivalent of a full simulation of the circuit, thereby indentifying a vector which gives rise to the greatest number of transitions overall. However, this in general is not practical for most circuits. Their size and the number of primary inputs simply precludes any such prospect. The tree exploration has to be terminated at some point once some user-defined time or resource limit has been reached. If a power budget has been assigned to the circuit, exploration may, of course, be terminated as soon as it is met. The longer the exploration, the tighter is the upper bound on switching activity, and hence on the power estimate.

The alternative to global analysis is local analysis. The idea here is to pick certain regions of the circuit and perform exhaustive analysis. If the regions are chosen correctly, it may be more profitable, in terms of resources used, to conduct an in-depth analysis after a certain amount of global analysis. The most obvious regions or sub-circuits to consider would be reconvergent regions since correlation between waveforms can be exploited.

3 Estimating Switching Activity

Switching activity in the circuit is measured by counting the number of rising and falling transitions on every node in the circuit. One way to achieve this is to simply count all the rising and falling transitions. However, this clearly results in an overestimate since any simple continuous *path*, a subset of the complete waveform, can only consist of a series of successive rising and falling transitions with appropriate stable 1 and stable 0 transitions interspersed between them. This is illustrated in Figure 13: the line in bold is one example of a *path*.

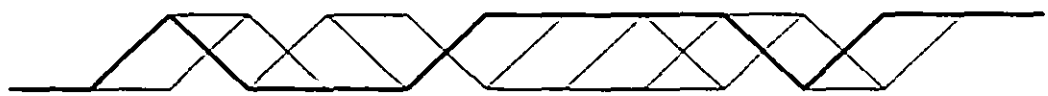


Figure 13: A sample path within a waveform

Such a path may or may not occur in reality. The merging of waveforms into the four classes described earlier increases the complexity of the waveforms – a class may have many more transitions in its component paths than were in the original waveforms. Currently it is not possible to distinguish the real paths from those that were created as a result of the merging.

An algorithm to extract such a *worst-case* path, maximal in the number of transitions (rising and falling), referred to as a *maximal simple waveform*, is presented in this section. The worst-case path in itself is not so much the issue as the number of rising and falling transitions contained within it. It is this value, referred to as an *exact count*, that is used in case analysis to determine which branch to explore in the case analysis tree. The exact count over a node is in reality an overestimate over the *true count* which may be obtained by exhaustive simulation over the entire circuit.

3.1 Counting Transitions on a Node

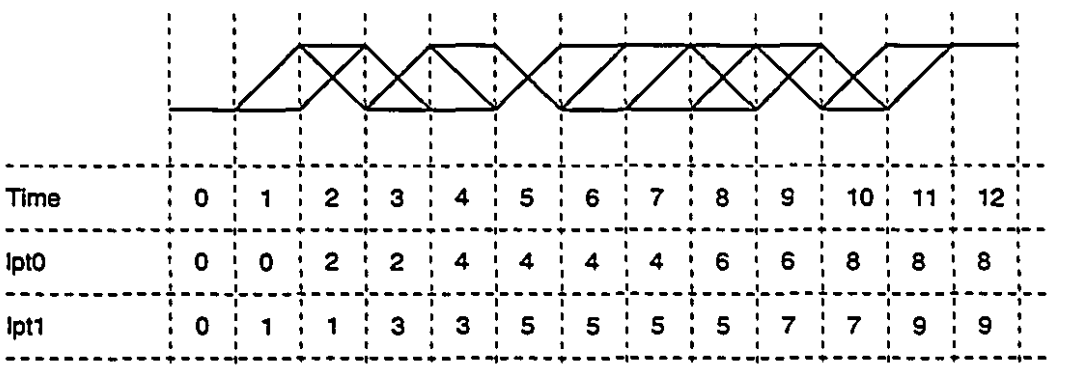
A complex waveform on a node represents the set of all possible transitions over a period of time. Of all the possible paths through such a waveform there is only one which

will actually occur in reality. The remaining paths are either false (creations of the merging process) or would occur in *non-maximal* cases – the *maximal* case being the set of paths on every node which give rise to the greatest number of transitions over the entire circuit. Of course, there could be more than one maximal case with their corresponding set of paths. The objective here is to find a path or subwaveform, which might or might not be real, with the greatest number of rising and falling transitions. An algorithm for this is presented in pseudocode in Figure 14:

```
for (t = 0 to Tmax)                          // Tmax is the time over which
{                                            // the waveform extends.  lpt0
    if waveform[t] has a rising transition   // and lpt1 keep track of the
        lpt1[t] = lpt0[t - 1] + 1;           // longest path to logical 0
    if waveform[t] has a falling transition  // and logical 1 respectively.
        lpt0[t] = lpt1[t - 1] + 1;           // waveform[t] contains a des-
}                                            // cription of the waveform at
                                             // every point in time
```

Figure 14: Pseudocode for counting transitions on a node

The working of the algorithm is best illustrated through Figure 15. A sample complex waveform along with the values of the variables at each point in time is indicated below. The



Time	0	1	2	3	4	5	6	7	8	9	10	11	12
lpt0	0	0	2	2	4	4	4	4	6	6	8	8	8
lpt1	0	1	1	3	3	5	5	5	5	7	7	9	9

Figure 15: Value of variables over time for a sample waveform

possible paths from time 0 to time 12 which end at logic 0, can contain a maximum of 8 rising and falling transitions as indicated by the value of `lpt0` at time 12. Similarly the maximum number of transitions in any path which ends at logic 1 is 9 as indicated by the final value of `lpt1`. In the implementation of this algorithm the value of the variables `lpt0` and `lpt1` is

not kept for every time as may be implied by Figure 15. Instead other variables are used to temporarily store the values of `lpt0` and `lpt1` of the previous time unit only.

The advantage of this algorithm is that its running time is linear with respect to the length, or extension over time, of the waveform. The complexity of the waveform or the number of possible paths, which may be maximal in the number of transitions since there is usually more than one, does not affect the running time at all. The variables `lpt0` and `lpt1` keep a cumulative transition count for all the possible waveforms at any point in time which is what makes the algorithm so efficient. Depending on the class of the waveform for which the transition count is being done, the appropriate variable is taken as the exact count; for instance counting transitions on the waveform c00 and c10 would mean taking the final value of the variable `lpt0` and conversely for the c01 and c11.

3.2 Counting Transitions on a Set of Nodes

The problem is essentially to determine the maximum number of transitions possible in the waveforms of a set of functionally interconnected nodes. This is primarily intended as a limited form of local analysis as opposed to the global nature of case analysis. The complex waveforms at any node represent the set of all possible simple waveforms as described previously and some additional artificial ones. The simple waveforms within the complex waveforms are related by the topological functionality of the entire circuit not just the sub-circuit being considered. Limiting the analysis to a sub-circuit still provides a transition count which errs on the side of pessimism with respect to switching activity. Enlarging the sub-circuit to include more functional elements, or gates, would be equivalent to imposing additional constraints on the system and would result in an even more accurate transition count in the sense that the reduction in pessimism would be greater. The tradeoff is between larger sub-circuits (correspondingly more accurate transition counts) and available resources (time, computation power and memory).

It is relatively simple to determine the set of all maximal simple waveforms at every node by making use of the transition counting algorithm for a node described in the previous section. These maximal complex waveforms, however, may well not be functionally consistent

over a sub-circuit. The maximal complex waveforms on a node or subset of nodes within the sub-circuit do not necessarily imply maximal complex waveforms on any other nodes. The largest transition count over a circuit is most likely given by a set of non-maximal complex waveforms. It would seem then that there is no alternative to that of exhaustive simulation to determine exactly those non-maximal complex waveforms and thereby the overall maximum transition count. In general this is true but there are two facets of the problem that can be exploited to improve the problem resolution. The first concerns topology and second is the analytical technique used; each one will be discussed separately in the subsections to follow.

3.2.1 Algorithm for Convergent Circuits

In convergent or tree-like circuits the exhaustive simulation method which is of exponential complexity can be avoided in favour of a "table" method which is considerably less complex. The key difference between a convergent circuit and a non-convergent one is that for a convergent circuit, one can always associate the set of transitions at each time unit in every waveform with another set of transitions in a uniquely corresponding time unit with respect to functionality. This is illustrated in Figure 16 where a *circuit time-unit* is as indicated. The circuit consists of three functional elements, each of unit delay. The transitions on a and b within the enclosed circuit time-unit are directly responsible for producing the transitions on e, similarly b and c for f, and e and f for g. This relationship is not always as unambiguous for non-tree circuits. In general, a time unit in one waveform cannot be uniquely associated with another time unit in another waveform as shown in Figure 17. The circuit in Figure 17 is a simple one of only three gates each of unit delay and exhibits reconvergent fanout. As can be seen there is no single unique circuit time-unit that can be associated with this circuit. Instead the temporal relationships between the nodes are complex.

For tree-like or convergent circuits the analysis can proceed one *circuit time-unit* at a time. The idea is essentially to avoid the exponential complexity of a full simulation, where every possible simple waveform on every node would have to be enumerated, by combining results obtained at intermediate stages. The complexity with this technique is limited to 4^n where n is the number of primary inputs. An example will serve to illustrate this technique best. Consider a trivial circuit, a single OR gate, with waveforms on the inputs as shown

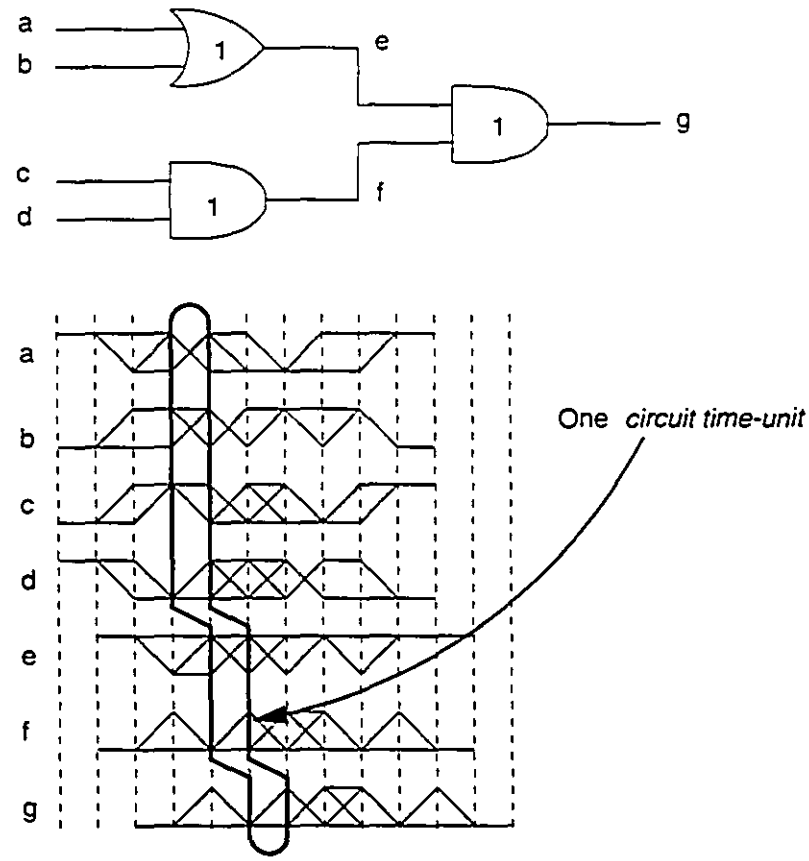


Figure 16: Temporal relationship between waveforms in tree circuits

in Figure 18. The 'table' that will be constructed for this case is Table 1. The 3rd and 4th columns in the table list all the possible transitions on the inputs of the OR gate. The 5th column lists the resultant output. The first column assigns a unique number, the 'index', to each input combination for easy reference. The second column lists the number of transitions for that set of inputs and output. The 'Links' column lists, for each input and output set, the set of possible inputs which could have occurred in the previous time unit, using the index. For instance input 00, index 1, can only be preceded by any one of 00, 0f, f0 and ff as indicated by the indices 1, 4, 13 and 16 in the 'Links' column. The analysis commences by considering the input transitions for time 1. From Figure 18 the possible inputs are 01 and r1. The number of transitions for each input combination is written into the appropriate row in the column for time 1. From time 2 onwards the procedure is slightly different. At time 2 the possible inputs are 01, 0f, 11 and 1f. For each of these the 'Links' column is used to

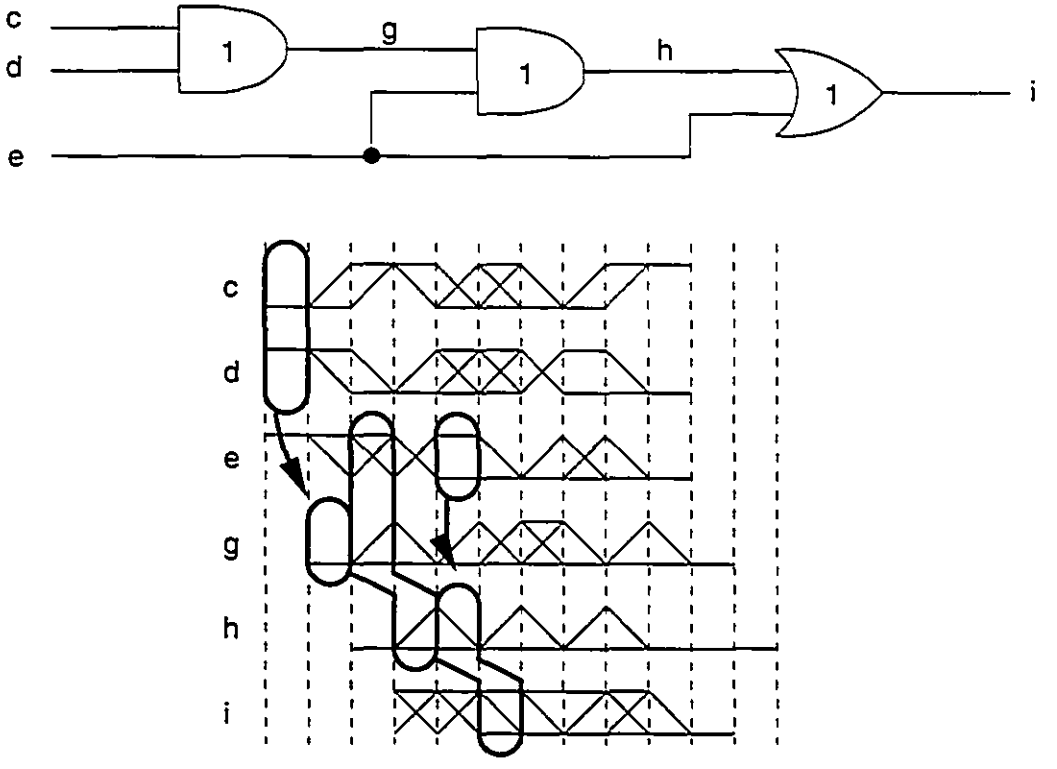


Figure 17: Temporal relationship between waveforms in non-tree circuits

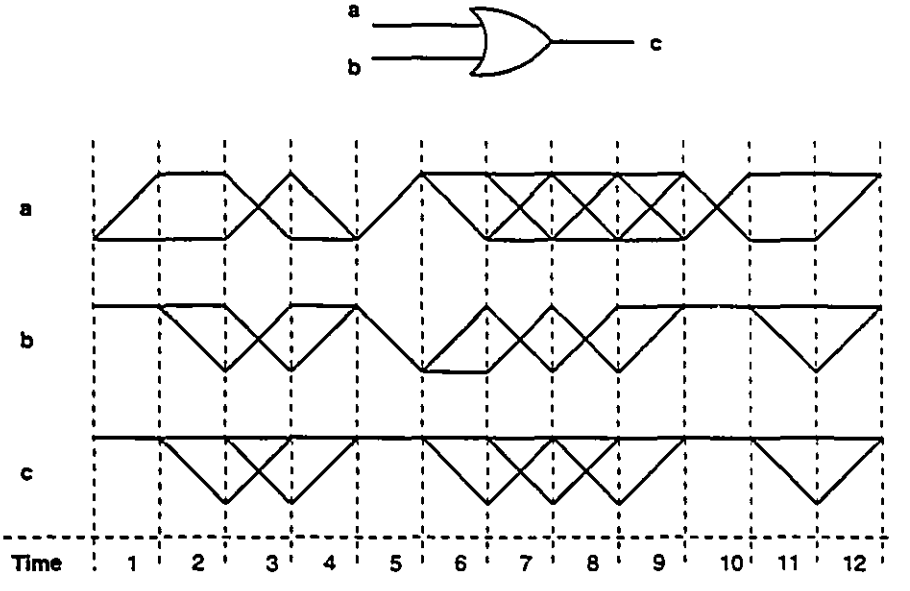


Figure 18: Trivial circuit of a single OR gate

Table						Discrete time units											
Index	# transitions	Input a	Input b	Output c	Links	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	0	1, 4, 13, 16												
2	2	0	r	r	1, 4, 13, 16				**6**			12	14	18			
3	0	0	1	1	2, 3, 14, 15	**0**	0		4					14		20	
4	2	0	f	f	2, 3, 14, 15		**2**					12	14			**22**	
5	2	r	0	r	1, 4, 13, 16												
6	3	r	r	r	1, 4, 13, 16			**5**				**13**	15	**19**			**25**
7	1	r	1	1	2, 3, 14, 15	**1**								15	19		21
8	2	r	f	1	2, 3, 14, 15			2		**8**		12	14				
9	0	1	0	1	5, 8, 9, 12						9						
10	1	1	r	1	5, 8, 9, 12						9	9	13	15			21
11	0	1	1	1	6, 7 10, 11		**1**							15		19	19
12	1	1	f	1	6, 7 10, 11		2					10	14			20	
13	2	f	0	f	5, 8, 9, 12						**10**						
14	2	f	r	1	5, 8, 9, 12			4	4		10	10	14	16			
15	1	f	1	1	6, 7 10, 11				**6**					16	**20**		
16	3	f	f	f	6, 7 10, 11			**4**				12	**16**				

Table 1: Analysis of sample convergent circuit in Figure 18

determine all the possible inputs in the previous time unit. The highest number of transitions is picked from that and added to the number of transitions in the current input transition. This is entered in the appropriate row for the current time. For instance, for input 0f at time 2, which has 2 transitions (including the output f), the only possible input in time 1 is 01 (with 0 transitions since the output is 1). The other possible inputs are 0r, fr and f1 but there are no entries in the table for these inputs because they cannot possibly exist at time 1. The entry for 0f at time 2 is then 2 (2 transitions at time 2 + 0 transitions at time 1). This process continues for all the remaining time units. For instance for input rf at time 5, the possible inputs, from examining the 'Links' column for rf, at time 4 are 0r, 01, fr and f1. Of these the largest number of transitions at time 4 is 6. The number of transitions for input rf is 2 and hence the entry for input rf at time 5 is 6 + 2 = 8.

By choosing the largest entry in the preceding time unit the worst-case input in the number of transitions is being chosen. Effectively the worst-case input (and output) is chosen every time and result in the final column at time 12 is a number of possible worst-case number of transitions over the entire waveforms. At any point in time one can look up the table to ascertain the worst-case up to that point.

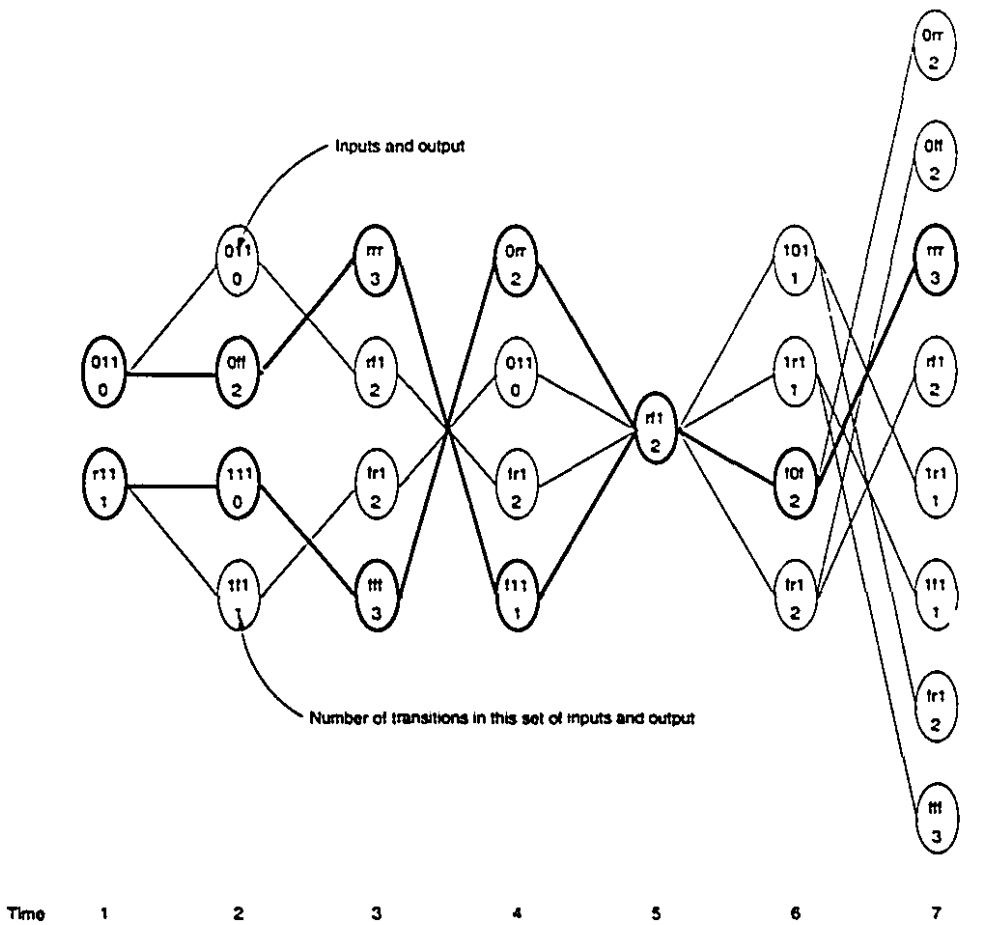


Figure 19: Inputs from time 1 to 7 and their 'links'

The possible inputs and the corresponding number of transitions for that input only are shown in Figure 19. Each node, representing inputs and an output at that time, is connected to the possible inputs (nodes) at the preceding and subsequent time units. Ignore the bold lines and nodes for the moment. The relationship between this tree and Table 1 is simple. In the table the entries are simply the sum of the number of transitions along any path in the tree. At nodes like rf1 at time 5 where there are four possible inputs at time 4, only the

highest value is taken to be the number of transitions at time 4. This is then added to the number of transitions for rf1 at time 5 to give the corresponding entry in Table 1.

The specific waveforms which give rise to the worst-case can be easily derived by traversing the table backwards from time 12. At time 12 pick the largest entry and, for that input, look up all the inputs pointed to by the links at time 11. Pick the largest entry of those links at time 11 and continue in this fashion until the time unit 1 is reached. The numbers in bold indicate the particular input-output combination at specific time points which lead to the desired waveform for this example. This is illustrated in Figure 19 by the bold lines and nodes. For instance, at time 7 pick the largest entry in the table which is 13 for the input rrr. The only possible input in time 6 for rrr is f0f so this chosen. Similarly for f0f the only possible input at time 5 is rf1 and this is therefore chosen. At this point, there are four possible inputs, 0rr, 011, fr1 and f11 at time 4 for the input rf1 at time 5. Only the highest values are to be chosen so both 0rr and f11 are chosen. The process continues in this manner taking each input, 0rr and f11, in turn. Compare this with the paths indicated by the bold lines in Figure 19.

This algorithm's running time is very fast, taking 0.04s on a Sparc 4 for the circuit in Figure 18, due to the fact that the table does not keep track of all possible simple waveforms over time but only the set of possible transitions any time unit. However, this technique is only applicable to small tree-like circuits and therefore limited in its usefulness. A simple version has been implemented in C as a concept demonstrator and no attempt has been made to optimize it in any manner.

3.2.2 Branch and Bound Algorithm

This is a technique of an exhaustive nature applicable to all circuits regardless of their topology since it essentially utilizes exhaustive analysis. For most circuits, this is the only way to determine the maximum number of transitions over all the nodes. It involves enumerating on each primary input of the circuit all the possible simple waveforms. The circuit is then analysed for all combinations of these simple waveforms. Clearly the complexity here is exponential because all possibilities are being tried. To ameliorate this, a branch and bound

technique is used. This involves attempting to predict during the analysis, of any set of simple waveforms on the primary inputs, whether it is in fact of any value to continue any further. This condition is evaluated at every point in time. If it can be discerned that further analysis would not be useful, the analysis of that set of simple waveforms is terminated. Analysis proceeds with consideration of other simple waveforms that have not yet been explored.

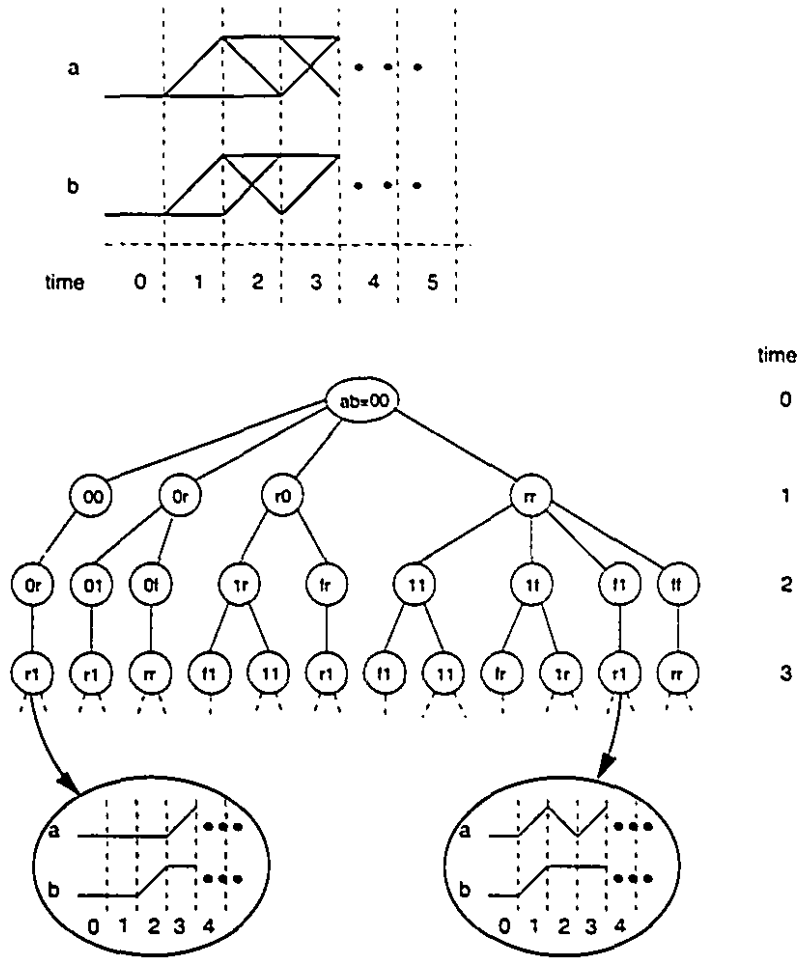


Figure 20: Sample analysis tree

The analysis proceeds by constructing a tree structure. Each node in the tree represents a particular set of transitions on all nodes of the circuit for a particular time unit. Each node has only one parent node but can have any number of child nodes. The child nodes represent the set of possible transitions in the next time unit from the parent node. This is illustrated in Figure 20. The diagram shows what an analysis tree would look like for the two waveforms a and b as shown at the top of Figure 20. At time 0 the only possible set of

transitions are 0 on both a and b. At time 1 there are r and 0 on both a and b which results in the four combinations 00, 0r, r0 and rr on a and b respectively as shown in the tree. Going down the tree in a depth-first fashion is equivalent to specifying a particular simple waveform on a and b as shown for two sample leaf nodes at the bottom of Figure 20. Although the figure would seem to imply that the tree is constructed breadth-first, this is not true. It is shown in this manner purely for explanatory purposes.

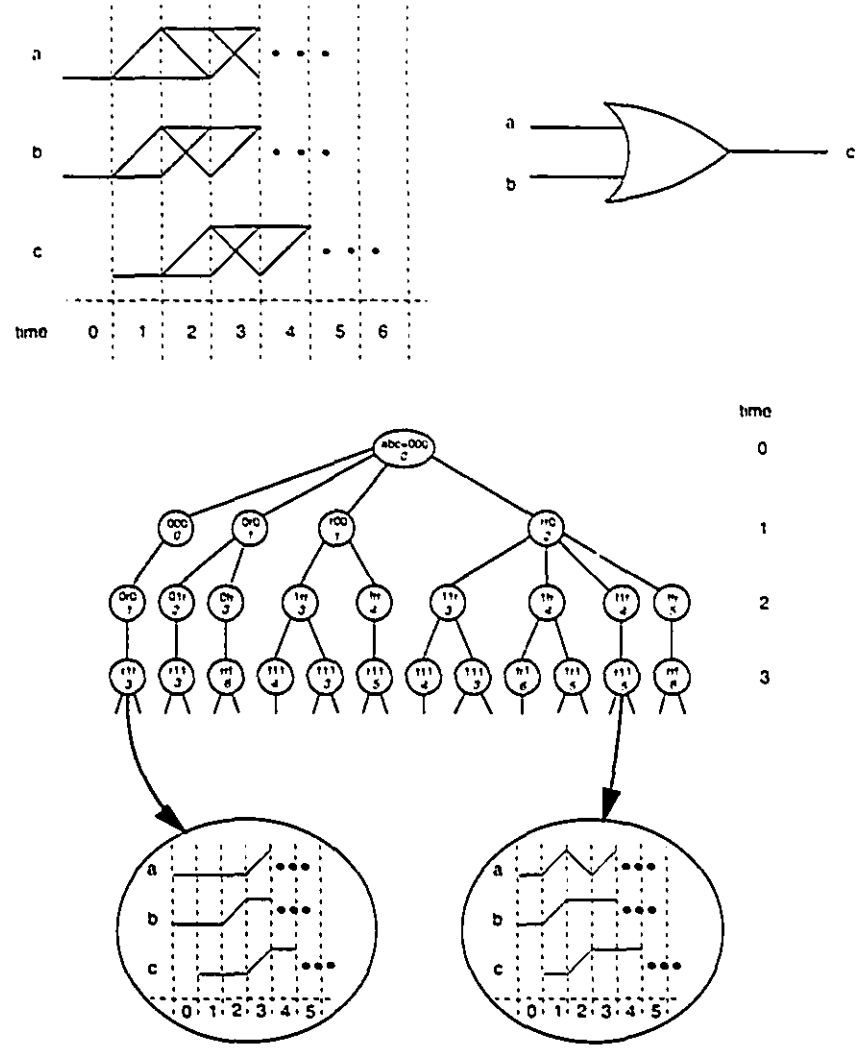


Figure 21: Sample analysis tree for an OR gate

When constructing such a tree for a circuit, each node would also store the values on all other nodes of the circuit at that time unit as well as the number of transitions so far. This is illustrated in Figure 21 where the analysis tree for a trivial circuit consisting of a single

OR gate is shown. There are two items in each node: the top item is the transition set on the inputs a and b and the output c, and the bottom item, a single number in italic font, is the number of transitions up to that time unit for the simple waveforms ending in that node. There are a number of other data items also stored in each node; these will be detailed later. With two primary inputs the maximum branching factor at each node is four. The branching factor, in general, is 4^n where n is the number of primary inputs.

The analysis tree is initially constructed in a depth-first manner. The objective is to explore to the leaf node with the largest number of transitions. The idea is to first carry out an analysis for one set of complete simple waveforms right down to the final time unit. This basically translates to exploring the tree once all the way down to one leaf node. The number of transitions at the final leaf node is noted. This number, denoted `maxODFsum` in the implementation, is then used as a basis of comparision for the rest of the tree exploration. It may be updated from time to time during the analysis. Depth-first exploration proceeds now with one additional constraint: a condition is evaluated prior to every branching. This condition decides whether continuing the analysis down that particular branch is a viable proposition. If it is not, then the entire subtree below that branch can be discarded. The condition can be stated as follows:

```
if (transitions(t) + worst-case_transitions(t)) > maxODFsum then
      branching is viable
else
      terminate analysis for this branch
```

`transitions(t)`	–	Number of transitions at the current node
`worst-case_transitions(t)`	–	Number of transitions in a maximal simple path from current time to final time

The value of `transitions(t)` is that which is kept within every node. However, `worst-case_transitions(t)` must be calculated prior to beginning any analysis. It is the number of transitions of a maximal simple path within the complex waveform from the current to the final time unit. It is certainly an overestimate since it is computed for the complex

waveform which is all the simple waveforms merged together. The reasoning behind this branching condition is that one is always looking for the largest possible value for the total number of transitions and there is no point in continuing down a particular branch if the worst-case number of transitions for that is lower than the current `maxODFsum`.

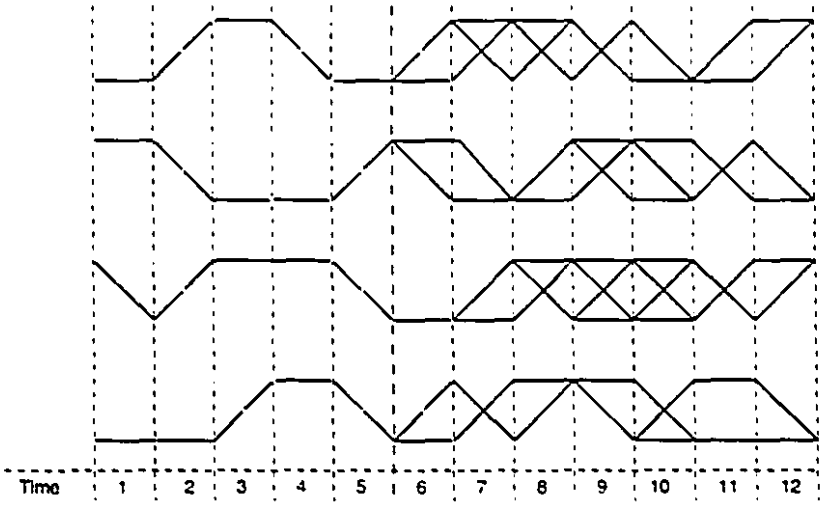


Figure 22: Waveforms at branching condition evaluation

This can be better understood by examining Figure 22 which is a figurative illustration of the state of analysis of a set of waveforms at a certain point in time. The analysis tree has been explored up to time 5 – simple paths exist for all waveforms up to and including time 5 and complex waveforms thereafter, which represent the unexplored part of the waveforms. The lines drawn in grey indicate the remainder of the original waveforms. At this point `transitions(t)` would have a value of 9 which is simply a count of the rising and falling transitions on all the simple paths from time 1 to time 5 inclusive. `worst-case_transitions(t)` would have a value of 21 which is the sum of the number rising and falling transitions in the maximal simple waveforms from time 12 to time 6 inclusive. From the branching condition, if `maxODFsum` is greater than 30 than the analysis would be terminated here; essentially the analysis upto time 5 has reduced the number of transitions overall so much that even assuming the worst-case from time 6 to 12 (ie. maximal simple paths on *all* waveforms) it still would not be worth it compared to the one particular analysis which resulted in a value of 30 for `maxODFsum`. Conversely if `maxODFsum` happened to be less than 30 then it implies that there is a possibility that this particular analysis, if continued further, might finally result in a number of transitions greater than the current `maxODFsum`. This possibility may well change

further down the analysis as the branching condition is evaluated at every point in time.

The algorithm for counting transitions on a single node comes into play when computing a value for `worst-case_transitions(t)`. Basically a table of values for `lpt0` and `lpt1` are created running along the complex waveform in reverse order, ie. proceeding from the final time unit to time 0. This table is created for every node in the circuit. If one proceeds down the table from time 0, it will give at any point in time the maximum, or worst-case, possible number of r and f transitions from that point onwards.

A particular branch of the analysis tree will be explored down to the final time unit only if the branching condition is consistently satisfied all the way down for every node. If the final leaf is reached, then the value of `maxODFsum` is updated to the new value of the number of transitions in the final leaf node. Effectively what is happening here is that a new set of simple waveforms on the primary inputs have been found which give rise to the largest number of transitions over the whole circuit. The analysis then proceeds as before. The value of `maxODFsum` will either increase or remain the same as the tree coverage increases. The larger the value of `maxODFsum` the more likely it is that the subtrees discarded will be larger. This is because the larger `maxODFsum` allows an earlier, or closer to the root of the tree, "prediction" of whether or not branching down any particular subtree is viable. It would seem that the analysis should be biased towards an exploration of those paths which are more likely to result in complete exploration to the final leaf nodes. In effect what is required is to accord a higher priority to branching in the direction of increased switching activity or r and f transitions.

A heuristic which attempts to produce exactly this bias has been implemented in addition to the branching condition. Every node whose children have yet to be explored holds a list of child nodes. As each child node is explored, that child node is removed from its parent node's list. The heuristic simply orders the list of child nodes in the order of decreasing number of transitions. Since each (child) node represents a set of transitions on the primary inputs, this in effect biases the analysis towards the more "active" simple waveforms. The assumption behind this is that the activity on the primary inputs will be propagated through the circuit. In general this assumption would seem to make sense since to have activity within the circuit one would first have to have it on the inputs.

The effect of the branching condition is somewhat unpredictable in the sense that it cannot be said precisely beforehand what percentage of the total tree will not have to be explored because of its use. It remains very much a heuristic; its effect is known but not its performance which will vary with the circuit and the waveforms on the nodes. The same holds for the ordering heuristic, for which it is known only that it gives a better performance in terms of the time taken for the analysis to complete, compared to random branching.

Clearly the larger the circuit the larger will be the analysis tree to be explored. The analysis tree is simply the means by which analysis is carried out. It is not necessary to maintain the tree or part of it in memory for any longer than it's needed. As the depth-first exploration progresses, nodes which have been explored are deleted to reduce memory usage.

The discussion so far has centred on circuits with one complex waveform on each of their nodes and this is the basic form of analysis. But each node in fact has an *abstract* waveform which is a set of four complex waveforms organized by class. This is true initially at least. As case analysis proceeds, however, nodes may not contain the full complement of four classes as constraints are progressively imposed upon the circuit. Transition counting on a sub-circuit takes cognizance of this situation and first performs what is termed as *class analysis*. This involves enumerating all possible combinations of the existing classes on all inputs of the sub-circuits and computing the corresponding classes on the remaining sub-circuit nodes, effectively restricting the nodes to a single complex waveform each.

An example of a simple circuit and some possible abstract waveforms are shown in Figure 23. Enumerating all possible combinations of classes on the inputs **A**, **B** and **D** and computing the corresponding classes which can exist on the remaining nodes for the circuit in Figure 23 would give the results shown in Table 2. Each of the entries in the table is referred to as a *class-set*. The sum of the transitions of the maximal simple paths on all the nodes is then taken for every class-set with the values as shown in the table. This list is then sorted in order of decreasing number of worst-case transitions and would then appear as shown in Table 3.

Since the objective is to determine the largest possible number of transitions over the set of circuit nodes, it is not usually necessary to analyse all the class-sets. The analysis proceeds by taking groups of class-sets which have the same overall number of worst-case

Inputs			Nodes		Worst-case
A	B	D	C	E	transitions
c00	c00	c01	c00	c01	8
c00	c00	c10	c00	c10	12
c00	c01	c01	c00	c01	7
c00	c01	c10	c00	c10	11
c00	c11	c01	c00	c01	8
c00	c11	c10	c00	c10	12
c01	c00	c01	c00	c01	7
c01	c00	c10	c00	c10	11
c01	c01	c01	c01	c01	5
c01	c01	c10	c01	c11	8
c01	c11	c01	c01	c01	6
c01	c11	c10	c01	c11	9

Table 2: Initial list of class-sets for the circuit in Figure 23

Inputs			Nodes		Worst-case
A	B	D	C	E	transitions
c00	c00	c10	c00	c10	12
c00	c11	c10	c00	c10	12
c00	c01	c10	c00	c10	11
c01	c00	c10	c00	c10	11
c01	c11	c10	c01	c11	9
c00	c00	c01	c00	c01	8
c00	c11	c01	c00	c01	8
c01	c01	c10	c01	c11	8
c00	c01	c01	c00	c01	7
c01	c00	c01	c00	c01	7
c01	c11	c01	c01	c01	6
c01	c01	c01	c01	c01	5

Table 3: Sorted list of class-sets

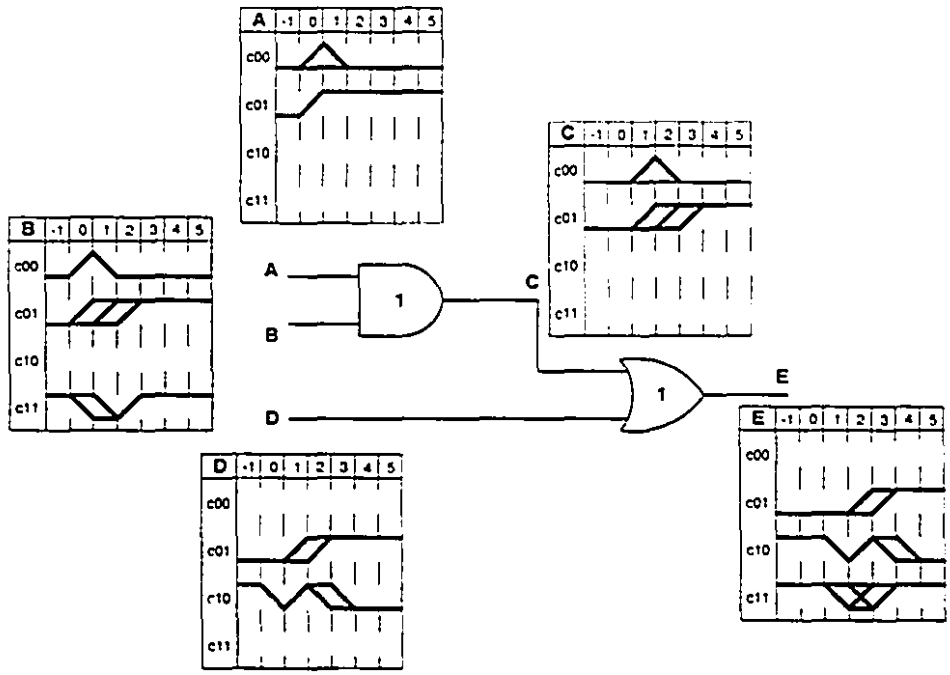


Figure 23: Abstract waveforms on a circuit prior to transition analysis

transitions. Each *class-group* is identified by its worst-case number of transitions, or *class-group value*. So for instance, *class-group 12* would refer to the first two entries in table 3. The transition counting algorithm proceeds in class-groups at a time – all the class-sets within a class-group are analysed and only the largest result (number of transitions) is retained as the *group exact count*. If this result is equivalent to the class-group value then clearly no further analysis is necessary. If the result is greater than the class-group value of the next class-group, then again the analysis can be terminated at this point. If the group exact value is less than the next class-group value then the analysis continues with consideration of the next class-group. The process is best understood by examining the pseudocode below:

```
next_cgv = class_set[0].trans;          // next class group value
curr_gev = 0;                           // current group exact value
i = j = 0;
while (i < num_class_sets)
{
    while ((next_cgv == class_set[j].trans) && (j < num_class_sets))
        next_cgv = class_set[++j].trans;
```

```
                    // next_cgv now contains the class group value of the
                    // next group in the list
    while ((i < max_num_class_sets) && (i < j))
    {
        curr_gev = max(curr_gev, analyse_circuit(class_set[i]);
                    // analyse_circuit performs the branch & bound
                    // analysis upon the circuit given a certain class-set
        i++;                            // increment index to next class-set
        if (curr_gev >= (class_set[i] - 1))
            return (curr_gev);
    }
    if (curr_gev >= next_cgv)
        return (curr_gev);
}
return (max(next_cgv, analyse_circuit(class_set[++i])));
                    // Takes care of boundary condition in while loop
```

For the circuit in Figure 23 with Table 3, class-group 12 would be analysed first. If the result of analysing the first class-set was 12 this would be the value returned immediately. Similarly if the value of the analysis was 11, this would be taken as the final value since this is guaranteed to be the largest given the list as in Table 3. But suppose the highest value obtained from analysing class-group 12 was 8, it would then be necessary to analyse class-group 11. If the analysis of class-group 11 resulted in a a group exact value of 10 then this would be returned as the final value since it is greater than the next class group value which is 9.

Where precisely the exhaustive analysis algorithm would fit within the entire system described in Section 2 has not been decided yet. That would depend partly on the time the algorithm takes to run. If it doesn't take too long then it might conceivably be used at every point in the case analysis when a transition count is desired; essentially immediately following the evaluation of a set of constraints. More accurate transition counts at every step have the potential to alter the exploration of the case analysis tree substantially. However, if the run times are considered too long to allow exhaustive analysis at every step, a more sparing use would have to be considered. This could involve some kind of criterion such as performing

the analysis for only one of the four classes that are evaluated at each step. Further research and experimentation is required in this aspect of the implementation.

The factors that affect the run times of the exhaustive analysis will be discussed in detail in the results section but for now it is important to realize that much depends on what kind of sub-circuits are picked. The user can control this through a set of command-line parameters. Thus, what parameters are specified must be taken into account when deciding whether to use exhaustive analysis sparingly or not.

4 Sub-circuit Picking Algorithm

As discussed earlier, the branch and bound algorithm is meant to be applied to small sub-circuits within a larger circuit. These sub-circuits have to be chosen judiciously to ensure the best performance from the transition counting algorithm. If the circuit is too large, too much time may be spent on computation. On the other hand if the circuit is too small, the gains from any analysis may be paltry. The best type of sub-circuits would be those of 'medium' size exhibiting reconvergence where there is a clear opportunity to enforce functional consistency between the nodes. Attempting to partition the entire circuit is inadvisable as most of the sub-circuits would undoubtably contribute little if anything primarily because most circuit topologies are unsuited for the kind of analysis being conducted. Among those sub-circuits identified, sharing of nodes or gates must be avoided since the branch and bound algorithm provides a transition count over an entire sub-circuit and shared gates or nodes would lead to erroneous overestimation.

As a preliminary step in investigating the performance of the transition counting algorithm, a procedure to identify sub-circuits was developed (the pseudocode is given in Appendix A). This is based on a simple greedy strategy; no overall topological analysis is conducted. The algorithm picks sub-circuits based on a list of nodes of the circuit sorted in decreasing order of fanout. The idea is to take a node and pick all the gates connected to it up to a certain logic depth. This then forms a sub-circuit. The gates surrounding this region are specially marked to form a boundary. Other sub-circuits may not contain the gates in this boundary and thus there are no overlapping regions. Sub-circuits are 'picked' by marking them with a unique identification number. Each sub-circuit has its own number; all the gates in that sub-circuit have that ID. Once a sub-circuit has been picked, it is then 'delimited'. This means marking all the gates which are connected to all the gates in a sub-circuit with a special ID number; these gates form the boundary.

The user must supply a number of parameters at the command line. Each of these are described below:

lldepth: This specifies the logic depth to which gates should be picked from a node. A good value is 3 or 4. A higher value means larger sub-circuits.

minfanout: Specifies that nodes being examined must have a fanout degree at least equivalent to or greater than this value. Nodes with fanout are attractive because sub-circuits containing such nodes may lead to regions of convergence or reconvergence both of which are good prospects for imposing functional consistency.

maxfanout: Specifies that nodes being examined cannot have a fanout degree greater than this value. There are two phases of sub-circuit picking: in the first phase only nodes which have fanout between *minfanout* and *maxfanout* are examined. In the second phase, all nodes which have a fanout degree greater than *minfanout* are examined. The rationale behind the fanout conditionality is that nodes with very high fanout are likely to result in large sub-circuits which is not desirable.

max_rPI: Only sub-circuits which have primary inputs equal to or below this value will be analysed. A good value for this is usually 6 though this may vary.

max_subPI: The first node from which the gate picking begins has a fanout restriction set by *max_rPI*; subsequent nodes, further down the logic depth, have a fanout restrcition set by *max_subPI*. This is way of limiting the size of sub-circuits.

mingates: This restricts the minimum size of sub-circuits in terms of gates. If a region is identified and found to be below this value, the region is unmarked and the gates and nodes will be available again to be picked as part of another sub-circuit. This is useful because small sub-circuits of 2 or 3 gates often identified are of no value for the transition counting algorithm.

The algorithm has a number of shortcomings, the main one being that it is not specifically targetted towards identifying regions of convergence or reconvergence. Where it does manage to include reconvergent regions in a sub-circuit it often picks additional gates that contribute little to the results. An example of this is illustrated in Figure 24. Ignore the functionality of the circuit for the moment, and consider simply the topology. If the gate picking begins at node *a* with the logic level depth set to 3, then the circuit within the broken line will be picked. However, it is only the circuit within the dotted lines that will really be useful in the exhaustive analysis. The remaining three gates cannot be expected to contribute anything much as far as reducing the pessimism in switching activity is concerned. The result is that the exhaustive analysis will not give as good results as it could have. The superfluous gates merely increase the computation time.

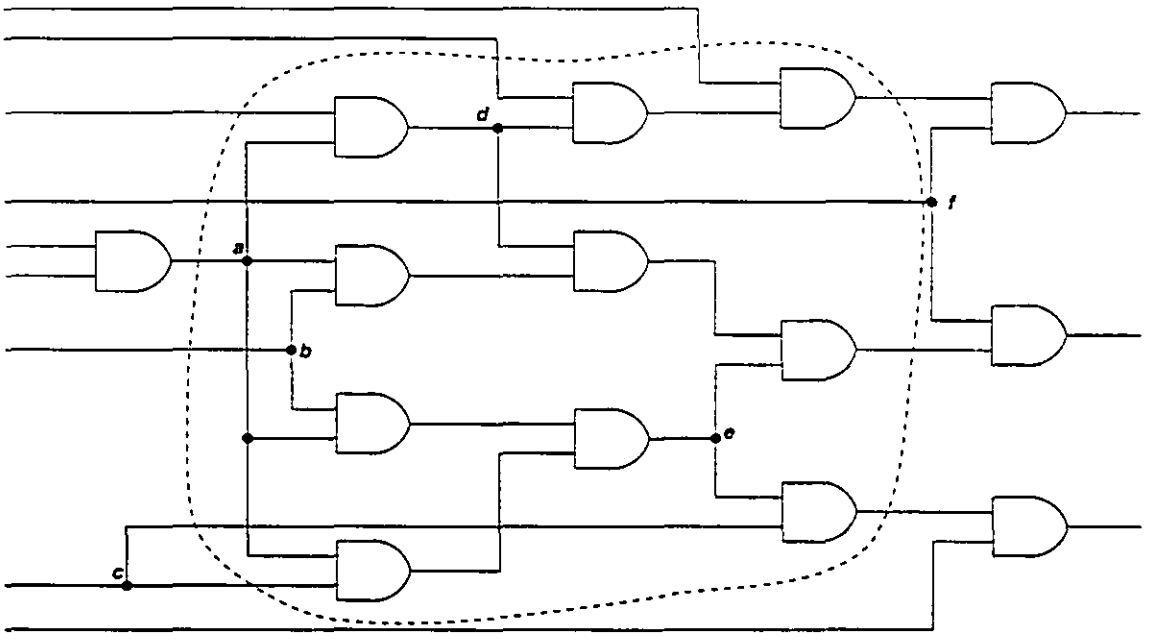


Figure 24: Sub-circuit picked with node *a*

The nodes from which the gate picking commences are very important in defining the sub-circuit that is picked. The current technique of prioritizing nodes according to their fanout is inadequate as demonstrated by the circuit in Figure 25 which can be assumed to be part of a larger circuit. If the sub-circuit-picking algorithm is applied to node *z* with the logic level depth set to 2 then the region within the broken line is the one identified.

Node *z* is the one most likely to be picked first as it has a higher fanout than nodes *x* or *y* or *w*. However, if either of nodes *x* or *y* were picked first then the sub-circuit picked would be as shown in Figure 26. Clearly the sub-circuit picked in Figure 26 encompasses more reconvergence then than that in Figure 25 and would be a better choice for the transition counting algorithm. This points out the insufficiency of merely considering nodes in order of their fanout.

When experimenting with the algorithm by varying parameters, it was found that a number of small sub-circuits of 1 or 2 gates were consistently being identified. These sub-circuits were quite useless to perform exhaustive analysis upon. This problem was dealt with by having a parameter, *mingates* which would restrict sub-circuit size above a certain minimum number of gates. The small sub-circuits were often being picked from the areas

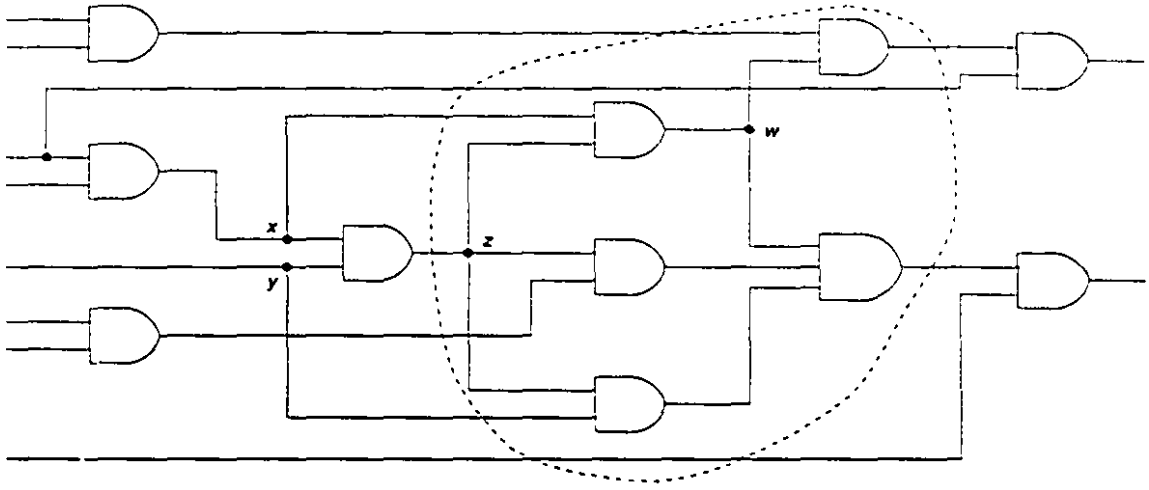


Figure 25: Sub-circuit picked with node z

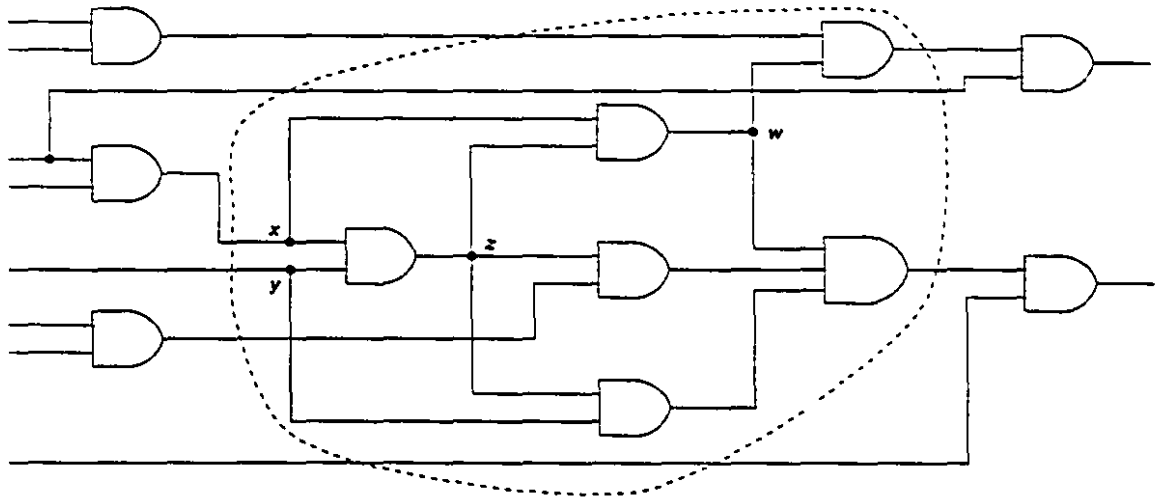


Figure 26: Sub-circuit picked with node x or y

between two or more larger sub-circuits; in a sense these were the 'leftover' gates. As such they have no inherent attractive topology that could be exploited.

Another serious shortcoming is that sub-circuits identified are too large to be exhaustively analysed. All sub-circuits which exceed *max_rPI* will be passed over by the transition counting algorithm. Attempts were made to deal with this problem by keeping track of the number of sub-circuit primary inputs, sPIs, as the picking proceeded but it was found this wasn't possible without an inordinate amount of computation. As this would have made

the algorithm very time-consuming, this approach was abandoned. A technique employing a heuristic was then attempted; this involved making an approximation of the number of sPIs the final sub-circuit would have during the picking process. However, the results the heuristic gave turned out to be too unpredictable to be of much use. The idea was that if one had an estimate of the number of sPIs during the gate picking process, one could then terminate the picking once the estimated number of sPIs reached a certain value (possibly close or equal to the user specified maximum number of sPIs).

This greedy sub-circuit picking algorithm is meant to be a simple one in the absence of a more comprehensive one which would accurately identify reconvergent regions in a circuit. As the results in the following section show, the transition counting algorithm for a set of nodes gives best results when presented with reconvergent regions.

5 Results

The algorithms for counting transitions over a sub-circuit and picking sub-circuits were implemented in the timing and power verification tool. Prior to conducting case analysis, sub-circuits are identified and analysed. A summary of the results is presented in Table 4. The

Circuit	# gates # nets	Number of sub-circuits	Average Size		Time to pick (s)	Time to count (min)	Reduction in count (%)	
			Gates	PIs			Sub-circuits	Overall
c880	383	15	5	3	0.63	15.0	10.4	2.8
	443	20	4	3	0.63	8.7	12.0	3.2
		15	5	3	0.66	1277.3	9.5	1.6
c432	160	9	4	3	0.21	0.12	7.6	1.7
	196	9	5	4	0.06	0.37	16.2	3.5
		4	6	5	0.06	1.44	8.3	1.3
		6	5	4	0.22	0.19	18.9	2.8
		5	5	5	0.20	0.78	13.6	1.7
c499	202	7	3	4	0.16	0.19	6.0	1.0
	243	5	3	4	0.15	0.06	5.2	0.6
		6	3	4	0.14	0.13	5.5	0.9
c1355	546	6	12	5	0.41	19.6	10.3	1.0
	587	9	10	4	0.42	3.4	3.4	0.6
c2670	1193	37	5	3	4.29	3.5	4.9	1.2
	1426	31	5	3	4.20	8.9	5.5	1.2
		36	6	3	4.20	806.7	6.5	1.6
		33	5	3	4.24	2.5	5.5	1.2

Table 4: Summary for some ISCAS circuits

table lists some of the ISCAS '85[1] circuits and results for each of them. All circuits have more than one set of results because the sub-circuit picking algorithm was run with different parameters each time. Depending on what parameters were set, the sub-circuits identified can vary. The number of sub-circuits can vary as well as the size (in terms of gates, nets and PIs). The time taken to pick sub-circuits is shown in the sixth column and the time that the branch and bound algorithm takes to analyse all the sub-circuits is shown in the seventh

column. The last two columns list the percentage reduction in transition count from the initial count, obtained using the algorithm described in section 3.1, against the more exact count given by exhaustive analysis. Two comparisions are made: the 'Sub-circuits' column considers only the sub-circuits analysed while the 'Overall' column considers the entire circuit.

The results in Table 4 were obtained by calling the function performing the exhaustive analysis *prior* to case analysis. This means that all the waveform classes are present on all the nodes of the circuit and the maximum number of class-sets for each sub-circuit have to be analysed. The reduction in count is also therefore the lowest that one might obtain. As constraints are progressively imposed during case analysis and classes of waveforms disappear, exhaustive analysis takes less time to run as well as giving higher reductions in transition counts. In general the relationship is described by the graphs in Figure 27 based on preliminary experimentation. The exact shape of the graphs would vary depending on circuit and sub-

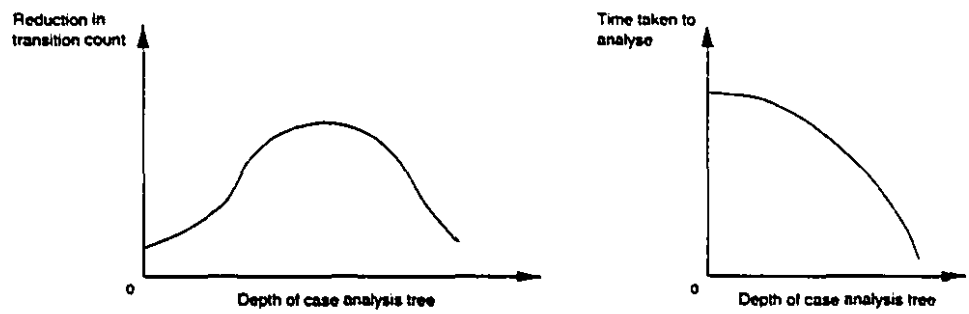


Figure 27: Performance of exhaustive analysis vs. case analysis tree depth

circuit topology, the order in which the constraints are imposed on nodes in case analysis and the size of the circuit. If the case analysis tree were explored in a purely breadth-first fashion, the performance would be exactly as depicted in Figure 27. However, a *frontier* approach is used and therefore there may well be significant deviations. Table 4 represents figures for the point at which the curves intersect the vertical axis. Up to a certain depth in the case analysis tree, reductions in transition count will increase after which they will start decreasing as the contraints increasingly narrow down possible activity within. However, the time taken by the analysis will decrease continuously with the tree depth as more constraints obviously mean less to analyse.

The overall reduction in count may seem meagre but these results are worst-case.

Based on some tentative investigation, it can be said that reductions in transition count once case analysis has proceeded for some depth are several times better than those in Table 4. Analysis times are similarly reduced.

Many of the sub-circuits identified do not show any reduction in transitions count at all. Had it not been for this fact, the average sub-circuit reduction (column 8 in Table 4) would have been higher. The performance of the branch and bound algorithm is very dependent on the topology of the sub-circuits identified. Those sub-circuits which exhibit reconvergence give the best results. However, the sub-circuit picking algorithm doesn't always manage to identify reconvergent regions. A detailed examination of the performance of both the algorithms will serve to illustrate this better.

5.1 Transition Counting over a Sub-circuit

The best results are obtained when the sub-circuit is entirely a reconvergent region. For instance the circuit in Figure 28 had a reduction in transition count of from 12 to 4 or 66.7%. A similar case, the circuit in Figure 29, had a reduction of 50%. These small circuits

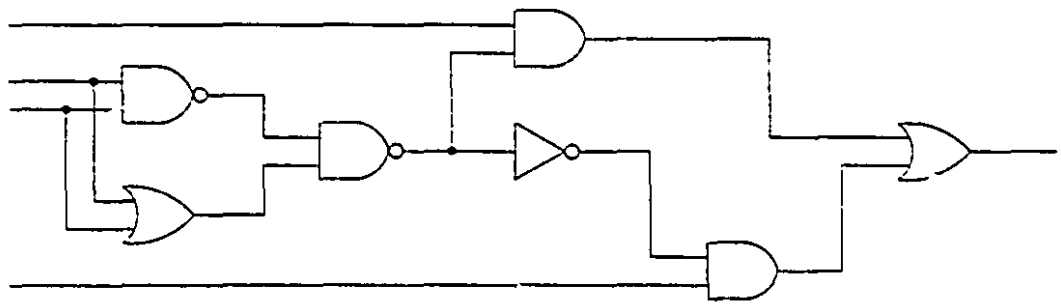


Figure 28: Sub-circuit exhibiting reconvergence from c880

show extremely good results. But combining two or more of these does not necessarily give an even larger reduction. The circuit in Figure 30 is basically a combination of two circuits of the type in Figure 29. But the reduction in transition count is only 14.6% which is considerably less than the 50% that was obtained for the circuit in Figure 29.

A rather larger circuit that had a reduction of 19.4% is shown in Figure 31. This

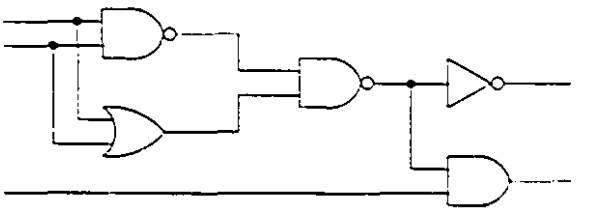


Figure 29: Sub-circuit exhibiting some reconvergence from c880

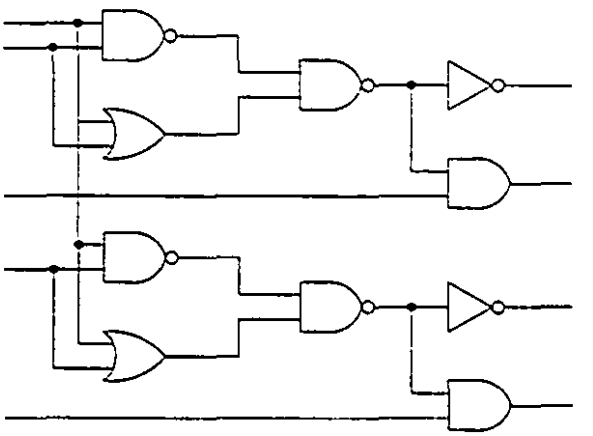


Figure 30: Sub-circuit from c880

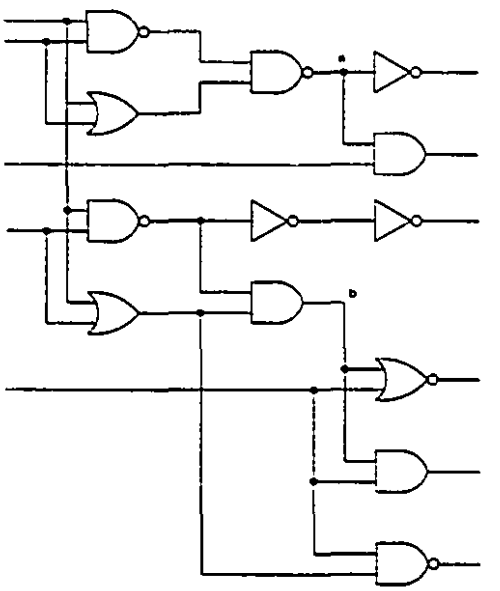


Figure 31: Large sub-circuit from c880

circuit has two reconvergent regions ending in nodes a and b as well as some superfluous gates. Regions which have no reconvergence at all will show very little reduction in their transition counts. The sub-circuit in Figure 32 showed a reduction of only 4.5%. The high

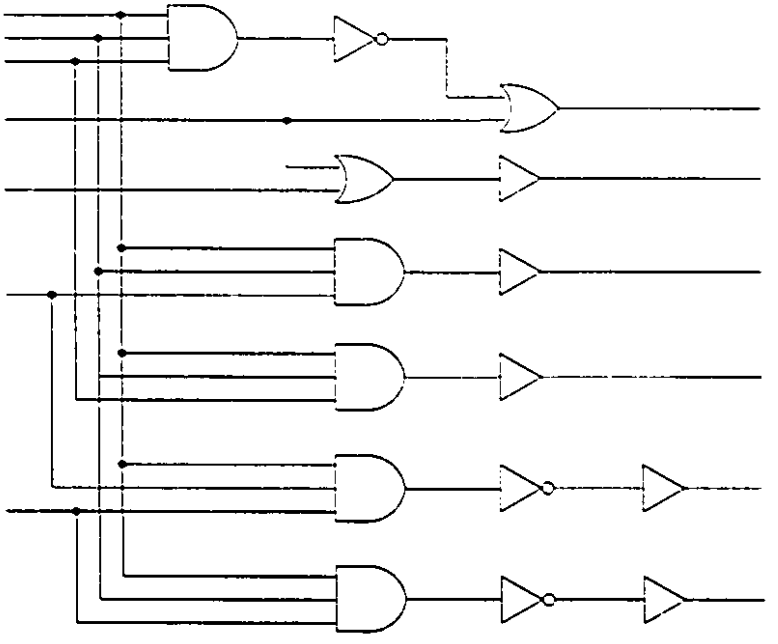


Figure 32: A sub-circuit with large fanout nodes

fanout degree of the input nodes allows some reduction to be achieved through accounting for correlation between waveforms. In general if the fanout is low then the reduction in transition count will be nil. The sub-circuit in Figure 33 for instance had no reduction at all in its transition count. It seems that among circuits that have no reconvergence, only those that

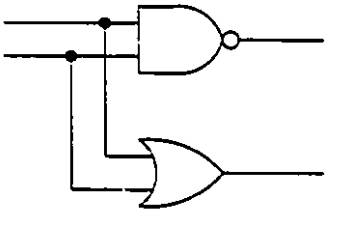


Figure 33: A sub-circuit with no reduction in transition count

have nodes with high fanout will show a little reduction in transition counts.

The sub-circuits illustrated so far were all taken from the results of analysing circuit

c880. It will be instructive to look at the detailed breakdown. The sub-circuit picking

Circuit	Sub-circuit	# gates	# nets	# sPIs	Worst-case	Exact	Time (s)	Reduction (%)
c880	1xx	16	30	14	sub-circuit too large to analyse			
	2	7	12	5	12	12	5.54	0.0
	3	8	15	7	sub-circuit too large to analyse			
	4	5	10	5	24	20	8.27	16.7
	5	5	8	3	8	6	0.32	25.0
	6	2	4	2	4	4	0.08	0.0
	7	10	15	5	39	35	6.94	6.94
	8	8	14	16	sub-circuit too large to analyse			
	9	20	31	11	sub-circuit too large to analyse			
	10	2	5	3	12	8	0.67	33.3
	11	2	5	3	13	13	0.28	0.0
	12	7	15	8	sub-circuit too large to analyse			
	13	12	24	12	sub-circuit too large to analyse			
	14	3	7	4	7	7	1.15	0.0
	15	4	7	3	63	44	434.69	30.2
	16	3	5	2	50	38	10.73	24.0
	17	3	5	2	32	24	0.65	25.0
	18	3	5	2	65	49	50.93	24.6
	19	2	4	2	8	6	0.08	25.0
	20	2	4	2	8	6	0.08	25.0
	21	4	7	3	9	9	0.32	0.0
	22	5	6	1	6	6	0.04	0.0
	23	5	8	3	13	13	0.32	0.0
	24	3	6	3	6	6	0.30	0.0
	25	3	6	3	6	6	0.29	0.0
	26	2	3	1	18	18	0.04	0.0
	27	10	24	14	sub-circuit too large to analyse			
Average	20	4	7	3	20	16	26.09	12.0

Table 5: Detailed results for c880

algorithm was run with the following parameters: *minfanout* = 2, *maxfanout* = 4, *lldepth* =

3, *max_rPI = 5*, *max_subrPI = 3* and *mingates = 1*. A total of 27 sub-circuits are identified in Table 5 of which only 20 are analysed. The rest are too large in terms of the number of PIs. Of the 20 that are analysed, fully half of them show no reduction at all. Most of these are similar topologically to those shown in Figures 32 and 33.

The entire analysis for c880 takes 521.7s of which 485.6s is occupied analysing sub-circuit 15 and 18 both of which are small reconvergent circuits with only a single output. The remaining sub-circuits take an average of 2s to analyse. This scenario, where a few sub-circuits monopolise almost the entire analysis time, is repeated a number of times. Two extreme cases of this nature can be seen in Table 4 for c880 taking 1277.3 minutes and c2670 taking 806.7 minutes to analyse. The breakdown of the analysis time shows that in c880 only two sub-circuits took 1276.1 minutes while the remainder took 69.7s. Of these two, one sub-circuit, shown in Figure 31, took 68.9 minutes while the other, shown in Figure 34, took 1207.2 minutes. Similarly in c2670 the circuit shown in Figure 35 took 792.1 minutes

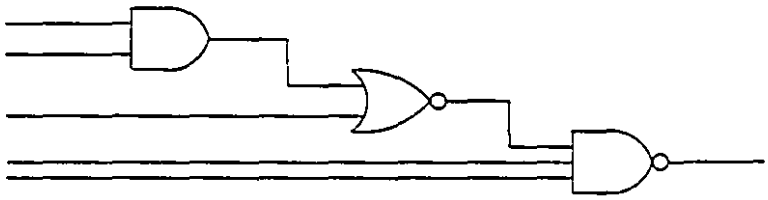


Figure 34: Sub-circuit from c880 which took the longest to analyse

to analyse. It is not clear why some circuits should take so much time while others take a

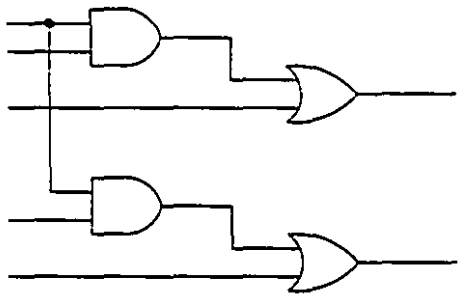


Figure 35: Sub-circuit from c2670 which took the longest to analyse

fraction of that. These circuits are not particularly large in terms of the number of gates or nodes. It can only be surmised that either the heuristic is at fault or the situation required

a large number of class-sets to be analysed. The heuristic may have failed completely in deciding which branches of the analysis tree are not worth exploring, resulting in the tree being fully explored.

It seems, from examining the table, that there is no one common denominator that would serve to indicate which circuits would achieve a greater reduction in transition count than another. The only conclusion in this regard that can be made is that circuits with two gates are of no use – such as the one in Figure 33. Among all the ISCAS circuits analysed, all sub-circuits which consisted of two gates experienced no reduction in their transition counts. Larger sub-circuits with similar results mostly had a single sPI. A few of these circuits are shown in Figures 36. Clearly there is not much room for any reduction in the transition count

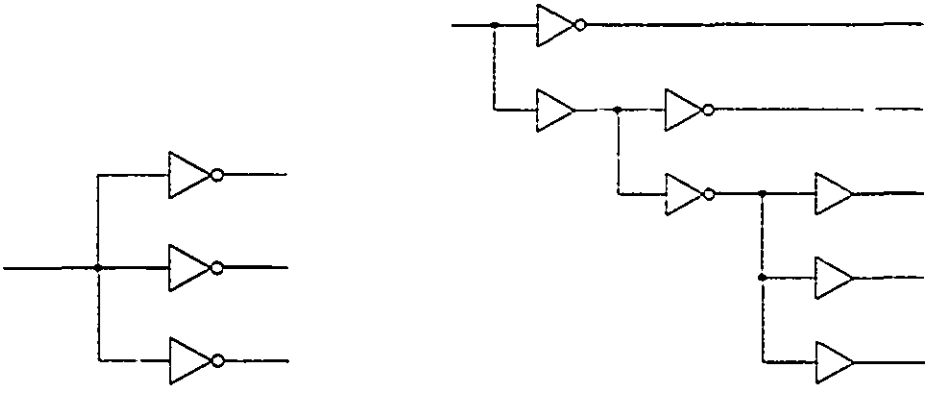


Figure 36: Sub-circuits from c2670 and c432

through any analysis. However, there were a few larger circuits which were reconvergent structures but experienced no reduction in transition count. These were all from the circuit c2670. One such circuit is shown in Figure 37. It was not clear why this should be the case.

Often circuits have regular structures and as a result the sub-circuits turn out to be identical, differing only in their location within the circuit. In c432 for instance, the results were as shown in Table 6. The first two sub-circuits were too large to be analysed and sub-circuit 11 is illustrated on the LHS of Figure 36. Sub-circuits 3 to 10 inclusive are topologically exactly alike, as in Figure 38. It seems from the identical results obtained for these sub-circuits that the waveforms on each of them may be the same as well. This could well be a recurrent feature of circuits with regular structures and analysis time could be saved by exploiting this

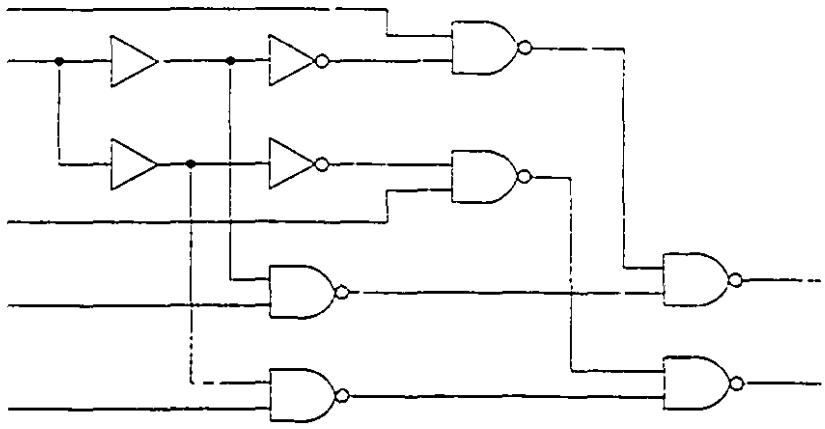


Figure 37: Large c2670 sub-circuit which experienced no reduction

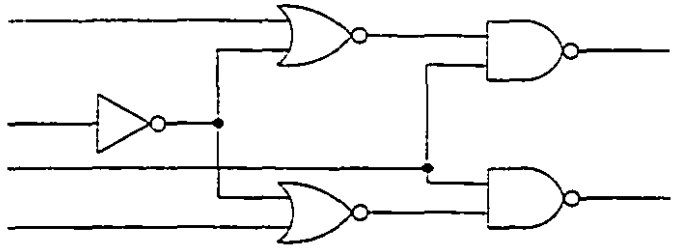


Figure 38: One of several identical sub-circuits from c432

since only one of such identical sub-circuits need be analyzed.

Small sub-circuits do not necessarily show no reduction in their transition counts; circuit c499 shows a low reduction as the results in Table 7 illustrates. The sub-circuits are very small in size, consisting of only two or three gates. It is not only size which plays a part but also functionality. In this case the sub-circuits were constituted entirely of XOR gates as shown in Figure 39. Sub-circuits consisted of only these two illustrated topographies. XOR

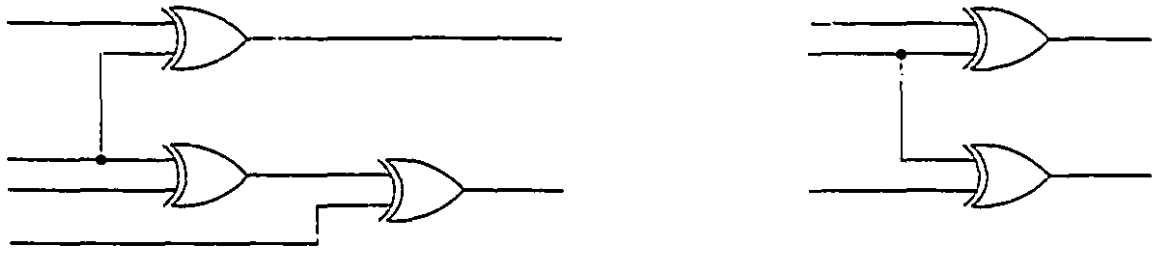


Figure 39: Typical sub-ciruits from c499

Circuit	Sub-circuit	# gates	# nets	# sPIs	Worst-case	Exact	Reduction (%)	Time (s)
c432	1	10	22	12	sub-circuit too large to analyse			
	2	12	27	15	sub-circuit too large to analyse			
	3	5	9	4	22	18	18.2	2.75
	4	5	9	4	22	18	18.2	2.74
	5	5	9	4	22	18	18.2	2.76
	6	5	9	4	22	18	18.2	2.73
	7	5	9	4	22	18	18.2	2.75
	8	5	9	4	22	18	18.2	2.75
	9	5	9	4	22	18	18.2	2.76
	10	5	9	4	22	18	18.2	2.75
	11	3	4	1	20	20	0.0	0.04
	Average	5	8	4	22	18	16.2	2.45

Table 6: Detailed results for c432

gates are clearly good candidates for transition analysis because they have no controlling value and hence no controlling transition. An examination of the transition evaluation table shown in Figure 40 will clarify this.

In summary, it would seem that the key factor which determines the extent of reduction in switching activity is topology. Sub-circuits which are reconvergent regions (eg. Figure 28) or contain reconvergent regions within (eg. Figure 29) give the best results. The next most influential factor would seem to be functionality as evidenced by the small simple circuits

Circuit	Sub-circuit	# gates	# nets	# sPIs	Worst-case	Exact	Reduction (%)
c499	1	3	7	4	20	19	5.0
	2	3	7	4	20	19	5.0
	3	2	5	3	18	17	5.6
	4	2	5	3	18	17	5.6
	5	3	7	4	20	19	5.0
	Average	5	3	6	19	18	5.2

Table 7: Detailed results for c499

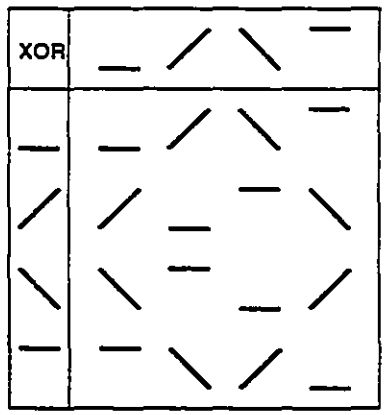


Figure 40: Transition evaluation for the XOR function

in Figure 39 showing reductions of the order of 5% even though these circuits exhibit no reconvergence and have very low fanout nodes. In particular XOR gates would seem to be the critical contributing factor here.

The results also indicate that fanout seems to be an element that comes into play at times. High fanout, as in the circuit in Figure 32, can have an effect although not a very substantial one. Last but not least, the waveforms on the nodes or nets are very important. Waveforms with few transitions may well not result in much reduction in transition counts. Obviously those with a higher number of transitions will have a greater chance of showing a reduction.

It is difficult to draw conclusions about the analysis times for sub-circuits from the results. In general, the larger the number of PIs to a sub-circuit, the longer it takes to analyse. However, there are sufficient deviations from this in the results to detract from any such statement. This aspect of exhaustive analysis certainly needs more investigation.

5.2 Sub-circuit Picking Algorithm

As described earlier the user specifies a few parameters which guide the algorithm towards the type of sub-circuits desired. For the ISCAS circuits analysed, it is generally most practical to consider nodes with fanout between 2 and 5 and logic level depths of 3 or 4. These settings seem to give the best results. Increasing the maximum fanout results in larger

Circuit	Sub-circuit	# gates	# nets	# sPIs	Reduction (%)
c432	1	10	22	12	sub-circuit not analysed
minfanout = 3	2	26	55	29	sub-circuit not analysed
maxfanout = 5	3	3	6	3	0.0
logic depth = 4	4	3	6	3	0.0
max_rPI = 6	5	3	6	3	0.0
max_subPI = 5	6	3	6	3	0.0
	7	3	6	3	0.0
	8	5	9	4	22.7
	9	5	9	4	22.7
	10	5	9	4	22.7
	Average	4	7	3	8.5
c432	1	10	22	12	sub-circuit not analysed
minfanout = 3	2	12	27	15	sub-circuit not analysed
maxfanout = 5	3	5	9	4	18.2
logic depth = 3	4	5	9	4	18.2
max_rPI = 6	5	5	9	4	18.2
max_subPI = 5	6	5	9	4	18.2
	7	5	9	4	18.2
	8	5	9	4	18.2
	9	5	9	4	18.2
	10	5	9	4	18.2
	11	3	4	1	0.0
	Average	5	8	4	16.2

Table 8: Detailed results for c432

sub-circuits which cannot be analysed by the transition counting algorithm in reasonable time

or exceeds the maximum number of sPIs specified. Setting the minimum fanout below 2 gives a large number of small circuits which give little or no reduction in switching activity. A similar reasoning holds for the logic level depth setting. Setting it much higher than 5 gives sub-circuits that are too large and setting it lower than 3 results in sub-circuits which are too small. It must be noted that these numbers may well be different for other circuits depending on topology. What is best can only be determined through experimentation. A good illustration of this are the results in Table 8 obtained for two different sets of parameters.

The first set of parameters differs from the second set only by the logic level depth which is 4 and 3 respectively and yet the results, in terms of transition count reduction, are markedly different. One would have expected the results to be the other way around since deeper sub-circuits would be expected to give better results. The reason for this state of affairs can be easily discerned. Sub-circuit 1 is identical for both sets of parameters but sub-circuit 2 is much larger when the logic depth is set to 4. The effect of this seems to have been that gates which might have been picked to be part of the rest of the sub-circuits are not available because they have been picked to be part of sub-circuit 2. Contrast this with the case when the logic depth is set to 3; sub-circuit 2 is too large to analyse but small enough that it does not take gates away from other sub-circuits. The sub-circuits analysed in the first set of results which showed a reduction of 22.7%, are all of the type shown in Figure 41 while the sub-circuits analysed in the second set of results are all of the type shown in

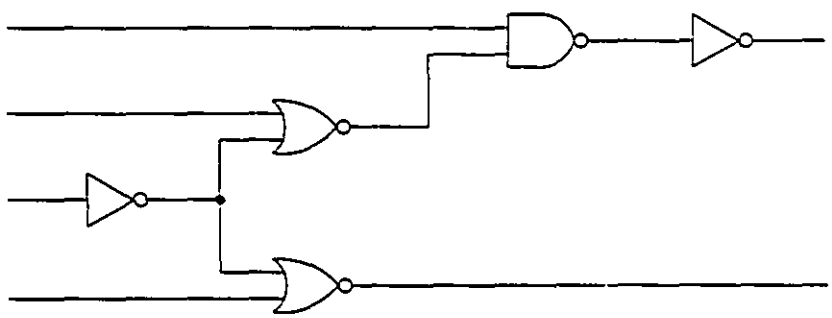


Figure 41: Typical sub-circuit from c432

Figure 38. In general a larger value for the logic level depth results in fewer sub-circuits which contain more gates, resulting in the transition counting algorithm giving a greater reduction in switching activity. However, changing the parameters further for c499 did not make much difference. The sub-circuits identified and the results were the same as those in Table 7. The

only thing that changed were the size of sub-circuits too large to be analysed.

One key point to note is that the *gate coverage*, the percentage of gates included in the sub-circuits compared to the size of the entire circuit, was on average 15% for all the ISCAS circuits considered. This is one reason the overall transition count reductions are so small: 15% of the gates cannot be expected to be contribute much. Higher overall reductions can be obtained if the fraction of gates analysed is increased, though this may lead to higher analysis times.

The running time of this algorithm is generally very small, under 1 second in most cases, though proportional to the size of the circuit. A listing of the run times is in Table 4. Most of the time is taken up in printing the results of the sub-circuit picking to the screen. This is especially true for c2670 which is a large circuit of 1193 gates and there are a large number of sub-circuits.

From the results, it is clear the performance of exhaustive analaysis is critically dependent on the topology of the circuits it analyses. Certain topologies, regardless of size, seem to cause the heuristics employed to fail resulting in a massive increase in analysis time. Considering how frequently it is envisaged exhaustive analysis will be used within the framework of case analysis, this is unacceptable. At present the only solution is to experiment with different settings for the sub-circuit picking and ascertaining that analysis takes a reasonable amount of time before proceeding with case analysis.

6 Conclusions

An algorithm has been presented for computing the maximum amount of switching activity in a given circuit while maintaining the functional relationships between nodes. The work is a contribution to the power verification tool described in [48, 47] and is based on the concepts elaborated therein. The algorithm has been demonstrated to work for small circuits. Results presented have shown that the amount of reduction in switching activity from the worst-case primarily depends on topological characteristics. In general, reconvergent circuits give the best results. However, in rare cases the algorithm may take an unacceptable amount of time to complete its analysis. The cause of this has yet to be determined but it appears to stem from a failure of the branch and bound technique thereby resulting in a complete exploration of the analysis tree. This tentative conclusion is borne out by the extremely large number of tree nodes that were created in these particular cases.

The second most influential factor in determining the performance of the algorithm seems to be the functionality of the sub-circuit. XOR gates, in particular, because of their lack of a controlling value, or transitions in this case, offer the greatest potential for a substantial reduction in switching activity. As shown in the results, circuits containing XOR gates, not withstanding their poor topology with regard to reconvergence or fanout, still manage to post a reduction in switching activity.

A heuristic was used to speed up the algorithm based on the assumption that a greater amount of switching activity on the inputs of a sub-circuit will translate into greater activity on the internal nodes. This improved the overall running time by a significant amount. However, the assumption this heuristic is based on may not be applicable to certain circuits where the functionality dictates otherwise. An obvious instance of this is a circuit composed of XOR gates.

Further work is needed to investigate other heuristics which might help speed up the analysis tree exploration. Aspects of the implemenation can be improved too. In particular, instead of enumerating when necessary the set of transitions possible in the next time unit, given the current set of transition on the inputs, it is possible to create a table of this information prior to commencing the tree-exploration. However, this information grows expo-

nentially and it is therefore necessary to develop a new way of reducing storage by exploiting redundancy.

A simple greedy algorithm for picking sub-circuits from a given circuit has been presented. The primary pupose of this algorithm was to allow a fast evaluation of the branch and bound transition counting algorithm in the absence of a more comprehensive circuit partinioning routine. The flexibility of this algorithm in specifying the characteristic of the sub-circuits being picked allowed investigation of what type of circuits would be best for transition counting analysis. The running time of this algorithm is negligible compared to the running time for entire tool.

The performance of this algorithm could be improved if a way is found whereby one could have a good idea of the number of sPIs the final sub-circuit would have during the gate picking process. This would allow termination of the gate picking before the sub-circuit grows too large. It would also leave more gates to be possibly picked to be part of other sub-circuits.

The running time of this algorithm is dependent on the topology and size of the circuits (in terms of the primary inputs) it is analysing. It is therefore necessary to have a good circuit partitioning routine not only to improve the running time but also for good results. Results presented have shown substantial reductions in switching activity for most circuits containing reconvergent regions, indicating the potential for even larger reductions should sub-circuits be entirely reconvergent rather than only partially.

7 References

[1] F. Brglez, H. Fujiwara, "A Neural Netlist of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN", *International Symposium on Circuits and Systems*, pp. 663-698, 1985.

[2] Lance A. Glasser, Daniel W. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Addison-Wesley Publishing Company, 1985.

[3] Olivier Lhomme, "Consistency Techniques for Numeric CSPs", 13th *IJCAI Conference*, 1993.

[4] J. W. McPherson, P. B. Ghate, "A Methodology for the Calculation of Continuous DC Electromigration Equivalents from Transient Waveforms", *Proceedings Symposium on Electromigration of Metals*, New Orleans, LA, pp. 64-74, October 7, 1984.

[5] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0", Technical Report, *Microelectronics Centre of North Carolina*, Research Triangle Park, North Carolina, January 1991.

Low-Level Techniques

[6] L. Benini, M. Favalli, P. Olovo, B. Riccò, "A Novel Approach to Cost-Effective of Power Dissipation in CMOS ICs", *European Design and Test Conference*, pp. 354-360, 1993.

[7] Na-Han Chan, "Rapid Current Analysis for CMOS Digital Circuits", Master's Thesis, McGill University, July 1994.

[8] Anthony M. Hill, Sung-Mo Kang, "Accuracy Bounds in Switching Activity Estimation", *IEEE Custom Integrated Circuits Conference*, pp. 73-76, 1995.

[9] Thomas Krodel, "PowerPlay - Fast Dynamic Power Estimation Based on Logic Simulation", *IEEE International Conference on Computer Design*, Cambridge, MA, pp. 96-100, October 1991.

[10] A. Nabavi-Lishi, N. C. Rumin, "Inverter-Based Models for Current Analysis of CMOS Logic Circuits", *IEEE International Symposium on Circuits and Systems*, pp. 13-16,

1994.

[11] A. Nabavi-Lishi, N. C. Rumin, "Delay and Bus Current Evaluation in CMOS Logic Circuits", *Proc. IEEE International Conference on Computer-Aided Design*, pp. 198-203, November 1992.

[12] A. Nabavi-Lishi, N. C. Rumin, "Inverter Models of CMOS Gates for Supply Current and Delay Evaluation", *Proc. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 10, pp. 1271-127, October 1994.

[13] Jiing-Yuan Lin, Tai-Chen Liu, Wen-Zen Shen, "A Cell-Based Power Estimation in CMOS Combinational Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 304-309, 1994.

[14] F. Rouabti, B. Haroun, A. J. Al-Khalili, "Power Estimation Tool for Sub-Micron CMOS VLSI Circuits", *IEEE/ACM International Conference on Computer-Aided Design*, pp. 204-209, 1992.

[15] G. Ruan, J. Vlach, J. Barby, "Current-Limited Switch-Level Timing Simulator for MOS Logic Networks", *IEEE Transactions on Computer-Aided Design*, Vol. 7, No. 6, pp. 659-667, 1988.

Statistical Techniques

[16] R. Burch, F. N. Najm, P. Yang, T. N. Trick, "A Monte Carlo Approach for Power Estimation", *IEEE Transactions on VLSI Systems*, Vol. 1, No. 1, pp. 63-71, March 1993.

[17] Tan-Li Chou, Kaushik Roy, "Statistical Estimation of Sequential Circuit Activity", *IEEE International Conference on Computer-Aided Design*, pp. 34-37, 1995.

[18] B. Kapoor, "Improving the Accuracy of Circuit Activity Measurement", 31st *ACM/IEEE Design Automation Conference*, San Diego, California, pp. 734-739, 1994.

[19] F. N. Najm, S. Goel, I. N. Hajj, "Power Estimation in Sequential Circuits", 32nd *ACM/IEEE Design Automation Conference*, San Francisco, California, pp. 635-640, June 1995.

[20] M. G. Xekellis, F. N. Najm, "Statistical Estimation of the Switching Activity in Digital Circuits", 31st *ACM/IEEE Design Automation Conference*, San Diego, California, pp. 728-733, 1994.

Probabilistic Techniques

[21] David I. Cheng, M. Marek-Sadowska, Kwang-Ting Cheng, "Speeding up Power Estimation by Topological Analysis", *IEEE Custom Integrated Circuits Conference*, pp. 623-626, 1995.

[22] Tan-Li Chou, Kaushik Roy, Sharat Prasad, "Estimation of Circuit Activity Considering Signal Correlations and Simultaneous Switching", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 300-303, 1994.

[23] Mehmet A. Cirit, "Estimating Dynamic Power Consumption of CMOS Circuits", *IEEE Intl. Conference on Computer-Aided Design*, pp. 534-537, November 9-12, 1987.

[24] A. Ghosh, S. Devadas, K. Keutzer, J. White, "Estimation of Average Switching Activity in Combinational and Sequential Circuits", 29th *ACM/IEEE Design Automation Conference*, Anaheim, California, pp. 253-259, June 8-12, 1992.

[25] Radu Marculescu, Diana Marculescu, Massoud Pedram, "Switching Activity Analysis Considering Spatiotemporal Correlations", *Proceedings International Conference on Computer-Aided Design*, pp. 294-299, 1994.

[26] Radu Marculescu, Diana Marculescu, Massoud Pedram, "Efficient Power Estimation for Highly Correlated Input Streams", 32nd *ACM/IEEE Design Automation Conference*, San Francisco, California, pp. 618-622, June 1995.

[27] H. Mehta, M. Borah, R. M. Owens, M. J. Irwin, "Accurate Estimation of Combinational Circuit Activity", 32nd *ACM/IEEE Design Automation Conference*, San Francisco, California, pp. 618-622, June 1995.

[28] F. N. Najm, R. Burch, P Yang, I. N. Hajj, "Probabilistic Simulation for Reliability Analysis of CMOS VLSI Circuits", *IEEE Transactions on Computer-Aided Design*, Vol. 9, No. 4, pp. 439-450, April 1990.

[29] F. N. Najm, "Transition density: A New Measure of Activity in Digital Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 2, pp. 310-323, February 1993.

[30] F. N. Najm, Michael Y. Zhang, "Extreme Delay Sensitivity and the Worst-Case Switching Acitivity in VLSI Circuits", 32nd *ACM/IEEE Design Automation Conference*, San Francisco, California, pp. 623-627, June 1995.

[31] S. C. Seth, V. D. Agarwal, "An Exact Analysis for Efficient Computation of Random Pattern Testability in Combinational Circuits", 16th *International Symposium on Fault-Tolerant Computing Systems*, Vienna, Austria, pp. 318-323, July 1986.

[32] Chi-Ying Tsui, M. Pedran, A. M. Despain, "Efficient Estimation of Dynamic Power Consumption under a Real Delay Model", *IEEE International Conference on Computer-Aided Design*, Santa Clara, California, pp. 224-228, November 1993.

[33] Taku Uchino, Fumihiro Minami, Takashi Mitsuhashi, Nobuyuki Goto, "Switching Activity using Boolean Approximation Method", *IEEE International Conference on Computer-Aided Design*, pp. 20-25, 1995.

High-Level Techniques

[34] M. Alidina, José Monteiro, S. Devadas, A. Ghosh, M. Papaefthymiou, "Precomputation-Based Sequential Logic Optimization for Low Power", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 74-81, 1994.

[35] R. I. Bahar, G. D. Hachtel, E. Macii, F. Somenzi, "A Symbolic Method to Reduce Power Consumption of Circuits Containing False Paths", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 368-371, 1994.

[36] R. I. Bahar, H. Cho, G. D. Hachtel, E. Macii, F. Somenzi, "Timing Analysis of Combinational Circuits using ADDs", *IEEE European Conference on Design Automation*, Paris, France, pp. 625-629, February 1994.

[37] G. D. Hachtel, M. Hermida, A. Pardo, M. Poncino, F. Somenzi, "Re-Encoding Sequential Circuits to Reduce Power Dissipation", *IEEE Transactions on Computer-Aided*

Design of Integrated Circuits and Systems, pp.70-73, 1994.

[38] Sasan Iman, Massoud Pedram, "Multi-Level Optimization for Low Power", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 72-377, 1994.

[39] Chi-Ying Tsui, M. Pedran, Chih-Ang Chen, A. M. Despain, "Low Power State Assignment Targeting Two- and Multi-Level Logic Implementations", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 82-87, 1994.

[40] T. Villa, A. Sangiovanni-Vincentelli, "NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 9, pp. 905-924, September 1990.

Pattern-Independent Techniques

[41] Eduard Černý, Jindrich Zejda, "Gate-Level Timing Verification Using Waveform Narrowing", *European Design Automation Conference (EuroDAC)*, Grenoble, France, pp. 374-379, September 1994.

[42] Harish Kriplani, Farid Najm, Ibrahim Hajj, "Maximum Current Estimation in CMOS Circuits", 29th *ACM/IEEE Design Automation Conference*, pp. 2-7, 1992.

[43] Harish Kriplani, Farid Najm, Ibrahim Hajj, "Resolving Signal Correlations for Estimating Maximum Currents in CMOS Combinational Circuits", 30th *ACM/IEEE Design Automation Conference*, pp. 384-388, 1993.

[44] Harish Kriplani, Farid Najm, Ibrahim Hajj, "Pattern Independent Maximum Current Estimation in Power and Ground Buses of CMOS VLSI Circuits: Algorithms, Signal Correlations and their Resolution", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 14, No. 8, August 1995.

[45] Chin-Chi Teng, Anthony M. Hill, Sung-Mo Kang, "Estimation of Maximum Transition Counts at Internal Nodes in CMOS VLSI Circuits", *IEEE International Conference on Computer-Aided Design*, pp. 366-370, 1995.

[46] Paul Vanoostende, Paul Six, Hugo J. De Man, "PRITI: Estimation of Maximal Current

Derivaties in Complex CMOS Circuits using Activity Waveforms", *IEEE Journal of Solid-State Circuits*, Vol. 1, No. 1, pp. 347-353, 1993.

[47] Jindrich Zejda, Eduard Černý, Sandeep Shenoy, Nicholas C. Rumin, "Bounding Switching Activity in CMOS Circuits Using Constraint Resolution", *European Design and Test Conference (EDAC)*, Paris, March 1996.

[48] Jindrich Zejda, Eduard Černý, Sandeep Shenoy, Nicholas C. Rumin, "Gate-Level Power Estimation Using Transition Analysis", *Workshop on Design Methodologies for Microelectronics (DMM)*, Smolenice Castle, Slovakia, pp. 111-119, September 1995.

8 Appendix A

Two key algorithms, in pseudocode, for exhaustive analysis are given here. They do the bulk of the work involved in constructing the analysis tree. Both algorithms are presented in greatly simplified versions to allow one to focus on the main concepts. `Insert_Depth_First` creates the tree recursively in a depth-first fashion. `Insert_Terminating_Nodes` is used for the few nodes which represent waveforms on the internal nodes of a sub-circuit when there are no more transitions on the primary inputs.

```
void Insert_Depth_First (struct TreeNode *CurrentNode, int time, int rID)
{
    time++;
    if (time < max_PI_res)      // max_PI_res = time of last transition on PIs
    {
        evaluate_subcircuit (CurrentNode, rID);
                                        // evaluate sub-circuit with the
                                        // inputs as given by CurrentNode
        if (CurrentNode != tree root)
            CurrentNode->NumTransitions = count_transitions (
                CurrentNode->waveforms) + CurrentNode->Parent->NumTransitions;
                            // store cumulative count of transitions so far
        else
            CurrentNode->NumTransition = 0;
        if (the analysis tree has been fully explored to the bottom once)
        {
            BranchViable = Check_Branch_Viability(CurrentNode);
            if (!BranchViable)
            {
                delete (CurrentNode);   // if this branch of the analysis tree
                return;                 // is not worth analysing then delete
            }                           // CurrentNode and exit from function
        }
        CurrentChild = new TreeNode from NextInput->Input;
                                        // New child node created
```

```
        CurrentChild->Parent = CurrentNode;
        CurrentChild->time = time;
        if (time < max_PI_res - 1)   // if there are still transitions on
                                     // the PIs
        {
            Create list of possible inputs on PIs for (time+2) for
                  CurrentChild;
            Insert_Depth_first(CurrentChild, time, rID);
            CurrentNode->NextInput = CurrentNode->NextInput->next;
                            // advance pointer to next set of possible inputs
        }
        if (time == max_PI_res - 1)      // If this is the penultimate set of
                                         // transitions on the PIs
        {
            evaluate_subcircuit(CurrentChild, rID);
                           // evaluate sub-circuit with the inputs as given
                           // given by the waveforms on CurrentChild
            CurrentChild->NumTransitions = count_transitions (
                      CurrentChild->waveforms) + CurrentNode->NumTransitions;
                          // store cumulative count of transitions so far
            CurrentNode->NextInput = CurrentNode->NextInput->next;
                           // advance pointer to next set of possible inputs
            Insert_Terminating_Nodes (CurrentChild, (time+1), rID);
                           // as there are no more transitions on the PIs
                           // need to compute tree nodes that have transitions
                           // on the internal sub-circuit nodes only
        }
    }
}
```

```
void Insert_Terminating_Nodes (struct TreeNode *CurrentNode, int time, int rID)
{
    CurrentChild = new TreeNode;
    CurrentChild->time = time;
    CurrentChild->Parent = CurrentNode;
    evaluate_subcircuit (CurrentChild, rID);
                          // evaluate sub-circuit with the inputs as given
                          // given by the waveforms on CurrentChild
    CurrentChild->NumTransitions = count_transitions (
                    CurrentChild->waveforms) + CurrentNode->NumTransitions;
                          // store cumulative count of transitions so far
    if (CurrentChild->NumTransitions > maxODFsum)
        maxODFsum = CurrentChild->NumTransitions;
                    // If the current number of transitions for this tree
                    // branch > maxODFsum (the value for one complete tree
                    // branch analysis) then update maxODFsum
    if (there exist more waveforms to be computed on internal circuit nodes
        for (time+1))
        Insert_Terminating_Nodes (CurrentChild, (time+1), rID);
                    // even if there are no transitions on the PIs at the
                    // current time, there will be transitions on the internal
                    // nodes due to gate delays
}
```

9 Appendix B

The pseudocode for the sub-circuit picking algorithm is given here. It consists primarily of three functions. `pick_region` takes a node and marks a set of gates according to some given parameters returning the number of gates in that sub-circuit. `pick_regional_PIs` goes through the sub-circuit and marks those nodes which are primary inputs for this sub-circuit. `delimit_region` creates a boundary around the sub-circuit by marking the adjoining gates so that they cannot be picked as part of any sub-circuit.

```
int pick_regions (UDMwaveforms* fnode, int lldepth, int num_region, int
                                                        max_rPIs, int max_subPI)
{
    if (lldepth == 0)                   // if the limit on the logic level
    {                                   // depth has been reached, set the
        fnode->region_ID = num_region;  // current net to be part of the sub-
        return 0;                       // circuit and exit.
    }
    tmpgate = fnode->fanoutgate();
    if (tmpgate == NULL)                // if fnode is a primary output
    {
        if (fnode->region_ID == 0)          // if this net is not part of any
            fnode->region_ID = num_region;      // sub-circuit, pick this net
        return 0;                               // and exit.
    }
    for (fogate = fnode->fanoutgates; fogate != NULL; fogate++)
    {
        if (fogates->region_ID == 0)    // if this fanout gate has not been
        {                               // picked
            gatepick = 1;               // set flag to indicate gate is to be
            for (Inets = fogate->inputnets; Inets != NULL; Inets++)
            {
                if (Inets->region_ID == 0 || Inets->region_ID == num_region)
                {
                    // If this net has not been picked or is already part of
```

```
                // this sub-circuit
                Inets->region_ID = num_region;
            }
            else
                gatepick = 0;       // unset flag so gate is not picked
                                    // since the gate's inputs are already
                                    // part of some other subcircuit - must
                                    // maintain disjoint sub-circuits.
        }
        if (gatepick)
            fogate->region_ID = num_region;
        else                         // this gate is a boundary gate and
        {                            // cannot be part of any sub-circuit
            for (Inets = fogate->inputnets; Inets != NULL; Inets++)
            {
                if (Inets->region_ID == num_region) // make sure all the
                Inets->region_ID = 0;               // inputs to this gate
            }                          //  are not part of this sub-circuit
        }
        for (Onets = fogate->outputnets; Onets != NULL; Onets++)
        {                       // For each fanout net, if the logic level
                                // depth has not yet reached the limit,
                                // pick_region recursively from this net.
            if (lldepth > 0)
                pick_regions(Onets, lldepth - 1, num_region, max_rPIs);
        }
    }
    else                     // if the fanout gate has already been picked
    {
        if (fogate->region_ID == -1 && fnode->region_ID <= 0)
        {                       // if the current gate has not been picked
            gatevalid = 0;      // && the net is not picked or a boundary
                                // set flag to not pick the gate.
            for (figate = fnode->faningates; figate != NULL; figate++)
            {                    // Take care of case where a boundary
```

```
                                        // might be dividing the same sub-circuit.
                    if (figate->region_ID == num_region)
                        gatevalid = 1;
                }                       // Pick the gate to remove the boundary
                if (gatevalid)          // dividing the same region into two.
                    fnode->region_ID = num_region;
            }
        }
    }
}


int pick_regional_PIs(int r_ID)
{
    for (fogate = fnode->fanoutgates; fogate != NULL; fogate++)
    {                                    // for all agtes
        if (fogate->region_ID == r_ID)
        {                                 // if the gate is in the sub-circuit
            num_gates++;
            for (Inets = fogate->inputnets; Inets != NULL; Inets++)
            {                                          // for all the input nodes
                if (Inets->faningates == NULL)         // if they have no fan-in
                    Inets->set_rPI(r_ID);              // gates => they are PIs
            }
            for (Rgates = Inets->faningates; Rgates != NULL; Rgates++)
            {                                         // for all the fan-in gates
                if (Rgates->region_ID <= 0)            // of the input nodes, if
                    Inets->set_rPI(r_ID);              // they are not part of the
            }                                 // sub-circuit the input nodes are PIs
        }
    }
    return (num_gates);      // return the number of gates in this sub-circuit
}
```

```
void delimit_region(int rID)
{
    for (agates = all gates; agates != NULL; agates++;)
    {                                       // for all gates which are not
        if (agates->region_ID != rID)       // part of this sub-circuit
        {
            for (Inets = inputnets; Inets != NULL; Inets++)
            {                                       // for whom the input nodes
                if (Inets->region_ID == rID)        // are part of the sub-circuit
                {
                    agates->set_region_ID(-1);      // set that gate to be a
                    break;                          // boundary gate
                }
            }
            for (Inets = outputnets; Inets != NULL; Inets++)
            {                                       // for whom the output nodes
                if (Inets->region_ID == rID)        // are part of the sub-circuit
                {
                    agates->det_region_ID(-1);      // set that gates to be a
                    break;                          // boundary gate
                }
            }
        }
    }
}
```

How the three functions given above are used is illustrated in the following short `main()` program. First the list of nodes, arranged in order of decreasing fanout, is traversed taking only those nodes which have a fanout between `minFANOUT` and `maxFANOUT`, both parameters having been specified by the user. Sub-circuits are picked using these nodes. Then the list of nodes is traversed again, this time examing all nodes which have a fanout greater than `maxFANOUT`. The idea is to give priority to those nodes which may be most useful in producing 'good' sub-circuits, namely those which have a fanout between a certain range. The remaining nodes are more likely to produce circuits that are too large but the `max_subPI` parameter goes some way towards controlling this.

```
void main()
{
    while ((fanout_list != NULL) && (fanout_list->nodefanout >= minFANOUT))
    {
        if (fanout_list->node <= maxFANOUT) // if the node has fanout
                                            // between minFANOUT and maxFANOUT
        {
            pick_region(fanout_list->node, maxLLDEPTH, region_count, max_rPIs,
                                                                    max_subPI);
                                             // pick a region attached to it
            if (number of gates in region > minGATES)
            {                               // if the region has # of gates
                                            // greater than the min specified
                delimit_region(region_count);
                                            // create a boundary around it
                pick_regional_PIs(region_count);
                                           // mark the PIs to this sub-circuit
                region_count++;             // increment sub-circuit count
            }
            else
               delete_region(region_count);   // this region will not form a
                                           // sub-circuit since it is too small
        }
        (fanout_list->node)++;             // next node in list
    }
    fanout_list->reset;                    // reset list to point to first node
    while ((fanout_list != NULL) && (fanout_list->nodefanout >= minFANOUT))
                                          // if the node has fanout > minFANOUT
    {
        pick_region(fanout_list->node, maxLLDEPTH, region_count, max_rPIs,
                                                                    max_subPI);
        if (number of gates in region > minGATES)
        {
            delimit_region(region_count);
                                            // create a boundary around it
```

```
            pick_regional_PIs(region_count);
                                        // mark the PIs to this sub-circuit
            region_count++;              // increment sub-circuit count
        }
        else
            delete_region(region_count);;   // this region will not form a
                                        // sub-circuit since it is too small
    }
    (fanout_list->node)++;              // next node in list
}
```