Asynchronous Control Circuit Design and Hazard Generation: Inertial Delay and Pure Delay Models

by

Nozar Tabrizi

B.S.E.E (Sharif University of Technology) 1980
M.S.E.E (Sharif University of Technology) 1988

A thesis submitted for the degree of

Doctor of Philosophy

in the Centre for High Performance Integrated Technologies and Systems
(CHiPTec)

Department of Electrical and Electronic Engineering

The University of Adelaide

June 1997

Table of Contents

List of Figures

Abstract

The distribution of global clock signals in increasingly complex digital integrated circuits is considered to be a limiting factor in the near future. Clock skew, high power clock drivers, area penalty for global clock routing and a safety margin for clock period are the major factors giving rise to this limit. As an attempt to cope with this shortcoming, asynchronous methodologies have been considered a reasonable successor for the traditional synchronous technique by many researchers since the mid 1980's. However, the remedy, as a general rule, carries its own drawbacks. The major problem in asynchronous design techniques is the generation of possible spurious signal transitions which can easily cause a faulty state change and hence a permanent malfunction in the whole system. There are two usual methods to avoid these problematic signal transitions or *hazards*.

1. Delays which may be inserted to balance delays in different signal paths (that is hazard prevention), or to absorb[1] spurious transitions (that is hazard diminishing). Both approaches suffer from some overhead in terms of area and performance.

2. Introducing redundancy is another common technique for preventing hazards. This covers a vast area of research work from introducing redundant states as an attempt to eliminate races in the classical Huffman methodology, to some recent research work of gate level implementation of speed independent circuits.

One major objective in this Ph.D thesis is to relax the hazard problem through a third avenue, that is considering a realistic delay model. Although some researchers have assumed the inertial delay model in their work, this delay model has not been extensively investigated to uncover its effect on the existence of hazards.

We investigate two level logic synthesis of asynchronous circuits from STGs under the *inertial gate delay model* and the well behaved environment. We show that *multiple input high to low dynamic logic hazards* are ruled out under the inertial gate delay model in two level SOP logic circuits. Multiple input low to high dynamic hazards are then studied under the same conditions and it is shown that this type of hazard is unlikely to occur under the inertial gate delay model unless a liberal delay flexibility is required between the first level AND gates of SOPs.

1. Only inertial delay can be used for this purpose.

In the next stage of our work we assume the isochronic fork model for interconnection networks and first show that delay hazards are considerably reduced in two level SOP circuits under the pure bounded delay model. The gate delay model is then restricted to inertial and it is shown that delay hazards are further reduced and limited to virtually one type only.

The second major goal in this thesis is focused on design methodologies and hazard free implementations based on redundant logic where the inertial delay model does not help to avoid hazards.

1. Delay hazards are analysed in complex gate based implementations which may only be caused by inverters at some inputs under the isochronic fork assumption. We introduce *safe cells*, based on which well-formed STGs can be implemented free of delay hazards with no unrealistic assumptions about physical gates. Although this technique still compromises redundancy for the sake of preventing hazards, we show that it may achieve a significant area gain in comparison with the two-phase RS-implementation method [32] which is one of the few true speed independent implementation techniques that we are aware of.

2. Martin's asynchronous design technique [49] is a robust speed independent design methodology. In this thesis we introduce a tabular method to perform the last two of the four phases of Martin's compilation process for asynchronous circuit design. The method is then demonstrated with three examples, illustrating that our systematic method is very straightforward, flexible and convenient to apply, and hence it lends itself to automatic compilation.

In summary the contribution of this thesis is as follows:

We showed that two classes of hazards are ruled out or become considerably less likely to occur under the inertial delay model and the isochronic fork assumption. The significance of these achievements is that the logic designer is relieved of the removing of a major class of hazards.

We introduced safe cells to prevent delay hazards normally caused by inverters at some inputs of complex AND-OR-NOT gates under the unbounded gate delay model. We developed a theorem identifying some sufficient conditions to implement a node with a safe cell. We concluded that safe cells are not dual rail code based and hence can be mixed

with normal AND-OR-NOT gates. The significance of this theory is that the design becomes independent of the delay across inverters which has incorrectly been assumed negligible in some design techniques. This goal is achieved at some area overhead, depending on the number of safe cells used in the circuit.

We developed a tabular method to compile the output of the handshaking expansion stage of Martin's asynchronous design methodology into combinatorial / state holding logic operators in a complementary technology such as CMOS. We showed that this method is a flexible and straightforward alternative for the original method.

Statement of Originality

I declare that this thesis contains no material which has been accepted for the award of any other degree or diploma in any university. To the best of my knowledge and belief, this thesis contains no material previously published or written by any other person, except where the due reference is given in the text of the thesis.

I consent to this thesis being made available for photocopying or loan.

SIGNED: DATE: 30/6/97

Acknowledgement

I would like to express my gratitude to **Professor Kamran Eshraghian** my supervisor for his valuable guidance and encouragement during my Ph.D program.

My supervisor **Mr. Michael Liebelt** deserves special praise for his valuable time and much helpful advice on this research. Mike was always available to discuss during various stages of my research. His constructive reviews, comments and suggestions were always a great support for the progress of the research.

I should thank all academic and administrative staff and also postgraduate students specially in CHiPTec (the former GaAs) centre in our department who provided me with such a pleasant environment allowing me to always have enjoyable time during my study at the University of Adelaide.

I also thank my father, sister and my family. I appreciate their continual support and encouragement.

I should thank the Ministry of Culture and Higher Education of the Islamic Republic of Iran for providing me a scholarship and an opportunity for studying in Adelaide.

I also thank the Department of Electrical and Electronic Engineering, the University of Adelaide for kindly paying my tuition fee for the past two and half years.

Nozar Tabrizi

June 1997

Adelaide

List of the author's recent publications

1. Nozar Tabrizi, Kamran Eshraghian, Michael J. Liebelt, "A 256 x 4 Bit Three Phase GaAs D-RAM.", *in Proceedings of the 13th Australian Microelectronics Conference, (Micro'95),* pp 306-311, Adelaide, Australia, July 1995.

3. Nozar Tabrizi, Michael J. Liebelt, Kamran Eshraghian, "A Tabular Method for Guard Strengthening, Symmetrization and Operator Reduction for Martin's Asynchronous Design Methodology", accepted for publication in *the IEEE Transactions on Computers*, February 1997.

4. Nozar Tabrizi, Michael J. Liebelt, Kamran Eshraghian, "Dynamic Hazards and Speed Independent Delay Model", *in Proceedings of the Second International Symposium on Advanced Research in Asynchronous Circuits and Systems, (Async'96)*, pp 94-103, Aizu, Japan, March 1996.

5. Nozar Tabrizi, Michael J. Liebelt, Kamran Eshraghian, "Delay Hazards in Complex Gate Based Speed Independent VLSI Circuits", *in proceedings of the Sixth Great Lakes Symposium on VLSI (GLS-VLSI'96)*, pp 266-271, Iowa, USA, March 1996.

6. Nozar Tabrizi, Michael J. Liebelt, Kamran Eshraghian, "Delay Hazards in Two Level Asynchronous VLSI Circuits Synthesised from Signal Transition Graphs", accepted for presentation at *the 14th Australian Microelectronics Conference, (Micro'97)*, Melbourne, Australia.

To the memory of my ***mother***

Chapter 1

Motivation for Asynchronous Circuits

1.1 Introduction

Fabrication process dependent and hence unknown parasitic delays inherent in logic gates and manifested as different types of hazards (that is spurious signal transitions), have always been a restricting factor in logic design. In the mature synchronous logic design methodology these parasitic transitions are *hidden* through the use of a global clock signal. However, considering the fast growing technology of integrated circuit fabrication in terms of both chip area and transistor size (and hence transistor count), this signal itself is now becoming a restricting factor: clock skew, high power clock drivers, area penalty for global clock routing and more interesting, a safety margin (in terms of a pessimistically long clock period to guarantee the correct operation under the worst case data, voltage, temperature and process parameters) are the side effects entailing extra design efforts. Al-

though the asynchronous design methodology due to Huffman [32] was proposed in the mid 1950's, the resurgence in asynchronous logic design emerged in the mid 1980's to:

- relax clock distribution problems including

  clock skew,

  high power clock drivers and

  area penalty for global clock routing,

- reduce power consumption (by eliminating clock drivers and unnecessary signal transitions) and

- provide a better management for chip complexity resulting from the modularity inherent in asynchronous circuits.

In this design methodology, however, the above mentioned spurious signal transitions are manifested as *implementation deviations from specification,* which may not easily be ignored any more and hence are still a major concern in asynchronous logic design.

The following are some of the major benefits in more detail cited for asynchronous circuits:

1.1.1 Clock skew

Considering the clock drivers with limited derive capability and their highly capacitive loads, the clock signal is naturally delayed as it is distributed across the chip. This delay, known as *clock skew*, can easily cause circuit malfunction as demonstrated in Example 1.1.

Example 1.1: Consider the shift register shown in Figure 1.1 implemented with edge triggered flip-flops.

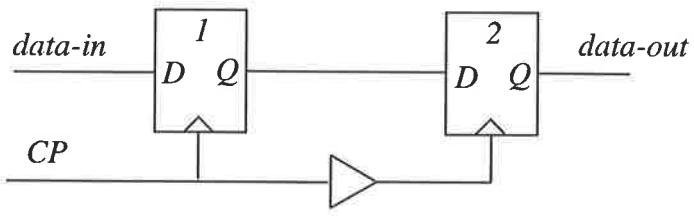


Figure 1.1: A shift register suffering from clock skew.

Due to the extra delay caused by the clock driver and the interconnection wires, the active edge of the clock signal is likely to be *seen* by flip-flop 2 after this edge has affected flip-flop 1, violating the hold time requirement of the second flip-flop. The result is incorrect triggering of flip-flop 2 as depicted in Figure 1.2.

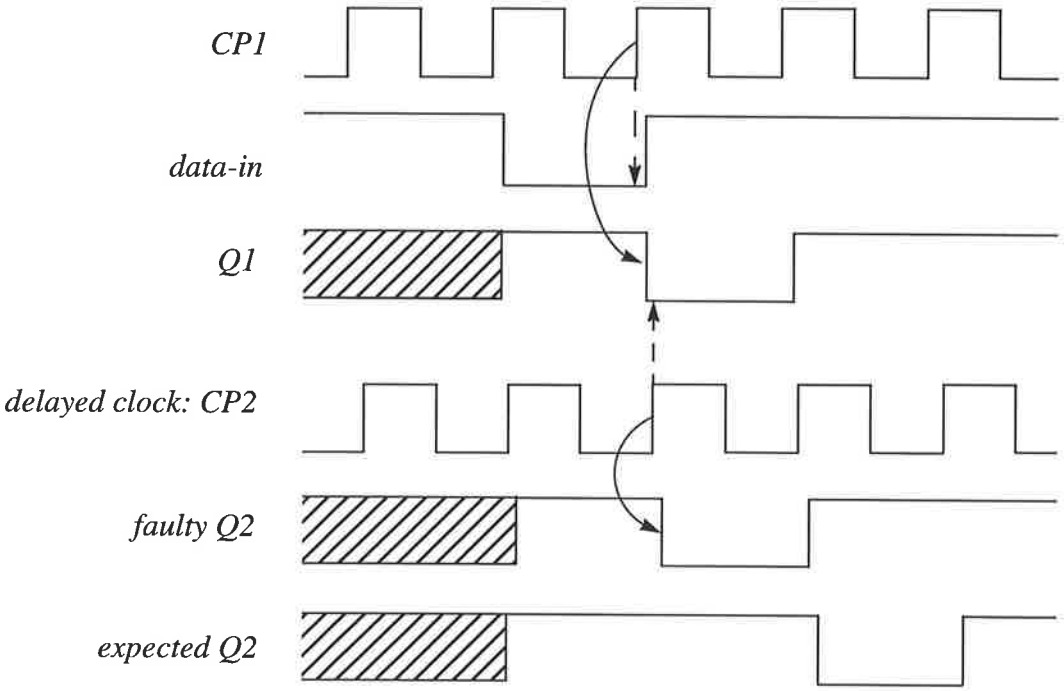


Figure 1.2: Malfunction caused by clock skew.

Notice that the data transferred from flip-flop 1 to flip-flop 2 does *not* undergo such a delay to counteract the clock skew, as the former is a *local* signal as opposed to the clock signal which is *global*.

Remedy to clock skew side effects

Normal level triggered flip-flops do not help at all, as clock skew can make the slave of flip-flop-1 and master of flip-flop-2 become transparent simultaneously resulting in a similar malfunction to that described above.

There are two solutions to solve this problem:

1- Edge triggered master-slave flip-flops

As the name implies these flip-flops are not level sensitive any more. Figure 1.3 shows how these flip-flops avoid malfunction caused by clock skew.

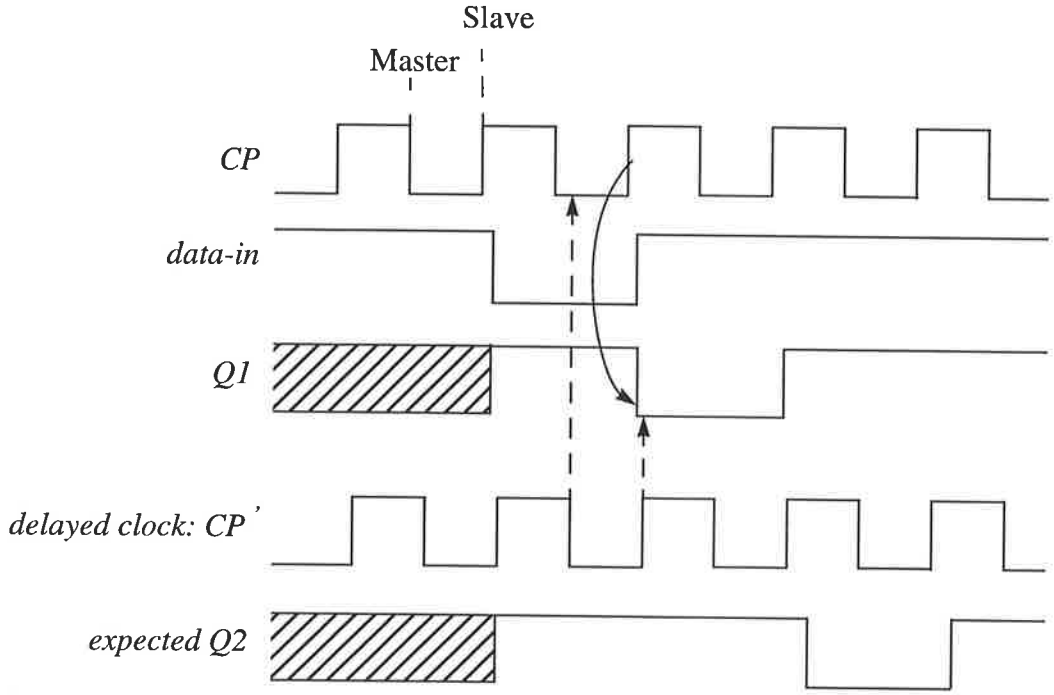


Figure 1.3: Faulty triggering avoided using edge triggered master slave flip-flops.

Let negative and positive edges of the clock signal trigger master and slave stages, respectively, of all flip fops in the chip. Notice how faulty triggering is avoided now: When the second stage samples its input through its master stage (that is the negative transition of the clock signal), the slave of the preceding stage is expected to trigger (and hence to change the input data of the second stage) half a period later, under conditions of no clock skew. Therefore, clock skew will not cause any malfunction as long as it does not exceed half the clock period, which may be chosen sufficiently long to guarantee safe operation. However edge triggered flip-flops are too complex to be normally considered for use in VLSI implementations.

The usual technique to cope with clock skew in VLSI implementations is the two phase clocking scheme [92]. Recall that the problem with level triggered master slave flip flops is caused by too small a gap between the time slots enabling slave and master stages, resulting in simultaneous transparency of slave stage n and master stage n+1. This problem can be solved if the gap is made controllable and hence sufficiently wide, that is a two phase non-overlapping clocking strategy as shown in Figure 1.4.

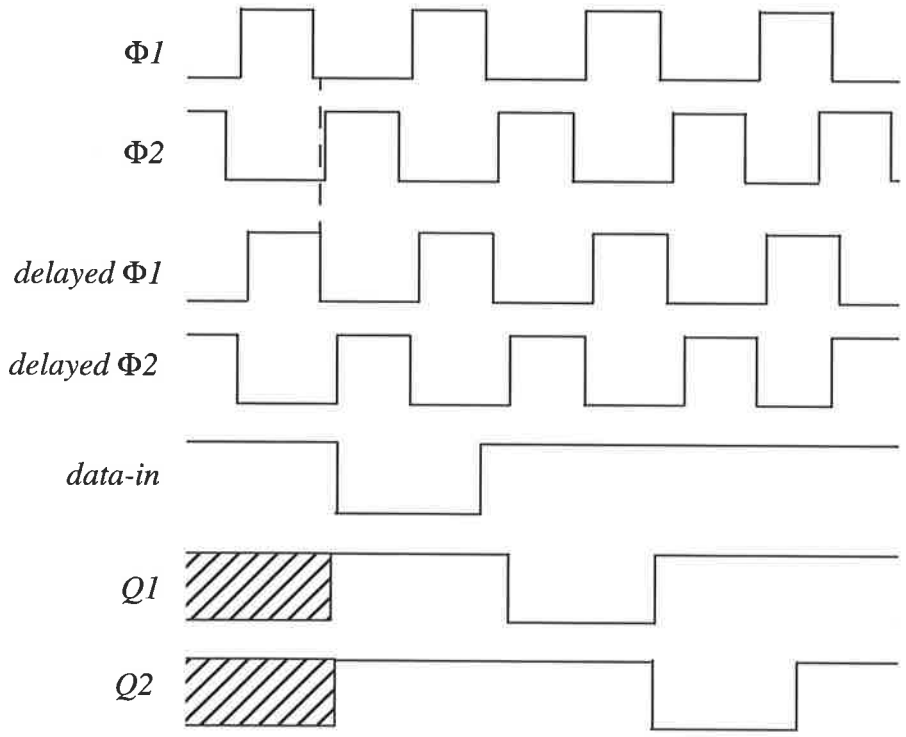


Figure 1.4: Faulty triggering avoided using two phase non-overlapping clock.

Notice how the overlapping de-active region following $\Phi 1$, that is the *dead time*, guarantees that a master stage and the preceding slave stage will not become transparent simultaneously. In other words as long as the clock skew does not exceed the dead time the operation of the flip-flop may be considered safe. However, the dead time is under the designer's control, so, safe operation can be guaranteed by choosing a sufficiently long dead time, no matter how long the clock skew is. The dead time itself is the obvious price to achieve this safety. The point is that as clock frequency and chip size increase, resulting in an increase in clock skew, the inefficiency caused by the clock skew, (that is the ratio of the dead time to the clock period) is increased as well. That is why clock skew is considered a major limiting factor as chip size and clock frequency are increased.

1.1.2 Power consumption

Considering the market demand for portable electronic products and environmental concerns, low power consumption is becoming a major issue in electronic circuit design. Asynchronous circuits seem promising in this aspect as well. This advantage stems again from the global clock signal being totally eliminated in this methodology. Recall that in

synchronous circuits the global (two-phase) clock signal has to be distributed across the chip and regularly trigger all synchronous memory elements, while many of these elements do not undergo any change in every clock signal transition. The net effect is that a considerable amount of power is wasted during every clock cycle. As an example the second-generation Alpha super scaler micro processor from Digital Semiconductor [22] dissipates 50 Watts from a 3.3V supply at 300 MHz. Notice that an asynchronous circuit is free of such a high frequency signal transition and its direct consequence, high dynamic power dissipation, in the current CMOS technology. More precisely, there is no signal transition other than required by the specification in an asynchronous circuit. So it makes sense to presume that an asynchronous circuit operates at the minimum possible dynamic power dissipation for a given implementation technology.

1.1.3 Variable computation time

Some asynchronous systems take advantage of data paths with true completion signals or *self-timed* logic [73] for each computational module[1]. This grants the unique privilege to each module to notify its environment as soon as the result of the computation becomes available. The overall effect may be a better performance for asynchronous systems as shown in the following idealized example.

Consider a n-stage pipeline with only one single data item. In the synchronous version the data word has to spend one clock cycle in every stage no matter how long the real process time is. However in a self-timed asynchronous version the data word will leave every stage as soon as the corresponding process terminates. The situation becomes more complex with non-empty pipe lines. Consider the two stage pipeline shown in Figure 1.5.

If the pipeline is synchronous the clock period has to be chosen as $Max\,[(tm_1), (tm_2)]$, with some safety margin, where $tm_1 = Max\,(computation\ time\text{-}1)$ and $tm_2 = Max\,(computation\ time\text{-}2)$ represent the worst case propagation delay time of the computational modules 1 and 2, respectively. Input data, voltage, temperature and process parameters are the parameters that determine worst case operation time. This of course is too pessimistic (but the most convenient method) to determine the clock period, as not all fabricated chips will operate at the maximum permissible temperature, nor under the worst case process pa-

1. This of course incurs some area and performance penalty, as the data path has to have some redundancy. The common technique to realize self-timed data path is *Differential Cascade Voltage Switch Logic, DCVSL* [31].

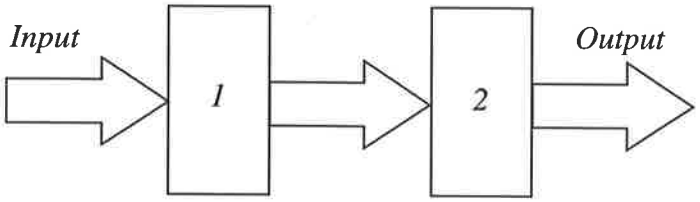


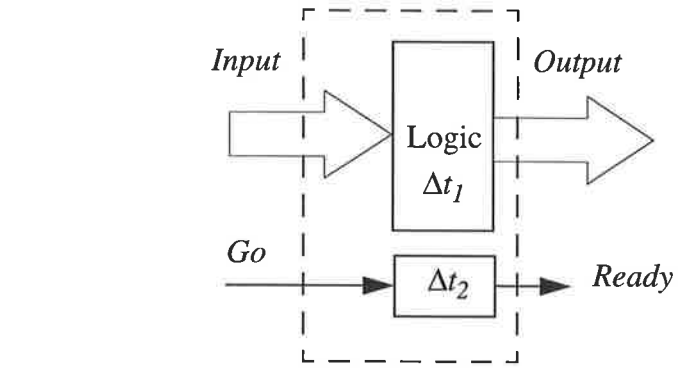
Figure 1.5: A pipeline structure to demonstrate a better throughput for asynchronous systems. The control unit has not been shown here.

rameters. Furthermore, not all clock cycles have to be wide enough to accommodate the worst case computation. Considering this worst case performance the latency and throughput of the synchronous pipeline is determined as $2 \times Max\ (tm_1,\ tm_2)$ and $\frac{1}{Max(tm_1,tm_2)}$ respectively.

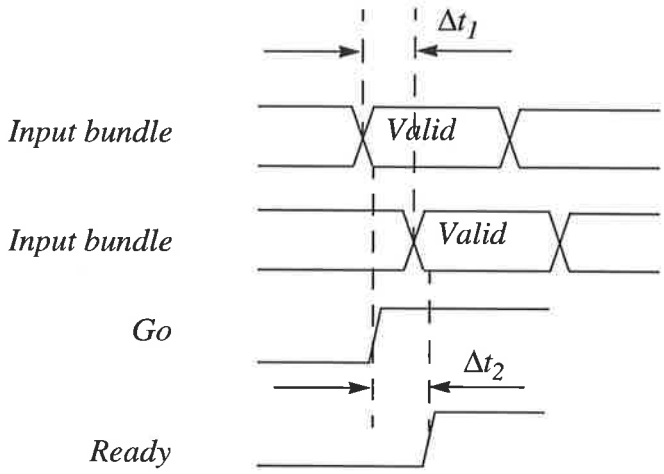
Now consider the asynchronous alternative with a full pipeline where the processed data in stage one does not have to wait for some pre-specified period of time to start the second round of processing in stage two. That is the output of stage one will proceed to stage two as soon as it becomes available provided that the following stage is ready as well to receive data. This clearly reduces the latency and increases the throughput of the pipeline. Assuming an average of half of the maximum delay time for the modules, the average latency and throughput can ideally be improved by a factor of two.

It is interesting to note that speed is adapted to the current operating circumstances of asynchronous circuits and hence is improved even in the absence of true completion signals for data paths. In such cases the delay of computational units are simulated by a fixed delay element with some safety margin as shown in Figure 1.6.

This technique which is to satisfy the timing constraints in the bundled data model in Micropipelines [75], although it lacks the data dependency flexibility introduced by true completion signal generation, still offers an adaptive operation speed, as both the data path and the corresponding delay element are under the same environmental conditions, that is process parameters and temperature. Therefore, the resulting design need not suffer from an extra safety margin to take the worst case process parameter and temperature into consideration. This is in contrast to synchronous circuits where the fixed period of the clock signal has to be sufficiently long to cope safely with the worst case process parameters and temperature.



(a)



(b)

Figure 1.6: Bundled data model with two wire handshake, (a) modelling delay in a processing element, and (b) timing diagram.

1.1.4 Modularity and upgradiblity

Asynchronous systems are known to be modular, as a module in such a system may be replaced with an upgraded version with no concerns about the environment of that module. This, however, is not the case with synchronous systems. Consider replacing a register in such a system, where some timing characteristics like *set up time* and *hold time* of the replacing module must be taken into consideration and verified against the environment requirements before such a replacement is performed legitimately.

In synchronous systems there would be no point in replacing a slow Adder, for ex-

ample, with a faster one if the bottleneck is known to be somewhere else, say, the multiplier of the system. Self-timed asynchronous systems, however, are more susceptible to an easy upgrading. In the above example since the computation time of all modules are data dependent and hence variable, a fast Adder might result in a better performance, as under some specific conditions the bottleneck may happen to be determined by the Adder itself.

This flexibility of asynchronous systems is manifested as the ease of technology migration as well. A robust asynchronous design works under several different technologies during the life time of the product.

Despite these interesting characteristics, asynchronous circuits still suffer from some drawbacks:

1- Asynchronous circuits are more difficult to design. This is primarily due to taking care of the transient response of gates which can easily result in faulty operation in asynchronous circuits. While in synchronous circuits the transient response is hidden by sufficiently wide clock pulses, every asynchronous design technique assumes one type of restriction or another which limits one or more of:

   1.1- the operation environment, such as single input change at a time in the classical Huffman methodology, or the non-subset requirement for two input bursts enabled in the same state, in the 3D burst mode asynchronous design methodology (see Chapter 2),

   1.2- the implementation flexibility, such as satisfying the isochronicity of forks in the implementation of speed independent circuits (see Chapter 5).

2- The verification of asynchronous circuits is more difficult than synchronous circuits [21][59][6][43], as the design procedure is more complex than that required for synchronous circuits. Furthermore, the verification requirements, time and memory, typically grow exponentially as the complexity of the circuit increases.

3- Asynchronous circuits suffer from much more difficult testability as well [42][5]. This is primarily due to the larger number and greater distribution of memory elements in asynchronous circuits. The redundancies introduced in asynchronous circuits such as redundancies to remove hazards, are another

source of complexity in testability of asynchronous circuits. Furthermore, incorrectly estimated transient response of asynchronous circuits may result in permanent non-functionality. More specifically a manufactured chip with a different implemented delay from the specification may not work properly at all, due to a glitch causing an incorrect state change. While in its synchronous counterpart this miscalculation in delays may easily be resolved by decreasing the clock frequency.

4- It is not yet clear whether an asynchronous methodology may result in a net speed gain, due to the overhead incurred by the specific handshaking protocol employed in the methodology, in spite of the partial performance gain due to average case instead of worst case operation discussed above.

1.2 Organization

This thesis is organised as follows:

In Chapter 2 some asynchronous design techniques are reviewed, with an emphasis on the specific delay model in each methodology. We first review state based techniques in Section 2.3 starting with the classical Huffman method and then continue with the first generation extensions to that, namely: one-hot coding, Friedman and Menon's technique, self clocked circuits, Q-modules and 3D asynchronous circuits.

Burst mode or self-clocked circuits are discussed in Section 2.3, which are considered as the second generation extensions to the Huffamn methodology.

In Section 2.4 event based circuits are studied. We start with Muller's theory of speed independent circuits based on Lattice theory, and proceed with a flow table based speed independent design which presents a redundancy based implementation technique to identify end of operation similar to the idea later utilized in self-timed data paths. This technique has been reviewed in this section, only because of its unbounded delay model assumption for logic gates, although it is a flow table based technique. Graph based specifications for Muller's theory (that is signal transition graphs (STGs) and change diagrams (CDs)) are then discussed which are considered as a bridge between the Lattice theory and the theory of Petri nets. This chapter is ended with reviewing some restrictions utilized in some extensions to the original STG based design techniques. A more detailed review of some of these design techniques will be presented in later chapters in order to provide a

deeper background for our work.

This chapter includes some discussions and comparisons between several sources. A good example is the two different interpretations for the notion of multiple transitions which has been covered in Section 2.2.

In Chapter 3, based on the work in [59], we investigate two level logic synthesis of asynchronous circuits from STGs under the *inertial gate delay model* in addition to the above constraints on the environment We show that multiple input high to low dynamic logic hazards are ruled out under the inertial gate delay model in two level SOP logic circuits. We then weaken the zero wire delay restriction and find an upper bound for the delay along critical interconnection wires and hence propose a *virtual isochronic fork* model for interconnection networks. Multiple input low to high dynamic hazards are then studied in two level logic implementations and it is shown that this type of hazard is unlikely to occur under the inertial gate delay model unless a liberal delay flexibility is required between the first level AND gates of SOPs. Static hazards, on the other hand, are shown to not be relaxed under the inertial delay model.

Delay hazards are investigated in two level logic and it is shown that even the well-behaved environment assumption does not necessarily lead to delay hazard free implementations, no matter whether the pure or the inertial delay model is considered. We show that this type of hazard has been overlooked in the design of the STG compiler, SIS [74].

In Chapter 4 we assume the isochronic fork model for interconnection networks and show that delay hazards are considerably reduced in two level SOP circuits under the bounded pure delay model. The gate delay model is then restricted to inertial and it is shown that delay hazards are further reduced and limited to virtually one type only.

In Chapter 5 hazards are analysed in single level logic family under the general pure delay model. We first discuss different types of interconnection forks. Then delay hazards are analysed which may only be caused by inverters at some inputs under the isochronic fork assumption. We introduce *safe cells*, based on which well-formed STGs [59] can be implemented free of delay hazards with no unrealistic assumptions about physical gates. Although this technique still compromises chip area for the sake of preventing hazards, we show that it may achieve a significant area gain in comparison with the two-phase RS-implementation method [33] which is one of the few true speed independent implementation techniques that we are aware of so far. Delay hazards are then analysed in complex

gate based circuits under some gate delay restrictions and hence theorems are developed to identify a subclass of delay hazards. We lastly show how logic hazards are relaxed in this logic class. Notice that these achievements apply to the pure and hence to the inertial delay model as well.

In Chapter 6 we introduce a tabular method to perform the last two of the four phases of Martin's compilation process for asynchronous circuit design. The method is then demonstrated with three examples, illustrating that our systematic method is very straightforward, flexible and convenient to apply, and hence it lends itself to automatic compilation. The technique is independent of the particular delay assumption considered in the design process.

Chapter 7 is the conclusion.

Chapter 2

Delay Constraints and Design Techniques of Asynchronous Control Circuits

2.1 Introduction

In this chapter we will consider different delay restrictions in some representative asynchronous design methodologies, as we review and assess the methodologies themselves. These are relevant observations as a major part of the work presented in this thesis is based on some specific delay assumptions.

Asynchronous control circuits are usually classified according to the particular delay model assumed in their synthesis:

2.1.1 Huffman classical method

The Huffman classical method and its extensions like [44][45][46] falling into the first category, assume the bounded but unrestricted gate-wire delay model. It is bounded in the sense that the environment has to know an upper bound on all stray delays (and delay elements if any) to determine the earliest instant of time to apply the next input vector (fundamental mode operation), otherwise the new input may arrive too early disturbing the circuit prematurely and causing a malfunction. As a result, if the value of a delay in a synthesised circuit is increased due to a variation in the process parameters or temperature the timing constraints on the environment must be revised accordingly. A group of these so called delay hazards in two and one level logic circuits are the topic of Chapters 5 and 6, respectively, in this thesis.

This model is also unrestricted in that sense that

1- there is no restriction in the relative values on the gate-wire delays assumed in the circuit. That is in the real chip a gate with an upper delay bound of 100 pico-seconds does not have to operate faster than another gate which has been specified by an upper delay bound of 150 pico seconds. Exceptions to these are some deliberately inserted delay elements to satisfy a particular inequality to guarantee the correct operation.

2- all stray delays can assume either type of models, inertial or pure, or even an inertial followed by a pure, except for delay elements deliberately introduced to absorb spurious transitions (spikes), which have to be of course of inertial type.

2.1.2 Speed independent circuits

The unbounded gate but skew free wire delay (Speed Independent) model was introduced by Muller [57]. In his methodology the bounded delay restriction is lifted from the gate model at the cost of another delay restriction on interconnection wires: the output signal of a gate must be absorbed at all fanout terminals with a negligible skew. This theory also requires an atomic model for all gates, which may not be easily achieved with today's technologies. Notice that both skew free and atomic delay model requirements have been shown to be only sufficient in [50] and [14], respectively, for correct operation.

In some extensions to Muller speed independent theory such as [2] Armstrong not only uses simple logic gates, but a bounded wire delay model as well, of course at the expense of some redundancy and hence area/performance overhead. Similar to Huffman

methods, this technique is flow table based as well, which again differs from Muller's.

2.1.3 Delay insensitive circuits

The most robust asynchronous design technique should obviously assume unbounded gate and unbounded wire delay model, resulting in the so called *delay insensitive* circuits. Although simple gate based delay insensitive circuits have been proven to be very limited [48] there are some delay insensitive synthesis techniques which are based on some specially designed modules with some internal timing assumptions [20] [58], or a one sided delay restriction on a single element [70].

Asynchronous circuit behaviour on the other hand may be classified as state based and event based as well: In the first category starting with Huffman's work and continuing with some extensions [65][66] an intermediate but abstract notion of *state* is explicitly introduced to link the specified inputs to the required outputs, so that at each instant of operation the outputs are determined by the current state and possibly the current inputs. In the event based methodology, however, the logic value of outputs are determined directly by the previous input/output vectors and the current input transitions, that is there is no intermediate layer of state as is the case with the Huffman methodology. As will be discussed in this chapter an interesting difference between these two methodologies is that in the Huffman methodology the initial design usually starts with more states than are really sufficient, that is the initial description is shrunk as the design proceeds, while in the event based methodologies the initial description not only is shrunk, but also is sometimes expanded to solve the CSC problem, described in this chapter.[1]

In this chapter we present a concise but descriptive perspective of some developments in asynchronous design techniques. We start with the Huffman methodology which is more suitable to model choice, and is later developed by Unger [83], and then proceed to its extensions. Then event based techniques will be addressed which are more amenable to model concurrency.

1. Transition tables in the Huffman methodology may also be expanded by newly introduced state variables to remove races, but not to increase the memory of the network, that is to solve the CSC problem.

2.2 State based techniques

2.2.1 Classical Huffman method

The classical asynchronous design methodology due to Huffman [32] has a close similarity with its synchronous counterpart as shown in Figure 2.1.

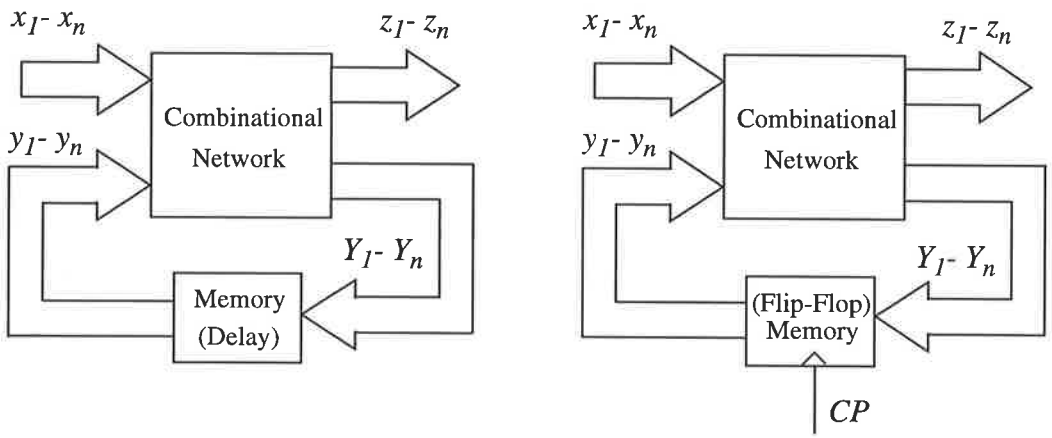


Figure 2.1: (a) Huffman model for asynchronous circuit, (b) the sequential counterpart.

In this figure x_i, z_i, Y_i, and y_i represent external input variables, external output variables, excitation variables and state variables, respectively. The principle idea behind both of these techniques is that the outputs of the combinational logic, that is z and Y, depend on (in addition to possibly the current external inputs, in Mealy circuits) some history of the previously applied inputs.

In an asynchronous circuit, starting with a stable state, that is a state in which $y_i = Y_i$ for all i, upon the arrival of a new input vector, the combinational logic is exposed to the new x vector but the previous y vector due to the delay elements which serve as a short term memory, generating $Y(x, y_{old})$. Now if it happens that $Y(x, y_{old}) = Y(x, Y(x, y_{old}))$ then the whole network has in fact reached a new stable state[2].

The high level description language in the classical Huffman methodology is a Flow Table (FT) which presents a more qualitative operation principle of the circuit. A FT is in

fact a *semi-symbolic Karnaugh map* for the combinational part of the circuit in Figure 2.1-a, where all possible input vectors are laid out horizontally and the *symbolic* internal states are laid out vertically. Each state variable vector y^i represents a *state* for the circuit. A state concatenated with an input vector is called a *total state*. For example the FT in Figure 2.2-a specifies that for the total state $y.x = y^2\text{-}01$ and $y^4\text{-}10$ the corresponding excitation output of the combinational circuit (that is the excitation variable vectors) would be Y^3 and Y^1, respectively.

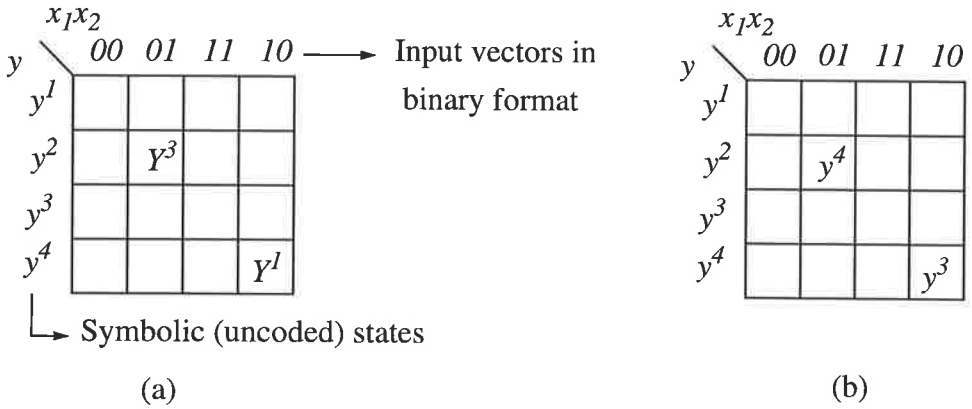


Figure 2.2: Flow table design.

Due to the delay nature of the memory elements which specifies $y(t) = Y(t + \Delta t)$, it is more descriptive to replace Y with y in this K-map, resulting in the FT shown in Figure 2.2-b.

Therefore, starting with a stable state, a FT shows the corresponding next state for every input change. In Figure 2.3 staring with the total stable state $y^2\text{-}00$, upon the input change of 00 to 01 the circuit changes its state from y^2 to y^4 and eventually settles in the stable total state $y^4\text{-}01$. Stable states are encircled.

Referring to Figure 2.3 consider the notion of *state*: in order to generate the required output the circuit should first reach the appropriate state. So, the notion of state seems an intermediate stage between external inputs and outputs. The same interpretation is true in Figure 2.1 where the output vector z is a function of both x and y.

2. This type of operation is called *normal* mode, as one stable state is reached through only one unstable state. Generally speaking a circuit may pass trough more than one unstable state to stabilise, resulting in a considerable speed penalty.

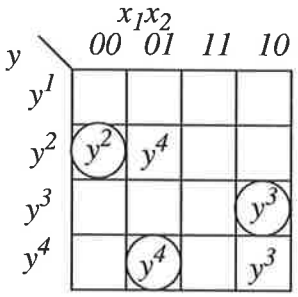


Figure 2.3: An asynchronous circuit behaviour partially modelled by a flow table.

When the circuit is ready to accept a new input (in the case of the Huffman methodology this happens when the circuit is in a stable state) the environment may have some *choice* of which particular input to apply to the asynchronous circuit.

Notice how *choice* is readily modelled by a flow table. In each stable state all possible choices are tabulated in different columns of the corresponding row.

The first step in the design procedure is to construct a flow table based on the verbal description of the problem. This is rather a non-routine procedure. Different designers may come up with different flow tables with different numbers of states, as it is generally not trivial to intuitively identify equivalent states to close up the flow table at early stages. This problem, however, is greatly relaxed by a mechanical technique for flow tables reduction [68] which normally follows the flow table construction in the standard design procedure. The resulting table has the minimum possible number of states, although it usually has to be expanded again to resolve the critical race problem as described below.

The next step is state assignment, in which unique binary vectors are assigned to each internal state in the simplified flow table just created in the previous stage. Numerical internal states usually introduce the *critical race* problem, in which the correct change of internal state (as a result of an input change) depends on the relative delays of different gates. Much work has been done to avoid critical races, while keeping a particular cost minimum [26][72] [45][80][81].

As an example starting with the total stable state of *1100* in Figure 2.4 the input change to *01* requires both the state variables to change from high to low. However, the circuit will not settle in the correct stable state (that is *0001*) if y_1 completes its transition before y_2 fires.

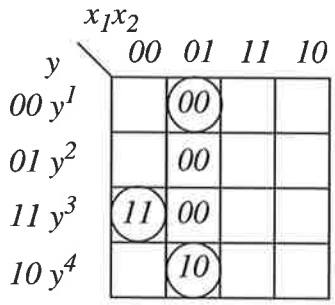


Figure 2.4: Critical race as a result of multiple state variable change.

This problem may always happen if there is a chance for multiple state variable change, revealing the critical role of this stage. An insufficient number of state variables and also inappropriate state coding (even with right number of state variables) may cause circuit malfunction, while a liberal use of state variables results in an inefficient circuit. Furthermore, introducing a (long) sequence of unstable states to make the circuit immune against critical races (that is *multiple transition time encoding*) entails as many passages through the combinational logic as the number of unstable states in the passage, resulting in a considerable degradation of performance, although the idea is straightforward. *Single Transition time* state encoding, on the other hand, entails only one state variable transition in each state change. Therefore, this coding technique not only avoids critical races but entails rippling through the combinational logic unit only once, most likely resulting in the best possible performance.

The final step in the classical Huffman asynchronous circuit design is hazard detection and removal. A hazard is a possible spurious signal transition on the output of a logic gate, which may have different sources. We will discuss this topic in later chapters in more detail.

Efficient computer aided design tools [93] have been developed to automatically perform the different stages of the Huffman design methodology.

2.2.2 One-hot coding

The one-hot coding technique due to Hollaar [29] is another flow table based technique to design asynchronous circuits. Recall that state coding is one of the critical stages in the Huffman methodology, which determines the overall structure of the prospective circuit. An appropriate state coding can remove all critical races with the minimum pos-

sible logic.

The one-hot coding technique totally bypasses the state coding (state variable assignment) stage by using as many state variables as the number of states in the flow table, so that each state is uniquely identified by one state variable asserted to logic high and the rest of the state variables to logic low. This implies one state flip-flop for each state or row of the flow table which of course incurs an exponentially increasing amount of redundancy comparing with encoded states as in Huffman methodology. The major achievement in this state coding technique is that every two adjacent states differ in exactly two bits avoiding any races with a little care as discussed below.

It is concluded in [29] that every excitation variable may be represented as $Y_n = \overline{R}.y_n + S$, where y_n is the corresponding state variable, and S & $\overline{R}$, called transition term and hold term in [28], are two logic functions independent of y_n. This equation shows that every state variable which represents one state in the one-hot coding can be implemented as a RS-FF with some extra logic. Therefore, unlike the case with the traditional Huffman methodology, one-hot coding approach does not only require state variable assignment, but also has a straight forward implementation as demonstrated here:

Rule 1: A state flip-flop, s, is set if and only if the current state is an immediate predecessor state of s and then a proper input is applied to the circuit specified by the flow table. That is the set input is a two level AND-OR logic function with as many AND gates as the number of immediate predecessors of state s.

Rule one specifies the logic for input S, that is the transition term of the state variable. The logic for input R is determined by Rule 2:

Rule 2: A state flip-flop is reset if and only if starting with that state a state changing input is applied to the circuit according to the flow table. So the reset input needs an OR gate with as many inputs as the number of immediate successor states.

Consider a flow table and the corresponding one-hot implementation shown in Figure 2.5.

Starting with the total state Q_1-*100*, suppose that x_2 goes up. This transition is followed by the following sequential transitions: $x_2^+ \rightarrow a_2^+ \rightarrow \overline{Q}_2^- \rightarrow Q_2^+ \rightarrow Q_1^- \rightarrow \overline{Q}_1^+$. This unique order of the transitions which is independent of any relative delays of the gates and or wires guarantees a race free transitions, as the flip flops 1 and 2 may not be

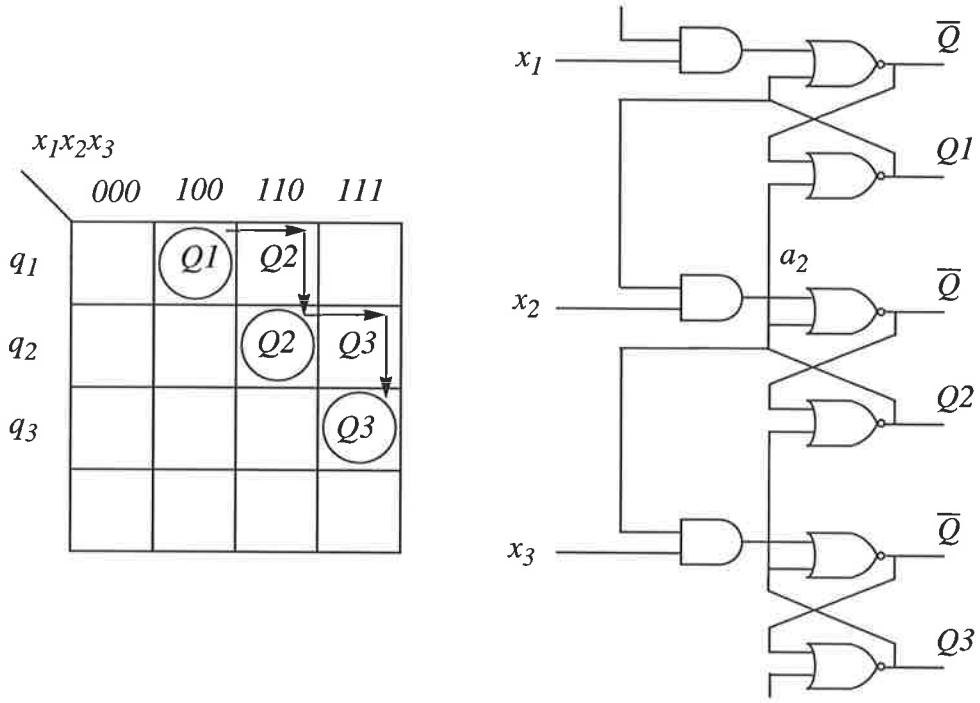


Figure 2.5: Direct implementation of a partial flow table using one-hot coding.

reset both at the same time.

The above technique may not be applicable any more if two states, Q_1 and Q_2, are immediately reachable from each other, that is $Q_1 \rightarrow Q_2$ and $Q_2 \rightarrow Q_1$. According to the above algorithm, since Q2 is an immediate successor of Q1, Q1 tends to set Q2. Likewise since Q1 is an immediate successor of Q2, Q1 tends to also reset Q2. This holds Q2 in an unstable situation with both outputs at logic high, preventing Q1 from being reset and hence resulting in a deadlock.

This problem is solved by introducing some redundancy to the circuit by either inserting an intermediate state in the flow table, (to make one of the states a non-immediate predecessor of the other one) or adding a product term to de-activate the R input of Q2 and hence let Q2 be pulled down freely.

In addition to structural and race free implementation, Hollaar further shows that his methodology relaxes the fundamental mode restriction so that now two consecutive input vectors in many cases may be applied closer to each other than is required by the traditional Huffman methodology. Notice, however, the main restriction of the bounded delay

model still exists.

2.2.3 Timing requirements in the Huffman methodology

Generally speaking two successive input vectors may differ in exactly one bit, (that is single input change) or multiple bits. The second case has two different interpretation in the literature:

1- All signal transitions occur during some pre-specified maximum (and normally short) time interval. So if the multiple input change $x_1\, x_2\, x_3$: 001 $\rightarrow$ 101 $\rightarrow$ 111 (or 001 $\rightarrow$ 011 $\rightarrow$ 111) occurs within δ_2, the circuit will consider the transitions on $x_1\, x_2$ as *simultaneous*, and hence will recognise the three input bit patterns, that is 001, 101 and 111 (or 001, 011 and 111) as *two* successive input vectors, that is *001* and *111*. Otherwise they may be considered as *three* consecutive input vectors, that is 001, 101 and 111 (or 001, 011 and 111) as discussed later in this section.

2- The second interpretation of multiple input change is that no matter how close to each other and in whatever order the transitions happen, the circuit reacts to this multiple input change or *input burst* only after all transitions have fired. The primary implication of this interpretation is that starting with a stable state no allowed input burst may be a subset of another legitimate input burst, other wise the circuit would face a non-deterministic situation if the subset transitions occur first.

The usual trend in the classical Huffman methodology is the first interpretation mentioned above, which virtually restricts this method to *single input change*, as simultaneous transitions are not very likely to occur. On the other hand, the new input vector in this methodology can only be applied after all delay elements have reached a steady state, that is every delay element (either explicit or implicit) has identical input and output logic levels. This inherent critical time interval, δ_1, categorises this methodology as *bounded* gate-wire delay model based[3], as the environment has to know an *upper bound* for the different delays in the circuit in order to be able to determine a right instant of time to apply the new input vector and avoid too early inputs which may even result in faulty state change. In summary two input vectors are considered *consecutive* if they are at least δ_1 time units apart. This mode of operation is called *fundamental* [54]. The second critical time interval in this methodology is δ_2 ($\delta_1 > \delta_2$) which identifies simultaneous transitions as being sep-

3. as opposed to the unbounded gate delay model to be discussed in this chapter.

arated at most δ_2 time units apart. Unfortunately, if an input transition satisfies none of these timing constraints, then the circuit reaction is not normally clear [83], unless some provisions like what are discussed in [82] are made.

In the next two sections three techniques are reviewed to relax the problem associated with the first interpretation for multiple transitions, that is to make δ_2 as large as required. As expected they impose some restrictions on the delay amount of the delay elements used in the design to achieve the goal. However, requiring the input transitions to become complete in some pre-specified time is itself a restriction as well, no matter how long it is. Prolonging, δ_2, on the other hand has a direct consequence of slowing the circuit down as now the circuit has to wait for a longer time interval to distinguish between consecutive and simultaneous input transitions.

The second (and more intuitive) interpretation for multiple input change is that all specified transitions can fire in any order and with any spacing. Notice that the flow table is already able to demonstrate this type of parallelism as shown in the partial flow table in Figure 2.6, for the parallel transition $y^4{:}11 \rightarrow 00$. This topic will be discussed in more detail later in this chapter.

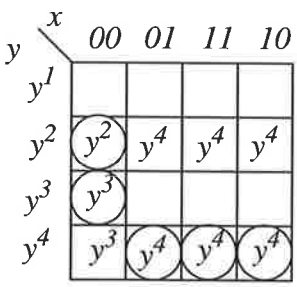


Figure 2.6: Multiple input change: second interpretation.

2.2.4 Friedman and Menon's methods to design multiple input change asynchronous circuits

Friedman and Menon have proposed three fundamental mode asynchronous design methods in [25] assuming simultaneous input changes. Considering the *simultaneous* interpretation, all input bursts are naturally supposed to complete within some pre-specified time interval, otherwise they might be recognised as two or more consecutive input vectors as described above.

In the first method the intended asynchronous circuit, *M,* is decomposed into a special type decoder called *source box* and another asynchronous circuit, *M'*, the inputs to which are *one-hot codes* only, that is each input vector to *M'* has exactly one bit at logic high, except for the *spacer input* (to be discussed shortly) which is *all-zeros*. For example a three input *M'* has at most four legitimate input vectors: 000 (spacer), 001, 010 and 100. The code conversion from binary (applied by the environment) to one-hot code is performed by the source box which of course incurs some redundancy and hence area overhead. The point is that, under fundamental mode assumption, circuit *M'* now does not undergo a multiple input change any more if all two consecutive input vectors to *M'* are separated by a spacer. This will prevent *M'* from faulty change of state if all transitions in the input burst are completed in some pre-specified time. Consider the multiple transition *000* $\rightarrow$ *111* which may be seen by *M* as *000* $\rightarrow$ *001* $\rightarrow$ *011*$\rightarrow$ *111* and finally translated to *0001* $\rightarrow$ *0000* $\rightarrow$ *1000* by the source box (before being applied to *M'*) *provided that* the whole transition is completed within the pre-specified time, *d*, which can be chosen by the designer, and through a single delay element, *D*, in the source box [25]. Notice that although *M* may see the above simultaneous input change as three (ambiguous) transitions 000 $\rightarrow$ 001, then 001 $\rightarrow$ 011 and then 011 $\rightarrow$ 111, this is always translated to *only two consecutive single input changes* eventually, as seen by *M'* no matter how many transitions the original multiple input change contains. This in fact means that the time interval δ_2 (in Huffman method) has now become under the control of the circuit designer.

All other classical problems pertaining to critical races and hazards need to be handled as usual. Furthermore, *M'* is to be so designed that neither the internal state nor the output may change as the spacer replaces a one-hot code.

In the second technique in [25] for the synthesis of simultaneous input change asynchronous circuits, in addition to the primary inputs x_1, x_1 ..., x_n, circuit *M'* receives a delayed version of the inputs as well, as shown in Figure 2.7.

The delay, *D*, has to be sufficiently large to guarantee that the input burst is completed before the delayed input starts its transition, and furthermore the circuit stabilizes in between. Figure 2.8 shows this timing restriction and different parts of the input transition $I_J \rightarrow I_K$.

Since the number of input bits is doubled, now the flow table *M'* has many more columns than the original *M*, however, the transition sequence between the columns of a par-

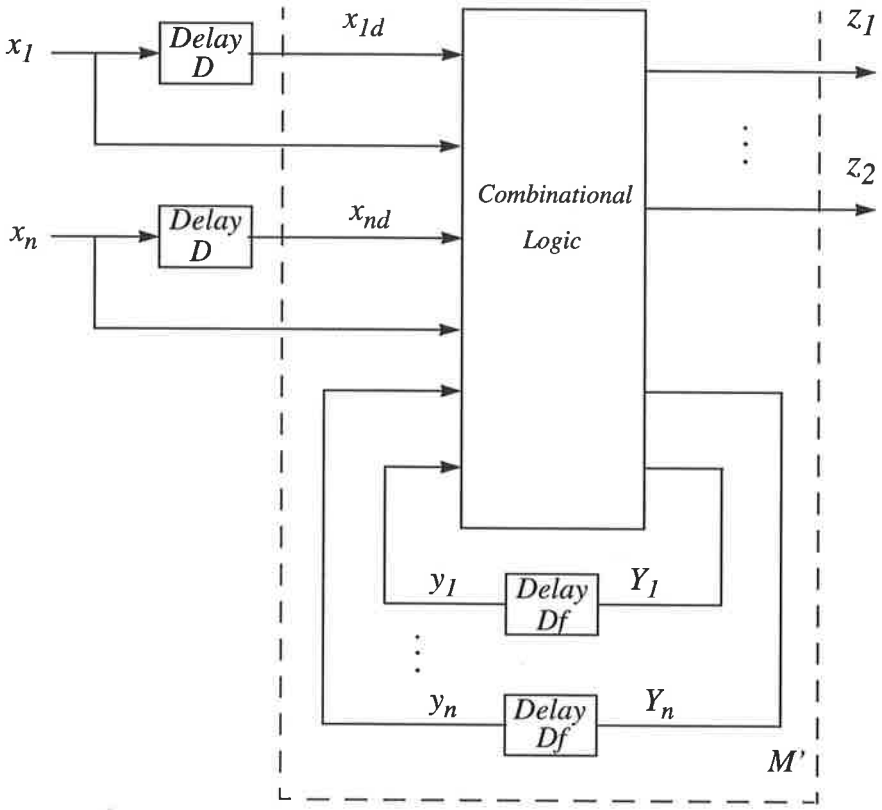


Figure 2.7: Circuit *M'* is driven by both primary and delayed inputs.

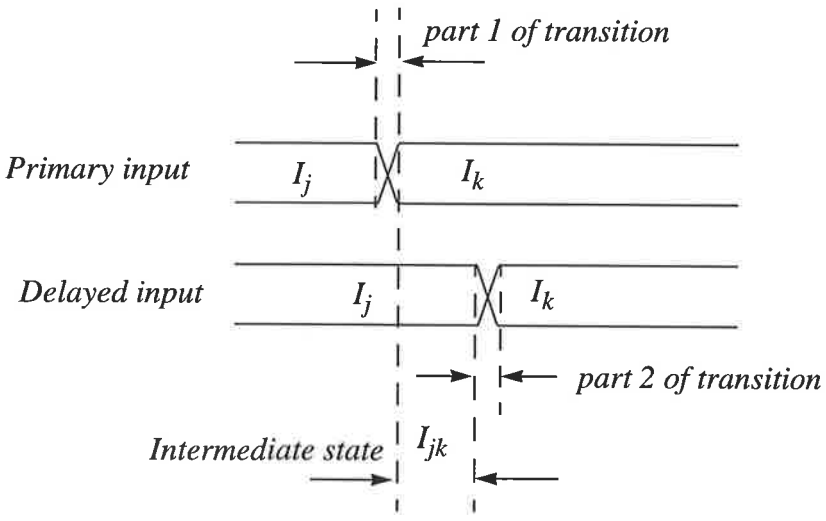


Figure 2.8: The timing of the Input transition $I_J \rightarrow I_K$ for the circuit in Figure 2.7.

ticular row due to an input transition does not include an input vector in which both primary and delayed input burst are incomplete, as it is a required condition that first the primary input burst must terminate and then the delayed input burst may start after the circuit has stabilised, as shown in Figure 2.8. Therefore, in the flow table *M'* and for the transition $(p, I_j) \rightarrow (q, I_k)$, the columns corresponding to the stable inputs are filled identically to the corresponding columns in flow table *M*. The rest of the states on the way of transition from I_j to I_k are considered a stable state, that is p.

Assuming a single transition time state assignment, the third method which is a straight forward extension to the basic time interval δ_2 is accomplished by placing a so called *Huffman delay box* in the standard block diagram of Figure 2.1-a following the combinational logic. Since the delay is of inertial type all spurious signal transitions including those resulting from multiple input changes on the outputs of the combinational logic will be absorbed provided that duration of these transitions and hence the completion time of the simultaneous input change do not exceed the added delay.

2.3 Burst mode or self clocked circuits

As an extension to the classical Huffman methodology some asynchronous design methodologies still in the category of bounded delay model have been developed to allow multiple input transitions in the second sense, that is intended multiple input transitions may occur in any order and with any spacing while assuming the fundamental mode restriction. The circuit, however, will react against this change only after all transitions have fired. In this method although the single input restriction which is normally attributed to the classical Huffman method is lifted, no allowed input vector or *input burst* may be a subset of another one in the same state. This usually reduces the number of input choices, therefore a better high level description language for this type of behaviour is a state diagram (rather than a flow table), presenting only the possible input bursts enabling the relevant state transitions.

2.3.1 Burst mode circuits using controlled excitation and edge triggered flip-flops

A self clocked synthesis method is proposed in [19] for fundamental mode multiple input change asynchronous circuits, using a local clock which makes the circuit behaviour close to Moore type synchronous state machines, as shown in Figure 2.9.

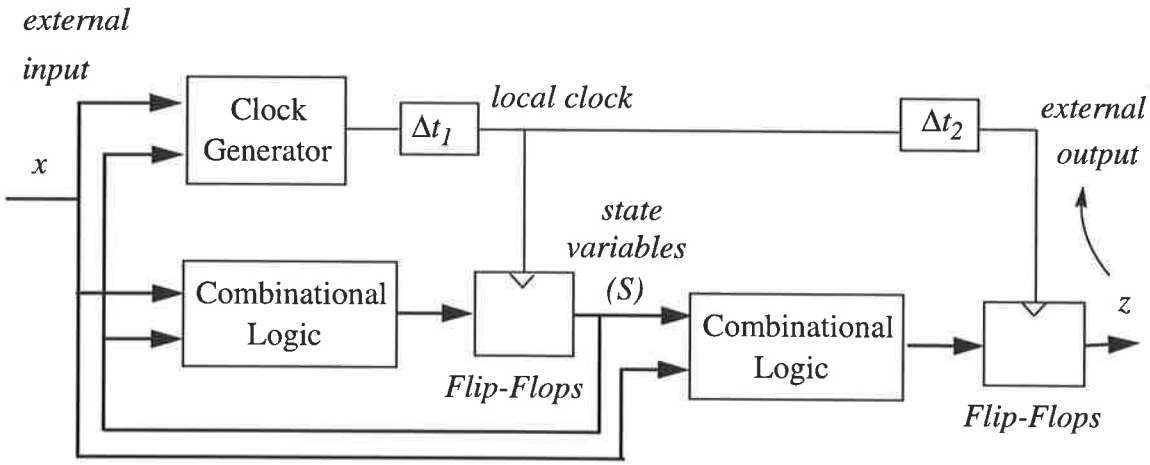


Figure 2.9: A general block diagram of burst mode circuits introduced by Chuang and Das.

A combinational logic circuit as usual is in charge of generating next state variables based on the current state variables and the input burst, which are eventually latched into the edge triggered flip-flops, as shown in Figure 2.9. Upon the arrival of a new legitimate input burst the combinational logic in charge of clock generation attempts to raise the local clock signal only if the state of the circuit is to be changed as a result of this input burst, while the next state variables are being worked out in the corresponding combinational logic. The delay element Δt_1 guarantees that the rising edge of the clock signal reaches the state variable flip flops after the data gets stabilized at the input of these flip flops, resulting in a new state. The output combinational logic now starts to evaluate the output burst based on the recent input burst and the new state variables just worked out. The new output vector, then, is loaded into the output flip flops by the sufficiently delayed (Δt_2) version of the same clock edge which triggered the state flip flops. The Moore type operation of the circuit is clearly shown: the output burst does not occur unless the new state has been reached. A typical state diagram is shown in Figure 2.10-a. Figure 2.10-b shows another alternative, that is the Mealy model.

Notice how the hazard problem is dealt with in this technique. The logic hazard of the input combinational logic is hidden by the sufficiently delayed (Δt_1) clock pulse. The delay element Δt_2 plays a similar role for the output combinatorial logic. On the other hand since the delay element Δt_1 is chosen to be of inertial type, all possible spurious transitions at the output of the clock generator combinational logic are absorbed assuming a suffi-

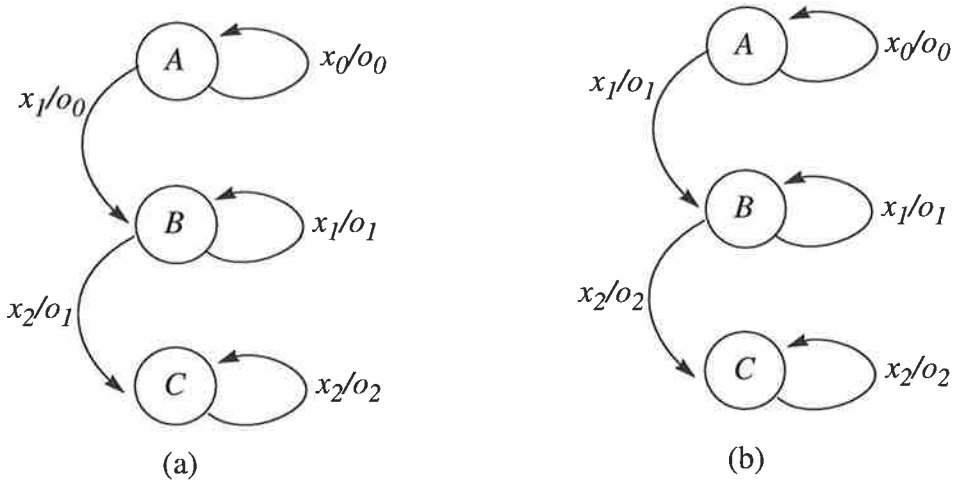


Figure 2.10: (a) Moore type and (b) Mealy type asynchronous state graph.

ciently long delay. Figure 2.11 shows a typical timing diagram.

In [19] function hazards have also been considered likely to occur in the clock generator combinational circuit. Function hazards, however, may not occur unless the fundamental assumption of burst mode circuits is neglected as in [19], where, say, both input transitions $00 \rightarrow 01$ and $00 \rightarrow 11$ are considered legitimate, resulting in *function hazards*. Suppose that the first input transition is specified to cause a state transition while the second one must not cause a clock pulse, leaving the circuit in the current state. However, if the input transition $00 \rightarrow 01 \rightarrow 11$ (as a real transition for $00 \rightarrow 11$) takes place too slowly, then the spurious transition on the output of the clock generator would be wide enough to be considered as a legitimate clock pulse to cause a faulty change of state.

In order to distinguish between the two input transitions $00 \rightarrow 01 \rightarrow 11$ and $00 \rightarrow 01$, in other words to absorb the spurious transition caused by the corresponding function hazard, the environment has to comply with a harsh restriction and complete the input burst within some pre-specified time interval, otherwise the spurious transition would pass through the delay element, possibly causing a malfunction.

A systematic self clocked method has been reported in [1] to implement asynchronous state machines on a PAL device 22Ip6 with no direct concern about hazards and critical races. This technique also assumes single input change and fundamental mode operation.

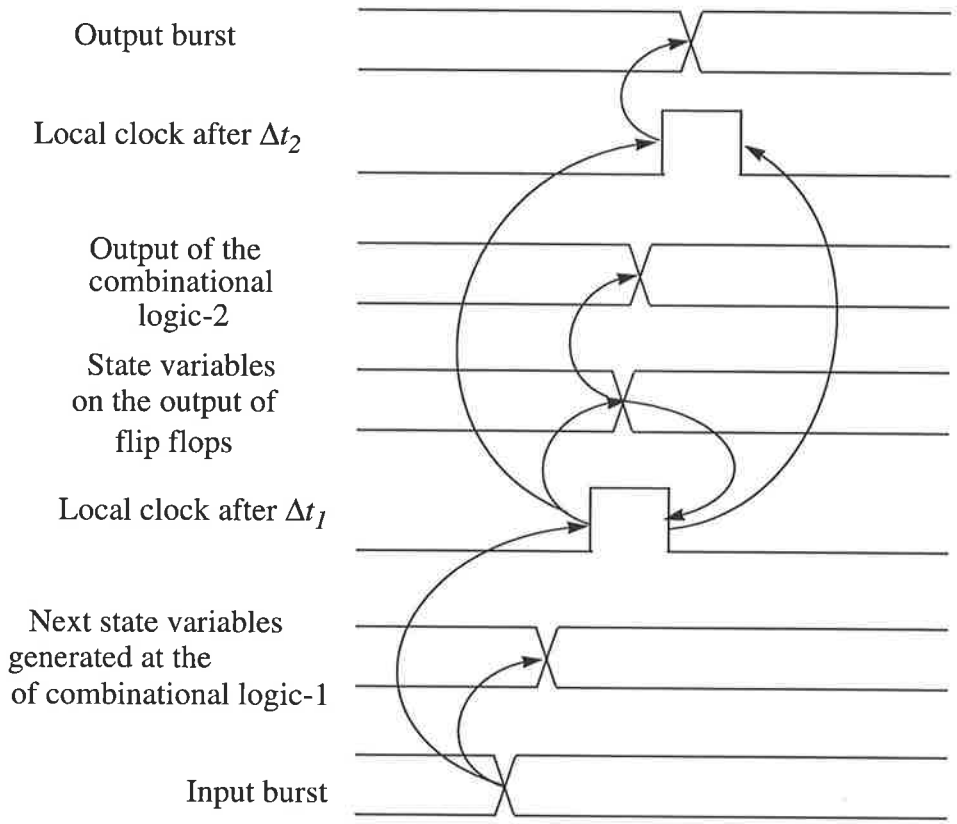


Figure 2.11: A general timing diagram corresponding to Figure 2.9.

2.3.2 Locally clocked asynchronous state machines

Nowick and Dill [65][66] introduced another burst mode synthesis method for asynchronous circuits based on a local clock which makes the circuit operation and modelling nearly identical to Mealy type synchronous state machines, but with some extra logic for local clock generation similar to that described in the previous section. This technique has been used to design some real life examples as reported in [67]. Figure 2.12 shows the general model of locally clocked asynchronous state machine.

Suppose that the two latches are transparent or disabled when the corresponding control signal is high or low, respectively. Therefore, considering the inverted input of stage-1 the two phase latch operates like a master slave flip flop. The clock signal is normally low, making the phase-1 and 2 latches normally transparent and normally disabled, respectively. In this method as a result of an input burst a clock pulse is generated if and only if a state change is to occur as a result of that input burst. However, no matter whether

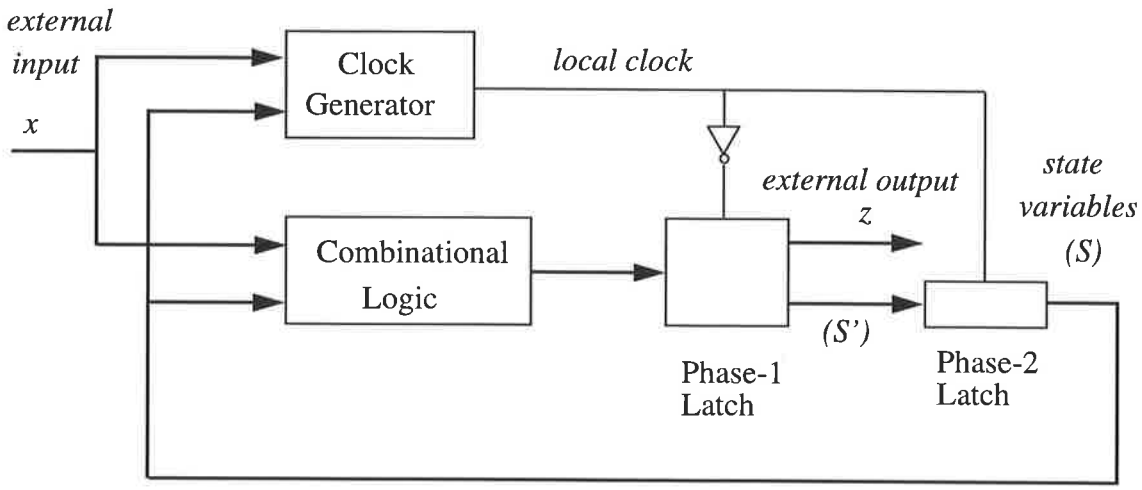


Figure 2.12: The general model of locally clocked asynchronous state machines.

the state changes or not, the output will change as specified by the state diagram, but this change always happens when the input burst is complete. This, however, is not a design but the specification restriction. In Chapter 3 an extended specification is discussed in which an output signal may change while the input burst in not complete yet, under the assumption of a well-behaved environment.

Starting with a stable state and upon the arrival of an input burst the external output and the pre-state variables S' take their desired logic value after some propagation delay time. Although any spurious transition on S' can be hidden by delaying the rising edge of the clock signal, the output signals (as well as the local clock generator, as in Chuang and Das's method) still suffer from possible logic hazards. Since the phase-2 latch is still disabled, the combinational logic and the clock generator (which is a combinational logic as well) still operate based on the previous state variables.

After the combinational logic stabilizes, the local clock generator raises the clock signal disabling phase-1 latch and enabling phase-2 latch. Upon the rising edge of the clock signal the machine enters the phase-2 of operation: The phase-1 latch becomes disabled and the phase-2 latch becomes enabled in a master-slave fashion operation, bringing the machine into its new state, as the updated state variables at the output of latch-1 now are transferred into latch-2, feeding back the new state variables to both the combinational logic and the clock generator. Notice that the external outputs are not affected by this change of state, as the output latch which is part of latch-1 was disabled at the end of phase

one. As the new state variables appear on the output of the phase-2 latch, the local clock generator pulls the clock signal back to logic low, disabling the phase-2 latch and enabling the phase-1 latch, and hence bringing the circuit back into the phase one of operation. Consider the timing restriction on the clock signal: the clock signal should go high after the combinatorial logic has managed to stabilize and the phase-1 latch has received its new data, that is the new state variables and the new output vector from the combinational logic. However, there is no timing restriction on the falling edge of the clock, as the clock (that is the effect) is pulled down after the new state variables (that is the cause) have been stabilized at the output of the phase-2 latch.

Figure 2.13 shows the general timing diagram for the two operation phases of this technique.

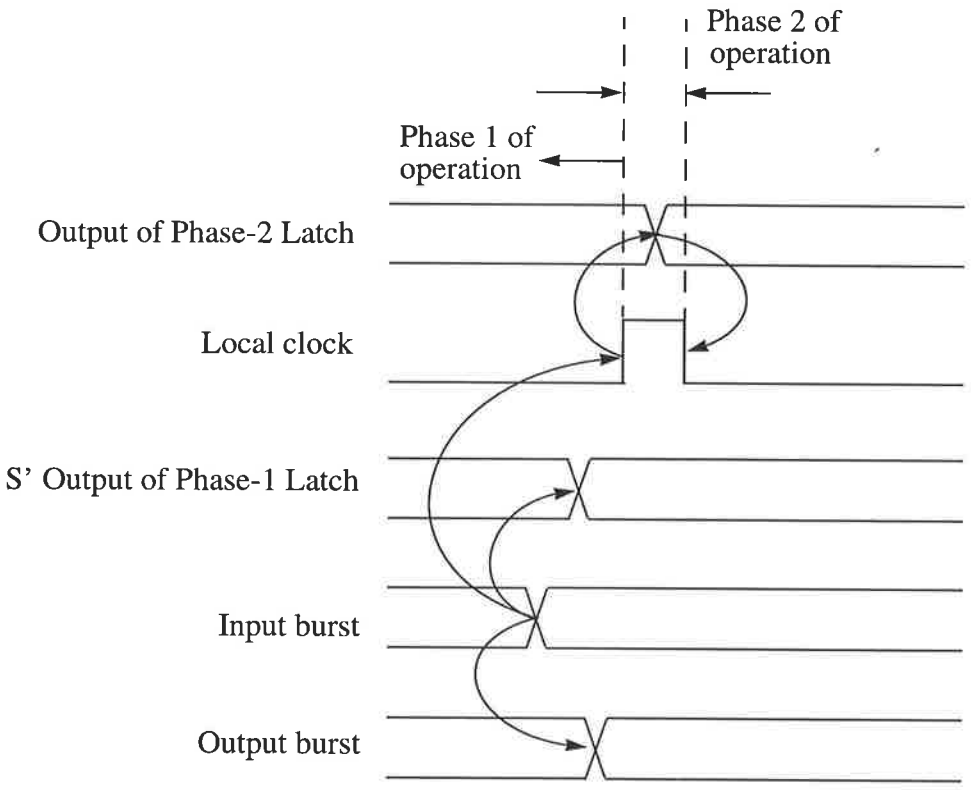


Figure 2.13: A general timing diagram for locally clocked asynchronous state machines.

2.3.3 Q-Modules

Although Q-Modules are usually cited in the delay-insensitive category, we discuss them here, as they are self clocked and furthermore they still require a delay constraint which entails a bounded delay model for some part of the circuit. In this section we see how the inherently hazardous behaviour in locally clocked asynchronous machines is avoided in Q-Modules, and furthermore how these modules achieve some other benefits at the cost of more circuit complexity and high power dissipation. Figure 2.14 shows a general block diagram of a Q-module.

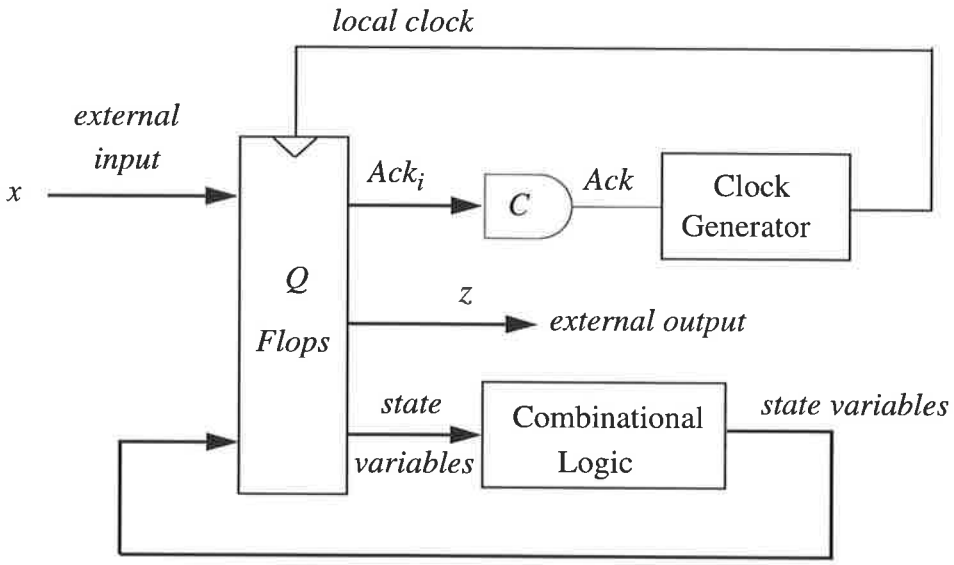


Figure 2.14: A general block diagram of a Q-module.

Each Q-flop in the Q-flop array is a master-slave flip-flop, with data and acknowledge outputs and some more features to be discussed later. The master and slave stages become transparent when the clock signal is low and high, respectively. Therefore, Q-flops sample the external inputs and the current state variables on the falling edge of the local clock and then make them appear on the output lines on the next rising edge. There are three different groups of output lines from the Q-flop array: external outputs, state variables and the acknowledge signals. When the i^{th} master stage of a Q-flop stores its own input data on the falling edge of the clock signal, it pulls down its Ack_i line indicating the resolution of possible metastability [15][16][34][35], as the input signal may change at any instant of time and asynchronously with the clock signal. Similarly, when the i^{th} slave stage of

the Q-flop array updates its output data on the rising edge of the local clock, the Ack_i line is pulled up. The *Ack* signal which can be generated by a Muller C element as shown in Figure 2.14, reflects the total situation resulting from all of the Q-flops: *Ack* is pulled down and up after all Ack_i*'s* have been pulled down and up, respectively.

Based on this introduction now the operation sequence and its limitation are considered. As the initialization stage suppose that the Q-flops are reset to an initial state and then the local clock is pulled down, sampling the first input vector. When all master stages receive valid outputs, *Ack* line is pulled down triggering the clock generator to raise the clock signal. The rising edge of the clock signal, on the other hand, updates the output of the slave stages as described above, resulting in a rise on *Ack* signal, indicating that all outputs of the Q-flops are valid and hence letting the combinational logic start its computation to evaluate the next state variables. Notice that the rising edge of the Ack signal is also in charge of triggering the clock generator to bring the clock signal down. However, *the clock generation must be slowed down* sufficiently to let the combinational logic stabilize and apply its final outputs to the Q-flops with appropriate set up time. This is *the only delay constraint* in this technique. The sufficiently delayed negative edge of the clock signal necessarily samples the input data present at that specific instant of time (no matter whether it is a new, partial or a complete input burst), and also the current state variables just worked out by the combinational logic, initialising another operation cycle of the Q-module. It is highlighted here that all signal transitions inside a Q-module occur delay insensitively except for the delay constraint mentioned above, as shown in Figure 2.15.

According to the operation principle of Q-modules presented above, the clock signal samples the input vector periodically (but with a possibly variable period due to possible metastability and hence a variable resolution time), no matter whether the input has been changed or not since the previous sampling, or whether the input burst has been completed or not. A very interesting and subtle point regarding the asynchronous nature of the input bursts is that *partial sampling* of an input vector resulting in a sampled input vector in the Q-flops consisting of some new and some old input data (which is very likely to happen due to unrestricted spacing in input transitions), does *not* cause any malfunction. Recall that as a general requirement of burst mode operation no allowed input burst may be a subset of another allowed input burst activated in the same state, or the circuit may not

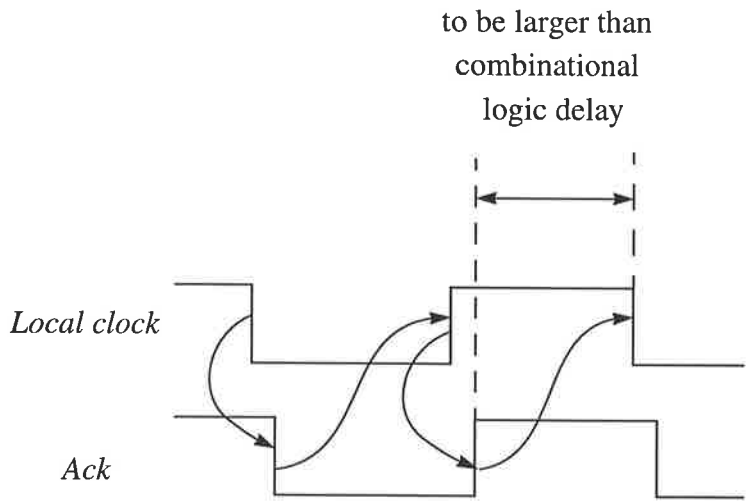


Figure 2.15: Semi-delay insensitive timing diagram of control signals in a Q-module: there is only one timing constraint.

operate deterministicly. Therefore, since a partially sampled input data is considered as an incomplete input burst, it shall not cause any state nor output change, resulting in an idle clock cycle. The idle clock cycle will continue until the input burst becomes complete, when the circuit reacts properly.

Since the output signals are taken from the slave stages of the Q-flops, all output bursts necessarily have monotonic transitions, and hence the output signals are hazard free. This technique, however, does not work for Nowick's locally clocked method mentioned above, as in this method no local clock pulse will be generated if there is no state change. This means that applying the Q-module clocking technique to Nowick's results in this malfunction that the output burst will not be seen in the outside world if the input burst causes output change only.

The above hazard immunity for Q-modules stems from the fact that in this technique the local clock is periodically (possibly with a variable period) generated no matter whether there are any input, output or state changes. This of course occurs at the cost of some extra power dissipation, due to free running clock signals similar to what normally happens in synchronous circuits.

Another issue which should be carefully considered in this technique is the asynchronous nature of the input transitions with respect to the local clock signal, which may result

in metastability as mentioned before. Non-digital circuits are normally used to deal with this problem but they are more expensive than their digital counterparts.

2.3.4 3D Asynchronous circuits

Following [65][66], Yun et al. [96][97] propose three dimensional asynchronous state machines which are the direct extension of Huffman methodology based on the second interpretation of multiple input change. Unlike [65][66], the 3D technique, however, does not use a local clock, nor any explicit storage element. The flow table in Figure 2.16 shows the basic idea used in this technique.

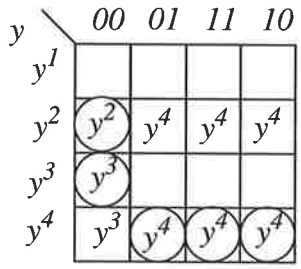


Figure 2.16: Multiple input change: parallel transitions.

Starting with the initial stable state $y^4 11$, suppose that the two input signals are to go low but with an arbitrary order and unknown (but bounded) spacing. The flow table clearly shows that no matter what the order and the spacing is, the final state would be $y^2 00$. Recall that as a general requirement for burst mode circuits with the second interpretation for multiple input changes, no input burst is allowed to be a subset of any other input burst allowed in the same internal state. In the design of the flow table or the state transition diagram care must be taken to keep the output(s) unchanged until the input burst is complete, that is the output burst may only occur after the stable state has been left, ($y^4 00$ or $y^2 00$ in the above example), as this is the implicit design specification. Notice that in this design technique the order in which the output signal transitions occur is not a design requirement, and is naturally determined by the specific implementation and also the stray delays.

In the method described in [96][97] the specified output signals are considered as state variables as well. This bypasses the state assignment stage in the classical Huffman methodology and also eliminates the output logic, although it may not result in the best

state assignment. Extra state variables, however, are introduced if the circuit needs more memory to remember the required past history. This style is similar to STG based design techniques in the sense that it also first addresses the input/output relation and then if necessary extra variables are introduced to satisfy complete state coding.

Different types of hazards and critical races are treated in similar ways to the classical Huffman methodology, and then sufficient conditions are presented to make the circuit hazard free.

This design technique has a further similarity with a spacial case of STG based design methodology under well-behaved environment [59][60] to be discussed in Chapter 3.

2.4 Muller's speed independent circuit theory

2.4.1 Introduction

The above review shows that in the classical Huffman methodology and its extensions the environment needs to know some upper bound for the propagation delay of the circuit in order to determine when to apply the next change in the input, putting this technique in the *bounded delay model* category. (This in fact is the timing function of a clock pulse, the need for which still exists in the Huffman model). Otherwise an early change in input may disturb the network prematurely, resulting in an erroneous state and/or output and hence in incorrect operation. Furthermore, the classical Huffman method and its extensions are not powerful in modelling the combination of sequencing and concurrency, that is *partial ordering*. As an example consider a partial description of a circuit with four outputs, O_1, O_2, O_3 and O_4 (initially at logic low), which are supposed to go high, but in a specific order, as the input I_1 is pulled down, so that O_1 and O_2 may rise as soon as I_1 goes down but the transitions on O_3 and O_4 may only occur if the transitions on O_1 and O_2 have already been completed. The cause-effect relationship describing this behaviour is shown in Figure 2.17.

These shortcomings are relaxed in Muller's speed independent circuit theory [57]. The first restriction is removed by introducing suitable handshaking signals and the second shortcoming is overcome by manipulating and monitoring all nodes in the network explicitly, as discussed in the following sections.

Muller [57] used a directed graph called *State Transition Diagram* (STD) to model the behaviour of a circuit with no external inputs, the so called *autonomous* or *complete*

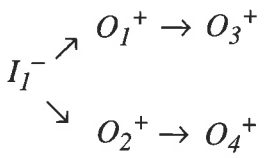


Figure 2.17: A restricted order for output signal transitions which is difficult to model in flow table based techniques.

circuits. A circuit is a network of some logic gates or operators, so that the output of each gate, that is each node of the network, may be an arbitrary function (either combinational or sequential) of all or some of the variables in the circuit. The STD shows how the state of the network changes. A bit pattern representing the logic values of all nodes in the network at a particular instant of time is called the *state* of the network at that particular instant of time, which is totally different from the previous notion of state in the Huffman design methodology. Muller shows that every autonomous circuit has a unique STD describing the circuit behaviour, but the converse in not true. Figure 2.18 shows an autonomous circuit with the corresponding STD.

In this STD each state is three bits long representing the current logic values of the three participating nodes, *a, b* and *c* in the circuit. A * following any logic value means that the corresponding gate or node is *excited*. Otherwise the gate is *stable*. A gate is excited if its output is NOT consistent with its inputs at that specific instant of time, otherwise it is stable. State s_1 (like 11^*0^*) is connected with an arrow to another state s_2 (like 1^*00^*) if one of the excited nodes in s_1 changes its value in s_2.

Each gate is assumed to be atomic, that is no matter what function it performs it is modelled with the corresponding instantaneous decision element followed by some *unbounded inertial* delay. For example the *AND* gate together with its inverted input in Figure 2.18-a is considered a single atomic gate[4] although it might not be realizable in today's technologies. Non-atomic nature introduces new node(s) into the circuit and this might result in an implementation deviation from specification, that is a *hazard*. The output signal, on the other hand, is supposed to reach the fan-out points with negligible skew. Notice that this is less restrictive than negligible wire delay requirement, as a piece of wire

4. This in fact is a sufficient condition for speed independence. Some decompositions might not jeopardize speed independence. Burns in [14] shows the legitimate decompositions.

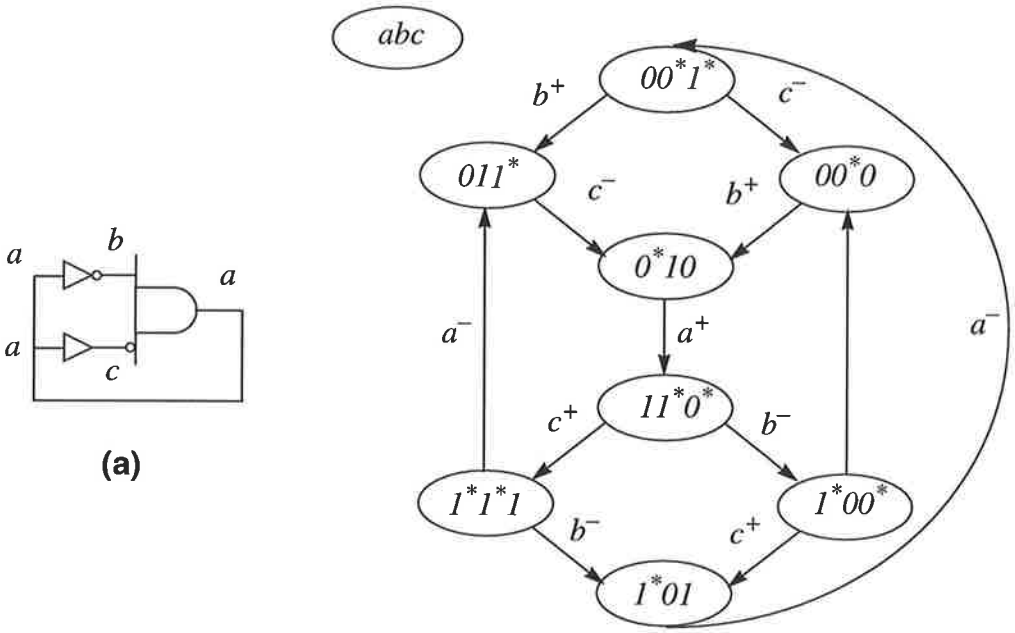


Figure 2.18: (a) An autonomous circuit and (b) its state transition diagram.

may be modelled as a non-inverting buffer with its own unbounded delay as demonstrated in Figure 2.18. These are the two major restrictions in Muller's speed independent theory.

Since mutual reachablity is an equivalence relation, the set of states in a STD are partitioned into mutually reachable sets of states, called *equivalence classes*. Figure 2.19 taken from [57] shows a symbolic STD with three equivalence classes, *{a,b}*, *{c}* and *{d}*.

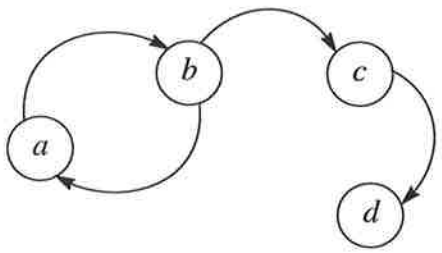


Figure 2.19: A symbolic state transition diagram. The equivalent states are {a,b}, {c} and {d}.

The following three definitions taken from [57] with minor changes give a clear and concise description for speed independent circuits:

The *terminal class* of a state transition sequence is the final equivalence class in that sequence.

A STD is called *speed independent* with respect to a state s_1, if there is only one terminal class reachable from s_1.[5]

A STD is called *speed independent* if it is speed independent with respect to each individual state.

Therefore, the STD in Figure 2.18 is speed independent, as every state is reachable from any other state, that is there is only one equivalence class in this STD. The STD in Figure 2.19, on the other hand, in not speed independent, as the initial state *b* may reach either of the equivalence classes, *{a,b}* and *{d}*.

The remarkable characteristics of a speed independent circuit lies in its robustness against the delay variations of the logic gates. So that the state transition sequence specified by the STD will eventually occur in the right order no matter what the individual gate delays are.

2.4.2 Two restricted types of speed independent circuits

In Figure 2.18-b two types of excitation are eminent: excited signals which eventually *fire* (that is change their logic values) before becoming stable, like 1^*in 00^*1^*, or excited signal which may become stable without changing their logic values, like node *b* (1^*) in 1^*1^*1 . The second situation shows a *choice*, where one of two (or more) excited signals manage to fire (that is are chosen) and in the meantime the rest of the excited signals become disabled.

In Muller's speed independent theory all signals are generated by the circuit itself, as there is no external signal applied form the environment. Therefore, in this model all signals involved in a choice are necessarily non-input signals, resulting in a non-deterministic operation for the circuit, which is normally not of much interest. Input choice, however, is a remarkable flexibility added to asynchronous circuits, and will be discussed

5. The definition for speed independent states is slightly different from [42], according to which {d} is the only final class in Figure 2.19, recognising the STD as speed independent.

in this chapter when input signals are allowed in describing circuit behaviour.

Semi-modular circuits[6] are a subset of speed independent circuits in which there is no choice, preventing the conflict situations mentioned above for Figure 2.18-b. In other words, in semi-modular circuits every excited signal will eventually fire, that is its excitation will not be withdrawn by firing another excited signal in the circuit[7].

Distributive circuits are, likewise, a subset of semi-modular circuits and are distinguished by the characteristic that, starting with any state in a STD, all disabled signals may only be enabled by (possibly different instances of) exactly one signal transition, that is the *excitation region* of a signal may only be entered by exactly one signal transition.

Definition 2.1: The *excitation region* of a signal transition is the largest possible set of connected states (in the STD) in which the transition is excited.

2.4.3 A flow table based speed independent circuit realization

Armstrong et al. in [2] propose two methods to implement both combinational and asynchronous circuits under the unbounded gate delay model. Both techniques are based on completion detection, of course at the price of some area overhead. In this section we study the combinational logic only. The basic idea is the same for sequential circuits, which undergo a flow table based procedure.

A general block diagram of this technique is shown in Figure 2.20.

Based on this model the environment knows when the next input vector may be applied, as the stable states are reported by the asserted *Req* lines, generated by the completion detector and based on some redundancy introduced in proper *coding* of stable inputs and outputs as explained here:

Method 1: spacer-data: In the first method two types of inputs are supplied by the environment and likewise two types of outputs are generated by the logic circuit as well.

---

6. Semi-modularity and distributivity (to be discussed shortly) are borrowed from the lattice theory. A STD is semi-modular (distributive) if the underlying cumulative diagram is semi-modular (distributive).
7. Martin [47] [50][51] has come up with a similar notion of *stability* of production rules as a necessary condition to avoid hazards in speed independent circuits. The rule of stability requires that if the guard of a production rule becomes true, it must not become false until the end of the computation.

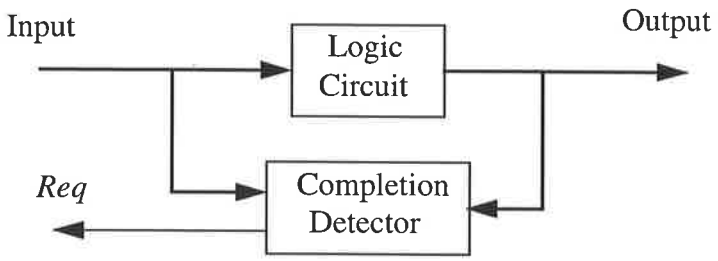


Figure 2.20: A general block diagram of Armstrong et al.'s design method.

Type one input/outputs are some coded data called *data words* (or codes), so that each data word applied to the logic circuit must generate a valid data word at the output. The second type of input/output is called a *spacer*. The input spacer must generate the output spacer.

There are two basic requirements in this design technique:

1- The input must alternate between data words and the spacer. If this sequence is applied to the circuit sufficiently slowly then the output will alternate between data words and the spacer as well, with some possible transient response.

2- During input (output) transitions from the spacer to a valid data word and also from a valid data word to the spacer no other valid input (output) word must be generated.

Condition 2 prevents any unspecified behaviour caused by the above transient response.

As an example data words may be coded as normal dual rail codes, in which a single bit *0* and *1* are coded as *01*, and *10*, respectively.

A valid data word detected at the output indicates a stable valid word at the input as well. Therefore, the completion detector has to only monitor the output lines to request a spacer. However, the case with the "data to spacer" transition is different, that is having detected a spacer on the outputs does not necessarily mean a spacer on the input lines, as the invalid input words during the input change may also cause a spacer on the output. This can result in premature input transition and eventually malfunction. This type of hazards (first reported in [2]) which are similar to what causes fundamental mode restrictions in Huffman methodology, is called *delay hazards*.

This problem may be solved by getting the completion detector to monitor both input

and output and not to request a new data word unless both input/output are settled at all-zeros. Figure 2.21 shows the timing diagram of this method.

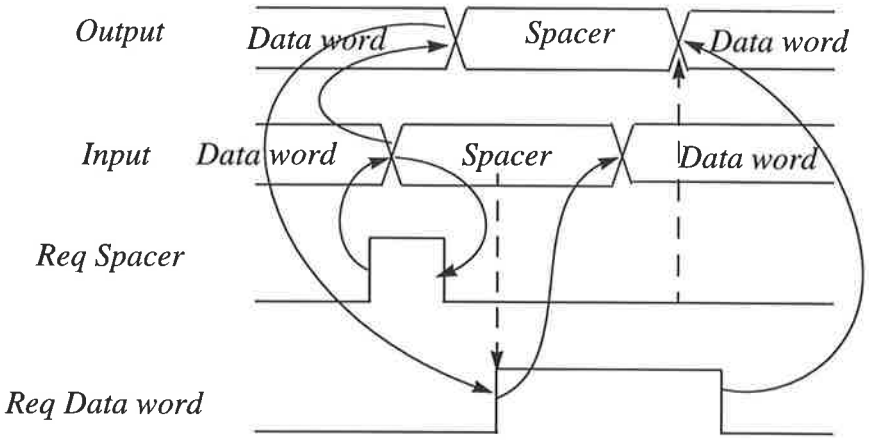


Figure 2.21: Timing diagram for the *data-spacer* method.

Method 2: alternating data: In The second method there are two separate sets of data words, so that no two inputs from one set must generate outputs from both of the sets, eliminating the need for an explicit spacer. The fundamental requirements now are as follows: 1-The input must alternate between these two set of data words. This implies that the output will alternate between the two sets as well. 2- During input (output) transitions from one set of data words to another one, no other valid input (output) word from either of the set must be generated.

The two required set of data words may be chosen again by allocating two coded bits like {*(01, 00), (10, 11)*} to the original uncoded bit *0* and *1*, respectively.

The second method has a better performance but at a considerable area penalty, due to data code alternation, although the principle idea is the same as the first method. In this technique input data alternating between the two sets of data words are applied to two combinational modules, each in charge of processing one group of data. The output of the irrelevant module is forced to all-zeroes. The request bus is again two bits wide: a data word of type-1 (type-2) is requested from the environment if the stable current output is of type-2 (type-1) and the module-1 (module-2)'s output is all-zeros.

2.4.4 High level graph specifications for asynchronous circuits

Two independent papers [71] [17] introduced a Petri net[8] based description for asynchronous circuits. The theory developed in these works may be considered as a bridge between lattice theory (the basis for Muller's work on speed independent circuits) and the theory of Petri nets, on the other hand.

Definition 2.2: A *Petri net* is a bipartite directed graph $G = \langle P, T, F, M_0 \rangle$, where P is a finite set of places or conditions, T represents a finite set of transitions, F is a flow relation: $F \subseteq (P \times T) \cup (T \times P)$ and M_0 is the initial marking. ◊

Figure 2.22 shows a Petri net which models the behaviour of a Muller C element if the transitions are interpreted properly. (See next section).

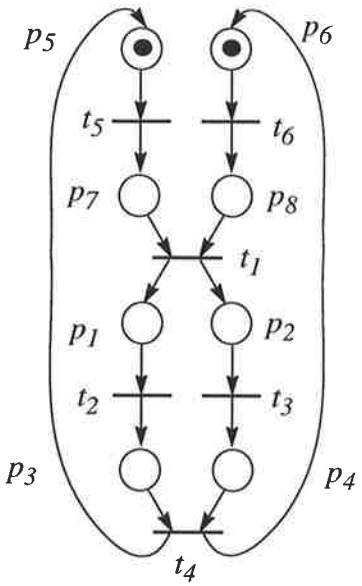


Figure 2.22: A Petri net modelling the behaviour of the Muller-C element.

Assuming a maximum of one token per place, the initial marking is represented by one token in some of the places of a Petri net (p_6 and p_5 in Figure 2.22). When a transition receives one token in all of its fanin places (that is the places which immediately precede that transition), then that transition is called *excited* and it may eventually fire. When a

8. This theory has originally been developed by Petri. For a comprehensive review see [61].

transition *fires* the tokens are removed from its fanin places and one token is placed in every fanout place of that transition (that is a place which immediately follows that transition). Notice that one token in every fanin place is necessary and sufficient to excite a transition, no matter what is happening in other parts of the net. This remarkable feature of Petri nets makes them an interesting tools to model asynchronous circuits. Using this capability partial ordering may be modelled readily.

Definition 2.3: A Petri net is called *safe* if no place can assume more than one token in any reachable marking.[9] ◊

Definition 2.4: In a *free choice* Petri net if a place has more than one fanout transition, then it is the only fanin place for all its fanout transitions. ◊

Definition 2.5: A *marked graph* is a Petri net with no choice, in other words in a marked graph no place has more than one input and more than one output transition. ◊

Figure 2.22 shows a marked graph Petri net, where there is no choice in the flow of transitions.

Definition 2.6: A *state machine* is a Petri net with no parallelism, in other words in a state machine no transition has more than one input and no more than one output place. ◊

Figure 2.23-a shows a state graph Petri net. Place p_1 shows a choice instance, while there is no parallelism in this Petri net.

2.4.5 Signal transition graphs

Definition 2.7: A *signal transition graph* (*STG*) [18] is a free choice Petri net in which each transition is interpreted as a physical signal transition on some node of a circuit. ◊

The signal transitions in a STG may be either input (applied by the environment) or non-input (generated by the circuit). STGs in general can model both choice and concurrency.

Figure 2.24-a shows the STG corresponding to the Petri net in Figure 2.22. Notice how the transitions in the Petri net have been *interpreted* as signal transitions. Rising and falling transitions of signal a are shown as $a^+$and a^-, respectively. Input signals are usu-

9. In [94] it is shown that some restrictions like safeness are not necessary, although they may lead to more efficient circuits. More examples are presented in [30].

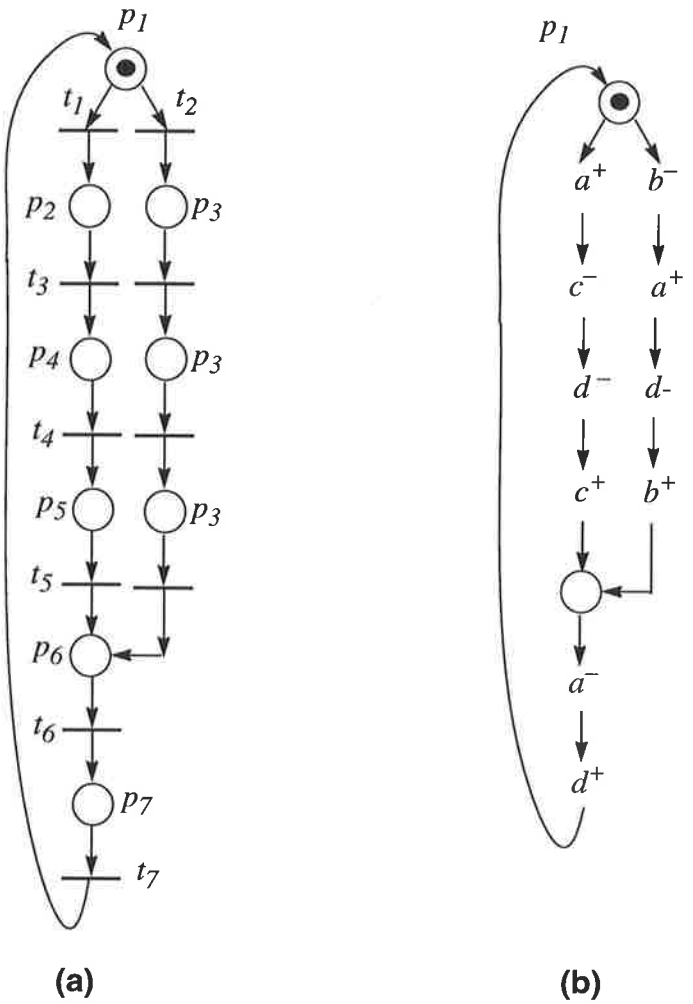


Figure 2.23: (a) A state machine Petri net, (b) the corresponding STG.

ally underlined, to be distinguished form internal and output signals. Internal signals are introduced to solve the CSC problem, to be discussed shortly. All places with one input and one output transition are eliminated from STGs for the sake of clarity.

Definition 2.8: A STG is called *well-formed* if it is live and its corresponding state transition diagram satisfies the complete state coding property.

Definition 2.9: A safe STG is *live* if every transition can be enabled through some transition sequence from every reachable marking. Furthermore, no two similar transitions of the same signal may follow each other unless they have one transition with the opposite direction in between.

Definition 2.10: A STG has the complete state coding (CSC) property if the state

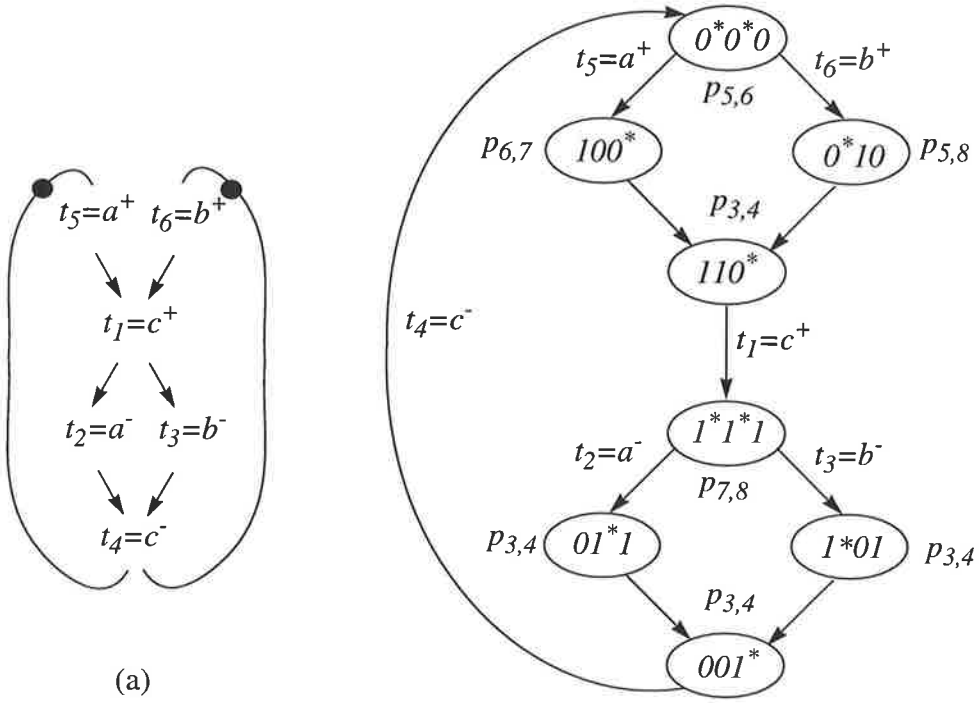


Figure 2.24: (a) The STG corresponding to the Petri net in Figure 2.22 with proper interpretation for the transitions, and (b) the resulting STD.

transition diagram corresponding to the STG has the CSC property.

Definition 2.11: A state transition diagram (or state diagram), STD is a two tuple $\langle S, E\rangle$ where S is a set of states and E is a set of edges so that $E \subset S \times S$. Each marking in the STG corresponds to one state in the STD. The value of each bit in a state bit vector is determined by the latest transition of that bit. There is an edge, E_{ij}, between S_i and S_j if and only if S_j *immediately* follows S_i.◊

Definition 2.12: A state, S_j, immediately follows another state, S_i, if M_j can be reached from M_i through the firing of a single transition, where M_j and M_i are the markings corresponding to S_i and S_j, respectively.

Definition 2.13: A STD has CSC property if there are no two identical states with different non-input excited variables. ◊

The STD corresponding to the STG in Figure 2.24-a is shown in Figure 2.24-b. Notice how a STD is generated as the STG is *executed*. In other words a STD consists of all possible markings in the STG, that is all reachable markings starting with an initial marking and following the above simple firing rule. Figure 2.24-b also shows the markings corresponding to each state. Therefore, a STG is, on one hand, an interpreted Petri net, and on the other hand a higher level of abstraction for the corresponding STD, which was originally introduced by Muller [57] based on the lattice theory.

Referring to the previous section note that a marked graph, that is a STG with no choice, generates a distributive STD.

Definition 2.14: A STD is semi-modular if all its excited variables fire eventually, that is no excited variable is disabled by the firing of another transition. In other words a choice free STD is called semi-modular.

Definition 2.15: A *detonant state* has a stable signal which becomes excited by more than one signal transition in more than one state immediately following that state. ◊

Detonant states may only be introduced in Change Diagrams to be discussed in the following section (and not in STGs) where weak precedence is allowed.

Definition 2.16: A STD is *distributive* if it is choice free with no detonant state.

Corollary 2.1: General STGs fall into a special class of *with choice-distributive lattices*, as there is no detonant state in STGs but (input) choice is allowed.

2.4.6 Change diagrams

A STG is able to model both *AND* and *Exclusive OR* (choice) casual relationships. For example the STG in Figure 2.24-b specifies that both a^+ and b^+ are required to fire in order to excite c^+. Figure 2.23-b on the other hand specifies a choice: when p_1 has a token, either a^+ or b^- may fire (but not both). Also, on the merging point of the two branches of the choice although either b^+ or c^+ may cause a^-, the transitions b^+ and c^+ are mutually exclusive. Change diagrams (CDs) introduced by Varshavsky et al. [91][33] do model *Inclusive OR* relationship between transitions while choice has not been considered in CDs. Therefore, the CSC property is reduced to the USC property while there is no input signal in the system:

Definition 2.17: A STD has USC violation (contradictory) if there are two identical

states but with different excited variables.

Figure 2.25 shows a change diagram, and the corresponding STD.

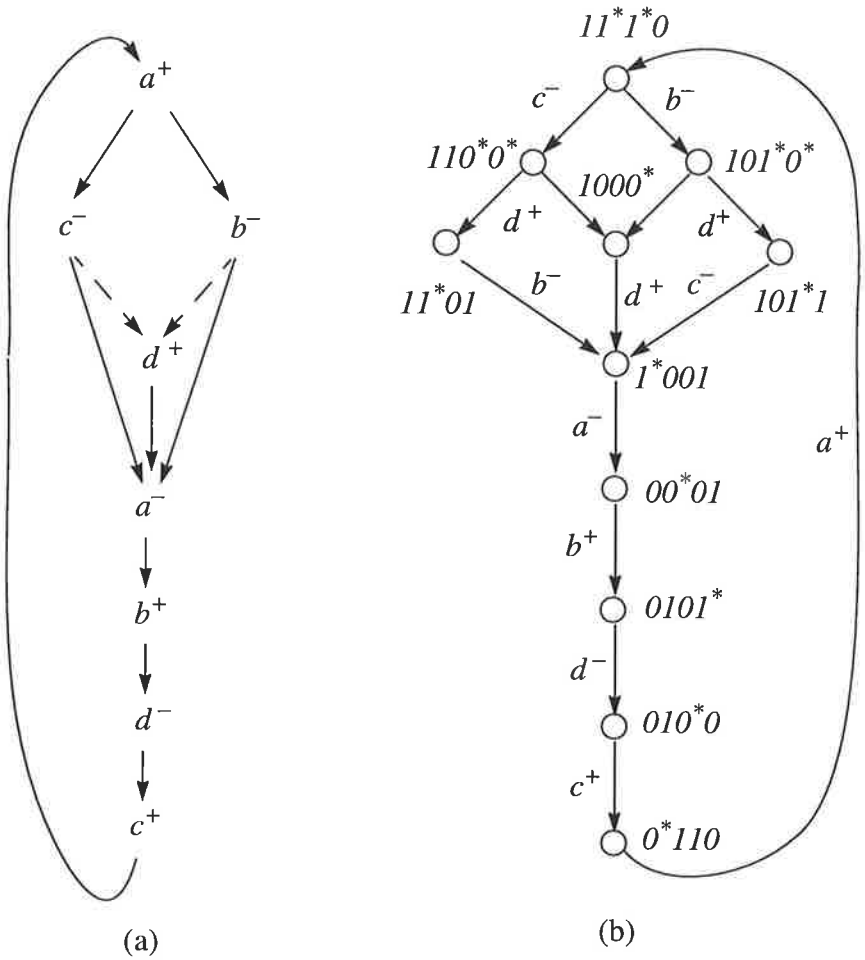


Figure 2.25: (a) A change diagram, and (b) the corresponding STD.

The *OR* causal relationship (weak precedence) is specified by dashed arrows in CDs, and the normal *AND* relationship by solid arrows (strong precedence). According to this CD both b^- and c^- can individually excite d^+, so that the sequences $c^-b^-d^+$, $c^-d^+b^-$, $b^-c^-d^+$ and $b^-d^+c^-$ all are legitimate now, a behaviour which may not be modelled by STGs.

Since no choices are modelled by CDs, the resulting STD (using the same algorithm as for STGs) are generally semi-modular, as no transition firing may disable another excited transition. A special class of CDs in which all causal relationships are of strong type are called *signal graphs*. This class of circuits corresponds to distributive STDs.

2.4.7 STG based implementations

STG's have been implemented as asynchronous circuits in different ways by different researchers. Chu [17][18] and Meng [55][56] have used complex gates to implement different nodes in the asynchronous network. This in fact is based on the exact assumption of the atomic model of gates in the Mullers's theory of speed independent circuits. A complex AND-OR-NOT gate almost satisfies speed independent requirements as long as no signal is used in the complementary form, as inverted inputs may easily violate isochronic fork assumptions. We will discuss this topic in more detail in Chapter 5.

Kishinevsky et al. [33] introduce 2 phase RS implementation, in which not only is the above problem resolved, but the technique also has two other interesting features to be discussed in Chapter 5. This technique however suffers from some area and performance overhead

Beerel [3][4][7] has developed a simple gate based technique to implement speed independent circuits. In his technique although extra nodes are introduced in the network, adhering to the two necessary and sufficient conditions of speed independency, (that is *acknowledgement* and *monotonicity*) he shows that the resulting network is hazard free. Acknowledgement requires that every signal request must be acknowledged before another request is sent to the same signal. Monotonicity requires that the request must reach the signal monotonically, that is the request has to be glitch free.

Moon [59][60] has assumed a well-behaved environment to implement asynchronous circuits from signal transitions graphs. In this technique the environment must be sufficiently slow to avoid delay hazards. Logic hazards are removed similarly to the conventional Huffman methodology.

Lavagno's simple gate design methodology [42] is based on the bounded wire delay model. In this technique delay hazards are avoided by proper delay padding on some of the wires, so the designer has to have exact knowledge about the delay bounds of different wires in the circuit. Logic hazards are again removed using a similar method to Moon's.

Myers [62][63] has introduced timed signal transitions, in which, based on the designer's knowledge of upper and lower delay bounds between any two consecutive signal transitions, some of the markings are found unlikely to occur, resulting in some area and hence performance improvement.

Martin [47][50][51], Brunvand [12] and van Berkel [85][87][88] (with some options for handshake circuits [69][90]) have introduced a systematic asynchronous design methodology starting with a CSP [27] like programming language. This high level description is then compiled in some stages into a speed independent circuit.

A more detailed review of some of these design techniques will be presented in different chapters in order to provide a better background for our work.

2.5 Conclusion

In this chapter some asynchronous design techniques with emphasis on the specific delay restriction were reviewed and it was concluded that all of these commonly used techniques assume one type of delay restriction or another, which is of course traded off against the complexity of the resulting implementation. In this context we highlighted two different notions of *simultaneous* and *concurrent* transitions. The closeness requirement of simultaneous transitions is only raised in systems with multiple environments. This situation can easily be modelled in a flow table based technique, but not in self clocked nor event based methodologies, in which an input burst may not be a subset of another input burst if both bursts are enabled in the same state. On the other hand, partial ordering can be modelled using event based descriptions like STGs, while they are too difficult to be represented with a flow table like language.

Chapter 3

Two Level Logic Implementation of Asynchronous Circuits from STGs under the Inertial Delay Model and the Well-Behaved Environment Assumption

3.1 Introduction

It was highlighted in Chapter 2 that the major advantage of the Petri net based description languages like STGs of asynchronous signal transitions, over the traditional flow table based techniques is their ability to model partial ordering, no matter what type of delay restrictions are taken into consideration to realize the STG as an asynchronous circuit. Under no restrictions for timing constraints for different signal transitions in a STG, that is

the unbounded gate delay model, the original speed independent circuit theory design techniques like [57][18][56] assume an atomic delay model, where each gate is modelled with an instantaneous decision element followed by an unbounded delay. As will be discussed in Chapter 5 a complex AND-OR-NOT gate reasonably satisfies this requirement, although they do not yet realize functions with both inverted and non-inverted inputs. In order to make all type of possible logic functions available under the atomic gate delay model the delay along inverters at some inputs of these gates has been assumed negligible by some researchers (that is atomic NOT-AND-OR-NOT gates).

Complex gates are only available in full custom VLSI designs which may incur a considerable waiting period between the design and the market. This fact has been the motivation for much research work to properly decompose complex gates into simple gates to reduce the variety of the gates required in a library and / or to relax the fanin restrictions, [36][7][44][14][37]although zero delay along inverters is still assumed. In addition to the above drawback of using complex gates, stacked transistors, which are invariably used in complex gates, suffer from too much noise margin degradation in technologies like n-channel MESFET Gallium Arsenide [24]. This fact does not allow speed independent circuits to be realized in such technologies, as the Muller C-element utilized in some decomposition techniques has to be implemented with two pull down stacked transistors[1] unless the *inefficient* NOR-NOR fashion is employed.

The unbounded gate delay model which was the first requirement in realizing speed independent circuits is considered too pessimistic by some researchers [42]. To relax this harsh requirement Lavagno [42] has developed an asynchronous design technique from STGs based on a bounded wire delay model, where he assumes that gates and wires realizing the circuit all have known bounded delay. Based on this knowledge the circuit is designed and fabricated with proper delay paddings to avoid delay hazards, (which of course incurs some overhead in terms of area, performance and testability). Therefore, the circuit will work according to the specification if the real delays do not exceed the presumed delay ranges.

Myers [63] has also developed an asynchronous design technique from STGs but

1. This implementation requires two p-channel stacked pull up transistors as well, which is a second reason why GaAs technology is unsuitable for speed independent circuits. These restrictions have recently been lifted by introducing complemetary GaAs technology [11].

again under a bounded gate delay model. The presumed delay bounds of the signal transitions of the STG are used to identify the non-reachable states in the corresponding state transition diagram. This redundancy removal normally results in a significant performance gain. The correct behaviour of the resulting implementations is again subject to conforming to the presumed delay ranges.

Moon [59] has developed an asynchronous design methodology from STGs but with different delay constraints which require the environment to not only comply with the signal transition ordering specified in the corresponding STG (as usual), but also be sufficiently slow to assure that the predecessor ordered transitions have already been stabilised and absorbed in the fanout gates before the new input transitions are activated. This technique does not suffer from stacked transistor limitation in the corresponding two level implementations and hence is suitable for implementation of STG based asynchronous circuits in technologies like GaAs, although the limited fanin restriction still exists.

In this chapter, based on the work in [59], we investigate the two level logic synthesis of asynchronous circuits from STGs under the *inertial gate delay model* in addition to the above constraints on the environment We show that multiple input high to low dynamic logic hazards are ruled out under the inertial gate delay model in two level SOP logic circuits. We then weaken the zero wire delay restriction and find an upper bound for the delay along critical interconnection wires and hence propose a *virtual isochronic fork* model for interconnection networks. Multiple input low to high dynamic hazards are then studied in two level logic implementations and it is shown that this type of hazard is unlikely to occur under the inertial gate delay model unless a liberal delay flexibility is required between the first level AND gates of SOPs. Static hazards, on the other hand, are shown to not be relaxed under the inertial delay model.

Delay hazards are investigated in two level logic under the well-behaved environment assumption[2] and it is uncovered that even this restriction does not necessarily lead to delay hazard free implementations, no matter whether the pure or the inertial delay model is considered. We show that this type of hazard has been overlooked in the design of the STG compiler, SIS [74].

2. A more general discussion of delay hazards in two level SOPs is covered in [79].

3.2 Basic notions and definitions

Definition 3.1: The logic function $f(x_1, x_2, \ldots x_n)$ of n input variables is a mapping from $\{0,1\}^n$ to $\{0, X, 1\}$.

Definition 3.2: A *literal* is a variable or its complement.

Definition 3.3: Each element in the domain of a function is called a *vertex*.

Definition 3.4: The subsets of vertices which are mapped to 1, X, and 0 are called the *on-set*, *don't care set* and *off-set* respectively of the function.

Definition 3.5: In an n variable environment a cube is the set of 2^k vertices, $0 \leq k \leq n$, which are identified by $(n - k)$ variables with fixed values and all 2^k possible combinations of two different values for the remaining k variables. So, a cube can be represented as a product term of the fixed literals (obviously with no complementary pair).

Definition 3.6: A cube A *covers* a cube B if all vertices in B belong to A as well.

Definition 3.7: If two cubes both cover the same one or more vertices, then they *intersect* each other. Notice that the intersection of two cubes is a cube as well.

Definition 3.8: When a literal is removed, the cube is *expanded*. Therefore the expanded cube covers twice as many vertices as the original one does.

Definition 3.9: An *implicant* is a cube which does not cover any off-set vertices.

Definition 3.10: If an implicant cannot be expanded into another implicant, then it is called a *prime implicant*.

Definition 3.11: An *on-set cover* (or simply cover) of f is a sum of product implementation of f in which the product terms (p-terms) are the implementation of a set of implicants by which all on-set vertices and possibly some don't care vertices are covered.

Definition 3.12: A cover from which one or more implicants can be removed is called *redundant*. Otherwise it is called *irredundant*.

Definition 3.13: During an input transition, if one or more spurious signal transitions may occur on the output of a combinational circuit under the delay models assumed for both logic gates and interconnection lines, then the *transition* has a *combinational hazard*.

Notice that the absence of any spurious transition on the output of a particular implementation does not necessarily imply a hazard free input transition.

In this paper hereafter by hazard we mean combinational hazard unless otherwise specified.

Definition 3.14: An input transition has a *static hazard* if as a result of that transition the output may change temporarily while it was expected to stay unchanged. A static hazard is called a 1-hazard or 0-hazard if the spurious pulses are low or high, respectively.

Definition 3.15: An input transition has a *dynamic hazard* if as a result of that transition the output may undergo a non-monotonic transition while it was expected to change monotonically.

Definition 3.16: An input transition is called

1- *adjacent* if the two input vectors differ in only one variable,

2- *non-adjacent* or *multiple* if the two input vectors differ in more than one variable.

The second case is unrealistic in the sense that it is very unlikely for all input variables to change simultaneously. Moreover, considering different parasitic delays with possibly independent values, the input change will in general not be sensed simultaneously even if they occur simultaneously. Therefore, it is more realistic to assume that input transitions happen one bit or more at a time and in an *arbitrary* order, that is *concurrent* or *parallel* transition of input variables.

Definition 3.17: The set of ordered input states which may occur in a multiple transition is called an *input route*.

Corollary 3.1: In a parallel input transition if the intermediate transitions happen one bit at a time, then there are $k!$ possible input routes, each k stages long, where k is the number of bits in the input vector which are to change.

Definition 3.18: The set of all input states appearing in all possible routes, is called the *transition cube*.

Definition 3.19: The set of ordered output states corresponding to the ordered states in an input route is called an *output route*.

Definition 3.20: If at least one of the output routes shows a non-monotonic output

transition, then the corresponding input transition has a *function hazard.*

Corollary 3.2: Function hazards cannot be eliminated.

Lemma 3.1: During a parallel input transition the real sequence seen on the output is not necessarily the corresponding output route.

Proof: Arbitrary delays may change the order by which the input variable changes are *seen* by the last gate. Moreover, different paths introduce different propagation delays even when a single variable change is propagated, so that during a transition the output may take a value which does not correspond to any vertex in the transition cube.Therefore,

Definition 3.21: If an input transition is free of function hazards but the output may still suffer from spurious transitions (which are due to parasitic delays of the circuit elements), then the transition has a *logic hazard.*

Corollary 3.3: All function hazards can be identified through the circuit's truth table, while the implementation logic equation and also the delay models are required to determine all existing logic hazards.

Definition 3.22: A delay element is called pure if all events on its input are transferred to the output after some propagation delay time. While in inertial delay the input pulse width must be sufficiently wide, that is greater than the propagation delay, to pass through the element. Shorter pulses will be absorbed and disappear.

3.3 Well-behaved environment

The following two definitions have been taken from [59]

Definition 3.23: The environment is *well-behaved* if

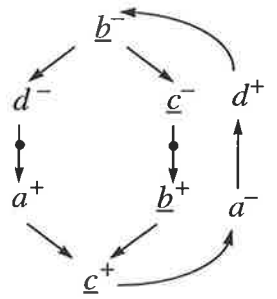
- it sends inputs according to the order specified by the STG, and
- it applies an input transition $s*$[3] only after the output signals which have signal s in their fanin set, are stable.

Definition 3.24: A signal is stable if that signal and all the signals in its transitive fanin have reached their steady-state values.

---

3. s* represents a signal transition on node s with unspecified direction.

Notice that well-behavedness is less restrictive than the fundamental mode requirement in which the next input vector may only be applied after the circuit has absorbed the previous input vector and reached the steady state condition, that is all delay elements both implicit and explicit have achieved outputs equal to the corresponding inputs. In other words the partial ordering feature of the specification is still valid in this methodology, similarly to other STG based implementations. Consider the STG shown in Figure 3.1, where the transitions b^+ and $a^+$are enabled. The well behaved environment is allowed to fire b^+ as soon as c^- fires and is absorbed, no matter what stage the transition d^- is, as it is positioned on a parallel branch.



$$a = \overline{b}.\overline{d}+\overline{c}.\overline{d}$$
$$b = \overline{c}+b.\overline{d}$$
$$c = \overline{(\overline{b}+\overline{c}).(\overline{a}+\overline{b})}$$
$$d = \overline{a+\overline{b}+\overline{c}}$$

Figure 3.1: Parallelized STGs allow output signal to fire concurrently with input transitions even under well-behaved environment assumption.

Corollary 3.4: From Definition 3.23 *we conclude*[4] that under the well-behaved environment assumption no two *sequential output transitions* are allowed in the STG if the (unrestricted) delay between these two may cause any delay hazards, otherwise there is no point in restricting the environment, that is sequential outputs can cause the same problem (that is, delay hazards) that the well-behaved environment assumption is to prevent.

Therefore, each cycle in a well-behaved environment based STG should normally look like a recent notion of burst mode machine [65][66][96][97] in which an input burst is followed by an output burst and vice versa. In other words this class of STGs are able to model burst mode machines while partial ordering may also be modelled.

4. Nothing has been mentioned regarding this restriction in the literature so far, as far as we are aware.

The details of the extraction of two level logic equations of different variables comprising an asynchronous circuit from STGs [18] are not repeated here. Therefore, we start with the required logic equations and study different types of hazards in two level logic under the inertial delay model.

3.4 Inherent function hazards

Theorem 3.1 has been proved in [59]. Here we use a more straightforward approach to achieve the same result.

Theorem 3.1: All parallel transitions in well-formed STG's are free of function hazards.

Proof: Referring to Definition 3.20 recall that function hazards are independent of the implementation and can be well conceptualized under the *no-delay gate model* (which is a special case of the speed independent unbounded atomic gate delay model) and are manifested as a (unwanted) non-monotonic output route. Notice, however, that for all parallel transitions modelled by STGs all *possible* output routes are *necessarily* taken into consideration, and explicitly extracted and displayed in the resulting state transition graphs (when the STG is executed) to (partially) satisfy the requirements of the *unbounded gate delay model*. In other words, for each atomic gate implementing the corresponding variable of the STG, each input route out of *k!* routes is considered possible and hence a valid input route. In the presence of function hazards not all input routes are valid, as some of them force the output to undergo spurious transitions, while in STG based specifications there is no signal transition other than those specified by the STG (under the *atomic gate model*) and hence there is no spurious signal transition causing non-monotonicity in any output routes. So the proof is complete. ◊

Therefore, all functions derived from well-formed STGs are free of function hazards. However as mentioned above and to be discussed in more detail later in this chapter, delay hazards used by precedence reversal are in fact newly introduced function hazards which have to be avoided in designing robust asynchronous circuits.

3.5 Multiple input change high to low dynamic hazards

3.5.1 Introduction

Different types of hazards have been studied extensively under the bounded gate and

wire delay model [9][10][83][64]. It is well known that under this delay model not all multiple input dynamic logic hazards can be removed from all two stage combinational logic circuits. In this section we restrict the delay model to the well-known inertial gate delay and show that under this model half of the dynamic logic hazards can no longer occur in two level logic circuits. We then weaken the zero wire delay restriction and find an upper bound for the delay along critical interconnection wires and hence propose a *virtual isochronic fork* model for interconnection networks.

3.5.2 Dynamic hazards

Any combinational logic function can be realized in the "two level" form of Sum of Product (SOP) logic possibly with inverters for some of the inputs. As a result of a multiple input transition, a product term in the corresponding implementation may stay low or high, may have a high to low or low to high transition, may have a static 0-hazard or a static 1-hazard or a dynamic hazard.

Definition 3.25: The different product terms described above are called a_0, a_1, t_0, t_1, h_0, h_1 and d, respectively.

Lemma 3.2: In a multiple input change causing a high to low output transition for a SOP circuit, the h_0 (with only one spurious pulse), a_0 and t_0-type product terms are the only possible product terms, under monotonic input transitions.

Proof: a_0 type product terms have no effect on the behaviour of the function.

No a_1 or t_1 type product terms may exist, as they require the output to reach a final 1-state.

An h_0-type with more than one spurious pulse, an h_1 and also a d type product term cannot exist either, as this could only occur if a cube is first exited and then re-entered, which would require that a variable had at least two transitions, which is not monotonic.

To further clarify notice that when the output of an AND gate (that is a p-term) is pulled down by an input signal f (so $f = 0$) during a multiple input change, the output cannot be pulled up again, as f is still at logic low, and will stay low until the end of the transition, because the transition is assumed to be monotonic.

A rising input followed by a falling one of an AND gate may cause an h_0 type product term. An h_0 type product term is consistent with the 1-0 output transition.

A t_0 type product term is the replica of the intended output and is caused by a falling input of an enabled AND gate.

Therefore, a_0, t_0 and h_0 are the only possible product terms in a multiple monotonic input change high to low output transition. ◊

In the following discussion we assume that there are no a_1 and t_1 type product terms in the input transitions. Otherwise, according to Lemma 3.2 the transition cannot be of 1-0 type. We further assume that the input transitions under consideration are monotonic and free of function hazards unless otherwise specified, therefore

Lemma 3.3: Under the bounded but unrestricted wire and gate delay model and in a SOP circuit, a multiple input transition has a high to low dynamic hazard iff there is at least one t_0 type and one h_0-type product term.

The following Lemma from [64] states the concepts of Lemma 3.3 in a different way:

Lemma 3.4: Under unrestricted gate-wire delay model, a 1-0 multiple input transition suffers from a dynamic hazard iff a p-term in the cover intersects the transition cube but does not cover the initial vertex (illegal intersections). ◊

In this section it is shown that under the inertial gate delay model these two Lemmas are not valid any more.

Considering these two Lemmas, in order to generate a high to low output transition with a dynamic hazard, first all t_0 type p-terms have to be exited (legal intersections terminate), so that the output of the circuit is pulled down as well and then one or more h_0 pulses (illegal intersections) occur, that is the output experiences one or more spurious pulses (see Figure 3.2).

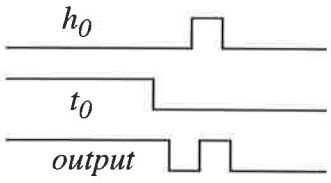


Figure 3.2: High to low dynamic hazard caused by one h_0-type p-term.

It becomes clear that h_0 type p-terms (illegal intersections) are the major cause of dynamic hazards, as the other partner, that is at least one t_0-type p-term (legal intersection)

is inevitable for an output high to low transition.

h_0 type p-terms have three different causes:

1- Function hazards (Dynamic function hazards are caused by static function hazards),

2- Static logic 1-hazard,

3- The rest, we call them *real dynamic hazards.*

In the following subsections the conditions under which function hazards may occur are first determined. These conditions must be avoided or dynamic hazards are inevitable, no matter what the delay model is. Then we discuss dynamic logic hazards caused by static logic hazards. These hazards can be removed using the traditional method of adding the required redundancy to the cover. This can always be performed no matter what the delay model is. These first two groups are in fact the minor groups of dynamic hazards. We show that the major group, that is the real dynamic hazards which are problematic under the unrestricted gate and wire delay model, cannot occur under the inertial delay model if they are of type high to low.

3.5.2.1 Dynamic function hazards (Type 1)

Definition 3.26: An implicant inside the transition cube which cannot be expanded to another implicant inside the transition cube, we call a *Transition Cube Prime (TCP) Implicant.*

In Theorem 3.2 and Theorem 3.3 normal transition cubes, that is transition cubes with at least two off-set vertices are considered. The special case of one off-set vertex is studied in Theorem 3.4.

Theorem 3.2: In a high to low output transition, if all on-set vertices in the transition cube are not covered by a single implicant, then the transition has a 1-0 dynamic function hazard iff the initial vertex is not within the overlapping area of the TCP implicants covering the on-set vertices of the transition cube.

Proof:

(Necessary) Notice that a dynamic function hazard occurs iff there is an input route which leaves the on-set area of the transition cube and then enters it again and finally terminates with an off-set vertex (see Definition 3.20). This, however, may only happen if

the first left area and the second entered area belong to two different cubes, as when one cube is exited it may not be re-entered under a monotonic input transition. Therefore, if the initial vertex, v, is in the overlapping area of the TCP implicants, no high to low dynamic function hazard may occur.

(Sufficient) Now suppose that there is an on-set vertex, x, in the transition cube but there is no on-set cube covering both v and x. It is shown here that such a transition suffers from a 1-0 dynamic function hazard.

Since there is no on-set cube covering both v and x, there must be at least one off-set vertex, w, which is less than x and greater than v in terms of the POSET theory[5] (See Appendix A). Therefore, in the path from v to w and then to x, there is a static function 1-hazard. Notice that the destination vertex, $z \neq w$, is the *greatest* element in the transition cube, that is, it is reachable from all vertices within the transition cube, including x. This means that $vwxz$ is one or more input routes with a non-monotonic output route, that is, a 1-0 dynamic function hazard.◊

Example 3.1:

Consider the multiple input transition $t \rightarrow z$ in Figure 3.3. Here the transition cube is the whole 4×4 map with the on-set vertices not covered by one cube only. Notice that the starting vertex, t, is located inside the overlapping area. Therefore, the transition is free of function hazard.

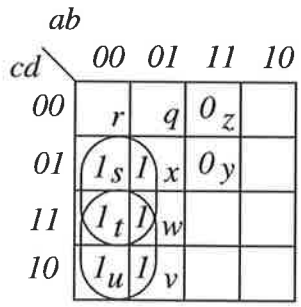


Figure 3.3: A transition cube (the whole map) with on-set vertices not covered by one cube.

5. In Appendix A we show that every transition cube in a parallel monotonic input transition to a logic circuit realizes a *distributive lattice* under the *follows* partial ordering.

Now consider the transition $u \rightarrow y$, which has the same transition cube but the starting vertex is outside the overlapping area, which causes a function hazard, as an input route, say, *ursxy* corresponds to a non-monotonic output route, *10110*. ◊

3.5.2.2 Dynamic logic hazards caused by static logic 1-hazards (Type 2)

Figure 3.4 demonstrates a logic circuit not implemented with the prime implicant as a p-term.

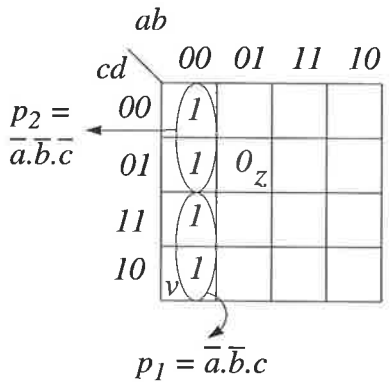


Figure 3.4: Dynamic hazard caused by static hazard.

Consider the multiple transition *0010 (v)* $\rightarrow$ *0101 (z)*, for which the transition cube is $\overline{a}$. This implementation suffers from a static 1-hazard, as during the transition the enabled p-term, p_1, may turn off being replaced with another newly enabled p-term, p_2, which eventually manifests itself as a 1-0 dynamic logic hazard. This would have never occurred if the prime implicant $\overline{a}.\overline{b}$ had been implemented as a p-term and replaced those two implicants.

Now consider the same transition and hence the same transition cube, but in a different function $f = \overline{a}.\overline{b} + a.\overline{b}$, shown in Figure 3.5.

Notice that one of the p-terms, $a.\overline{b}$ falls outside the transition cube, so that there is no risk of switching from one p-term to the other one, that is why TCP implicants ($\overline{a}.\overline{b}$ in this example), are significant, although prime implicants are obviously sufficient to guarantee a static hazard free transition, and furthermore, are normally implemented as individual p-terms in practice. We will later show an example in which both type 2 and 3 dynamic hazards are involved.

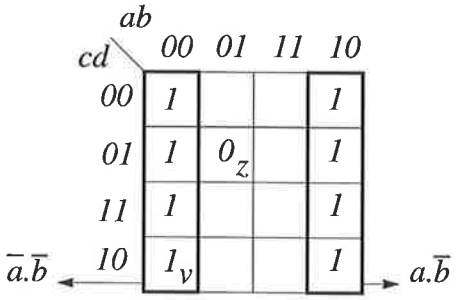


Figure 3.5: TCP or prime implicants, either can be used to avoid static hazards.

Notice that one characteristic of type 2 dynamic hazards is that the output of a function may be pulled down before exiting the on-set area of the transition cube (see Figure 3.2).

3.5.2.3 Real dynamic hazards (Type 3)

We assume that in the multiple input transition under consideration all t_0-type TCP implicants (or other implicants which cover t_0-type TCP implicants) exist in the corresponding implementation, unless otherwise specified. It was shown how the lack of this requirement results in dynamic hazards caused by static hazards. Notice that it is acceptable to expand a TCP implicant out of the transition cube, as discussed in the example shown in Figure 3.5.

Before introducing the major theorem, consider how a real dynamic hazard may occur: Starting from a (on-set) vertex in the overlapping area of all the t_0-type p-terms and moving toward the (off-set) destination, one or more h_0-type p-terms may become excited and then affect the output of the circuit after all t_0-type p-terms have been exited. These problematic obstacles are those implementation p-terms overlapping the t_0-type p-term(s) being travelled in the transition cube.

Example 3.2

Consider the implementation shown in Figure 3.6 with one t_0 and two h_0-type p-terms, in which the multiple transition *abcd*: *0011 (v)* $\rightarrow$ *0100 (z)* occurs. The transition cube corresponding to this transition is $\bar{a}$. During this transition one or both of these h_0 type p-terms may get excited but be sensed by the OR gate only when p_1 has been pulled down, resulting in a non-monotonic output change, that is a dynamic hazard. ◊

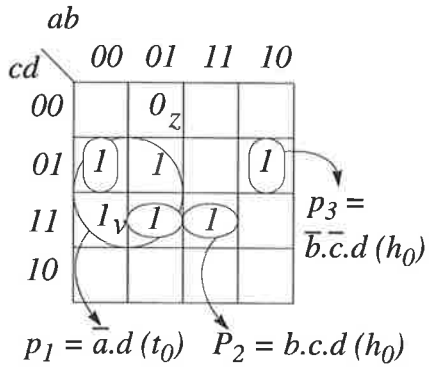


Figure 3.6: High to low real dynamic logic hazard generation in unrestricted delay model.

Notice that here unlike type 2, the output of the circuit necessarily stays high as long as the on-set vertex area has not been exited.

Considering the dynamic hazard generation mechanism, the following Theorem [6]states a relationship between the disabling sequence of t_0 and h_0-type p-terms:

Theorem 3.3: In a multiple input transition 1-0 dynamic logic hazard, under the bounded but unrestricted wire and gate delay model, each h_0-type p-term is necessarily disabled by at least the same input transition which disables one or more of the t_0 type p-terms.

Proof: Consider the possible situations that a h_0 type p-term may have in a multiple input transition: It may be isolated from all t_0 type p-terms or may overlap one or more t_0 type p-terms. The first case introduces a function hazard according to Theorem 3.2, as the t_0 type p-terms are supposed to be at least TCP implicants.

We discuss the case of one t_0 type p-term. Two or more t_0's cases may be treated similarly.

Recall that the intersection of two cubes p_1 and p_2 is a third cube $p_3 = p_1 \wedge p_2$. Now consider two overlapping p-terms p_1 (h_0) and p_2 (t_0) shown in Figure 3.7.

---

6. The notion in this theorem has been stated in [59] as well but in a different way from our approach, which was performed independently.

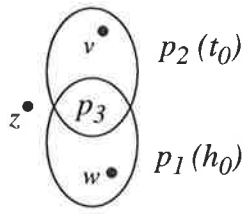


Figure 3.7: Graphical representation demonstrating t_0 and h_0 p-terms participating in a possible 1-0 dynamic hazard.

Starting from an initial vertex such as v, when p_2 is exited then either p_1 is exited as well (vertex z), or a vertex such as w in the non-overlapping area is entered. However, if w is outside the transition cube, then it is NOT reachable, and if it is inside the transition cube there is a function hazard according to Theorem 3.2. Therefore, if p_2 is exited, p_1 is also necessarily disabled with the same input transition as well, in a function hazard free environment, even if it might already have been disabled by one or more other input transitions or it might have never been excited at all.

Notice that z cannot belong to another h_0, as the same reasoning also applies to that. ◊

Example 3.3:

Consider the parallel transition *abcdef*: 000 100 (x) → 010 111 (z) for which the transition cube is $\bar{a}.\bar{c}.d$ (the highlighted area in Figure 3.8). Starting from x, the t_0 p-term, p_2, may be exited by any of transitions a^+, b^+, d^-, or e^+. Notice, however that a^+, and d^- are not possible, otherwise the transition cube boundary will be violated. On the other hand, b^+ or e^+ can individually disable the p_1 product term as well. ◊

Now we consider the special case in which there is only one off-set vertex in the transition cube. Theorem 3.4 uncovers sufficient reasons to prove that such a transition is necessarily hazard free.

Theorem 3.4: If there is exactly one off-set vertex in the transition cube, then the starting vertex is necessarily the unique intersection vertex of all TCP implicants.

Proof: Suppose that $z = (a_1, a_2, \ldots a_n, b_1, b_2, \ldots b_m)$ is the only off-set vertex of a function f, where b_i's are the fixed literals which specify the transition cube, and a_i's are the variable literals which determine the specific vertex in the transition cube. For the sake of brevity we ignore the b_i's. Therefore, the starting vertex, v, becomes $(\bar{a}_1, \bar{a}_2, \ldots \bar{a}_n)$ and the

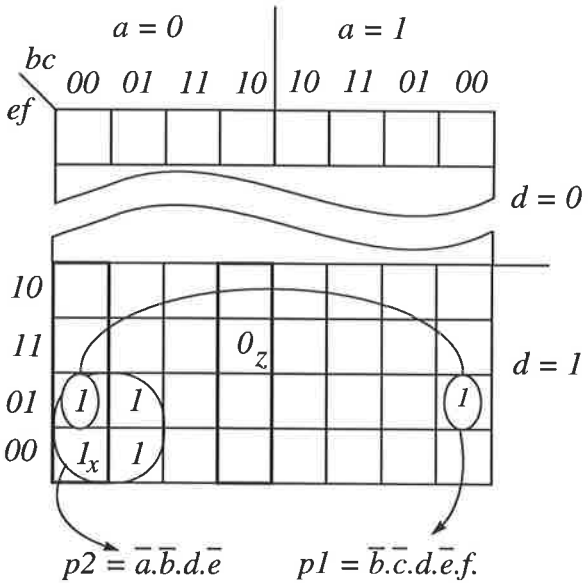


Figure 3.8: An example demonstrating how an overlapping h_0 p-term is disabled when the t_0 p-term is disabled.

function f realizing the transition cube may be represented as

$$f = \overline{a1.a2. \ldots .an}$$

or

$$f = \bar{a}_1 + \bar{a}_2 + \ldots + \bar{a}_n$$

The above equation clearly shows that the function f has n (TC) prime implicants intersecting in the unique vertex, $(\bar{a}_1, \bar{a}_2, \ldots \bar{a}_n)$, which is the starting vertex of the transition.◊

Corollary 3.5: A single off-set vertex transition cube has a unique starting vertex for a high to low transition.

Lemma 3.5: A transition cube with exactly one off-set vertex represents a dynamic hazard free transition.

Proof: First notice that one off-set vertex transition cube is free of dynamic function hazards, as in order to have a non-monotonic output transition such a hazard needs at least two off-set vertices.

Now, according to Theorem 3.4 the starting vertex is in the overlapping area of all TCP implicants, in other words all p-terms implementing the on-set area of the transition

cube are of t_0 type, that is there is no h_0 type p-term in the implementation if all TCP implicants are implemented as p-terms, unless there are other p-terms added to the set of these t_0's. This, however, does not cause any hazards, as each newly added h_0 vertex is necessarily covered by one t_0 type p-term, and this means that, with similar argument to Theorem 3.3, there is no high to low dynamic logic hazard. ◊

No restriction to the delay model has been assumed so far, that is, both wire and gate delays have been considered unrestricted but bounded. Now, we restrict our discussion to the model in which a logic gate is modelled with an instantaneous decision element followed by an arbitrary inertial delay. Interconnection wire delays, however, are assumed negligible. We will relax this restriction later. Notice that what is crucial is the lack of any restriction on the relative size of delays. Hence, the following theorem presents some sufficient conditions for multiple input transitions to occur hazard free:

Theorem 3.5: There is no *1* to *0* dynamic logic hazard for a monotonic multiple input transition in a SOP circuit under the inertial gate delay model.

Proof: According to Lemma 3.3 a high to low dynamic logic hazard necessitates at last one h_0 and one t_0 type p-terms in the corresponding SOP implementation. On the other hand Theorem 3.3 specifies that the h_0 type p-term has to be disabled by the same signal transition that disables a t_0 type p-term. This implies a minimum logic circuit as shown in Figure 3.9. Suppose that a^- is the input transition disabling both p_2 (t_0) and p_1 (h_0) and b^+ is the input transition enabling p_1.

Therefore, $b^+ \rightarrow {p_1}^+$, $a^- \rightarrow {p_1}^-$, $b^+ < a^-$ and $a^- \rightarrow {p_2}^-$, or

$$a^- < {p_2}^- \qquad (3.1)$$

where $\rightarrow$ shows the causal relationship[7] and $e^+ < g^-$ is a temporal relationship meaning that e^+ happens before g^-. In other words e^+ and g^- may be interpreted as the corresponding transition instants. b^+ is the input variable transition causing the h_0 node to rise. Notice that whether variable b is inverted or not does not affect the generality of the discussion. An inverted b can only defer the rising edge on p_1, if any.

Now it is shown that under the above mentioned conditions, the h_0 p-term cannot

7. Notice that here the sign $\rightarrow$ is used for a different purpose from that for which it was used earlier, that is to show a transition from one input vector to another one. The intended meaning is clear from the context.

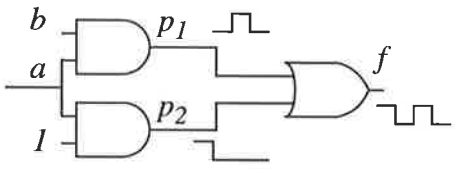


Figure 3.9: Dynamic hazard in the absence of wire delay in SOP circuits.

cause a dynamic hazard. The same reasoning applies to other h_0's and also other t_0's.

Since the gate delay model is assumed inertial, p_1^+ has to occur before a^-, otherwise there is no spurious pulse on p_1 and hence no spurious transition on f, that is

$$p_1^+ < a^- \qquad (3.2)$$

On the other hand, comparing (3.1) and (3.2) yields

$$p_1^+ < p_2^- \qquad (3.3)$$

Therefore, the OR gate *sees* its inputs according to the temporal precedence shown in (3.3), which under the atomic gate model guarantees a hazard free transition. ◊

In the remainder of this section an example including both type 2 and 3 hazards are presented:

Consider the multiple transition *abc*: *111* → *001* for the logic function $f = \bar{a}b+ac$ shown in Figure 3.10. Figure 3.11 shows a possible hazardous timing diagram for this transition.

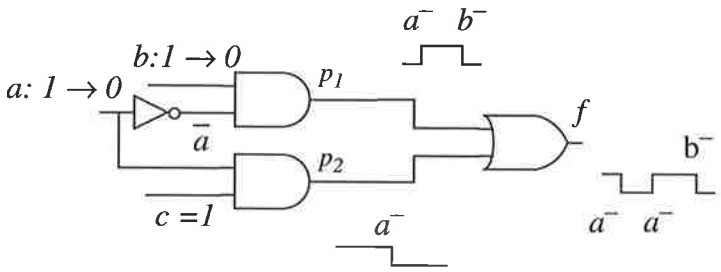


Figure 3.10: Dynamic hazard caused by static hazard assuming SI model.

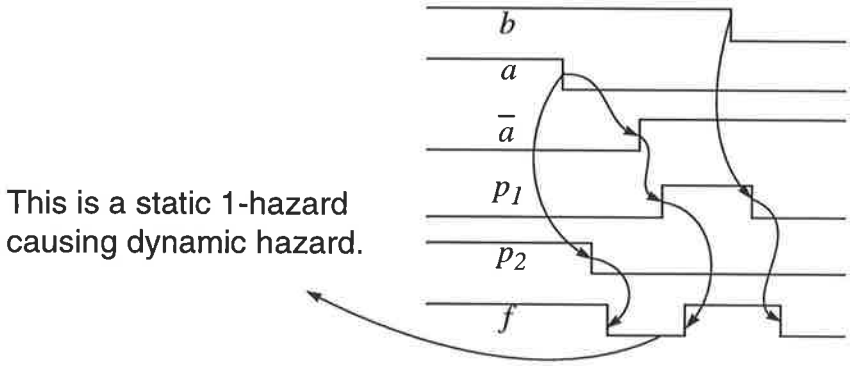


Figure 3.11: Timing diagram showing dynamic hazard caused by static hazard.

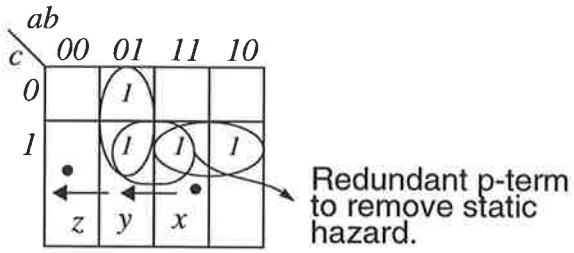


Figure 3.12: K-Map showing the role of static hazard in causing dynamic hazard.

Here in Figure 3.11 the output spurious logic low pulse is in fact a *static 1-hazard* caused during the transition from x to y, that is an effect of a^-, as shown in Figure 3.12. Applying the classical solution (that is introducing the redundant cube bc to the cover) removes this static hazard and hence the corresponding type 1 dynamic hazard. On the other hand, now referring to Theorem 3.5, the h_0-type p-term, that is $\bar{a}.b$, cannot cause a type 3 dynamic hazard either. (Notice that cube bc is exited by the same input transition as cube $\bar{a}b$ is exited, that is b^-.) However, under the unrestricted wire-gate delay model this dynamic hazard could not be eliminated without introducing another hazard. Figure 3.13 shows a modified timing diagram.

Notice that 0-1 dynamic hazards cannot be eliminated even under the inertial delay model, although they are relaxed under this delay model, as shown in a later section.

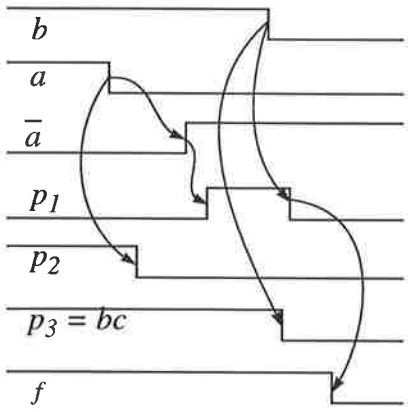


Figure 3.13: Static hazard and hence dynamic hazard is now removed under the inertial delay model.

The following example shown in Figure 3.14 clarifies this difference between the two types of dynamic hazards: Consider the multiple transition *abcd*: *0001 (v)* $\rightarrow$ *0010 (z)*, in which both p_1 and p_2 are entered as c^+ occurs, and then p_2 is exited by d^-. Now if the inertial delay of gate p_1 is so large that before p_1 is being pulled up, p_2 experiences both its transitions, then a 0-1 dynamic hazard is inevitable.

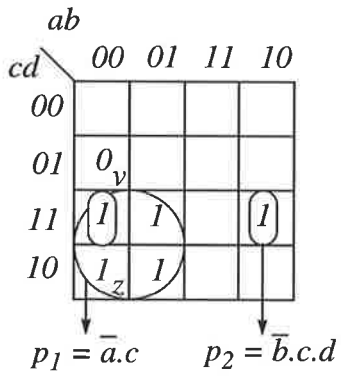


Figure 3.14: Inertial delay model does not prevent low to high dynamic hazards.

3.5.3 Delay bound for critical interconnection lines

Definition 3.27: Referring to Lemma 3.3, the h_0-type AND gates, their outputs and the corresponding AND to OR interconnection wires are called *critical gates, critical*

nodes and *critical wires* respectively. Also, the t_0- (and t_1 for Section 3.6) type AND gates, their outputs and the corresponding AND to OR interconnection wires are called *transition gates, transition nodes* and *transition wires* respectively.◊

In the theory of speed independent circuits, although wire delays are assumed negligible, they can be modelled with a delay element (as if there were a non-inverting buffer) if desired. This element does not cause any problems as long as that wire drives only one fan-out gate. However, for two or more fan-outs if some delay is attributed to the wire, then some restrictions, like isochronic forks [48][86], are required[8].

In our discussion regarding dynamic hazards, however, an arbitrary delay could no longer be allocated to a critical wire even if only one fan-out gate were driven by that wire, as this newly introduced delay can easily cause equation (3.3) to not be satisfied any more. This, however, is not more restrictive than isochronic forks and is less restrictive than extended isochronic forks [89]. As shown below, a similar assumption to isochronic forks on transition and critical wires, which is weaker than zero wire delay, still guarantees hazard free operation. We call this type of interconnection wires connecting the critical and transition nodes to the OR gates, *virtual isochronic forks*.

3.5.3.1 Critical wire delay restriction and virtual isochronic forks

Referring to Figure 3.9 and considering the line delays as shown in Figure 3.15, in order to have a hazard free transition, the two transitions p_1^- and p_2^- must be seen by the final OR gate in the following temporal order

$$p_1'^{+} < p_2'^{-} + \Delta t_{OR}^{+} \quad (3.4)$$

where

$p_1' = p_1 + \Delta t_1$

$p_2' = p_2 + \Delta t_2$

$p_1 = b + \Delta p_1$

$p_2 = a + \Delta p_2$

in which p_1 is the transition instant on node p_1, Δt_1 and Δt_2 are the delays of critical and transition wires, respectively and Δt_{OR} is the inertial propagation delay of the OR

8. Except for the limited class of delay insensitive forks (see Chapter 5).

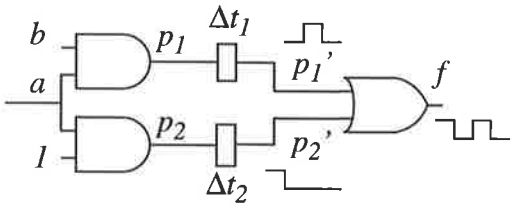


Figure 3.15: Dynamic hazard in the presence of wire delay.

gate. Substituting for p_1' and p_2' in (3.4) yields

$$b^+ + \Delta p_1^+ + \Delta t_1^+ < a^- + \Delta p_2^- + \Delta t_2^- + \Delta t_{OR}^+$$

or

$$\Delta p_1^+ + \Delta t_1^+ < (a^- - b^+) + \Delta p_2^- + \Delta t_2^- + \Delta t_{OR}^+ \qquad (3.5)$$

where Δp^+ and Δt^- are the low to high and high to low propagation delay of the AND gate p and an interconnection wire, respectively.

Now considering that due to the inertial delay model of the critical AND gate, Δp_1^+ cannot exceed $a^- - b^+$ (or there is no static 0-hazard), equation (3.5) is simplified as follows:

$$\Delta t_1^+ < \Delta p_2^- + \Delta t_2^- + \Delta t_{OR}^+ \qquad (3.6)$$

Under the assumed gate model and in order to have a hazard free output transition, equation (3.6) shows the required restriction (that is, an upper bound) on the delay of the critical wire. Assuming equal rise and fall times for the wire delays and according to (3.6), if the critical and transition wires have similar delays (an assumption similar to isochronic forks), then equation (3.6) and hence the hazard free operation requirement is satisfied with some safety margin. We call this type of matched interconnection wires *virtual isochronic forks*. In the example presented above an asymmetric virtual isochronic fork is

9. When the term Δt_{OR} is added to this equation, the OR gate is in fact allowed to operate in a so called non-semi-modular fashion, as the excited output may be disabled by the rising transition of p_1. On the other hand, p_1 may also be disabled by a^-. These, however, do not cause any concern, as having satisfied equation (3.4), no hazards may be introduced on the output resulting from this signal disabling. Notice that internal nodes like p_1 of this SOP are not normally supposed to be seen by any other nodes in the network, in other words they are invisible in the corresponding STGs.

adequate to satisfy the hazard free requirement. However, if a pair of critical and transition wires happen to swap their statuses in two separate transitions, then a symmetric virtual isochronic fork is required.

3.5.4 Dynamic hazards in multi-level logic

Our findings may be generalized to multi level logic provided that the delay of the critical AND gate branch is guaranteed to not exceed the transition branch delay. This is of special importance in gate decomposition for the purpose of limiting the fanin of standard gates (due to technological constrains). Consider the decomposed circuit in Figure 3.16.

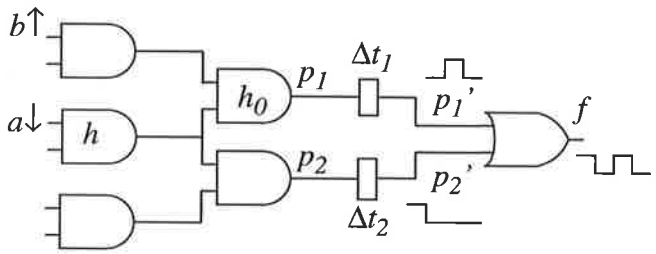


Figure 3.16: Multi-level combinational circuit which is still 1-0 dynamic hazard free under inertial gate delay model.

In this figure the first level AND gates have been decomposed into two levels. Again assuming the inertial delay model for the second level AND gates the circuit is free of high to low dynamic hazards. Of course for all possible dynamic hazards resulting from different sets of parallel transitions, all h_0 type p-terms have to be disabled through a signal transition injected to gate h in Figure 3.16.

3.5.5 Example

The following example is taken from [59]. As shown in Figure 3.17, x is a state holding variable for which the parallel transition $a^- \parallel d^-$ is considered suffering from a 1-0 dynamic hazard. Using the cube reduction technique in [60], the dynamic hazard is removed through adding some redundancy, so that the modified logic is $x = \bar{a}cd + ax + dx$. Using the hazard removal algorithm in [59] the following multi level implementation is achieved

$$x = dy + ax,\ c = \bar{a}c + y,\ y = \bar{a} + x$$

However, notice that the dynamic hazard is of 1-0 type and therefore, according to Theorem 3.5, the above multiple transition is free of this hazard under the inertial delay model.

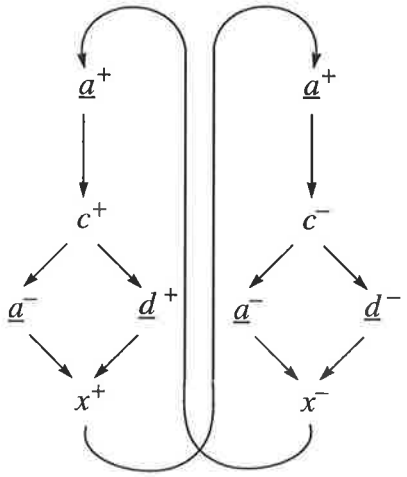


Figure 3.17: A STG resulting in a possible dynamic hazard on $x = \bar{a}d + ax + dx$ for $a^- \,//\, d^-$.

3.6 Multiple input change low to high dynamic logic hazards

In the previous section we showed that multiple input change high to low dynamic logic hazards may not occur in two level logic circuits under the inertial delay model and virtual isochronic fork assumptions. In this section we extend the previous work to multiple change low to high dynamic logic hazards and show that this type of hazard can be avoided through a reasonable delay constraint imposed to the first level AND gates in SOP implementations.

The following theorems are the duals of Theorem 3.2 and Theorem 3.3 respectively:

Theorem 3.6: In a low to high output transition, if all on-set vertices in the transition cube are not covered by a single implicant, then the transition has a 0-1 dynamic function hazard iff the final vertex is not within the overlapping area of the TCP implicants covering the on-set vertices of the transition cube.

Proof: The proof is the dual of the proof presented for Theorem 3.2.

Example 3.4: Consider the multiple input transition $t \rightarrow z$ in Figure 3.18 which is the complement of the function shown in Figure 3.3, where the transition cube is the whole *4* $\times$ *4* map with the on-set vertices not covered by one cube only. Notice that the terminating vertex, *z*, is located inside the overlapping area. Therefore, the transition is free of function hazard.

Now consider the transition $u \rightarrow y$, which has the same transition cube but the terminating vertex is outside the overlapping area, which causes a function hazard, as an input route, say, *ursxy* (the same input route referred in Section 3.5.2) corresponds to a non-monotonic output route, *01001*. ◊

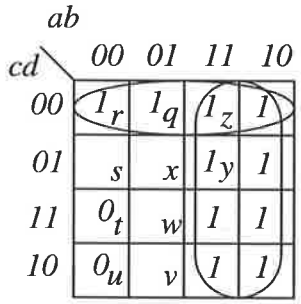


Figure 3.18: A transition cube (the whole map) with on-set vertices not covered by one cube to demonstrate a 0-1 dynamic function hazard.

Theorem 3.7: In a multiple input transition 0-1 dynamic logic hazard, under the bounded but unrestricted wire and gate delay model, each h_0-type p-term is necessarily enabled by at least the same input transition which enables one or more of the t_1 type p-terms.

Proof: The proof is the dual of the proof presented for Theorem 3.3.

Example 3.5:

Consider the multiple transition *abcdef: 011111* $\rightarrow$ *000100* shown in Figure 3.19. The transition cube $\bar{a}.d$ is highlighted. The t_1 p-term (that is $p_1 = \bar{a}.\bar{b}.d$) is enabled by $\bar{b}^+$ which is in the fanin set of p_2 as well. Therefore, prior to p_1 being enabled (by $\bar{b}^+$), p_2 cannot be enabled by any signal.

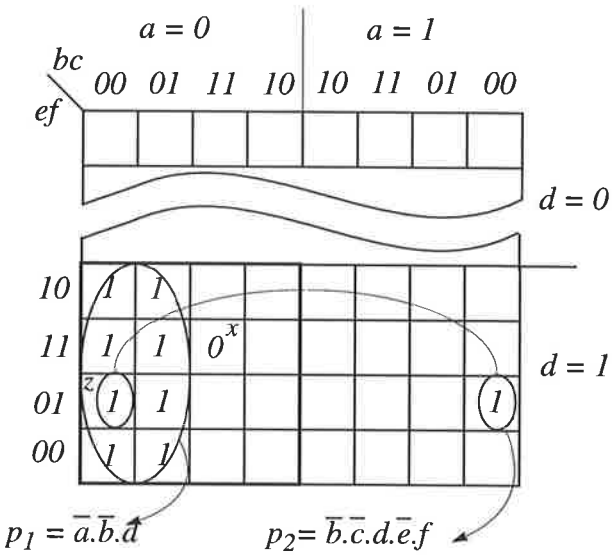


Figure 3.19: An example demonstrating how an overlapping h_0 p-term is disabled when the t_0 p-term is disabled.

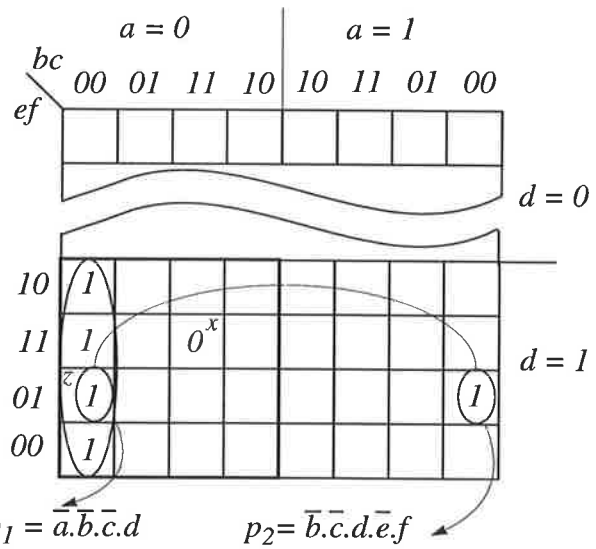


Figure 3.20: An example demonstrating how an overlapping h_0 p-term is disabled when the t_0 p-term is disabled.

Example 3.6: Figure 3.20 shows the same multiple input transition applied to a different logic implementation, in which the t_1-type p-term (that is, $p_1 = \overline{a}.\overline{b}.\overline{c}.d$) is enabled only after both $\overline{b}^{+}$ and $\overline{c}^{+}$ have fired. Notice that p_2 may not rise either until these two transitions occur.

Theorem 3.8: There is no *0* to *1* dynamic logic hazard for a monotonic multiple input transition in a SOP circuit under the inertial gate delay model if the h_o-type AND gate does not have less than half the delay of the t_1-type AND gate.

Proof: Consider the *0-1* dynamic hazard shown in Figure 3.21.

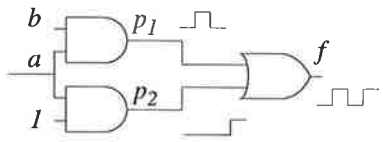


Figure 3.21: A 0-1 dynamic hazard.

Referring to Theorem 3.7 suppose that a^+ is the signal transition causing $p_2{}^+$ and $p_1{}^{+10}$. In order to guarantee a hazard free operation the transitions $p_2{}^+$ and $p_1{}^-$ have to be seen by the OR gate at times such that

$$p_2{}^+ - p_1{}^- < \Delta t_{OR} \qquad (3.7)$$

where Δt_{OR} is the inertial delay of the OR gate and $p_2{}^+$ is given by

$$p_2{}^+ = a^+ + \Delta p_2 \qquad (3.8)$$

where Δp_2 is the inertial propagation delay of the transition (t_1 type) AND gate. Substituting for $p_2{}^+$ in (3.7) from (3.8) yields

$$a^+ + \Delta p_2 - p_1{}^- < \Delta t_{OR} \qquad (3.9)$$

On the other hand considering the inertial nature assumed for the delays for the AND gates the pulse generated on the critical line has to be wider than the inertial delay of the critical AND gate, that is

$$p_1{}^- - p_1{}^+ > \Delta p_1 \qquad (3.10)$$

where Δp_1 is the inertial propagation delay of the critical (h_0 type) AND gate and $p_1{}^+$ is given by

$$p_1{}^+ = a^+ + \Delta p_1 \qquad (3.11)$$

10. Recall that this is the worst case, as the t_1 type p-term may be enabled before the h_o type p-term has been enabled. This precedence of course reduces the possibility of spurious transitions occurring.

Substituting for Δp_1 in (3.10) from (3.11) results in (3.12):

$$p_1^- > a^+ + 2\Delta p_1 \qquad (3.12)$$

Now substituting for p_1^- in (3.9) from (3.12) yields

$$\Delta p_2 - 2\Delta p_1 < \Delta t_{OR} \qquad (3.13)$$

Equation (3.13) (the so called 50% delay requirement) shows that if the critical AND gate is not over twice faster than the transition AND gate, then with a reasonable safety margin, that is Δt_{OR}, the possibility of a *0-1* dynamic hazard is ruled out under the inertial gate delay model and the proof is complete. ◊

Notice that this is a *one sided* requirement which deals with p-terms of the *same* SOP, that is the p-terms which are normally fabricated physically close to each other. This locality makes it very likely that the AND gates will satisfy the 50% delay requirement. Also notice that if in a particular specification two AND gates happen to be both transition and critical gates with respect to the each other, the delay restriction becomes double sided. Rewriting restriction (3.13) for Δp_2 and Δp_1 results in (3.14) with some safety margin, Δt_{OR}, not mentioned any more:

$$\frac{\Delta p_1}{2} < \Delta p_2 < 2\Delta p_1 \qquad (3.14)$$

or equivalently

$$\frac{\Delta p_2}{2} < \Delta p_1 < 2\Delta p_2 \qquad (3.15)$$

3.7 Static logic hazards

Recall that static 1-logic hazards exist in two level SOPs whenever a disabled p-term is enabled to substitute for another enabled p-term which is disabled in the transition cube. In other words this type of hazard exists iff the transition cube is not covered by one p-term. The inertial delay model does not alter this behaviour. Therefore, this delay model cannot relax the static 1-hazard problem.

It has been shown in [83] that static 0-hazards may not occur in SOPs unless both inverted and non-inverted literals of the same variable are applied to the same p-term, which does not happen in practical two level SOP circuits.

3.8 Delay hazards under the well-behaved environment assumption

The well-behavedness assumption of the environment is known in the literature as a sufficient condition to implement hazard free two level circuits from well-formed signal transition graphs. In this section we uncover this fallacy and show that non-atomic operator based implementation of well-formed STGs may result in hazards even in the absence of problematic sequential outputs discussed above and no matter how slow the environment is.

Having obtained the logic equations for all non-input nodes from a well-formed STG and shown that races cannot occur under the existing conditions, the methodology in [59] is focused on the removal of different types of logic hazards for each individual node in the network. This is as performed in the classical Huffman methodology in a rather similar basis. However, the guarantee for proper settling intervals for the sequential signals has been overlooked, so that these signals can be generated and applied close enough together to violate the fundamental mode requirement and hence to lead to an implementation deviation from the specification, (a *hazard*). To avoid this problem imposed by the environment, it is assumed in [59] that the environment is slow enough to meet this requirement or proper delay elements must be inserted in the path of outgoing signals in order to guarantee a proper gap between an output signal (ready signal) and its following input signal. However as shown below this does not guarantee a proper gap between an input signal and its following output signal[11]. Consider the STG and the corresponding logic equations generated by the automatic STG compiler SIS [74] in Figure 3.22, where c, d and e are input signals.

Suppose that after b^+ fires the environment raises e with a sufficiently long pause to let the preceding output transition f^+ to get stabilised and absorbed. Notice, however, that the transition f^- is out of the environment's control any more, and may fire too early after e^+ has fired, so that if in Figure 3.22 the NOR gate is faster than the AND gate, then input y of the OR gate would be pulled down before the input x has been pulled up, no matter whether the delay model is inertial or pure. Considering the logic high state for node b during these transitions, a static 1-hazard is likely to occur on node b as a result of the precedence reversal by parasitic delays.

11. A similar precedence violation is also likely when introducing internal signals in sequence with output signals in an attempt to solve the CSC problem.

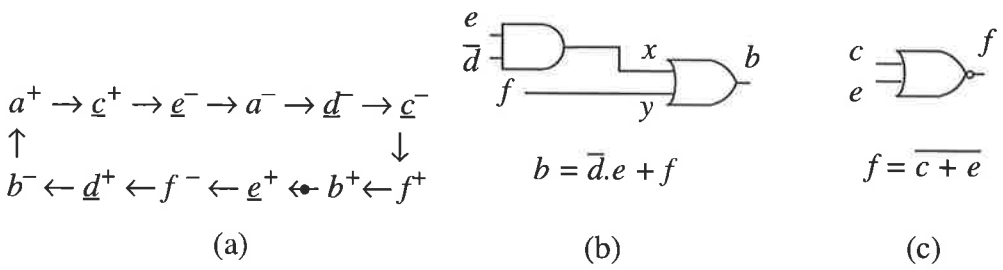


Figure 3.22: A STG and the resulting two level logic implementation for node *b* to show hazard generation due to non-atomic state model.

Delay hazards may be avoided if the function can be changed based on a proper use of don't care states. For example Equation (3.16) shows a delay hazard free function for node b which happens to be a state holding operator.

$$b = b.\overline{d} + f \qquad (3.16)$$

3.9 Conclusion

In this chapter we studied the design of two level hazard free asynchronous circuits from signal transition graphs under the *inertial delay model* and well-behaved environment. Different sources of dynamic hazards were first investigated, and then a major group of this class, that is *multiple input high to low real dynamic hazards*, were identified which we proved could not occur under the inertial gate delay model in two level SOP logic circuits. Notice that the inertial delay model restriction only applies to the first level AND gates. The inverters before this stage can still be modelled with the unrestricted delay model, while the final OR gate may be modelled as an instantaneous decision element followed by an inertial delay and then a pure delay. For this gate the inertial delay model can even be eliminated at the cost of losing the term $\Delta t_{OR}{}^{+}$ in Equation (3.6). We then relaxed the zero wire delay assumption and determined an upper bound for the delay of critical wires and introduced the notion of *virtual isochronic forks*, under which hazard free operation is still guaranteed.

We next proved that under a reasonable delay constraint all simple gate based SOP circuits are free of multiple input change 0-1 dynamic hazards when the inertial delay model can be assumed. These two results relieve the designer of all dynamic hazard problems in many cases. Static hazards, on the other hand, are not relaxed under the inertial

delay model and hence they have to be prevented through the classical method of introducing some appropriate redundancy to the cover of the function.

We uncovered a fallacy and showed that even the well-behaved environment assumption might result in delay hazards in non-atomic two level gate based implementations, although delay hazards are significantly reduced by the well-behavedness assumption of the environment. In order to eliminate this type of hazards a different cover has to be found with no hazardous behaviour. We showed this possibility through an example.

Complex SOP/SOP CMOS gates introduced in [Kudva][1] are static hazard free under an unrestricted wire and gate delay model [Kudva]. If the wire delay is restricted to the isochronic fork model, then complex gates (both SOP/SOP and standard CMOS) become immune to dynamic hazards, as shown in chapter five of this thesis for standard complex gates, no matter whether the gate delay model is pure or inertial. However the inertial delay model by itself does not provide this immunity in the absence of the virtual isochronic fork (symmetric or asymmetric, whichever is relevant).

Notice that as feature sizes shrink and transistor, and hence gate, speed increases regularly [Matzke][2], at some stage it makes sense to imagine that in a VLSI design wires are connected to each other through gates, rather than gates through wires, as wires do not scale well. This of course jeopardizes the isochronic fork model assumed in this chapter and some other research works. Recall that an isochronic fork model assumes negligible delay skew between the branches.

However if the (virtual) isochronic assumption remains valid by, say, accurately adjusting branch sizes in the forks, our findings in this chapter remain valid as long as the first level AND gate delay model is assumed inertial. In other words the inability to scale interconnection wires, predicted in [Matzke], does not affect the relaxation of dynamic hazards presented in this chapter. Therefore what a design tool must do in order to eliminate 1-0 dynamic hazards is to accurately adjust the branches of all symmetric virtual isochronic forks by proper placement and routing. These forks can be identified by considering all possible signal transitions during a pre-pass phase of the compilation.

To make sure of having sufficiently short rise and fall times, an accurate transistor level simulation may be necessary. Signal transitions with too long rise/fall times entering critical AND gates may not let 1-0 dynamic hazards disappear. In such circum-

Chapter 4

Delay Hazards in STG Based Two Level Logic Asynchronous Circuits

4.1 Introduction

STGs have been realized as asynchronous circuits in different ways with different delay assumptions. Chu [17][18] and Meng [56][55] have assumed an atomic gate model which can realize all logic functions with a single delay element at the output of the gate. All (logic and delay) hazards are ruled out under this atomic gate assumption under either the pure or the inertial delay model. Martin [47][50][51] has used a library of gates but with isolated inverters for the inverted inputs, which may be problematic if the circuit becomes sufficiently complex. Moon [59][60] uses two level logic based on the pure delay model but with a well-behaved environment as discussed in Chapter 3. Recall that in this methodology the environment must be sufficiently slow or delay hazards may be inevitable. Logic hazards, on the other hand, are eliminated by introducing some redundancy. Lavagno [42] has considered a simple gate implementation based on a bounded pure wire

delay model. In this methodology delay hazards are avoided by delay padding on the outputs of some of the logic gates in the asynchronous network, while logic hazards are again eliminated by adding some redundant terms. Kishinevsky et.al [33] have introduced a two phase RS speed independent implementation based on atomic AND-OR-NOT gates. As discussed in Chapter 5 each node in this methodology is implemented as two modelling variables, resulting in true speed independent circuits with some area and performance penalty. Beerel [3][4][7] has developed a design methodology for a simple gate based speed independent implementation in which simple gates with as many inverted inputs as required are considered atomic.

In this chapter we assume the isochronic fork model for interconnection networks and show that delay hazards are considerably reduced in two level SOP circuits. The gate delay model is then restricted to inertial and it is shown that delay hazards are further reduced.

The following definitions are repeated here for the sake of clarity:

Definition 4.1: A delay element is called *pure* if all events on its input are transferred to the output after some propagation delay time, while in *inertial* delay the input pulse must be sufficiently wide, (that is, greater than the propagation delay) to pass through the element. Shorter pulses will be absorbed and disappear.

Definition 4.2: As a result of a monotonic input transition if the output of a logic circuit may undergo a non-monotonic transition under the zero gate-wire delay model, then the transition suffers from a *function hazard.*

Definition 4.3: In a stable SOP if an input transition is free of function hazards but the output may still suffer from spurious transitions (which are due to parasitic delays of the circuit elements), then the transition has a *logic hazard.*

Definition 4.4 In a logic circuit if an input is applied too early, that is before the previous input has been absorbed, then the output may undergo a spurious transition. The early transition is said to incur a *delay hazard.*

In this chapter we assume that an AND gate with as many inverted inputs as needed is an atomic gate, unless otherwise specified. Our theory is based on the fundamental and *obvious* assumption that given a correct implementation of the STG, the right order of all signal transitions is always preserved according to the corresponding signal transition

graph. Therefore, the crucial point is that delay hazards may only be caused by ordered signals and unknown delays of the first level AND gates. This may best be understood if one realizes that parallel transitions are already allowed to fire in any order specified by the STG and with any spacing, so that the delays introduced by the individual gates comprising a logic circuit for a variable can only produce *another legitimate order* for the parallel signal transitions, unless *two ordered signals* overtake each other, as seen by the final OR gate in a two level AND-OR implementation. That is, possible delay hazards may only be generated due to the delays in the first level gates of a two level logic realization of the node under consideration. Therefore, we only need to consider ordered transitions to identify this kind of hazard. In the following section we first classify delay hazards and then address each one individually.

4.2 A classification of delay hazards

Notice that the notions of *static* and *dynamic* used in logic hazards cannot be identically applied to delay hazards, as in logic hazards the type of hazard is determined by the starting and the ending vertices in the transition cube. When considering delay hazards the SOP logic may not be stable and hence the transition cube is not sensible any more, as the circuit might not have responded to an input transition when the next input transition is applied. Therefore, we first define static and dynamic hazards in the context of delay hazards, and then address different types of hazards separately.

Definition 4.5: The *fanin set* of node x is the set of all variables arriving in the first level NOT-AND gates realizing the SOP circuit of node x. The fanin set is formally defined as

$$fi(x) = \{v \mid x = f(v) \ \& \ v \in V\}$$

where V is the set of all variables in the *STG* and f is the logic function realizing x.

Notice that since a node may be realized with different logic functions, it may have different fanin sets in different implementations.

Notice that the inputs of *OR* gates in a *SOP* realization of node x that are not single fanin inputs, are not STG specified signals and hence are the source of all possible delay hazards.

Definition 4.6: A *trigger enabling fanin signal transition*, of node x, $tef(x)$, is an

immediate predecessor of x^+.

Definition 4.7: A *trigger disabling fanin signal transition*, of node x, $tdf(x)$, is an immediate predecessor of x^-.

Definition 4.8: The set of all signal transitions in a simple cycle of a STG occurring after a positive transition of signal x and before the first negative transition of x, is called a *basic hazard region* of node x, abbreviated as $\mathcal{BHR}(x)$, where x is the output (not the inverted output) of the corresponding SOP[1]. A basic hazard region of node x is formally defined as follows:

$$\mathcal{BHR}(x) = \{v^* | v \in V, x^+ < v^* < x^- \ \& \ all\ v^*'s\ belong\ to\ same\ cycle\}$$

where V is the set of all variables in the *STG* and $a^+ < b^-$ means that a^+occurs before b^-.

Definition 4.9: The set of

1- all transitions in parallel with at least one transition in a $\mathcal{BHR}(x)$ in conjunction with

2- all transitions in the $\mathcal{BHR}$, and

3- all transitions in parallel with x^+, is called a *hazard region*. A hazard region is formally defined as follows:

$$\mathcal{HR}(x) = \{v^* | \ v \in V, v^* \| y^* \ \& \ y^* \in \mathcal{BHR}\} \cup \mathcal{BHR} \cup \{v^* | \ v \in V, v^* \| x^+\}$$

Definition 4.10: In a STG based asynchronous circuit suppose that a hazard region is entered with stable logic gates. If the output of a SOP may undergo spurious transitions as a result of early input transitions in a hazard region, then the delay hazard is of *high to low dynamic* type if the spurious high pulse may only occur on the output after each *tdf* has propagated through *at least* one p-term to the output node. Otherwise, the spurious transitions are *called static one delay hazards.* ◊

According to this definition in the hazardous SOP output shown in Figure 4.1 if transition t_1 is caused by $tdf(x)$, then this non-monotonic transition demonstrates a high to low dynamic delay hazard, otherwise it is a static one delay hazard.

Definition 4.11: When a signal specified in the STG goes low freely of static one and

1. All of the achievements for SOPs can of course be applied to POSs as well by considering proper dualities.

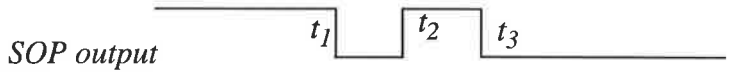


Figure 4.1: 1-0 dynamic or static 1- (delay) hazard.

high to low dynamic delay hazards, any possible spurious transition caused afterwards (as a result of early input transitions outside a hazard region) is called a *static zero delay hazard.* ◊

The reason that only one type of hazards is attributed to the outside of hazard regions will become clear in the following section.

Corollary 4.1: According to the above two definitions dynamic low to high hazards are not defined any more in the context of delay hazards.

In the following sections we study all different types of delay hazards, that is high to low dynamic hazards and also static one and zero hazards in SOPs, under both the *inertial* and *pure* delay models.

4.3 Static 0-delay hazards

In this section we identify a major part of STGs which are hazard free in the corresponding two level implementations under the pure delay model. Therefore, the results automatically apply to the inertial delay model as well.

Definition 4.12: A p-term is *input enabled* if its inverted and non-inverted inputs are asserted to logic low and logic high, respectively. Otherwise, it is called *input disabled.* Notice that being an input enabled or input disabled p-term does not by itself necessarily determine the state of the output of the AND gate, because of the propagation delay time of the gate.

In this chapter the prefix *input* is left out whenever the intention is clear from the context.

Corollary 4.2: The output of a disabled p-term may be either *1** or *0* (neither *0** nor *1*) under inertial gate delay model, but it can be *0** under the pure delay model.

Corollary 4.3: The output of an enabled p-term may be either *1* or *0 ** (neither *0* nor *1**) under inertial gate delay model, but it can be *1** under the pure delay model.

The following theorem identifies a necessary condition to cause delay hazards and hence shows the importance of hazard regions in generating delay hazards in SOPs:

Theorem 4.1: Under the pure gate delay model and the isochronic fork assumption[2] a signal transition may *directly* cause a spurious transition on the output, x, of a SOP only if the transition falls inside a hazard region of x. (a necessary condition.)

Corollary 4.4: Under the pure gate delay model and the isochronic fork assumption, all SOP implementations are free of static 0-delay hazards.

Notice that because of this hazard free area, we did not define low to high dynamic hazards separately in Definition 4.11, Section 4.2.

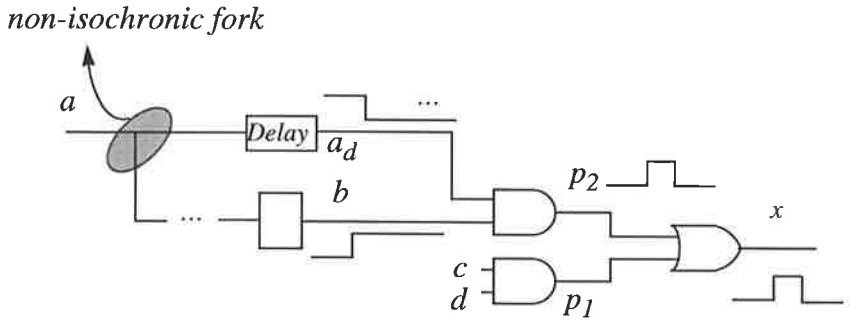
Proof: Recall that outside a hazard region of node x, the specified value for x is zero or excited to zero. We first need to prove that under any delay attributes of the first level AND gates, the output of two level logic gates would not undergo a spurious transition if the output signal is specified to stay low during some sequential signal transitions. Notice that a spurious low to high transition on the output of a SOP, x, necessitates such a transition on a p-term, say, p_1, that is the relative delays of no two AND gates may cause any concern of generating spurious low to high signal transition on the output node of a SOP. Therefore, a spurious low to high transition on x may only be caused as a result of two sequential fanin transitions, t_1 and t_2, of that node, implying that one of these transitions, say t_2, is an enabling and the other one, t_1, is a disabling fanin transition. This, moreover, necessitates that the enabling fanin transition fires before the disabling one, that is $t_2 < t_1$. On the other hand under the intended operating conditions there is no spurious transition on x. Therefore, p_1 is first disabled by t_1 before getting a chance to become enabled by t_2, that is $t_1 < t_2$. Notice, however, that both t_1 and t_2 are STG specified signals and cannot lose their specified orders. Furthermore, the implementation is skew free fork based, so that the right order is preserved at the input terminals of the AND gate p_1 as well, that is the output cannot be pulled up. The proof is complete. ◊

It is worth highlighting that in the above theorem, unlike static 0-*logic* hazards, the skew free assumption is inevitable. The output may easily suffer from a static 0-delay-hazard under an unrestricted delay assumption. Consider the situation depicted in Figure 4.2.

2. See Chapter 5 for more details about isochronic forks.

$$\rightarrow a^- \rightarrow \ldots \rightarrow b^+ \rightarrow \ldots$$

(a)



(b)

Figure 4.2: (a) A partial STG and (b) a typical two level implementation demonstrating delay static 0-hazard generation under fork skew assumption.

In Figure 4.2 if the falling transition a^- is so delayed that gate p_2 sees a_d^- after node b has been pulled up, a spurious positive pulse on node p_2 and hence on the output x would be likely to appear. Notice that an asymmetric isochronic fork would be sufficient to avoid this delay hazard as far as this partial STG is concerned, that is a larger delay on the branch going to node b, than the delay on the other branch still guarantees a hazard free operation. The *bounded wire delay model* and hence the above possibility of delay hazards have been addressed in [41] where all delay hazards are avoided by proper delay padding based on the exact knowledge about the delay limits of different wires prior to the circuit design. However, static zero delay hazards are avoided under the pure gate delay model and the isochronic fork assumption as proved above.

Notice that the isochronicity assumption, on the other hand, may be violated by inverters at some of the inputs, resulting in delay hazards, similar to what was mentioned regarding wire delays. There are many published works in the design of speed independent circuits assuming negligible delay across inverters [3][4][18][56], that is each basic logic gate together with all its input inverters are assumed atomic. This, however, is not so restricting, as in many cases these inverters may properly be relocated to asymmetric

forks (like the example in Figure 4.2) or even to delay insensitive forks, relaxing the atomic gate restriction. Therefore, under the pure gate delay model and the isochronic fork assumption, and outside hazard regions possible spurious transitions may only be caused by inverters (fork isochronicity violation) at the inputs of some AND gates. Regions outside hazard regions are hazard free if the problematic inverters are properly relocated or the library cells have been carefully designed to make every gate, together with the input inverters, behave as an atomic gate. In this chapter we also assume that these inverters are non-problematic unless otherwise specified.

It should be highlighted that here some transitions with appropriate delays *outside* the hazard region may be able to prevent some delay hazards. In other words these transitions are not the *direct cause* of the spurious transitions, that is why the term *directly* has been included in Theorem 4.1. See Example 4.4 for more details.

Notice that the above theorem does not restrict the delay model to the inertial model. The result is general and applies to the pure delay model as well.

4.4 Static 1-delay hazards

In the previous section we showed that static 0-delay hazards may not occur under the pure gate delay model in SOP circuits assuming isocronic forks where necessary. In other words, it was demonstrated that a major part of STGs are immune to hazards. Later on we will show that this immunity is further extended to dynamic hazards as well, under the inertial delay model. Now, in this section static 1- delay hazards are addressed which are the only type of delay hazards to be concerned about in SOP implementations.

Theorem 4.1 showed that static delay hazards may only happen in hazard regions. Furthermore notice that the hazard generation mechanism is similar to logic hazard generation: a spurious transition may occur on the output of a SOP as a p-term is disabled and replaced with an already enabled p-term. This mechanism will be demonstrated through some examples in this section.

Definition 4.13: Two p-terms in a SOP are called *input overlapping* if during a hazard region one may become disabled after the other one has become enabled. Notice that this does not guarantee an overlapping period for the asserted outputs of these two gates, that is *output overlapping*, under an unrestricted delay model.

Definition 4.14: Two p-terms are called *output overlapping* if the first output is not

pulled down unless the second output has been pulled up.

Corollary 4.5: Input overlapping is neither necessary nor sufficient condition for output overlapping in simple gate SOPs under an unbounded delay model. It is necessary and sufficient under the assumption of a atomic complex gate implementation with unbounded delay.

Therefore, generally speaking, two level SOPs suffer from delay hazards under an unbounded delay model. In order to avoid hazards some delay restriction is inevitable, which implies a bounded delay model as well for the relevant gates.

Definition 4.15: A *primary p-term* (pp-term) of node x is a p-term which may excite the output node under some specific combination of the relevant parasitic delays.

Primary p-terms are normally expected to be excited by *tef*(x). We call these pp-terms type-1. This, however, is not always the case, as we will see in this chapter. A pp-term which is not excited by *tef*(x) is called a pp-term type-2. On the other hand pp-terms of the same type may not be unique either. Therefore, primary p-terms are not deterministic in many cases as demonstrated in this section.

Primary p-terms may be excited simultaneously as well. Therefore, two simultaneous primary p-terms are necessarily enabled by the same signal transition but may be disabled by different signal transitions.

Notice how logic high may be maintained on the output of a SOP logic circuit realizing variable x during a hazard region: x^+ is excited through a pp-term and possibly with *tef*(x). Generally speaking, the pp-term becomes disabled at some stage in the hazard region but after another p-term has become enabled, to maintain input overlapping. This switch over process between input overlapping p-term pairs may continue a few times until the final enabled p-term(s) becomes disabled by *tdf*(x), eventually resulting in pulling x down. Therefore, in a hazard region it is guaranteed there is at least one sequence of input overlapping p-term pairs starting with a pp-term and possibly initialized by *tef*(x) as described above. This guarantee is made by the *specification*. However, in order to avoid static 1-hazards there must be a sequence of output overlapping p-terms which may only be guaranteed by the *implementation*, say, providing sufficiently long input overlapping times between consecutively enabled p-terms. Therefore, to prevent possible output glitches due to delay variations in the AND gates, it has to be guaranteed that there is a

sequence of output overlapping p-terms (a necessary condition). This requirement in 2 level SOP based implementations of course incurs some delay restrictions on some of the AND gates. In this chapter we assume that the output overlapping sequence is provided through the corresponding input overlapping sequence with proper delay constraints, where necessary. However, notice that we are not developing an algorithm for optimal delay padding in this thesis.

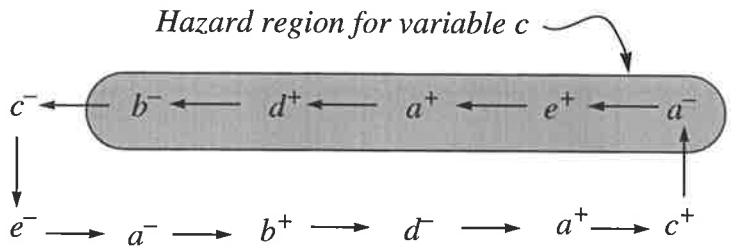
Definition 4.16: Every sequence of input overlapping p-term pairs in a hazard region starts with a pp-term and possibly continues with some *auxiliary p-terms*, *ap-terms*.

Therefore an *auxiliary p-term*, is enabled before the primary (or a previous auxiliary) p-term is disabled and is disabled after the primary (or the previous auxiliary) p-term is disabled. Furthermore, an auxiliary p-term is disabled after another ap-term has already been enabled, unless it is the final ap-term which is disabled by $tdf(x)$ and hence terminating the $\mathcal{HR}$.

An auxiliary p-term, p_2, may happen to be enabled by the same signal transition that disables the preceding enabled auxiliary or primary p-term, p_1. This particular situation shows a static 1-logic hazard in two level logic which can be removed by imposing a delay restriction or introducing a redundant cube, p_3, to the onset cover of the variable. Now p_3 is a newly introduced ap-term.

Example 4.1: Consider the STG, the hazard region and the logic equation for variable c in Figure 4.3. Notice that the hazard region here is reduced to the basic hazard region as there is no parallelism in the STG. The pp-term is ab which is enabled by $a^+$and is later disabled by a^-, however, notice that the auxiliary p-term, $c.\overline{d}$, has already been enabled by c^+, so that if the delay of the AND gate corresponding to the p-term $c.\overline{d}$ is less than the total delay from c^+ to a^-, (that is the two level logic realizing node a plus the delay of the AND gate corresponding to the p-term $a.b$,) then the output of the two p-terms $a.b$ and $c.\overline{d}$ will be overlapping and hence no spurious pulse might occur on node c during this switch over process.

The auxiliary p-term $c.\overline{d}$ itself is later on disabled by d^+, however, now the former pp-term, $a.b$ is playing the role of the new auxiliary p-term, as it has already been enabled by a^+ in the hazard region. This second switch over between the two p-terms can be the cause of a second static 1-delay hazard, unless a similar delay constraint is imposed on the relevant gates.



$$a = \bar{c}.\bar{d} + e$$

$$b = \bar{a} + \bar{d}$$

$$c = a.b + c.\bar{d}$$

$$d = a.e + \bar{b}$$

$$e = \bar{a}.c + c.e$$

Figure 4.3: A STG, the hazard region for node c, and the logic equations of all variables.

Notice that if signal x has exactly one *tdf*, then the *tdf* may not cause a static 1-delay hazard for variable x, as no matter how slowly it fires, node x may not be pulled down before its unique predecessor fires.

Corollary 4.6: If x^- has exactly one *tdf* then the basic hazard region is redefined as

$\mathcal{BHR}(x) = \{v^* \mid v \in V, x^+ < v^* < x^- \ \&\ v^* \neq \bullet x^- \ \&$ all v^*'s belong to same cycle.$\}$

where V is the set of all variables in the *STG* and $\bullet x^-$ is the immediate predecessor of the transition x^-.

In multiple *tdf*, however, even an immediate predecessor may cause a static 1-delay hazard as demonstrated in the following example:

Example 4.2: Figure 4.4 shows a STG and the hazard region for node a.

The pp-term for node a is $\bar{d}.\bar{c}$ which is disabled by c^+ in the hazard region and the auxiliary p-term is $a.b$. If gate 2 is so slow that y (see Figure 4.4-b) goes high (as a result of transition a^+) after x goes low, (as a result of transom c^+) a may undergo a spurious transition. Notice that here c^+, a *tdf* of a, is causing a static 1-delay hazard

The hazard generation mechanism becomes more complex as the hazard region in-

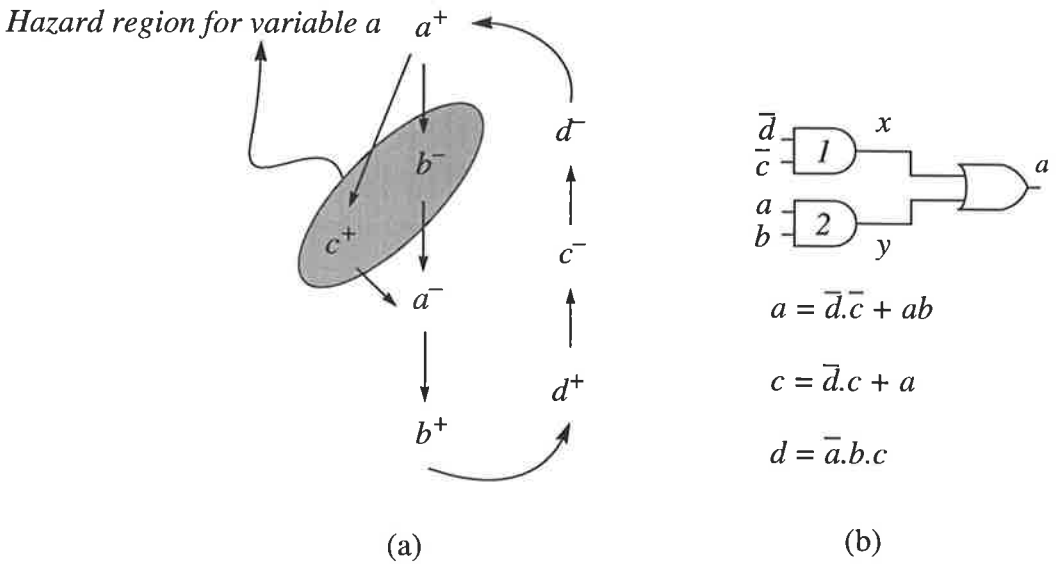


Figure 4.4: (a) The STG and hazard region for variable *a*, and (b) an implementation for node *a* and the logic equations in Example 4.2.

cludes more parallel transitions, as now all parallel branches may affect the node under consideration.

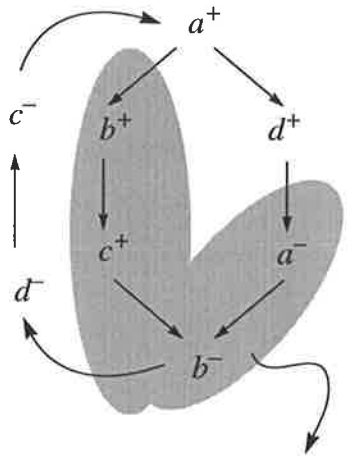
Example 4.3: Consider the STG and the hazard region of node d in Figure 4.5.

Here a is the unique pp-term for signal d which is later disabled by a^-, while the corresponding ap-term is $\bar{c}.d$. This ap-term itself is disabled by c^+, on the other branch, however, c^+ has its own preceding ap-term, b^+. This requires that

1- b goes high before the p-term $d.\bar{c}$ is output disabled by c^+ which in this specific circumstance is always the case, as b is a single input p-term, and

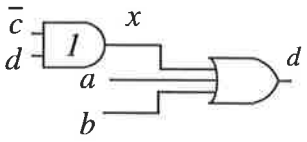
2- on the other branch $\bar{c}.d$ must be output enabled before a goes low, that is $x^+ < a^-$ and $b^+ < x^-$ in order to guarantee static one delay hazard free behaviour. (See Figure 4.5-a)

Output overlapping for the p-terms of variable d may be achieved by different delay restrictions: if b goes high before a^- occurs, node d is free of static 1-delay hazards. This type of delay restriction, that is between two parallel transitions, results in eliminating some states from the state transition diagram. For example under the above assumption the state $abcd = 0001$ may not occur at all. This type of restriction, as discussed under *timed* asynchronous circuits [62][63] may become non-feasible if a signal like b is an in-



Hazard region for variable d

(a)



$a = a.\bar{d} + \bar{c}.\bar{d}$

$b = a + \bar{c}.d$

$c = d.c + b$

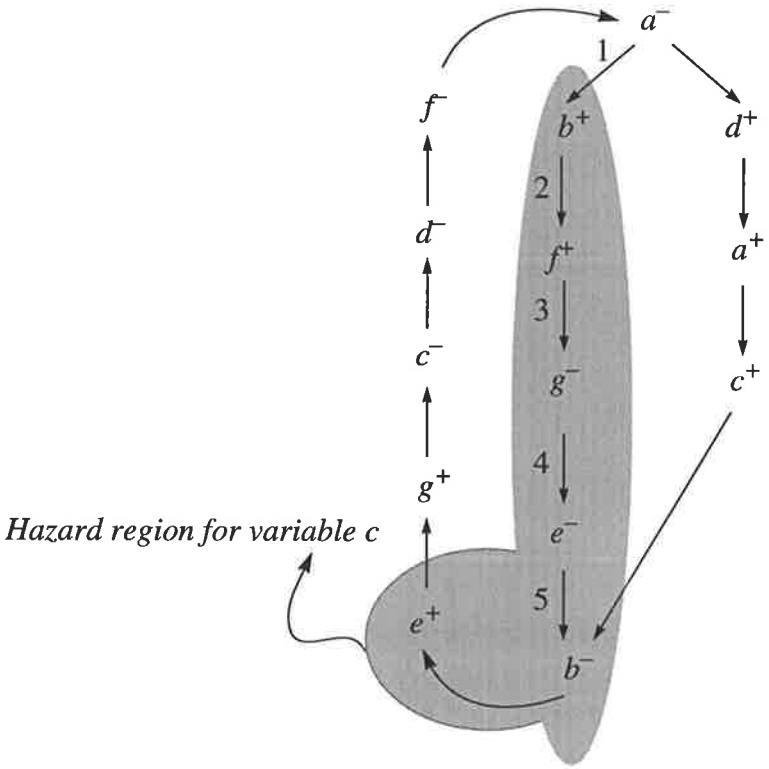
(b)

Figure 4.5: (a) The STG and hazard region for variable *d*, and (b) an implementation for node *d*, and the logic equations in Example 4.3.

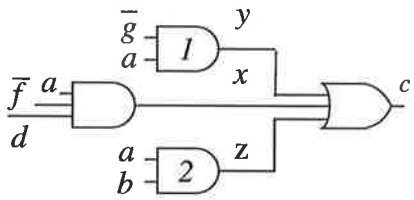
put signal for which no upper bound for the delay restriction may be assumed.

Notice that the pp-term may be non-deterministic, that is it may be different for different situations on the parallel branches and/or different delays of the relevant first level AND gates, as demonstrated in the following example:

Example 4.4: A STG and the hazard region for node c is shown in Figure 4.6 for which the pp-term is non-deterministic, that is which p-term becomes the pp-term, (the successful pp-term) depends on which transition is enabled on the left branch in the hazard region when a^+ fires and also depends on the delays of the relevant first level AND gates to which signal a is a fanin signal. Different situations are summarized in Table 4.1 assuming that the first input enabled p-term is the first output enabled one as well. Notice that all pp-terms here are of type-1.



(a)



$$c = a.d.\bar{f} + a.\bar{g} + a.b$$

(b)

Figure 4.6: (a) The STG and hazard region, and (b) the logic equation and an implementation for variable *c* in Example 4.4.

Table 4.1

position of token	pp-terms
1	$a.d.\overline{f}$
2	$a.d.\overline{f}$, ab
3	ab
4	ab, $a.\overline{g}$
5	ab, $a.\overline{g}$

As shown in this table any p-term may happen to be the pp-term unlike the previous examples in which the pp-term was deterministicly unique. Again if we assume the two parallel branches to be independent, a robust design requires the worst case condition to be satisfied, that is we need to satisfy two equations $z^+ < x^-$ and $y^+ < z^-$ in order to guarantee a glitch free signal on node c under the worst case delay restriction where $a.d.\overline{f}$ is the pp-term.

Notice how controlled delays *outside* hazard regions may avoid hazards as well: In Figure 4.6 if the right branch is sufficiently slow, so that f^+ is seen before a^+ is seen by the relevant final OR gates, then the first restriction determined above is ruled out. Furthermore, this delay restriction is now of type 2, where a delay restriction is imposed among parallel transitions.

Considering the generation mechanism of static one delay hazards addressed above the following theorem is proved:

Theorem 4.2: Under the unbounded gate delay model an input overlapping sequence suffers from a static one delay hazard if and only if one or more auxiliary p-terms in the sequence have two or more literals.

In addition to the non-determinism among multiple possible pp-terms enabled by *tef*'s (pp-terms type-1), this type of pp-term is not necessarily the first p-term to enable the output of a SOP, as demonstrated in the following example:

Example 4.5: A two level logic equation for variable b and the corresponding hazard region are shown in Figure 4.7.

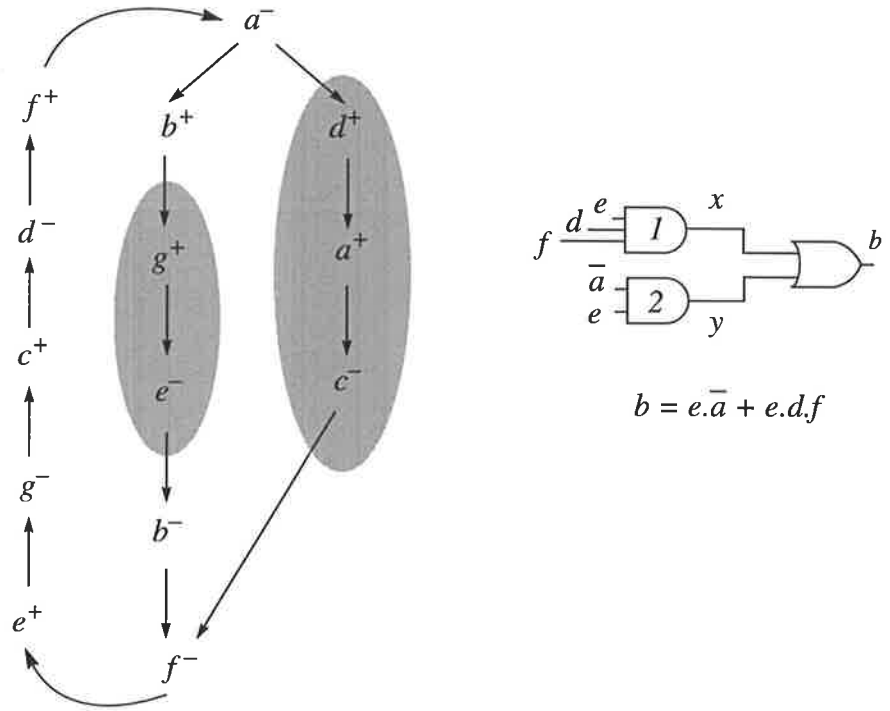


Figure 4.7: A STG and the hazard region (the shaded area) for node *b* (see Example 4.5).

The only type-1 pp-term in this hazard region is $e.\bar{a}$, however, this is not necessarily the p-term which excites node b, as d^+ may fire and enable the corresponding p-term ($e.d.f$) before the pp-term $e.\bar{a}$ has been enabled.

Now the delay restrictions are considered for different possibilities to avoid static one delay hazards:

Suppose that pp-term type-1 $e.\bar{a}$ is first pulled up. Since the corresponding ap-term is $e.d.f$, the delay of gate *1* must be less than the total delay of the logic from d to a, plus the delay associated with gate 2, in order to avoid static one delay hazards.

Consider the second alternative where the p-term type-2 $e.d.f$ is asserted first. This pp-term, however, is not disabled in the hazard region except with e^- which is the *tdf(b)*. Therefore, under the second alternative there is no static one delay hazard.

Even if the above circuit might be static hazard free under the specified conditions, it would still suffer from dynamic hazards: if the transition from a to d and then d to x is slow enough, then under the pure delay model the p-term *1* may be asserted to high after

node y has been pulled down by e^-. This type of hazards is the topic of the next section.

4.5 High to low dynamic delay hazards

The mechanism of static 1-delay hazard generation was discussed in the previous section. It was shown that if there is an output overlapping sequence of p-terms for every possible pp-term in a hazard region, then static 1-delay hazards will be avoided under a bounded pure gate delay model assumption. This, however, does not rule out the possibility of all spurious transitions on the output of a SOP, as the output may still suffer from dynamic delay hazards under a pure delay model. In this section the generation mechanism of high to low dynamic hazards is addressed and then it is shown that they are automatically avoided under the inertial gate delay model.

Referring to Example 4.4 (Figure 4.6) suppose that when a^+ fires both p-terms x and z are enabled but z happens to be the pp-term. Since there is no delay restriction for x^+, this may fire after c has been pulled down resulting in a dynamic high to low hazard.

A similar hazard is likely to occur if p-term y happens to be the pp-term. Consider the situation that when a^+ occurs e^- has already been excited so that both z and y become excited as well. Now due to some particular parasitic delays if y goes high first, there is no guarantee for z^+ to fire before y goes low by g^+. This can also result in a high to low dynamic delay hazard.

Definition 4.17: The pp-term which enables the output of the SOP in reality is called the *successful* pp-term.

Definition 4.18: A *secondary p-term* (*sp-term*) is a p-term which is enabled during a hazard region but is not a part of the sequence of the overlapping p-terms corresponding to the successful pp-term.

Theorem 4.3: Under the bounded pure delay model if there is a sequence of output overlapping p-terms for node x in its hazard region, then node x will not suffer from a high to low dynamic delay hazard if there are no secondary p-terms in the hazard region.

Proof:

Suppose that a variable x goes high not as a result of *tef*'s and after all *tdf*'s have fired. The hazardous rising transition may not have been caused by a p-term in the output overlapping sequence, as the output of such p-terms are only pulled up before the output of the

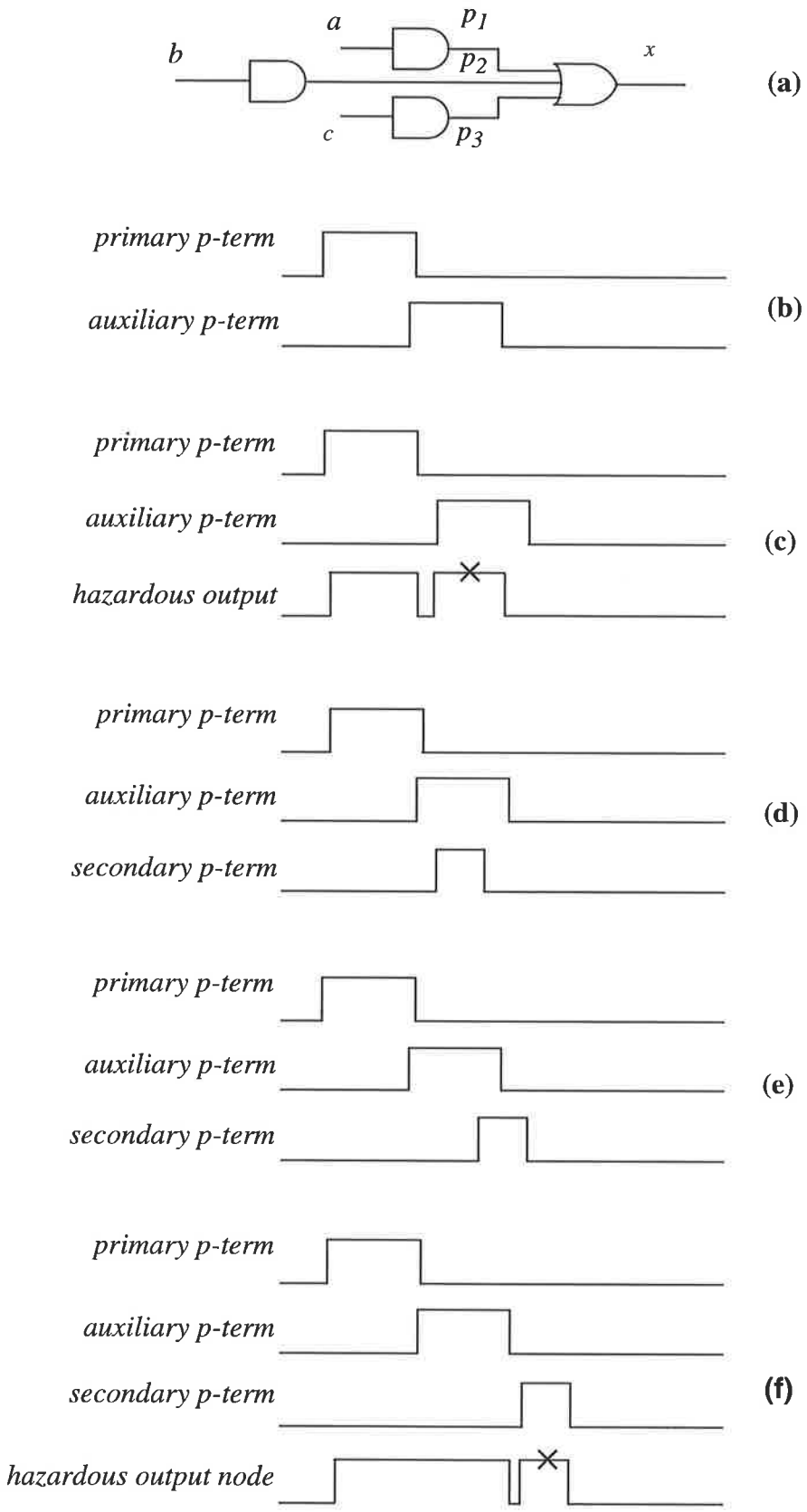


Figure 4.8: (a) a typical SOP circuit, (b) - (f) different delay behaviour of different p-terms in SOPs.

previous p-term in the sequence has been pulled down resulting in no transitions on the output x. Therefore, the spurious transition must have been caused by a p-term outside the sequence of the output overlapping p-terms, that is a secondary p-term. So the proof is complete. ◊

Consider the two level implementation based on a well-formed STG and the different timing situations for its p-terms shown in Figure 4.8.

In Figure 4.8-b the primary and the auxiliary p-terms are output overlapping. So the output is not hazardous as far as these two p-terms are concerned. The two p-terms are not output overlapping in Figure 4.8-c, resulting in a static 1-delay hazard. Figure 4.8-d and e show a secondary p-term which is pulled up before the last auxiliary p-term is pulled down. It is clearly shown that no matter when the secondary p-term is pulled down the output is hazard free. Finally Figure 4.8-f shows a situation in which the secondary p-term is pulled up after the previous auxiliary p-term has been pulled down. This situation suffers from a high to low dynamic delay hazard. Notice that the two seemingly similar behaviours in Figure 4.8-c and f are of different nature. In Figure 4.8-c the apparent dynamic hazard has been caused by a static 1-delay hazard, that is the pulse on the auxiliary p-term can be of type logic *1*, while in Figure 4.8-f the pulse on the secondary p-term is of 1^* type. In other words when the output of this p-term goes high the corresponding AND gate has already been input disabled.

Example 4.6: Consider the STG in Figure 4.9 taken from [7].

There are two hazard regions for node *c* in this STG corresponding to the two possible choices, as shown by the shaded areas in the figure. The first one is empty so there is no delay hazard on node *c* in the first hazard region.

For the second region d^+ is the *tef(c)*, the pp-term *d* is a single fanin (no gate), d^- is the transition which disables the pp-term and gate *1* realizes the multiple input auxiliary p-term, *bc*. Therefore, according to Theorem 4.2 transition d^- causes a static one delay hazard, under the unbounded gate delay model.

Now consider the secondary p-term *2* which is enabled by transition a^-. According to Theorem 4.3 this p-term is the cause of a dynamic delay hazard. Therefore, there are two independent sources for delay hazards in hazard region 2.

Notice that the delay model that we have assumed so far in this section is the pure de-

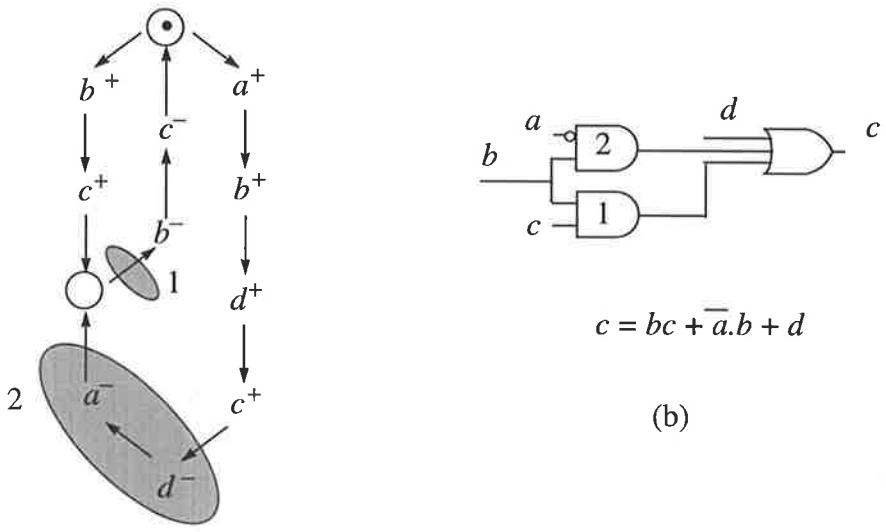


Figure 4.9: (a) Two hazard regions for node *c*, (b) a simple gate implementation for node *c*.

lay model. In the following theorem we restrict the delay model to the inertial delay model and show that dynamic delay hazards can no longer occur in two level logic circuits.

Theorem 4.4: Under the inertial delay model high to low dynamic delay hazards are automatically prevented in SOP implementations.

Proof:

In Theorem 4.3 we showed that such a spurious pulse is caused by a secondary p-term, which is enabled during a hazard region but outside the output overlapping sequence.

Let t_1 and t_0 be the instants of time at which the secondary p-term is output enabled and the output of the last disabled p-term in the output overlapping sequence goes down, respectively.

Notice that a spurious transition may only occur if $t_0 < t_1$. At t_0 all *tdf*'s and hence the disabling transitions of the secondary p-term have already fired in the hazard region. The equation $t_0 < t_1$ then implies that the output of the secondary p-term has not been pulled up yet at t_0 while one or more of its inputs has/have already been disabled. This situation corresponds to an output of 0* for this p-term while the input is disabled which is not feasible under the inertial gate delay model according to Corollary 4.2. Therefore, the output

of the secondary p-term may only be pulled up while a p-term in the output overlapping sequence is still high and this rules out the possibility of high to low dynamic hazards under the inertial delay model assumed for the first level NOT-AND gates in two level SOP circuits.◊

Referring to Example 4.6 under the inertial gate delay model transition a^- in Figure 4.9 is no longer hazardous, according to Theorem 4.4.

4.6 Conclusion

In this chapter different types of delay hazards in STG based two level asynchronous circuits were addressed. We first identified a major region in STGs which are immune to hazards and hence showed that static 0-delay hazards have no chance of occurring under the pure delay model and isochronic fork assumption. Furthermore, according to our hazard classification low to high dynamic hazards are no longer defined. We then showed that static 1-delay hazards may be avoided if there are sufficient sequences of output overlapping p-terms in the hazard region. We further showed that these sequences eliminate all dynamic high to low delay hazards as well, under the inertial gate delay model and isochronic fork assumption.

Chapter 5

Hazards in Complex Gate Based VLSI Circuits

5.1 Introduction

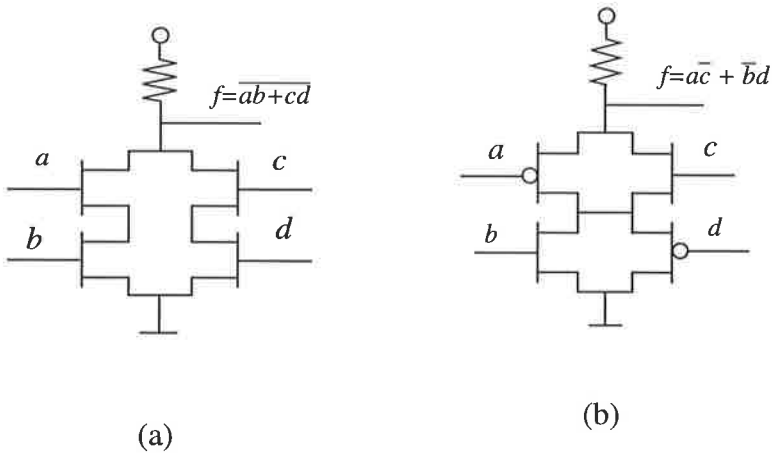
In Chapter 4 delay hazards were analysed in two level SOP implementations and it was shown that this class of logic become more immune to delay hazards under the inertial delay model. In this chapter hazards are analysed in single level logic family under the general *pure* delay model and it is concluded that the inertial delay model does not alleviate the hazard problem. We first discuss different types of interconnection forks. Then delay hazards are analysed which may only be caused by inverters at some inputs under the isochronic fork assumption for wire forks. We introduce *safe cells* based on which well-formed STGs can be implemented free of delay hazards with no unrealistic assumptions about physical gates. Although this technique still compromises chip area for the

sake of preventing hazards, we show that it may achieve a significant area gain in comparison with the two-phase RS-implementation method [33] which is one of the few true speed independent implementation techniques that we are aware of so far. Delay hazards are then analysed in complex gate based circuits under some gate delay restrictions and hence a theorem is developed to identify a subclass of delay hazards. We lastly show how logic hazards are relaxed in this logic class. Notice that these achievements apply to the pure and hence to the inertial delay model as well.

5.2 Different types of forks

5.2.1 Isochronic forks

A complex AND-OR-NOT gate as shown in Figure 5.1-a, reasonably satisfies requirements for speed independence, as no matter how close together two sequential inputs occur, they affect the circuit in the same order.



(a) (b)

Figure 5.1: Two AND-OR-NOT complex gates realized with (a) real, (b) ideal transistors.

Note that the only factors [86] jeopardizing its robustness are the differences between

- the threshold voltages of the transistors participating in the sequential inputs, and
- the rise and fall times of the corresponding input signals.

These differences usually happen to be within a safety margin including

- the time gap between two sequential inputs, as considering the causal relationship, no two sequential inputs may occur simultaneously. Otherwise, a simple NAND gate would not be a speed independent gate either,
- the inertial delay nature of the gate itself, which absorbs short signal glitches.

The situation becomes different if interconnection networks are taken into consideration. Consider a two branch fork shown in Figure 5.2 where the branches have been modelled with two delay elements. Generally speaking all gates, such as Gate 2 are expected to see a transition on node x (that is x^*) before x^* propagates to any output node like G_1. This entails $\Delta t_2 < \Delta t_1 + \Delta t_{gate\text{-}1}$, or

$$\Delta t_2 - \Delta t_1 < \Delta t_{gate\text{-}1} \qquad (5.1)$$

Otherwise Gate 2 will see x^* after it sees the possible transition on G_1 caused by I_1, that is a change of the specified order of signal transitions.

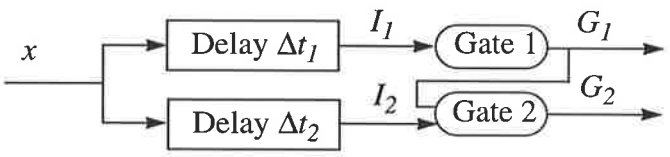


Figure 5.2: A fork with different branch delays.

The dual of this conclusion leads to the following equation

$$\Delta t_1 - \Delta t_2 < \Delta t_{gate\text{-}2} \qquad (5.2)$$

Considering Equations (5.1) and (5.2) the following equation is concluded in order to guarantee the correct order of signal transitions:

$$| \Delta t_1 - \Delta t_2 | < Min.\ (\Delta t_{gate\text{-}2}, \Delta t_{gate\text{-}1}) \qquad (5.3)$$

Therefore, some fork skew less than the relevant gate delay is still accepted without disturbing the right order of signal transitions and hence the correct operation of the circuit. Notice that the delays can be the pure type.

In the above discussion Δt_1 and Δt_2 include both the corresponding branch delay and the delay caused by the threshold voltages of the fanout transistors together with the non-zero rise/fall times of the relevant signal transitions. An interconnection fork on which a

transition is detected in the fanout gates with a negligible skew is called *isochronic* [50]. Assuming an isochronic fork model the complex gate in Figure 5.1-a necessarily becomes a legitimate gate for speed independent asynchronous circuits as demonstrated here. In Figure 5.2 consider the transition $x^* \rightarrow {G_1}^*$ which may be split into

$$x^* \rightarrow {I_1}^* \rightarrow {G_1}^* \tag{5.4}$$

Equation (5.4) means that

$${I_1}^* < {G_1}^* \tag{5.5}$$

Assuming the two branch fork x to be isochronic, implies that

$${I_1}^* = {I_2}^* \tag{5.6}$$

Substituting for I_1 in (5.5) from (5.6) yields

$${I_2}^* < {G_1}^* \tag{5.7}$$

Equation (5.7) shows that Gate 2 sees the two sequential transitions $x^* \rightarrow {G_1}^*$ in the right order under the isochronic fork assumption.

Under the isochronic fork assumption complex AND-OR-NOT gates provide a complete collection of speed independent gates as long as all input signals are unipolar. However, as soon as both the inverting and non-inverting literals of a variable are used in the circuit, the basic speed independent theory assumption is violated, (unless the fork under consideration is delay insensitive or asymmetric as discussed in the following subsections). This is because the inverted and non-inverted signal transitions may not reach all fanout points simultaneously due to the delay along the inverter, unless the implementation process allows the use of complementary transistors in the pull down trees as shown in Figure 5.1-b.

5.2.2 Delay insensitive forks

The following definitions are repeated here for the ease of reference:

Definition 5.1: In a STG a signal transition is *acknowledged* by its immediate successor(s).

Definition 5.2: In an implementation a fork is called *delay insensitive* if an unbounded but finite pure delay can be attributed to any branch of the fork, while the circuit still works according to the specification. ◊

Different delays across the different branches, including possibly inverted branches, in a fork distributing signal x, introduce some new nodes, x_1, x_2, ... x_n modelling the end points of n branches, so that in the *extended* STG, x^* (in both + and - directions) is replaced with the parallel transitions shown in Figure 5.3. The extended STG for node x is represented as STG_x.

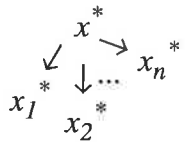


Figure 5.3: A set of parallel transitions replacing *x when signal *x* is propagated through a *n* branch fork.**

After this substitution if each x_i^* becomes an immediate predecessor of at least one transition in STG_x, then the implementation complies with the specification. Therefore [47]

Corollary 5.1: A fork is *delay insensitive* if and only if all newly introduced signal transitions (that is x_1^*, x_2^* ... x_n^* in Figure 5.3) in the corresponding extended STG are acknowledged. ◊

Corollary 5.2: In each direction *n-1* transitions out of *n* transitions in Corollary 5.1 will have immediate successors with multiple immediate predecessors.

Proof: Consider the partial STG in Figure 5.4-a.

In the corresponding complex gate implementation suppose that signal a propagates through a two branch fork with the end nodes called a_1 and a_2 as shown in Figure 5.4-b. Transition a_1^* is explicitly acknowledged by transition b^*. We assume that transition a_2^*, on the other hand, is implicitly an immediate predecessor of another transition, d^* as shown in Figure 5.4-c, that is a_2^* is a necessary condition to enable d^*. According to Muller's theory of speed independent circuits a_1^* and a_2^* are expected to simultaneously reach and be absorbed by the gates realizing variables b and d, respectively, that is $\Delta t_1 = \Delta t_3$, where Δt_1 and Δt_3 are the propagation delays along the two branches a_1 and a_2 respectively. On the other hand Δt_2, the propagation delay from a_2^* to d^* is finite but unbounded under the speed independent model. Therefore, the specification does not restrict the de-

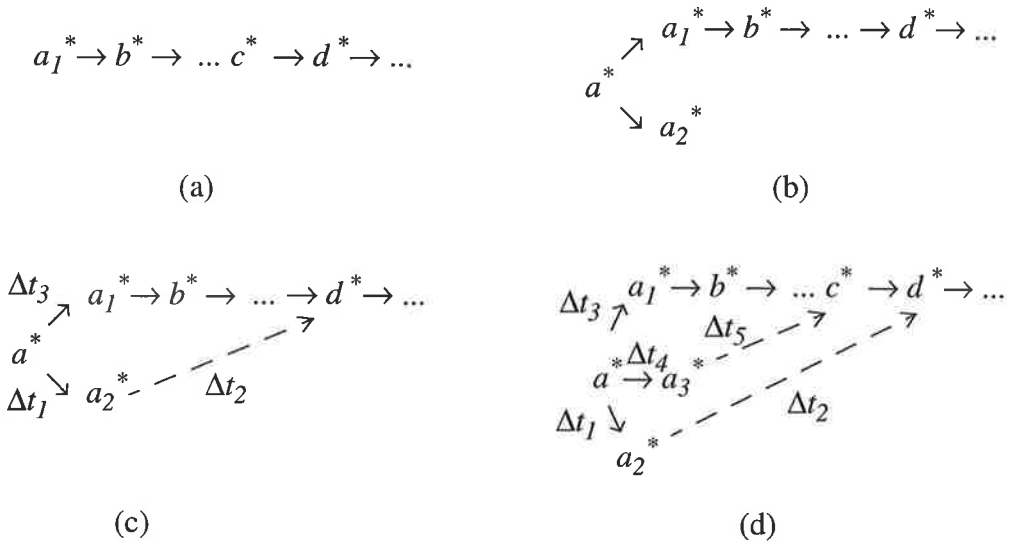


Figure 5.4: (a) A STG, (b) the extended STG with a two branch fork for literal *a*, (c) the same extended STG with the implicit transition $a_2^* \rightarrow d^*$, (d) a three branch version of (c).

lay along the sequential transitions $a^* \rightarrow a_2^* \rightarrow d^*$, that is $\Delta t_1 + \Delta t_2$. In other words the total delay $\Delta t_1 + \Delta t_2$ is specified finite but unbounded. Therefore, Δt_1 can be assumed unbounded as well and hence independent of Δt_3. This implies that fork a is delay insensitive.

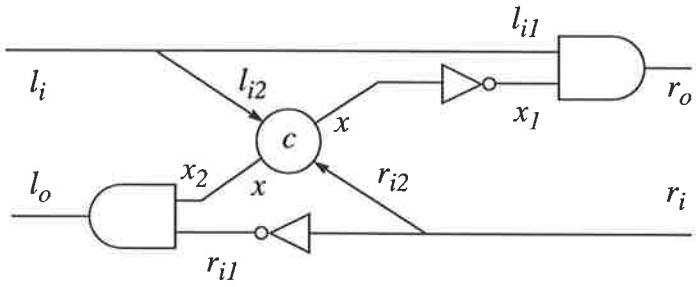
Since in the original graph, Figure 5.4-a, all transitions have at least one immediate predecessor, all newly introduced (that is the dashed) arrows, as in Figure 5.4-c will be the second or later input arrow to the corresponding immediate successor.

The idea can easily be generalised to forks with three or more branches. Figure 5.4-d shows the three branch version of the fork. In a similar way it can be shown that the propagation delay time Δt_4 along the third branch of the fork can be unbounded, if there is a dashed arrow from a_3^* to a transition c^* in the STG. ◊

As an example consider fork x in Figure 5.5 [50], with two end nodes x_1 and x_2. Notice that branch x_1 is inverted.

The resulting extended STG is shown in Figure 5.6, where both branches are acknowledged [1] on both rising and falling transitions. Therefore, fork x is a delay insensitive

1. We use the term "branch" to imply one or more of the corresponding falling / rising signal transitions, as long as the intention is clear from the context.



(a)

$$l_i^+ \rightarrow r_o^+ \rightarrow r_i^+ \rightarrow x^+ \rightarrow r_o^-$$
$$\uparrow \qquad\qquad\qquad\qquad\qquad \downarrow$$
$$l_o^- \leftarrow x^- \leftarrow l_i^- \leftarrow l_o^+ \leftarrow r_i^-$$

(b)

Figure 5.5: L/R element: (a) logic circuit, and (b) STG.

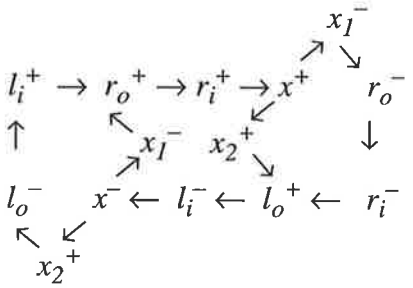


Figure 5.6: Both branches x_1 and x_2 of the isochronic fork x in Figure 5.5 are acknowledged on both transitions.

fork, as mentioned in [47] as well.

A non-delay insensitive fork need not be isochronic yet. Consider a fork x with two branches x_1 and x_2, where x_1 is acknowledged but x_2 is not. Further assume that the delay across the branch x_1 is known to be larger than that of x_2's. Then, there would be no implementation deviation from the specification, as by the time that x_1 fires x_2 has already fired. In other words the transition acknowledging x_1 may be considered the (worst case) acknowledgement of x_2 as well. This type of fork is called an *asymmetric isochronic fork*, which is the topic of the following section.

5.2.3 Asymmetric isochronic forks

In the previous section delay insensitive forks were reviewed, and it was concluded that after inserting the newly introduced transitions representing the end points of a fork, if the extended STG still remains connected (that is these transitions are acknowledged), then the fork is delay insensitive. Notice that delay insensitive forks are very rare in asynchronous circuits, which is why simple gate based delay insensitive asynchronous circuits are very limited [48]. In this section asymmetric isochronic forks, another class of interconnection forks, are considered.

The STG in Figure 5.5 has been extended for fork l_i in Figure 5.7.

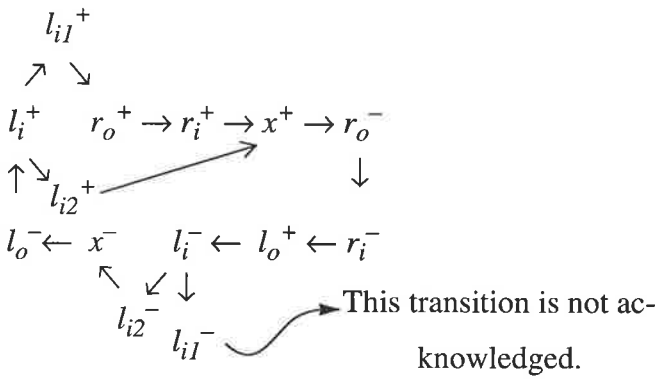
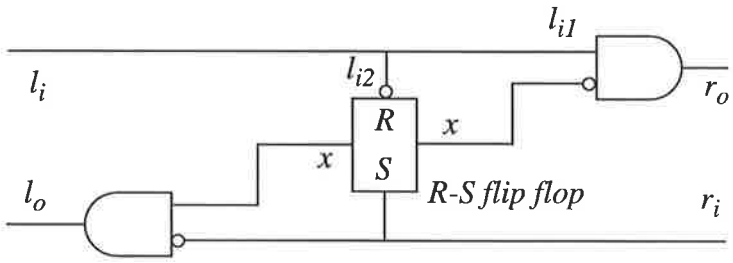


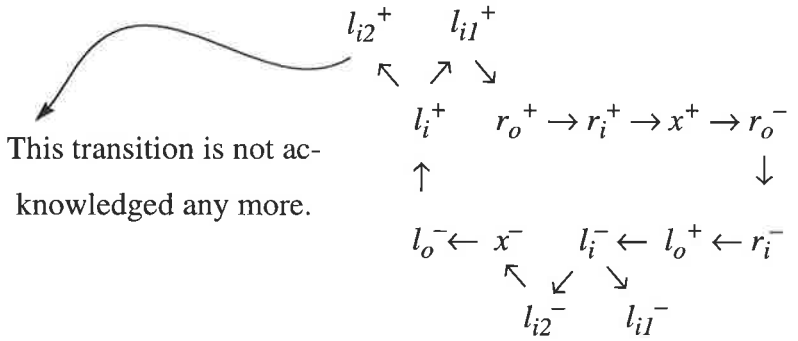
Figure 5.7: STG_{li}: the two branch fork l_i is an asymmetric isochronic fork. (See the logic circuit in Figure 5.5)

In this extended STG both transitions on l_{i2} are acknowledged. However, the other branch, l_{i1}, is acknowledged only on the positive transition. Therefore, this fork is not delay insensitive, but it does not have to be isochronic either. Since the branch l_{i2} is always acknowledged, there is no implementation deviation from the specification as long as the delay along the branch l_{i2} exceeds that of the branch l_{i1}. Asymmetric forks are particularly useful when one branch of a fork is inverted. Now if the inverted branch is always acknowledged, the implementation is correct if the delay along the inverted branch dominates the delay along the other branch, which is a reasonable assumption in many cases.

Notice that the type of an interconnection fork may depend on the specific implementation, as shown in Figure 5.8. In this figure the L/R element has been implemented with



(a)



(b)

Figure 5.8: (a) L/R element implemented with a R-S flip-flop, (b) extended STG for fork l_i.

a R-S flop-flop, instead of a Muller C-element as shown in Figure 5.5. As a result of this particular implementation, the fork l_i has to be isochronic, as, unlike the extended STG in Figure 5.7, the transition l_{i2}^+ is not acknowledged any more, as shown in Figure 5.8-b.

5.3 Relocation of problematic inverters

In this section we discuss how hazardous inverters may be relocated in the circuit in order to possibly avoid delay hazards. Consider the problematic inverter in Figure 5.9-a where a^* immediately follows b^* in the corresponding STG. Assume that in fork b the inverted branch is not acknowledged on at least one direction. Then, this inverter has to be relocated or the circuit may undergo spurious transitions. Applying De-Morgan's law to gate y makes fork b free of inverters, as shown in Figure 5.9-b, although now fork a has received one inverter. If fork a happens to be delay insensitive or its inverted branch is acknowledged then the behaviour of this part of the circuit would be hazard free. Other-

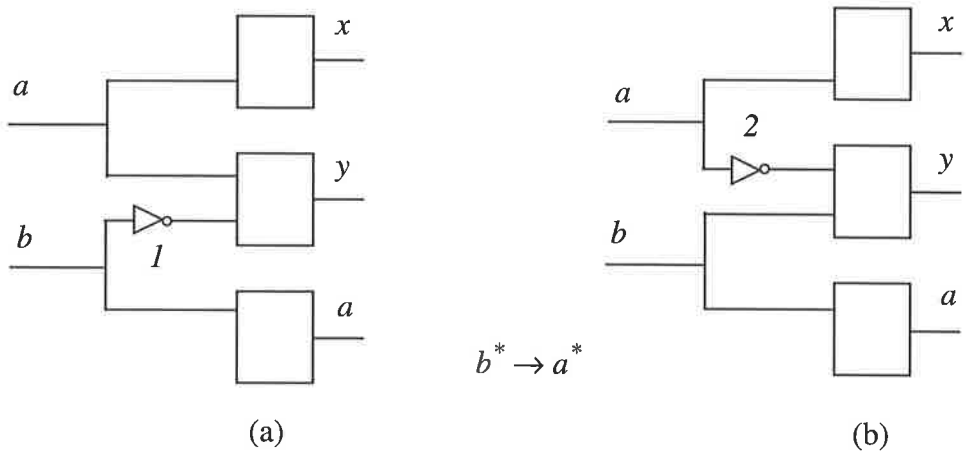


Figure 5.9: Using De-Morgan's law a problematic inverter (1) is moved to a asymmetric fork (2).

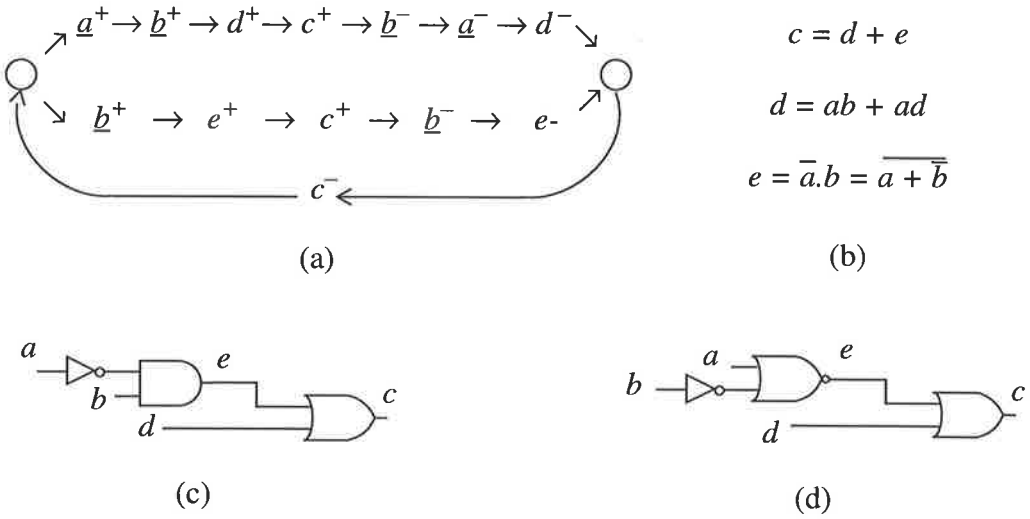


Figure 5.10: A problematic inverter which cannot be eliminated by applying De-Morgan's Law, (a) STG, (b) logic equations, (c) and (d) two possible 2-level implementations.

wise as shown in the following example the inverter would remain problematic even if De-Morgan's law were applied.

Figure 5.10 shows a STG, the logic equations for nodes c, d and e, and the two possible implementations for node e. Notice the transition $a^+ \rightarrow b^+$ may generate a spurious transition on node e in Figure 5.10-c as a result of the delay along the inverter. This prob-

lem may not be solved by applying De-Morgan's law (Figure 5.10-d), as now another pair of transitions, that is $b^- \rightarrow a^-$ becomes hazardous.

The situation becomes more complex when multiple branch forks are considered, although the principle remains the same.

Every gate, including a safe cell (to be discussed later) may have two visible complementary outputs:

Definition 5.3: The *primary output* (*PO*) of a gate is the visible output generated first.

Definition 5.4: The complement of a primary output is called the *secondary output* (*SO*).

In other words *PO* and *SO* are the non-inverted and inverted branches of a two branch fork corresponding to the variable being propagated through the fork. Notice that having assumed the isochronic fork model for the possible wire fork following the inverter in the *SO* branch, it is sufficient for only one branch in the wire fork to be acknowledged (in order to have a hazard free implementation) if no *PO* type branch has already been acknowledged. This is further clarified by the following example. Consider the extended STG in Figure 5.11.

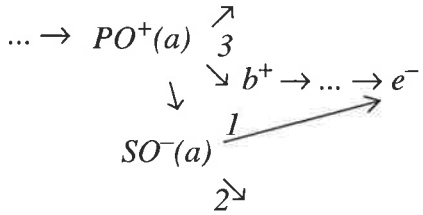


Figure 5.11: A hazardous fork on SO(a).

In the STG $SO^-(a)_1$ is acknowledged and the other branch, $SO^-(a)_2$, is not. However, what makes the inverted fork *a* problematic, as far as this partial STG is concerned, is that $SO^-(a)_1$ is not acknowledged before $PO^+(a)_3$ has been acknowledged. In other words the acknowledgement of $SO^-(a)_1$, (that is e^-) does not fire before the acknowledgement of $PO^+(a)_3$, (that is b^+) has fired. This may cause the node which has $SO(a)_2$ in its fanin set, to erroneously see the transition $SO^-(a)$ fire *after* some transitions such as b^+ have fired, due to the unbounded delay assumed along the inverter generating $SO^-(a)$.

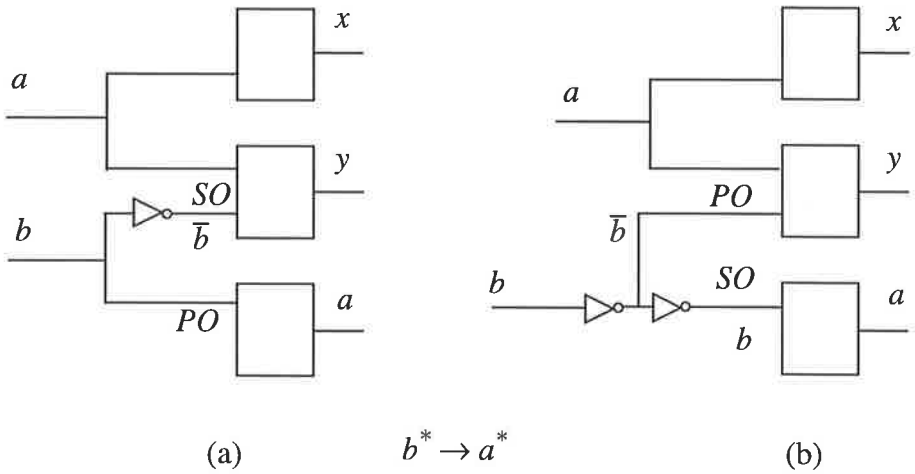


Figure 5.12: Using double inversion the required order, that is b^* and then a^*, is guaranteed.

Further to De-Morgan's law, "double inversion", that is converting a *PO* type output to a *SO* type and vice versa may also help solve the delay hazard problem as shown in Figure 5.12.

In this technique the problematic inverter remains in the same fork but is moved to another branch. In Figure 5.12-a suppose that $SO(b) = \overline{b}$ is not acknowledged so the implementation is hazardous. In Figure 5.12-b literal b (instead of $\overline{b}$) has become *SO(b)*, and since it is acknowledged according to the assumed transition pair $b^* \rightarrow a^*$, the implementation becomes hazard free, as far as this partial STG is concerned. Notice that this technique may only solve the problem under special conditions. Consider a common situation in which *SO(b)* is acknowledged in only one direction and *PO(b)* is acknowledged only in the other direction. Then, no matter which literal of the variable b happens to be *SO(b)*, the implementation would be hazardous under the speed independent model.

Kishinevsky, et al. [33] have introduced a two-phase RS-implementation which eliminates the delay hazards caused by inverters. Although this technique suffers from some speed and area penalty due to doubling the nodes in order to introduce modelling nodes and the modulo-2 operations for extraction of the original signals, it has two interesting features apart from its robustness against gate delays:

- the complete state coding (CSC) problem is automatically solved by doubling the nodes,

- using a simple algorithm, the logic equation for each node is easily determined with *no* need for the state transition diagram, which grows exponentially as the number of parallel transitions increases.

The methodology will be discussed in more detail in the following section.

5.4 Safe cells

In the previous sections different types of interconnection forks were discussed, and it was concluded that in complex AND-OR-NOT based circuits an inverter causes delay hazards if it happens to be placed on a branch of an isochronic fork. Remember that if two sequential inputs *SO(x)* and *PO(y)* ($x^+ \rightarrow y^+$) are applied to a logic operator z, then due to the delay along the inverter on the *SO* branch, the node z may see the rise of x after y has gone up and this may result in a hazard and hence non-speed independence. On the other hand, a safe cell, introduced in this subsection, is so realized that its output (the effect) cannot change unless both its *SO* and *PO* type input transitions (that is, the cause) have already been stabilized and absorbed. Now in the above example if y were a safe cell, then z would never see y^+ before $\sim x^-$ occurs, as y would never have been pulled up if both x and $\sim x$ had not been absorbed yet in the fanout points, where $\sim x$ is the inverse of x. (We will use the negation sign ~ instead of an overline for the sake of clarity whenever necessary.)

A safe cell is realized with two complementary trees, *T1* and *T2*, and one 2-input Muller C-element as shown in Figure 5.13 in which $\forall i,\ 1 \le i \le n,\ b_i = \overline{a}_i$.

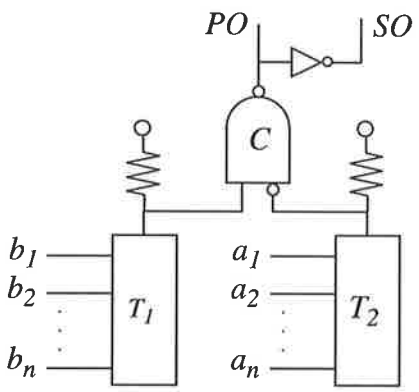


Figure 5.13: The basic structure of a safe cell.

If a gate receives a *PO* type input transition its safe version will receive both the *PO* and *SO* type transitions according to Figure 5.13. As a result of a *PO* type input transition the output of the Muller-C element may not change unless both the trees have been stabilised which necessitates the absorption of both *PO* and *SO* type input transitions, that is the output will not change until the *SO* type transition has fired as well. Both *PO* and *SO* type transitions are acknowledged in safe cells. This eliminates the hazardous effect of inverters.

Notice that the operation of safe cells is not dual rail coding based. Instead, they are driven by and generate normal single rail logic levels. Therefore, these cells can easily be mixed with normal complex gates or even simple gates as far as logic levels are concerned.

Theorem 5.1: In order to remove delay hazards resulting from a non-acknowledged *SO* type transition, it is sufficient to make safe all logic operators corresponding to the *immediate successors* of the corresponding *PO* type transition. ◊

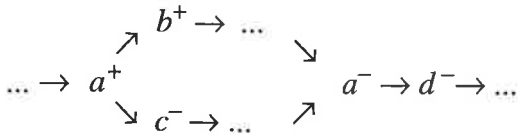
Proof: Considering the above introduction the theorem is proved.

Consider the partial STG and the corresponding extended STG in Figure 5.14-a and b, respectively. Transition a^+ is not acknowledged on the *SO* side. Therefore, according to Theorem 5.1 it is sufficient to make both nodes *b* and *c* safe in order to avoid the hazardous effect of the transitions on node *a*.

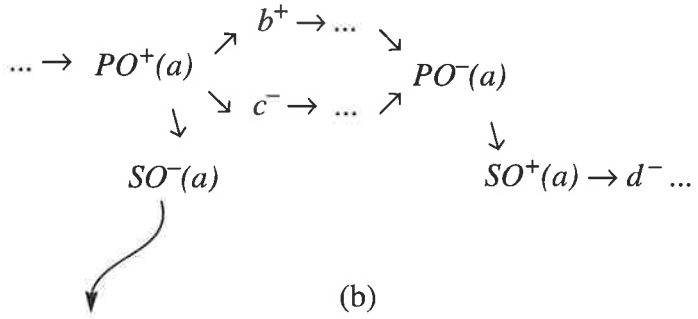
The dashed arrows in Figure 5.14-c show how the graph would be modified if nodes *c* and *b* were made safe, where none of the sequences starting with b^+ and c^- might proceed unless $SO^-(a)$ fires. Notice that unlike the implicit acknowledgement in Figure 5.4, making the graph connected by only one dashed arrow, (that is only one acknowledgement) is not always sufficient to avoid delay hazards in situations like Figure 5.14-c, as because of the incorrect state of logic high on *SO(a)*, generally speaking some nodes may be affected erroneously while the non-acknowledging sequence of transitions proceeds forward.

In this example nodes *b* and *c* are not of course the best candidates for being safe, as using the double inversion technique the new extended STG in Figure 5.15 requires only one node, that is *d*, to be implemented safe, as shown by the dashed arrow.

Notice that Theorem 5.1 only specifies some sufficient conditions to make a circuit

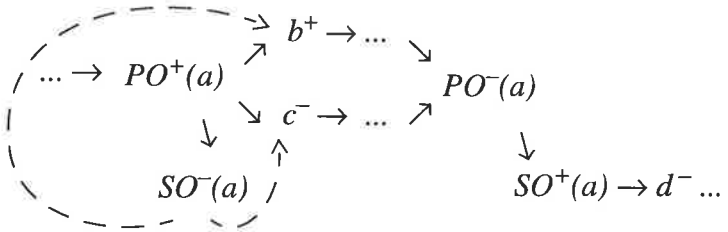


(a)



(b)

This transition is not acknowledged.



(c)

Figure 5.14: (a) A partial STG, (b) the corresponding extended STG, and (c) the same STG with two safe node *b* and *c*.

hazard free, that is not all immediate successors of non-acknowledged *SO* type transitions have to be always safe. We discuss one situation here. More work is needed to formally determine the necessary conditions to implement a variable as a safe cell.

Example: Consider the STG shown in Figure 5.16-a, and the extended STG in Figure 5.16-b. We show that although $SO^-(a)$ is not acknowledged and c^- is an immediate successor of $PO^+(a)$, (that is the sufficient conditions in Theorem 5.1 are satisfied), node *c*

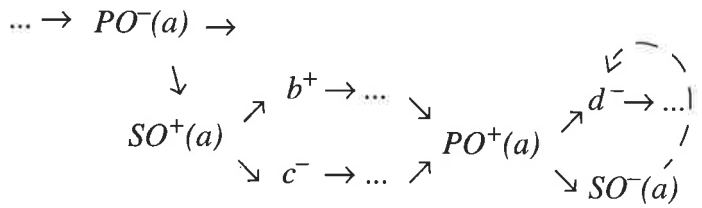


Figure 5.15: A second arrangement for the extended STG in Figure 5.14. Now only one node, *d*, needs to be safe.

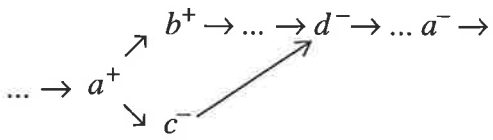
does not necessarily have to be implemented as a safe cell.

Notice that the only node which may directly be affected by a late transition of $SO^-(a)$ is the node which has $SO(a)$ in its fanin set. Spurious transitions on this node may of course result in further deviations from the specification on some other nodes. Assume that node b (but not c) is safe as shown in Figure 5.16-c. Therefore, the only erroneous state in Figure 5.16-b occurs when c^- has fired while $SO(a)$ is still erroneously high. This of course may only occur if b^+ has not fired yet, as we assume that variable b has been implemented as a safe cell as shown in Figure 5.16-c. Otherwise if b^+ has already fired, $SO(a)$ has necessarily been stabilized (to logic low) as well (because of the dashed arrow in Figure 5.16-c) and hence no spurious transition appears on any outputs. Thus, as far as this partial STG is concerned, the node which has $SO(a)$ in its fanin set may have spurious transitions only if $SO(a) = 1$ and then c is pulled down.

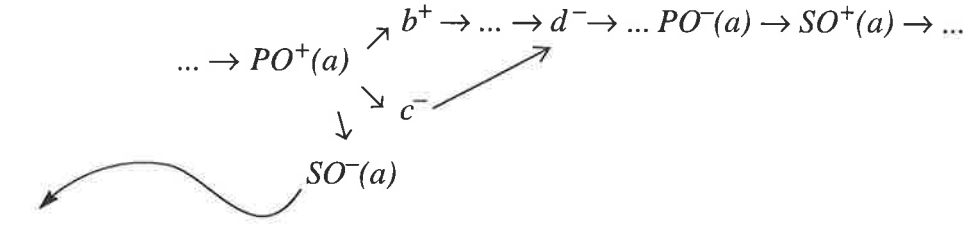
On the other hand a literal at an erroneous logic high level (that is $SO(a) = 1$) may only tend to pull the output of a SOP-NOT, x, down while x has been specified to stay high. Therefore, if $SO(a)$ and $\bar{c}$ are not in the same p-term of the complex gate implementing the above node, no p-term may be enabled by the erroneous input $SO(a) = 1$ and hence no output will suffer from spurious transitions.

Example: In this section the STG in Figure 5.17, taken from [4][33][59], is implemented using both the 2-phase RS technique and safe cells, and it is shown that the second method results in a considerable area gain.

In the 2-phase RS method two modelling signals, x_1 and x_2, are introduced for every original one, x. Considering the equivalent two phase STG (not shown here), one consisting of all positive and the other one consisting of all negative modelling variables, the log-

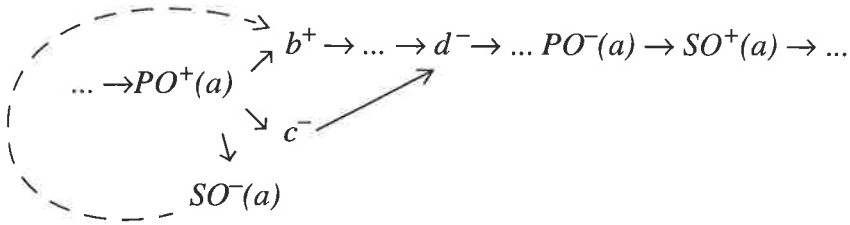


(a)



This transition is not acknowledged.

(b)



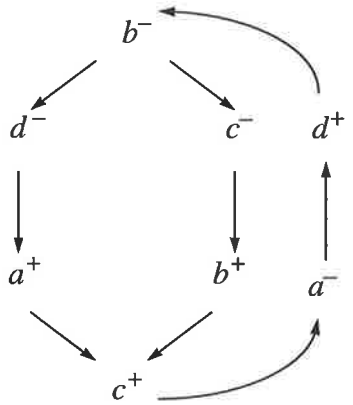
(c)

Figure 5.16: (a) A partial STG, (b) the corresponding extended STG, (c) the same STG with a safe node *b*.

ic equation for these variables can be readily determined as follows using a simple algorithm described in [33]. Table 5.1 shows the set and reset functions[2] for the RS flip-flops realizing different nodes, as depicted in Figure 5.18, for node a_2 as an example.

Then each original signal is determined using a modulo-2 function as shown in Figure 5.19 for the variable a, as an example.

2. As shown in Figure 5.18, the original signal corresponding to the immediate predecessor('s) of the modelling signal being generated has to be considered in the R and S functions as well. See [33] for details.



$$a = \overline{\bar{b}.\bar{d}+\bar{c}.\bar{d}}$$
$$b = \bar{c}+b.\bar{d}$$
$$c = \overline{(\bar{b}+\bar{c}).(\bar{a}+\bar{b})}$$
$$d = \overline{a+\bar{b}+\bar{c}}$$

Figure 5.17: A STG and one corresponding group of logic equations.

Modelling Variables	a_1	a_2	b_1	b_2	c_1	c_2	d_1	d_2
Set	d_1	c_2	$\bar{d}_2$	c_1	b_1	$a_1.b_2$	b_1	a_2
Reset	$\bar{d}_1$	$\bar{c}_2$	d_2	$\bar{c}_1$	$\bar{b}_1$	$\bar{a}_1.\bar{b}_2$	$\bar{b}_1$	$\bar{a}_2$

Table 5.1 : Set and Reset functions for the modelling variables.

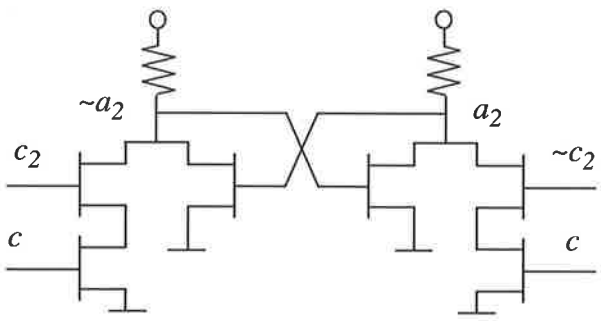


Figure 5.18: A modelling node, a_2, implemented as two cross coupled complex gates.

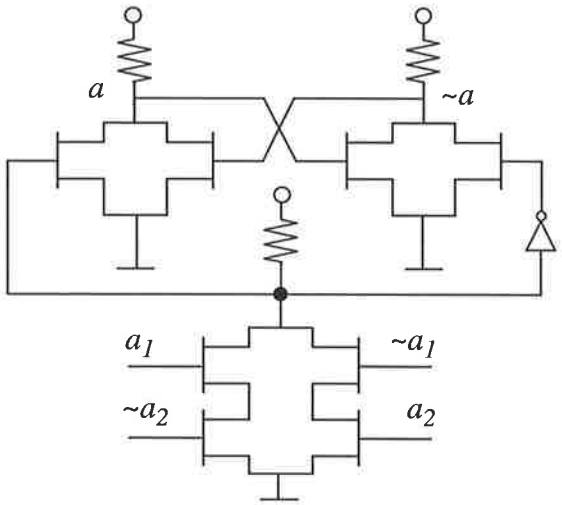


Figure 5.19: Modulo 2 function to extract original variables from modelling variables.

IS Table

PO	*IS*
$\bar{a}$	*c, d*
b	*c, d*
c	*a, b*
d	*a, b*

SI Table

IS \ *PO*	$\bar{a}$	*b*	*c*	*d*
a			✓	✓
b			✓	✓
c	×	✓		
d	✓	✓		

Figure 5.20: IS and SI Tables.

Now, the same circuit is implemented using safe cells. According to the logic equations shown in Figure 5.17, all *PO's* are inverted. The *IS Table* in Figure 5.20 shows all immediate successors of the inverted *PO's* from the STG.

Next, it is checked whether any *PO* (from the above list) excites any of its own immediate successors. Such cases are shown with a × in the *SI (Speed Independent) table* in Figure 5.20. Notice that here *b* is the corresponding *PO*. Referring to Theorem 1, the SI table shows that only node *c* may need to be implemented as a safe cell.

The above implementation needs only one safe cell, showing a significant area gain over the corresponding two-phase RS-implementation.

Considering the two methods in Section 5.3 to relax the delay hazards caused by inverters further work is required to optimize a complex gate based asynchronous circuit using safe cells.

5.5 Delay hazards analysis and verification

The unbounded delay model is sometimes considered too pessimistic [42], and hence unnecessarily expensive in terms of area and performance of the resulting implementations. With this motivation we relax the unboundedness constraint only for *inverters* in this section and assume that a logic operator in a network may only be affected by the change of precedence of only two consecutive transitions (that is, one transition and its immediate successor) due to the delay along possible inverters. In other words in a sequence of transitions $a^* \rightarrow b^* \rightarrow c^*$ we assume that the delay along the inverter in the fork of signal a is sufficiently small so that $SO^*(a)$ fires before $PO^*(c)$ fires. Based on this assumption different types of signal transitions in only *sequential* STGs (that is state machines) are analysed and hence the hazardous situations are identified.

Definition 5.5: A STG in which all transitions are of *PO* type, we call a *complete* STG. ◊

Hereafter all partial STGs are assumed to be complete unless otherwise specified.

For each pair of signal transitions like $a^* \rightarrow b^*$, two possible groups of nodes, that is, *non-immediate successor* and *immediate successor* nodes, possibly suffering from delay hazards, are considered.

5.5.1 Static delay hazards

In this section we study the possibly hazardous behaviour of a node, x, which has two transitions a^* & b^* ($a^* \rightarrow b^*$) in its fanin set, but is not the immediate successor of b^*. Therefore, during the transition $a^* \rightarrow b^*$ node x is not supposed to undergo a transition, according to the specification. However, the unknown delay along inverters may disturb the normal operation, and cause node x to experience spurious transitions.

Theorem 5.2: Suppose that node x has the two variables a and b in its fanin set. Further assume that $PO(b)^*$ immediately follows $PO(a)^*$ in the STG, that is $PO(a)^* \rightarrow PO(b)^*$, but x^* is not an immediate successor of b^*. Two different cases are considered according to the initial state of x, where x is the immediate output of the corresponding

AND-OR-NOT gate:

If x is initially at logic high, then it suffers from a delay hazard (in the form of a static 1-hazard) iff *PO (a)* undergoes a low to high transition, and both transitions $SO(a)^-$ and b^+ (where transition b^+ can be either $PO(b)^+$or $SO(b)^+$) are applied to the same p-term of node x, and this p-term is not disabled by another signal during the transition pair $PO(a)^* \rightarrow PO(b)^*$.

If x is initially at logic low, then it suffers from a delay hazard (in the form of a static 0-hazard) iff *PO (a)* undergoes a high to low transition, and transitions $SO(a)^+$ and b^- (where b^- can be either $PO(b)^-$ or $SO(b)^-$) are applied to two different p-terms of node x, and the p-term including b is the only enabled p-term of the cover during the transition pair $PO(a)^* \rightarrow PO(b)^*$.

Proof: Assuming two sequential transitions a^*& $b^*(a^* \rightarrow b^*)$ being applied to a two level atomic AND-OR-NOT circuit with an output x, the proof is based on the following partial proofs:

1. If the transitions are in the same direction, the change of their order will not cause any spurious transitions on the output, x, as two transitions with similar directions are tending to stabilize the output x at the same logic level.
2. If the transitions are in different directions, then a change in their order may cause a hazard as demonstrated here:

    2.1. Assume that the specified directions of the transitions are $a^+ \rightarrow b^-$, but b^- overtakes a^+ in time, then two different situations may be considered:

        2.1.1. The two signals are applied to the same p-term. Notice that if the specified order of the transition pair $a^+ \rightarrow b^-$ is *reversed*, that is $b^- < a^+$, then the p-term is necessarily disabled during this transition pair, as either a or b is at logic low. On the other hand, the only way that the transition pair $a^+ \rightarrow b^-$ in its *specified* order may affect the p-term is to enable the p-term by a^+, and then disable it by b^-. This in turn could only cause a negative pulse on the output of the AND-OR-NOT gate, x, pulled down by a^+, and then pulled up by b^- if the output was at logic high. Otherwise, it has no effect on the output, x. However,

according to the specification the transition a^+ does not affect the output, x. Therefore, the change of the precedence of the two transitions a^+& b^- will not violate the specification, as this change of order may only keep the p-term disabled.

2.1.2. The two signals a and b are applied to two p-terms, pa and pb, respectively. If the output x is specified to be at logic high (which entails all p-terms being disabled), no matter which transition, a^+or b^-, arrives first, the output will stay at the same level, as both p-terms must already have been disabled by other inputs regardless of the order of this pair of transitions. Now suppose that the output is at logic low, and moreover pb is the only enabled p-term, then $a+$ must be in charge of enabling another p-term (that is pa) which is intended to replace pb in order to still keep the output at logic low. This is a hazardous situation, as the enabled p-term, pb, is disabled before the disabled p-term, pa, is enabled. Therefore, a change in the order of the two transitions $a^+$and b^- may pull the output, x, up temporarily,

2.2. Assume that the specified directions of the transitions are $a^-\rightarrow b^+$, but b^+ overtakes a^- in time. Again two different situations may be considered:

2.2.1. The two signals a and b are applied to two different p-terms, pa and pb, respectively. Notice that the only way that the transition pair $a^- \rightarrow b^+$ in its specified order may affect the output of the two p-terms is to disable pa by a^-, and enable pb by b^+. This in turn might only produce a logic high pulse on the output x, which would be enabled by a^-, and then disabled by b^+ if x was originally at logic low. If the order of the transitions pair, $a^-\rightarrow b^+$, was reversed then no pulse would appear on the output, x, as at least one p-term at a time is enabled (pb is enabled before pa is disabled). However, according to the specification, the transition a^- in its correct order does not affect the output, x. Therefore, the change of the precedence of the two transitions a^- and b^+ will not violate the specification.

2.2.2. The two signals a and b are applied to the same p-term. The correct order of the transition pair $a^- \rightarrow b^+$ first disables the p-term by a^-. However, the reversed order would enable the p-term by b^+, and then disable it by a^- if the p-tern was not disabled by any other signal. This spuriously enabled p-term would cause a spurious negative pulse on the output x if x was at logic high.

All possible sequences resulting from a PO type transition pair, $PO(b)^* \rightarrow SO(b)^*$, and seen by the logic operator x are tabulated in Figure 5.21. The first column shows the four possible specified situations for the transition pair $PO(b)^* \rightarrow SO(b)^*$. The second column shows the two possible literal transition sequences for each situation. The first transition sequence in each row in which b is excited by $SO^*(a)$ is free of delay hazards, as all these literal transitions happen sequentially. In other words $SO^*(a)$ is acknowledged by b^* in all of these situations. The second sequence in each row in which $SO^*(a)$ is not acknowledged[3], however, may cause delay hazards as demonstrated below.

1. First consider the sequence 1-2 in Figure 5.21. The literal transitions for this case may be seen as one of the following sequences:

   1.1. $PO(a)^+ < PO(b)^+ < SO(a)^- < SO(b)^-$ [4], with one change in the precedence which is between $PO(b)^+$ and $SO(a)^-$, that is a positive transition, $PO(b)^+$, has erroneously preceded a negative transition, $SO(a)^-$. According to partial proof 2.2.2 if these two transitions are in the same p-term, the output might undergo a spurious logic low pulse only if it was initially at logic high.

   1.2. $PO(a)^+ < PO(b)^+ < SO(b)^- < SO(a)^-$, with two precedence changes between $SO(b)^-$ and $SO(a)^-$, and between $PO(b)^+$ and $SO(a)^-$. The first change of precedence does not cause any hazards, according to partial proof 1 above, as both of these transitions are in the same direction and hence tending to stabilize the output x at the same logic level. However, the precedence change between $PO(b)^+$ and $SO(a)^-$ may cause a delay hazard, in the same way as explained above. If $PO(b)$ and $SO(a)$ are applied to two

3. Notice that now a SO type transition being not acknowledged is not sufficient to cause hazards, as the delay along inverters is assumed to be bounded in this section.
4. $a^* < b^*$ indicates a temporal relationship meaning that a^* occurs before b^*.

PO transitions	*Literal transitions*
$1\text{-}\ PO(a)^+ \rightarrow PO(b)^+$	$1\text{-}1.\ PO(a)^+ \rightarrow SO(a)^- \rightarrow PO(b)^+ \rightarrow SO(b)^-$ $1\text{-}2.\ PO(a)^+ \rightarrow \begin{matrix} \nearrow SO(a)^- \\ PO(b)^+ \end{matrix} \rightarrow SO(b)^-$
$2\text{-}\ PO(a)^- \rightarrow PO(b)^-$	$2\text{-}1.\ PO(a)^- \rightarrow SO(a)^+ \rightarrow PO(b)^- \rightarrow SO(b)^+$ $2\text{-}2.\ PO(a)^- \rightarrow \begin{matrix} \nearrow SO(a)^+ \\ PO(b)^- \end{matrix} \rightarrow SO(b)^+$
$3\text{-}\ PO(a)^+ \rightarrow PO(b)^-$	$3\text{-}1.\ PO(a)^+ \rightarrow SO(a)^- \rightarrow PO(b)^- \rightarrow SO(b)^+$ $3\text{-}2.\ PO(a)^+ \rightarrow \begin{matrix} \nearrow SO(a)^- \\ PO(b)^- \end{matrix} \rightarrow SO(b)^+$
$4\text{-}\ PO(a)^- \rightarrow PO(b)^+$	$4\text{-}1.\ PO(a)^- \rightarrow SO(a)^+ \rightarrow PO(b)^+ \rightarrow SO(b)^-$ $4\text{-}2.\ PO(a)^- \rightarrow \begin{matrix} \nearrow SO(a)^+ \\ PO(b)^+ \end{matrix} \rightarrow SO(b)^-$

Figure 5.21: Four possible signal transitions and the corresponding literal transition graphs.

different p-terms, the output will not experience a transition, according to partial proof 2.2.1. However, if these two transitions are in the same p-term, the output might undergo a spurious logic low pulse only if it was initially at logic high, according to partial proof 2.2.2.

A similar seasoning may be applied to the sequence 3-2 in Figure 5.21. The two possible literal transitions are as follows:

1.3. $PO(a)^+ < PO(b)^- < SO(a)^- < SO(b)^+$, where the only precedence change is between $PO(b)^-$ and $SO(a)^-$. This, however, does not cause any hazards, according to partial proof 1, as both of these transitions are in the same direction and hence tending to stabilize the output x at the same logic level.

1.4. $PO(a)^+ < PO(b)^- < SO(b)^+ < SO(a)^-$, with two precedence changes between $PO(b)^-$ and $SO(a)^-$, and between $SO(b)^+$ and $SO(a)^-$. The first change of precedence does not cause any hazards, according to partial proof 1 above, as both of these transitions are in the same direction and hence tending to stabilize the output x at the same logic level. However, the precedence change between $SO(b)^+$ and $SO(a)^-$ may cause a delay hazard. If $SO(b)$ and $SO(a)$ are applied to two different p-terms, the output does not experience a transition. However, if these two transitions are in the same p-term, the output might undergo a spurious logic low pulse only if it was initially at logic high, according to partial proof 2.2.2.

2. Now consider the sequence 2-2 in Figure 5.21. The literal transitions for this case may be seen as one of the following sequences:

2.1. $PO(a)^- < PO(b)^- < SO(a)^+ < SO(b)^+$, with one change of precedence between $PO(b)^-$ and $SO(a)^+$. If $PO(b)$ and $SO(a)$ are applied to the same p-term, the output does not experience a transition, according to partial proof 2.1.1. However, if these two transitions are applied to two different p-terms of node x, the output may undergo a spurious logic high pulse only if the output was initially at logic low, and the p-term including $PO(b)$ was the only asserted p-term, according to partial proof 2.1.2.

2.2. $PO(a)^- < PO(b)^- < SO(b)^+ < SO(a)^+$ with two precedence changes between $SO(b)^+$ and $SO(a)^+$, and between $PO(b)^-$ and $SO(a)^+$. The first change of precedence does not cause any hazards, according to partial proof 1 above, as both of these transitions are in the same direction and hence tending to stabilize the output x at the same logic level. However, the precedence

change between $PO(b)^-$ and $SO(a)^+$ may cause a delay hazard. If $PO(b)$ and $SO(a)$ are applied to the same p-term the output again does not experience a transition, according to partial proof 2.1.1. However, if these two transitions are applied to two different p-terms of node x, the output may undergo a spurious logic high pulse only if the output was initially at logic low, and the p-term including $PO(b)$ was the only asserted p-term, according to partial proof 2.1.2.

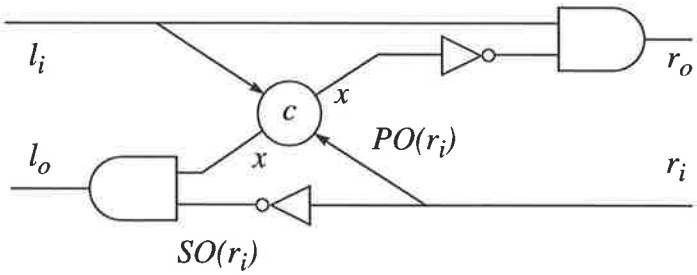
A similar reasoning may be applied to sequence 4-2. The literal transition sequence corresponding to this sequence are as follows:

2.3. $PO(a)^- < PO(b)^+ < SO(a)^+ < SO(b)^-$, in which the only change in the precedence is between $PO(b)^+$ and $SO(a)^+$. This, again, does not cause any hazards, as both of these transitions are in the same direction and hence tending to stabilize the output at the same logic level, according to partial proof 1.

2.4. $PO(a)^- < PO(b)^+ < SO(b)^- < SO(a)^+$, with two precedence changes between $PO(b)^+$ and $SO(a)^+$, and between $SO(b)^-$ and $SO(a)^+$. The first change of precedence does not cause any hazards, according to partial proof 1 above, as both of these transitions are in the same direction and hence tending to stabilize the output x at the same logic level. However, the precedence change between $SO(b)^-$ and $SO(a)^+$ may cause a delay hazard. If $SO(b)$ and $SO(a)$ are applied to the same p-term the output again does not experience a spurious transition. However, if these two transitions are applied to two different p-terms of node x, the output may undergo a spurious logic high pulse only if the output was initially at logic low, and the p-term including $SO(b)$ was the only asserted p-term, according to partial proof 2.1.2.

Example: A STG and the corresponding logic circuit[5] for a L/R element [50] are shown in Figure 5.22.[6]

5. The procedure of logic equation extraction is not discussed here. The interested reader may refer to [18].
6. This STG has been taken from [78]. In [78] we show the close similarity between STGs and the output sequence of the handshaking expansion phase of Martin's methodology.

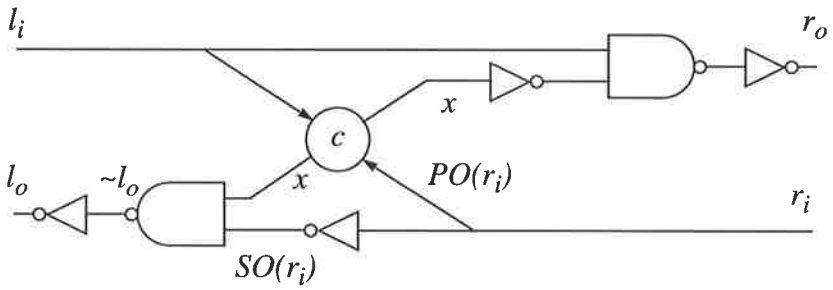


$$l_i^+ \rightarrow r_o^+ \rightarrow r_i^+ \rightarrow x^+ \rightarrow r_o^-$$
$$\uparrow \qquad\qquad\qquad\qquad \downarrow$$
$$l_o^- \leftarrow x^- \leftarrow l_i^- \leftarrow l_o^+ \leftarrow r_i^-$$

Figure 5.22: L/R element: logic circuit and STG.

Figure 5.23 shows an equivalent circuit (in which the AND gates have been replaced with NAND gates in order to show the immediate outputs of the logic operators according to our assumptions) and the corresponding complete STG. Notice that node x has been assumed to be the corresponding PO although the conclusion in this example would be valid if x were supposed to be the corresponding SO as well.

In this example the lower NAND gate generating $\sim l_o$ has the two variables r_i and x in its fanin set. Furthermore, two transitions of these variables are consecutive in the STG, that is $PO(r_i)^+ \rightarrow PO(x)^+$. Considering that this NAND gate is not disabled by any other



$$l_i^+ \rightarrow \sim r_o^- \rightarrow r_i^+ \rightarrow x^+ \rightarrow \sim r_o^+$$
$$\uparrow \qquad\qquad\qquad\qquad \downarrow$$
$$\sim l_o^+ \leftarrow x^- \leftarrow l_i^- \leftarrow \sim l_o^- \leftarrow r_i^-$$

Figure 5.23: Complete STG and SOP-NOT implementation of L/R element.

signal, case 1 of Theorem 5.2 applies to this example, as the immediate output of the gate, (that is $\sim l_o$) is at logic high during the transition pair $PO(r_i)^+ \rightarrow PO(x)^+$, $PO(r_i)$ undergoes a low to high transition in this transition, and both transitions $SO(r_i)^-$ *and* x^+ are applied to the same p-term of node $\sim l_o$. Therefore, the transition pair $PO(r_i)^+ \rightarrow PO(x)^+$ is hazardous for node $\sim l_o$. Notice that according to Theorem 5.2 this conclusion is valid no matter whether x is of *SO* or *PO* type.

The hazard problem in this specific case can be resolved by applying De-Morgan's law to the lower NAND gate, as shown in Figure 5.24, as r_i is now applied to the NOR gate in its *PO* type. It is interesting to note that the newly introduced inverter on the lower branch of fork x is not problematic at all even under unbounded delay model, as the fork is delay insensitive as discussed in Section 5.2.

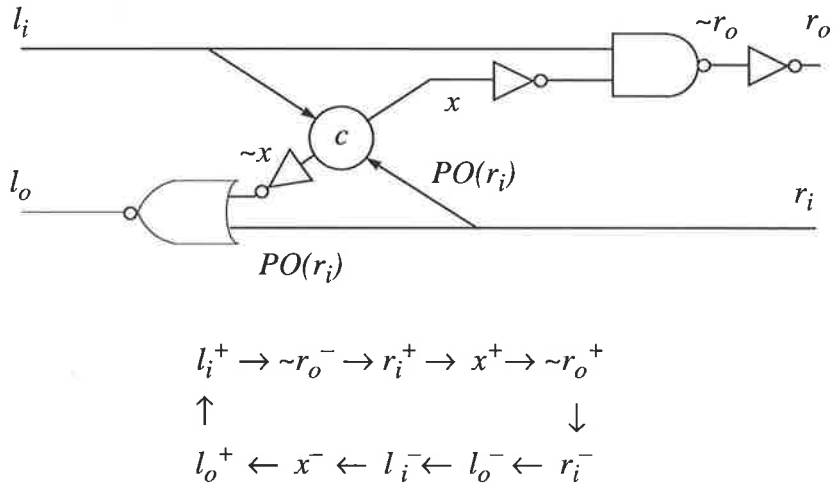


Figure 5.24: Complete STG and modified implementation of L/R element.

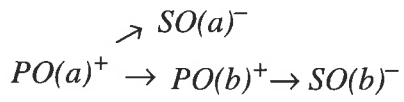
A similar situation exists for node $\sim r_o$ which has the two variables l_i (in *PO* type) and x in its fanin set, with two consecutive transitions in the STG, that is $PO(l_i)^- \rightarrow PO(x)^-$. Since $SO(l_i)$ is not in the fanin set of $\sim r_o$, a necessary condition does not exist any more and hence the transition pair $PO(l_i)^- \rightarrow PO(x)^-$ is free of delay hazards for node $\sim r_o$.

5.5.2 Dynamic delay hazards

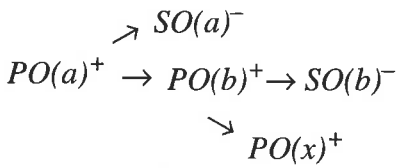
In the rest of this section the behaviour of the immediate successor of b^* in the pair of transitions $a^* \rightarrow b^*$ is investigated, that is $a^* \rightarrow b^* \rightarrow x^*$, to see how x can undergo a non-

monotonic transition by re-ordering the two preceding transitions, $a^*$and b^*. Remember that in the discussion in the previous section we assumed that node x is specified to stay at a fixed logic level during a pair of transitions. Therefore, delay hazards manifested themselves as static hazards. However, now delay hazards will manifest themselves on variable x as dynamic hazards, as x is specified to undergo a transition after the sequence $a^* \rightarrow b^*$. Since AND-OR-NOT gates are assumed atomic, at least three out of order transitions are required to generate a dynamic hazard, as a dynamic hazard causes at least three transitions on the hazardous output. Therefore, under our assumption of only two out of order transitions, dynamic hazards are ruled out in sequential STGs.

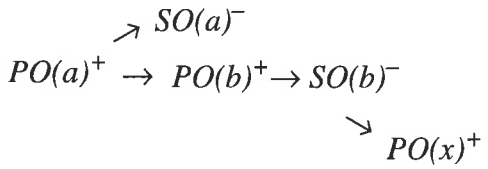
The above conclusion may be verified more closely by looking at all possible literal transitions in Figure 5.21. All literal transitions in the first row of every situation have all transitions acknowledged, and hence are free of delay hazards. The second row of each situation, however, has some out of order transitions. We study case 1-2 which is repeated in Figure 5.25-a for the sake of clarity. The rest of the sequences may be treated similarly.



(a)



(b)



(c)

Figure 5.25: (a) Literal transition 1-2 from Figure 5.21, (b) and (c) two possible positions for the third transition, $PO(x)^+$.

The signal transition corresponding to x may be added to this literal transition sequence according to either Figure 5.25-b or Figure 5.25-c. Without the loss of generality the third transition has been assumed to be of positive type. In Figure 5.25-b $PO(x)^+$ is excited by $PO(b)^+$. Considering the atomic nature of the AND-OR-NOT gates used in this section, the late literal transition $SO(a)^-$ can only pull $PO(x)^+$ down, therefore its delay in firing has no effect on $PO(b)^+$. In Figure 5.23-c $PO(x)^+$ is excited by $SO(b)^-$. Notice that now the delayed literal transition $SO(a)^-$ is in the same direction as the immediate predecessor of $PO(x)^+$, that is $SO(b)^-$, therefore the only effect of this delay can be delaying $PO(x)^+$.

5.6 Logic hazards in complex logic gates

Different types of logic hazards have been studied extensively in two stage combinational logic [83]. In this section complex AND-OR-NOT gates with possible inverters at some inputs are analysed and are shown to be more robust against logic hazards than their normal two stage simple gate counterparts. Although complex gates have been used by some researchers as mentioned earlier, we are not aware of a systematic hazard analysis in this family so far. Furthermore, in most design strategies the delay along the inverters is assumed negligible [17] [55] as mentioned in the introduction.

Remember that the delay model assumed in this chapter is the *pure gate delay model.* In this section we assume all logic circuits are in the form of *sum of product (SOP)* (unless otherwise specified) possibly with some inverted inputs, although taking proper duality into consideration, our discussions would apply to *product of sum* implementations as well. It is also assumed that all transitions are free of function hazards.

Lemmas 1, 3 and 5 are adapted mainly from [83][64], while the rest which deal with complex gates, have been developed in this work.

Lemma 5.1: A single input transition in simple gate SOPs is free of static 1-hazard iff both vertices are covered by a single product term (p-term). ◊

The example in Figure 5.26 demonstrates a hazardous single input change, *abc: 011 → 111*. A redundant cube is needed to remove the hazard. Notice that here both directions $q \rightarrow x$ & $x \rightarrow q$ are hazardous, while according to the following Lemma in complex gate based SOPs only one direction is hazardous.

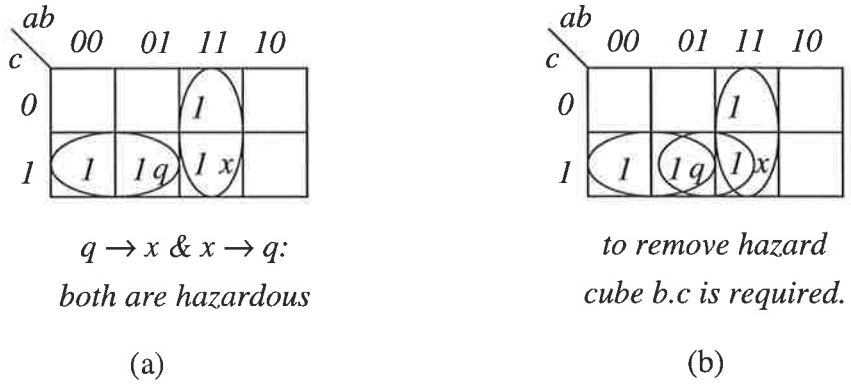


Figure 5.26: In simple gate based SOPs (a) both transitions $q \rightarrow x$ & $x \rightarrow q$ are hazardous, (b) a redundant cube *b.c* is introduced to remove the hazard.

Lemma 5.2: A low to high single *PO* type input transition may not have a static 1-hazard in complex gate SOPs. A high to low transition of a *PO* type input, however, has a static 1-hazard in this family iff both (on-set) vertices are not covered by a single p-term.

Proof: A single input transition may disable and enable two enabled and disabled, respectively, p-terms. If the enabled p-term is disabled before the other p-term is enabled then the output will suffer from a low glitch. In complex gates, however, the transition on the *PO* always reaches the atomic AND-OR-NOT gate before the *SO* transition reaches it, under the isochronic fork assumption. Therefore, it is always guaranteed that the disabled p-term is first enabled if the *PO* transition happens to be low to high. Similarly a low to high *SO* type transition always reaches the gate after the high to low type *PO* has reached the gate. This always carries the risk of spurious transitions. ◊

Therefore the transition *q: 011* → *x: 111* is hazard free in Figure 5.27, as *PO(a)* undergoes a low to high transition. However, the transition *x: 111* → *q: 011* is hazardous, because now *SO*(*a*) (and not *PO*(*a*)) experiences a low to high transition.

Lemma 5.3: A multiple input transition in simple gate SOPs is free of static 1-hazards iff the transition cube is covered by one p-term. ◊

Figure 5.28-a shows a hazardous multiple high to high input transition due to switch over between the p-terms similar to that shown in Figure 5.26, where unlike the case in Figure 5.28-b the transition cube is not covered by one cube. The following Lemma shows that complex gates are more immune to multiple input change static 1-hazards as well. However, this immunity is not of any practical advantage, as it is never economical to im-

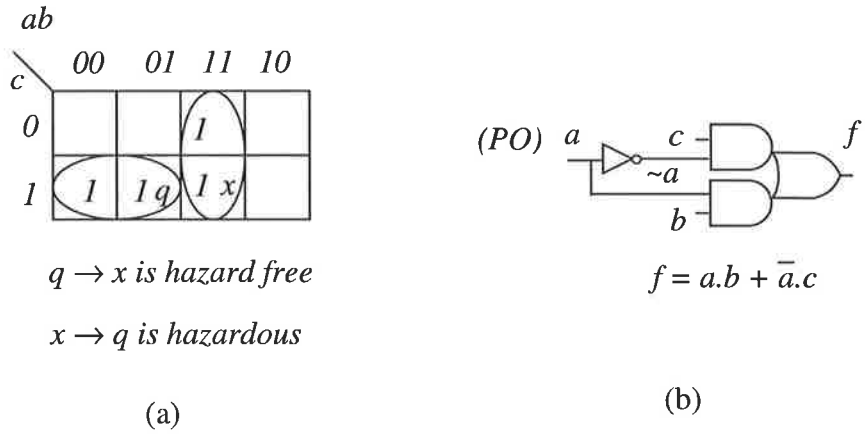


Figure 5.27: (a) The immunity of complex gates against single input change static hazards, (b) a typical logic diagram.

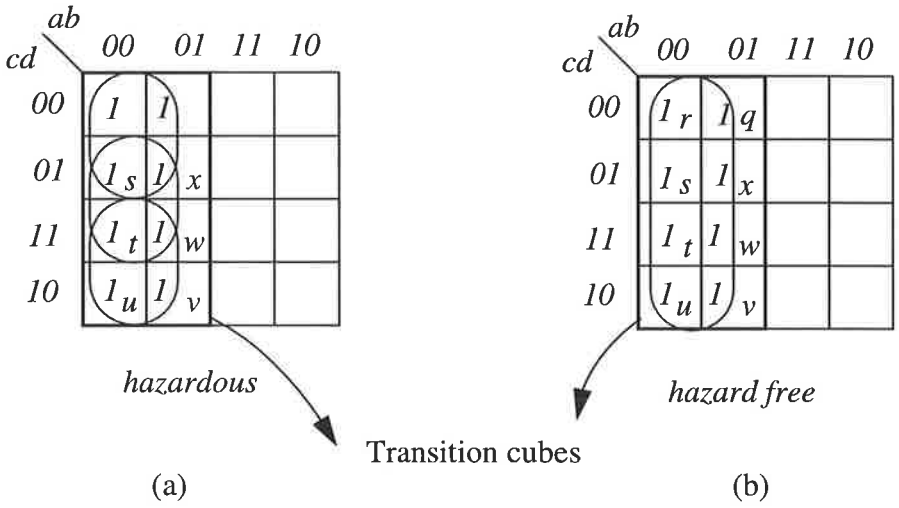


Figure 5.28: (a) Hazardous and (b) hazard free multiple input (static 1-hazard) in simple gate based SOPs.

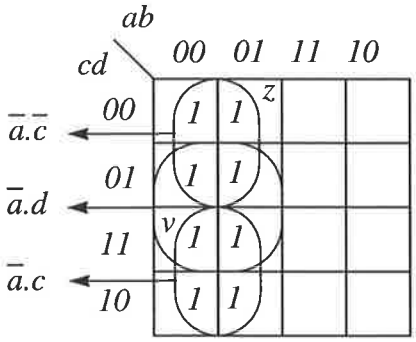
plement an all-one transition cube with multiple p-terms.

In a 1-1 multiple input change, an originally disabled p-term which may be enabled during the transition, has one or more *enabling inputs*:

Lemma 5.4: A multiple input transition is free of static 1-hazards in AND-OR-NOT complex gate based SOPs iff each vertex in the transition cube belongs to an initially enabled p-term, or a p-term with no *SO* type enabling inputs.

Proof:

Similarly to single input change, suppose that a low to high SO type transition enables



$v \rightarrow z$: $\sim c^+, \sim d^+ (SO) \Rightarrow$ *hazardous*

$z \rightarrow v$: $c^+, d^+ (PO) \Rightarrow$ *hazard free*

Figure 5.29: Hazardous and hazard free multiple input change (static 1-hazard) in complex gate based SOPs.

a disabled p-term to replace another enabled p-term which is disabled during the transition. Due to the unknown delay of the inverter on the way of the SO type transition the p-term may be enabled after the enabled p-term becomes disabled, resulting in hazards.

Suppose that the output of a SOP suffers from a spurious low glitch when an enabled p-term is disabled and replaced with a newly enabled p-term. The signal glitch is again caused by late enabling of the disabled p-term. Under the atomic gate model for complex AND-OR-NOT gates this, however, may only happen if the enabling low to high transition is of SO type, which reaches the gate in an unknown propagation delay time. ◊

For example consider the implementation shown in Figure 5.29. The transition *z: 0100* $\rightarrow$ *v: 0011* is hazard free, as both enabling transitions, c^+ and d^+, are of *PO* type. However, the reverse transition, that is *v: 0011* $\rightarrow$ *z: 0100* is hazardous, because the two enabling transitions, that is $\sim c^+$ and $\sim d^+$ are of *SO* type.

Lemma 5.5: A multiple input transition in simple gate SOPs is free of 0-1 dynamic hazards iff no product term in the cover intersects the transition cube unless it also covers the final vertex[7].

Figure 5.30 shows an example demonstrating a low to high dynamic hazard. If the p-term p_1 is sufficiently fast that p_1 is enabled and then disabled before p_2 has had a chance to become enabled, the output may undergo a non-monotonic transition.

7. We show in [76] that multiple input transitions are free of 1-0 dynamic hazards under the inertial delay model for SOP circuits.

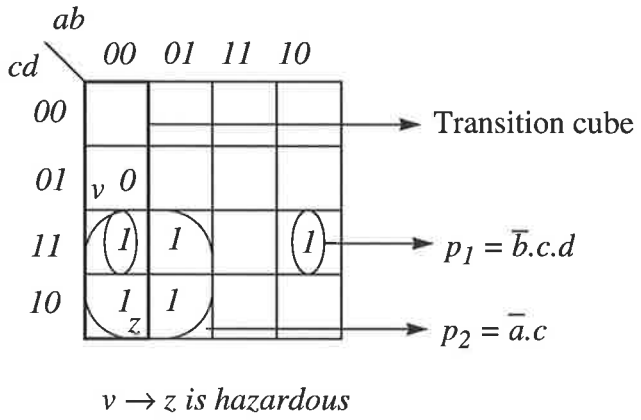


Figure 5.30: Multiple input low to high dynamic hazards in simple gate based SOPs.

Lemma 5.6: Multiple input transitions in AND-OR-NOT complex gate based SOPs are free of dynamic hazards. ◊

Specifically speaking, we mean *real* dynamic hazards in Lemma 5.6. See Chapter 3 for different types of dynamic hazard.

Proof: For two level SOP logic, referring to Theorem 3.3 recall that in a multiple input transition 1-0 dynamic logic hazard, each h_0-type p-term is necessarily disabled by at least the same input transition which disables one or more of the t_0 type p-terms. Since now there is no first level gate delay in our intended implementation, the complex gate *sees* the falling transitions of both h_0-type (if any) and t_0-type p-terms at the same time. Therefore the output is free of spurious transitions.

For multiple input transition 0-1 dynamic logic hazard, through a similar argument in the proof of Theorem 3.3 it is shown that each h_0-type p-term is necessarily enabled by at least the same input transition which enables one or more of the t_1 type p-terms. Since the t_1-type p-term is an ideal instantaneous AND gate, the complex gate will NOT see its transition after it sees the positive transition of the h_0-type p-term. Therefore the output will not undergo any spurious transitions.◊

5.7 Conclusion

In this chapter we first studied different types of interconnection forks in STG based speed independent circuits and concluded that inverters on the input of some complex AND-OR-NOT gates may cause delay hazards in many cases where isochronic forks are

inevitable. With this motivation safe cells were introduced and shown to be robust against delay hazards. Then, some sufficient conditions were identified to determine those nodes in a network which need to be realized as safe cells. This method was compared with the two phase RS-implementation technique in an example, and it was concluded that safe cells might result in a significant chip area gain.

Safe cells are so designed that they acknowledge both the *SO* and *PO* type input transitions. Therefore, no matter how slow the inverters are, the correct order of transitions are always guaranteed by safe cells.

We then relaxed the so called pessimistic assumption of unboundedness of delays in inverters, studied delay hazards in complex gate based circuits and identified a subclass of delay hazards in different nodes of the network.

Finally combinational hazards in complex AND-OR-NOT gates were discussed and it was shown that this logic family is more immune to those hazards than the two level AND-OR logic.

Chapter 6

A Tabular Method for Guard Strengthening, Symmetrization and Operator Reduction for Martin's Asynchronous Design Methodology

6.1 Introduction

In this chapter we introduce a tabular method to perform the last two of the four phases of Martin's compilation process for asynchronous circuit design. The method is then demonstrated with three examples, illustrating that our systematic method is very straightforward, flexible and convenient to apply, and hence it lends itself to automatic compilation. The technique is independent of the particular delay assumption considered in the design process.

6.2 Overview of Martin's methodology for asynchronous logic design

Martin [47][50][51][53] has developed an asynchronous design methodology[1] starting with a high level description similar to Hoare's CSP [27]. The source program is compiled into some asynchronous circuits in four stages called *process decomposition, handshaking expansion, production rule expansion* and *operator reduction.*

During the **process decomposition** stage, the initial (complex) program is translated into some sub-programs with simpler control structures using the following *subroutine call* fashion algorithm:

Replace the process P containing a section S with two processes P_1 and P_2 communicating on the channel (c_1, c_2). Here P_1 is the same as P, but replacing S with a communication action c_1 on channel (c_1, c_2), and P_2 is the process $*[[\bar{c}_2 \rightarrow S\ ;\ c_2]]$, where $\bar{c}_2$ (called a *probe*) indicates to P_2 that the calling process (P_1) is waiting for P_2. Having found $\bar{c}_2$ asserted, the process P_2 executes S and then proceeds to complete the suspended handshake cycle, c_2, (see below) initiated by the process P_1.

The **handshaking expansion** phase replaces each communication action with the corresponding handshaking signal transitions. To carry out this transformation, each so-called software channel is implemented with two wires between the two communicating modules: for example two wires c_{in2} -c_{out1} and c_{out2}- c_{in1} for channel (c_1, c_2). Four phase handshaking has been utilised in Martin's methodology, that is on the P_2 side, the signal transition sequence $[c_{in2}] \rightarrow c_{out2}\uparrow \rightarrow [\neg c_{in2}] \rightarrow c_{out2}\downarrow$ replaces a full communication action on channel (c_1, c_2), where $\rightarrow$ indicates the cause-effect relationship, and $[x]$ means "wait until x is asserted". Notice that now c_{in2} in fact conveys the notion of the probe $\bar{c}_2$, which is the handshake initializer of the communication action c_2. Therefore, the process p_2 is eventually translated into $*[[c_{in2}] \rightarrow S;\ c_{out2}\uparrow\ ;\ [\neg c_{in2}]\ ;\ c_{out2}\downarrow]$ at the end of the handshaking expansion phase.

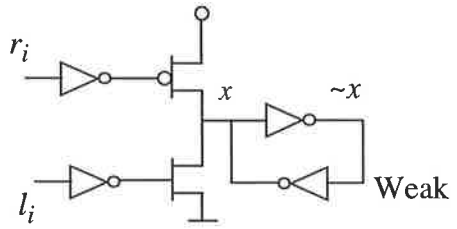
The **production rule expansion** consists of three sub-stages:

- State assignment, which is a heuristic solution for the lack of sufficient variables; that is to satisfy the complete state coding (CSC) condition [39][84][95]. As an example, x is not a pre-specified signal in Figure 6.1 (from Example 6.1), however, it provides the circuit with memory to remember

1. The design methodology has been automated in [13].

$$*[[l_i] ; r_o\uparrow ; [r_i] ; x\uparrow ; [x] ; r_o\downarrow ; [\sim r_i] ; l_o\uparrow ; [\sim l_i] ; x\downarrow ; [\sim x] \ ; \ lo\downarrow]$$

(a)



(b)

Figure 6.1: (a) The output of the handshake expansion of the L/R element. Variable x has been introduced to provide the circuit with enough memory, (b) an implementation for variable *x (see Example 6.1)*.

sufficient past history in order to distinguish different situations and hence behave according to the specification.

- Guard strengthening: adding some more restrictions to the input of some logic operators to enforce program sequencing. This guarantees that the signal transitions in physical design occur in the same order as they have been specified in the compiled program.
- Symmetrization, which is an attempt to avoid state holding operators as much as possible. (See Example 6.3.)

In the **operator reduction** stage, the required logic gates are determined based on the compiled program sequence generated in the previous stage.

Martin's research group reported in [52] the first asynchronous microprocessor designed and implemented based on this methodology. The successful fabrication of this chip and some other complex systems on first silicon, such as "a self-timed circuit for a distributed mutual exclusion" (see Example 6.3), show the robustness of this methodology.

In this article we introduce a tabular method replacing the current technique for the last two stages, specifically guard strengthening, symmetrization and operator reduction.

Notice, however, that this is a general method to realize well-formed STG's as complex CMOS combinational/sequential gates, as discussed in the following section.

6.3 Circuit realization

Martin has used a CMOS based implementation technique in which for every combinational cube covering some vertices in the on-set and off-set of each signal, $x_i = f(x_1, x_2, \ldots x_n)$, a pass transistor path is allocated from the corresponding node to *Vdd* and *GND* respectively. In order to avoid the threshold voltage loss, NMOS and PMOS transistors are used in the pull down and pull up trees, respectively. For the state holding gates, a cross coupled inverter pair is usually placed at the output, although it could be eliminated if a minimum refreshing frequency was guaranteed to counteract the charge leakage at the output. For example Figure 6.2 shows this type of realization for the logic equation $x = r_0 + x.\bar{l}_0$.

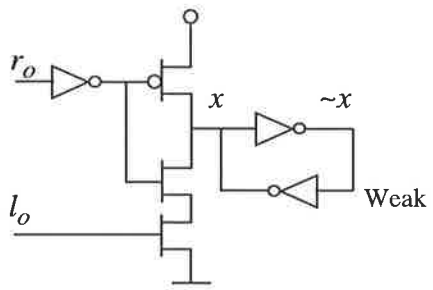


Figure 6.2: A complex gate implementation for the state holding logic operator $x = r_0 + x.\bar{l}_0$

6.4 Tabular method

6.4.1 Deriving new descriptions: STG and STD

Our method is applied starting from stage 3 (see Section 6.2), that is the production rule expansion, after the state assignment has been performed. We consider the wait states, in the output sequence of the handshaking expansion stage, as *normal* signal transitions. Therefore, the compiled program, so far, becomes the corresponding signal transition graph (STG) for which the state assignment problem (if any) has already been solved in the state assignment stage.

Notice that when there is no parallelism in the STG, the resulting STD has the simplest possible form with only $2 \times n$ states where n is the number of signals modelled by the STG, assuming one instance of each transition in the corresponding compiled program.

6.4.2 K-Map generation

When all transitions have a single instance in the STG, each variable, x_i, divides the STD into two disjoint subgraphs[2], $0SG_i$ and $1SG_i$, in which the off-set and on-set vertices of the corresponding variable are located. These subgraphs end with one or more *terminating states,* defined below:

Definition 6.1: In a *terminating* state of SG_i, x_i may change *immediately*; that is x_i is *excited.*

Definition 6.2: A KM_i is a normal Karnaugh map made for the variable x_i, in which x_i itself has been excluded from the map's co-ordinates. ◊

Now based on the resulting strings, two K-maps, $0KM_i$ and $1KM_i$ respectively, are generated for every node, x_i, applying the following algorithm based on [18]:

Algorithm - K-map generation: Every state in $0SG_i$ ($1SG_i$) causes a logic zero (one) to appear in the corresponding box of $0KM_i$ ($1KM_i$) except for the terminating state(s) which corresponds to a logic one (zero).◊

6.4.3 Operator extraction

Since these tables are derived from a well-formed STG, any logic extracted from them necessarily enforces the program order [19][59]. This in fact replaces the guard strengthening procedure in Martin's methodology. The other two steps are also performed within this framework as explained below.

Definition 6.3: The entries zero and one for *0KM* and *1KM,* respectively, we call *compatible* entries, otherwise they are called *incompatible.*

Lemma 6.1: For a variable, x_i, two different entries in identical positions in the two *KMi*'s must be compatible.

2. In the general case of n instances of each transition of x_i, the STD is divided into $2 \times n$ disjoint subgraphs by x_i, so that there are $n \times 0SG_i$'s and $n \times 1SG_i$'s in the STD.

proof- Incompatible different entries in identical locations of the two KM_i's mean that the corresponding signal is not a positive unate function of itself. On the other hand it is proved in [59] that each signal generated through a well-formed STG is a monotone increasing (positive unate) function of itself. Therefore, the two entries must be compatible.

Lemma 6.2: If the two maps $0KM_i$ and $1KM_i$, have two identically located different entries, the corresponding gate, x_i,is state holding. We call such entries *sequential entries*.

Proof- Notice that according to Lemma 6.1 two identically located different entries must be compatible. Now, two identically located different compatible entries for x_i mean that x_i may take both logic values for that particular combination of co-ordinate variables, x_j, $1 \leq j \leq n$, $i \neq j$, where n is the number of variables. In other words, x_i cannot be determined by that combination of variables, so it must be a memory element for that specific combination of the variables.

Lemma 6.3: Two identically located similar entries in the two maps *0KMi* and *1KMi,* represent a combinational state for x_i. We call such entries *combinational entries*.

Proof- Two identically located similar entries for x_i mean that x_i is determined by the particular combination of variables, x_j, $1 \leq j \leq n$, $i \neq j$, where n is the number of variables. Therefore, x_i is a combinational function of the co-ordinate variables for that particular K-map entry.◊

Having derived the proper $2 \times n$ K-maps for n nodes and considering the above propositions, operator extraction can be performed for each variable according to the following algorithm:

Algorithm - Operator extraction:

- 1: Check the two K-maps to see if there are any two identically located (different) compatible entries. Such a pair of entries identifies a state holding gate according to Lemma 6.2, otherwise the gate can be purely combinational, according to Lemma 6.3. Note that if there are two identically located different incompatible entries, then the maps are not correctly constructed.
- 2: For the combinational entries, group all zeros together with as many don't cares as required in as few and large groups as possible. Each resulting cube implies one path from the output to *GND*.

- 3: Similarly, group all combinational entries that are one together with as many don't cares as required in as few and large groups as possible. Each resulting cube implies one path from the output to V_{dd}.

This is because of the combinational nature of the circuit under this group of literals forcing the output to necessarily be pulled down (up) through a pass transistor path connecting the output to GND (Vdd) in case 2 (case 3), regardless of the previous history of inputs.

- 4: All remaining zeroes and ones with a corresponding don't care in the other K-map can be considered either sequential or combinational, whichever simplifies the final design; that is if the gate has already been determined as state holding, disregard such entries, otherwise check which choice incurs a lower cost.

Notice that these so called isolated zero's and one's may belong to either group, sequential or combinational entries[3], depending on how the corresponding don't care entry in the other K-Map is assigned a logic value by the designer.

Corollary 6.1: The implementation of isolated on-set or off-set vertices (discussed in number 4 above) as a combinational entry does necessarily entail some cost.

Proof: These vertices may obviously not be merged with other cubes, as they should have already been merged (in the second and third stages of the algorithm) if it had been feasible. Therefore, at least one new cube must be considered which incurs some extra circuitry.

6.5 Examples

In this section we demonstrate the convenience of our tabular method through three examples.

Example 6.1: Consider the output of the handshaking expansion stage for the L/R element [47] in which the CSC violation has been solved by introducing a new variable, x, as shown in Figure 6.3-a. The compiled program sequence in terms of the STG notation and the corresponding STD are shown in Figure 6.3-b and Figure 6.3-c, respectively.

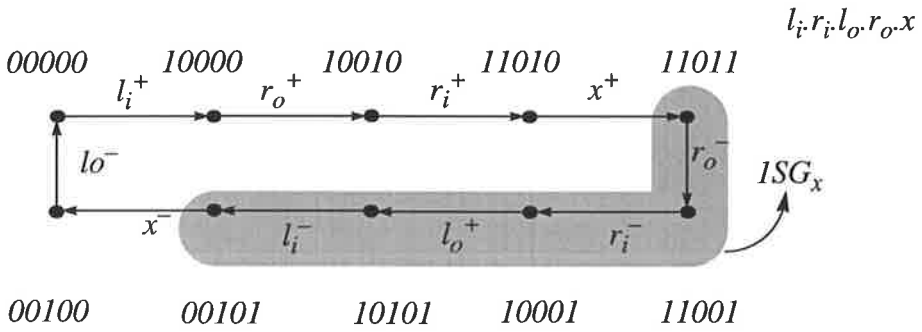
3. In other words they constitute a third group we call *don't care* group, as they may be assumed either sequential or combinational.

$*[[l_i]\ ;\ r_o\uparrow\ ;\ [r_i]\ ;\ x\uparrow\ ;\ [x]\ ;\ r_o\downarrow\ ;\ [\sim r_i]\ ;\ l_o\uparrow\ ;\ [\sim l_i]\ ;\ x\downarrow\ ;\ [\sim x]\ ;\ lo\downarrow]$

(a)

$$\begin{array}{ccccccccc} \underline{l_i}^+ & \rightarrow & r_o{}^+ & \rightarrow & \underline{r_i}^+ & \rightarrow & x^+ & \rightarrow & r_o{}^- \\ \uparrow & & & & & & & & \downarrow \\ l_o{}^- & \leftarrow & x^- & \leftarrow & \underline{l_i}^- & \leftarrow & l_o{}^+ & \leftarrow & \underline{r_i}^- \end{array}$$

(b)



(c)

Figure 6.3: (a) Output sequence, (b) STG and (c) STD for Example 6.1.

In this example node x is the only state holding node. It has been implemented in three different forms as shown in Figure 6.4[4], depending on how the don't care entries have been utilized. In Figure 6.4-a some don't cares have been shared between 1's and 0's to simplify the resulting circuit as much as possible. This approach, however, carries the risk of creating undefined logic levels at, say, the power on time, as the output node might be pulled both up and down by the unpredicted input vector $l_i.r_i = 01$ if the input signals l_i and r_i are not preset properly.

In order to resolve this problem the pull down and pull up cubes have been selected to be mutually exclusive in Figure 6.4-b, at the cost of one extra transistor and a larger fan-out for the upper inverter.

The same K-maps, on the other hand, may be translated into the well-known but more expensive Muller C-element by using fewer don't cares as shown in Figure 6.4-c. This is what has been considered in [47], as it has been based on the use of a standard library of logic operators to build up the asynchronous networks. Notice that there are two options here based on the logic value assigned to the don't care entries.

Example 6.2: In this example the newly introduced variable x in Example 6.1, is placed in the output sequence in a different position and in parallel with r_i^+ and l_i^-, as shown in Figure 6.5-a. The corresponding STG and STD are shown in Figure 6.5-b and Figure 6.5-c, respectively[5], where all 3 nodes are now of state holding type as shown in Figure 6.6.

Example 6.3: In this example the switch B, a more complex module and one of the two modules of a speed independent circuit for distributed mutual exclusion [49], is studied. The compiled program (taken from [49]) and the corresponding STG are shown in Figure 6.7.

Having removed the non-input choices [18] [6]from Figure 6.7-b the reshaped STG of Figure 6.7-c is obtained. Figure 6.8 shows the partial STG and the corresponding STD, K-maps and the circuit diagram for node b. According to the operator extraction

4. Generally speaking the inverters at some inputs of these operators violate the atomic assumption of the corresponding logic gates, which may result in hazards [77]. The situation becomes worse if two level (AND-OR) logic is used to implement the operators even under the well-behaved environment assumption. See [59][77] for more details.

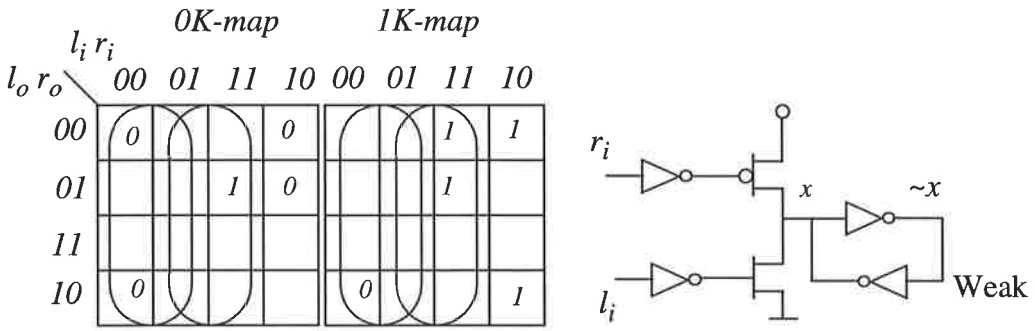
algorithm, b is the only sequential signal in this example. However, if the don't care entries were not properly assigned, then the combinational signals might also need output memory. As an example, if the K-map entries of signal s_o are grouped as shown in Figure 6.8-c, then s_o has in fact been assumed as a state holding signal; that is the don't care entry in the right table corresponding to the single zero in the other table has been assigned a one, otherwise these two entries should have been grouped and allocated a pass transistor path as well. This in fact is the *symmetrization stage* in Martin's methodology which is performed clearly here.

6.6 Conclusion

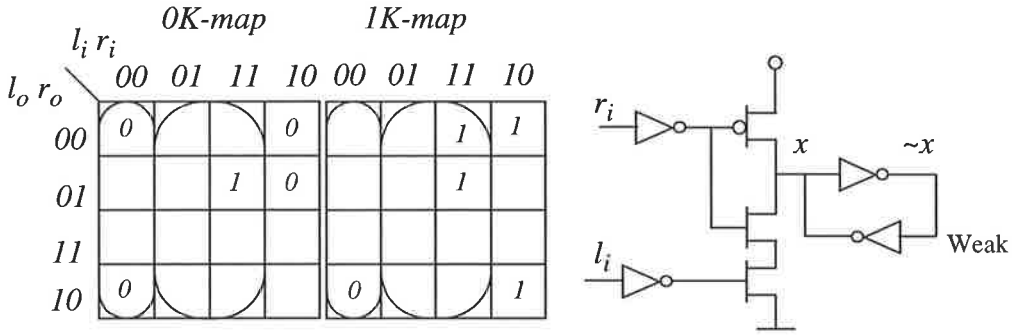
In this chapter we have presented a tabular method to compile the output of the handshaking expansion stage of Martin's asynchronous circuit design methodology into the CMOS combinational/state holding complex gates. As the examples demonstrate, our method is flexible, straightforward and descriptive. The method is STD based so, in common with several other asynchronous design methodologies, the size of STD grows exponentially for highly concurrent designs. Also notice that the major problem with this type of circuit implementation is the limited number of stacked transistors allowed in today's technologies.

---

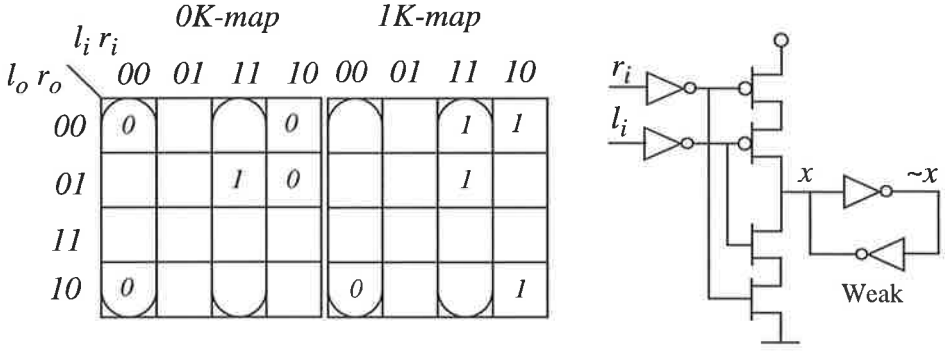
5. In this STG, x cannot immediately be followed by r_i nor by l_i, as the environment is supposed to monitor only r_o and l_o. Notice that having satisfied even this requirement, x may not take an arbitrary position in the STG, or the CSC property may still be violated. Also notice that in this implementation the mutual exclusion between the on-set and off-set cubes has been observed.
6. This has been performed by properly splitting the choice place *p1* in Figure 6.7-b into two places *p2* and *p3* in Figure 6.7-c. Notice that in *p1* the non-input signal b may have both logic values, while in *p2* and *p3* it may take only one logic value, that is *1* and *0*, respectively.



(a)



(b)

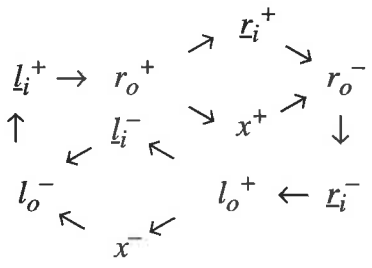


(c)

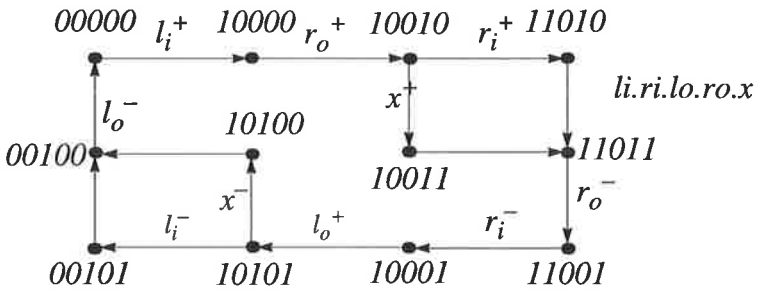
Figure 6.4: K-Maps and logic circuits realising state holding node x, assuming shared use (a), disjoint use (b) and not full use (c) of don't cares in Example 6.1.

$*[[l_i]\ ;\ r_o\uparrow\ ;\ [r_i]\ ,\ x\uparrow\ ;\ [x]\ ;\ r_o\downarrow\ ;\ [\sim r_i]\ ;\ l_o\uparrow\ ;\ [\sim l_i]\ ,\ x\downarrow\ ;\ [\sim x]\ \ ;\ \ l_o\downarrow]$

(a)

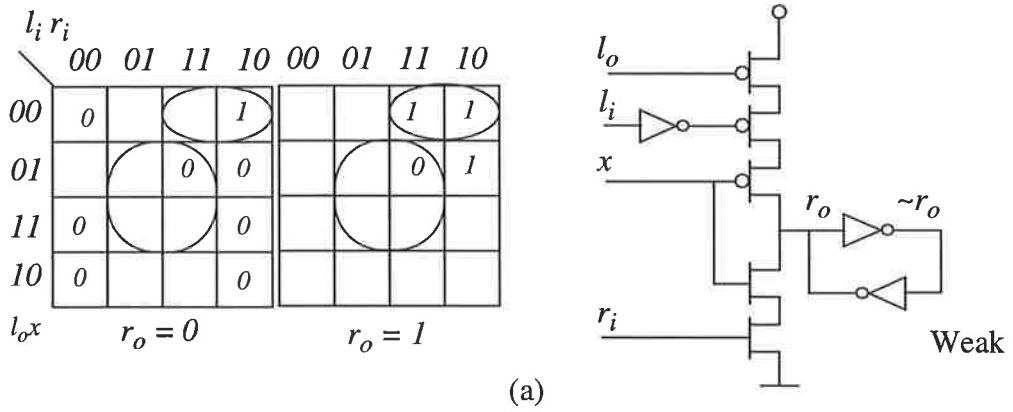


(b)

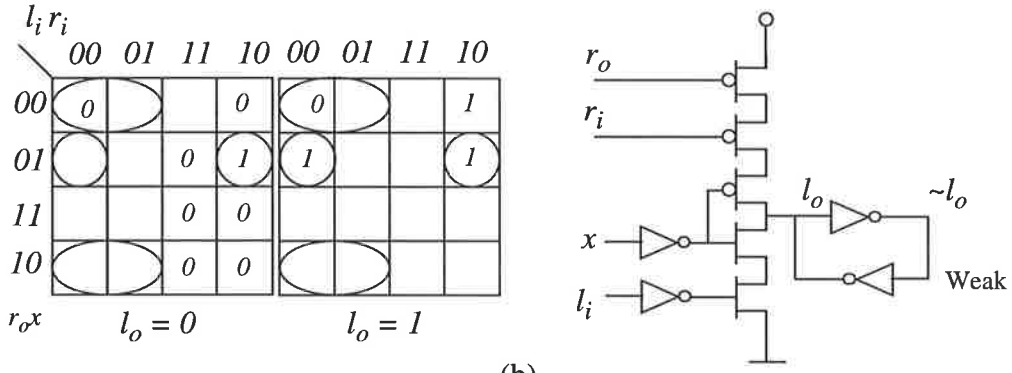


(c)

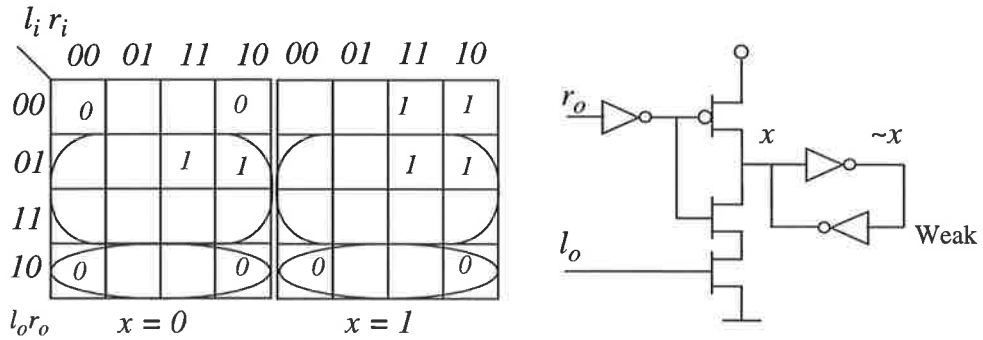
Figure 6.5: Output sequence (a), STG (b), and STD (c) for Example 6.2.



(a)



(b)



(c)

Figure 6.6: K-map pairs and corresponding circuit realizations for Example 6.2.

$$
\begin{aligned}
B \equiv * [[\ & q_i \wedge b \ \rightarrow\ q_o\uparrow\ ;\ [\neg q_i]\ ;\ q_o\downarrow \\
& \mid q_i \wedge \neg b \ \rightarrow r_o\uparrow\ ;\ [r_i]\ ;\ r_o\downarrow\ ;\ [\neg r_i]\ ;\ q_o\uparrow\ ;\ [\neg q_i]\ ;\ q_o\downarrow \\
& \mid s_i \ \rightarrow b\uparrow\ ;\ s_o\uparrow\ ;\ [\neg s_i]\ ;\ s_o\downarrow \\
& \mid t_i \ \rightarrow b\downarrow\ ;\ t_o\uparrow\ ;\ [\neg t_i]\ ;\ t_o\downarrow \\
&]]. \qquad\qquad (a)
\end{aligned}
$$

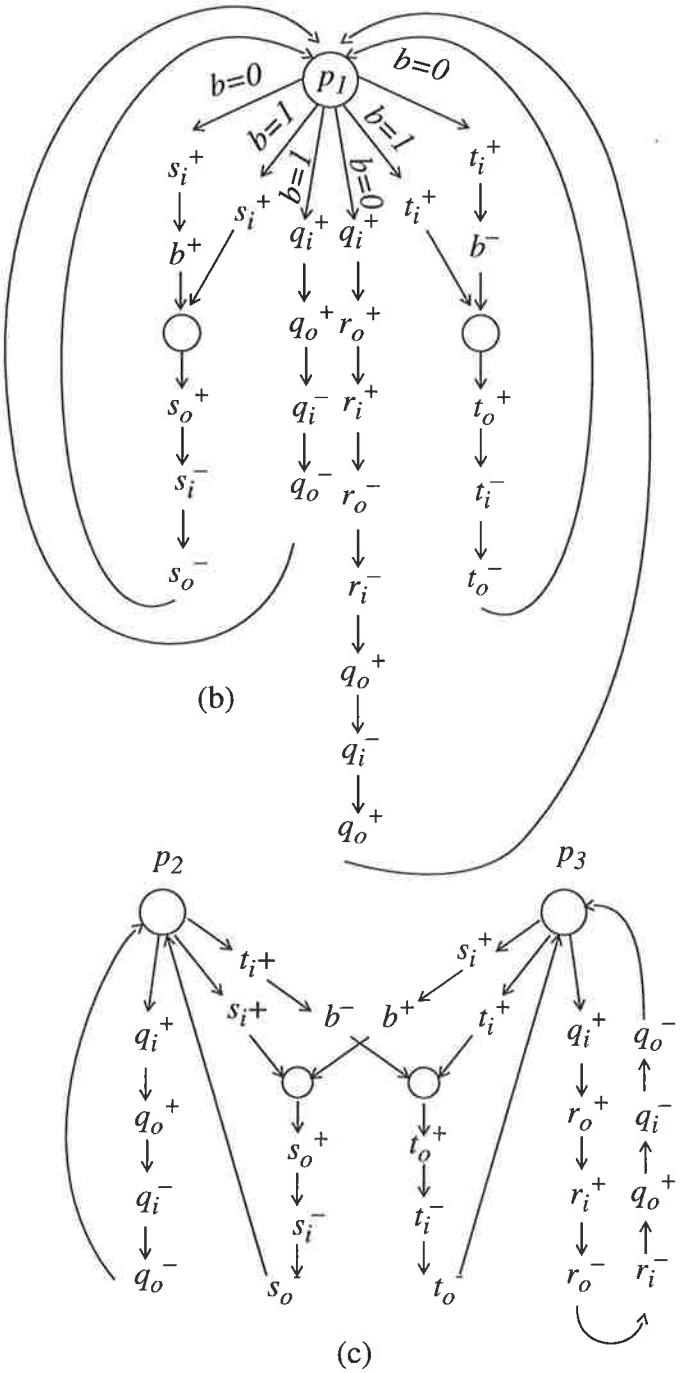
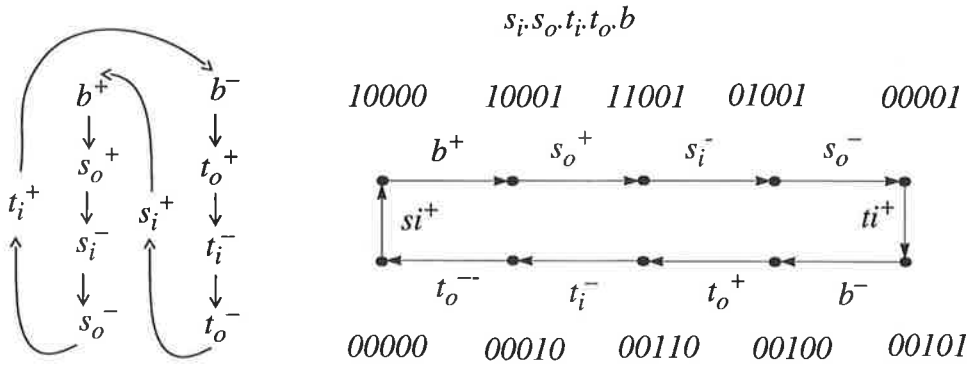
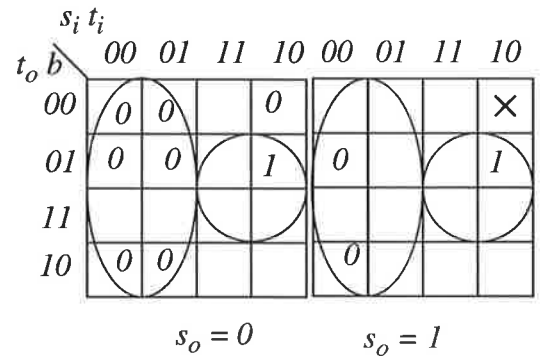


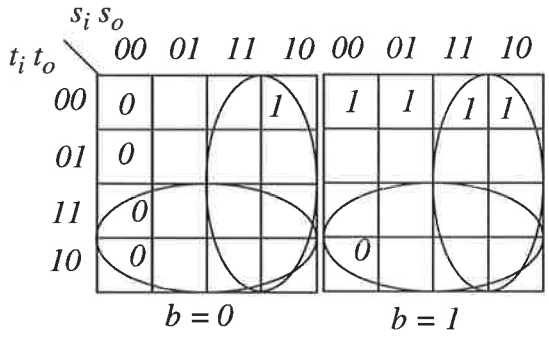
Figure 6.7: (a) Compiled program, (b) the corresponding STG, and (c) the reshaped STG for Example 6.3.



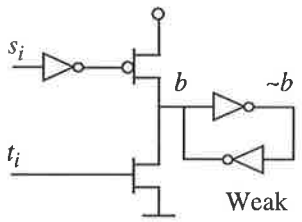
(a) (b)



(c)



(d)



(e)

Figure 6.8: (a) Partial STG, (b) the corresponding STD, (d) K-maps and (e) circuit diagram (an RS-Flip-Flop) for variable *b* (c), and K-maps for variable s_o (c) in Example 6.3.

Chapter 7

Conclusion and Further Work

This thesis considered asynchronous design techniques in two aspects. Identifying unlikely hazards in a bounded delay model, and preventing hazards through the introduction of some redundancy in the unbounded delay model. Following this technique, we introduced an alternative for the last two stages of Martin's four stage asynchronous design methodology.

1- Unlikely hazards were identified under the inertial gate delay model. We studied the design of two level hazard free asynchronous circuits from signal transition graphs under the *inertial delay model* and the well-behaved environment. Different sources of dynamic hazards were first investigated, and then a major group of this class, that is *multiple input high to low real dynamic hazards*, were identified which we proved could not occur

under the inertial gate delay model in two level SOP logic circuits. We then relaxed the zero wire delay assumption and determined an upper bound for the delay of critical wires and introduced the notion of *virtual isochronic forks*, under which hazard free operation is still guaranteed.

We next proved that under a reasonable delay constraint all simple gate based SOP circuits are free of multiple input change 0-1 dynamic logic hazards when the inertial delay model can be assumed. These two results relieve the designer of all dynamic hazard problems in many cases. Static hazards, on the other hand, are not relaxed under the inertial delay model and hence they have to be prevented through the classical method of introducing some appropriate redundancy to the cover of the function.

We uncovered a fallacy and showed that even the well-behaved environment assumption might result in delay hazards in non-atomic two level gate based implementations, although delay hazards are significantly reduced by the well-behavedness assumption of the environment. In order to eliminate this type of hazard a different cover has to be found with no hazardous behaviour. We showed this possibility through an example.

In the direction of uncovering unlikely hazards different types of *delay hazards* in STG based two level asynchronous circuits were addressed. We first identified a major region in STGs which are immune to hazards and hence showed that static 0-delay hazards have no chance of occurring under the pure delay model and the isochronic fork assumption. Furthermore, according to our hazard classification low to high dynamic hazards are no longer defined. We then showed that static 1-delay hazards may be avoided if there are sufficient sequences of output overlapping p-terms in the hazard region. We further showed that these sequences automatically eliminate all dynamic high to low delay hazards as well, under the inertial gate delay model and the isochronic fork assumption.

2- In our attempt at hazard prevention by introducing redundancy we first studied different types of interconnection forks in STG based speed independent circuits and concluded that inverters on the input of some complex AND-OR-NOT gates may cause delay hazards in many cases where isochronic forks are inevitable. With this motivation safe cells were introduced and shown to be robust against delay hazards. Then, some sufficient conditions were identified to determine those nodes in a network which need to be realized as safe cells. This method was compared with the two phase RS-implementation

technique in an example and it was concluded that safe cells might result in a significant chip area gain.

Safe cells are so designed that they acknowledge both the *secondary* and *primary* input transitions. Therefore, no matter how slow the inverters are, the correct order of transitions are always guaranteed by safe cells.

We then relaxed the so called pessimistic assumption of unboundedness of delays in inverters and studied delay hazards in complex gate based circuits and identified a subclass of delay hazards in different nodes of the network.

Finally combinational hazards in complex AND-OR-NOT gates were discussed and it was shown that this logic family is more immune to those hazards than the two level AND-OR logic.

Following the above redundancy based technique, we presented a tabular method to compile the output of the handshaking expansion stage of Martin's asynchronous circuit design methodology into the CMOS combinational/state holding complex gates. As the examples demonstrate, our method is flexible, straightforward and descriptive. The method is STD based so, in common with several other asynchronous design methodologies, the size of STD grows exponentially for highly concurrent designs. Also notice that the major problem with this type of circuit implementation is the limited number of stacked transistors allowed in today's technologies.

Notice that the outcome of both approaches may end up with some problematic inverters pointed out above. Therefore, in order to resolve this problem safe cells as general purpose gates may be utilized.

In summary we enhanced in this thesis the understanding of the role of gate and wire delay models in generating different types of hazards in asynchronous circuits. These achievements will affect the optimization techniques in the corresponding CAD tools. Then, based on the principle of acknowledgement we introduced safe sells in an attempt at removing delay hazards and hence implementing true speed-independent circuits.

Future research work may be directed as follows:

In this thesis unlikely hazards were investigated under the inertial delay model in two level logic circuits. This study should be extended to multi level logic as well. This will probably help to choose an optimum decomposition.

Unlikely hazards identified as dynamic logic and delay hazards *do* degrade noise immunity of logic gates. This degradation is in direct relation to rise and fall times of the relevant signal transitions, and the threshold voltages of the corresponding transistors. So that too slow signal transitions can eventually lead to hazards. Ratioed logic such as n-channel MESFET GaAs technology in which rise times significantly dominate the relevant fall times can give rise to this concern. As a part of the future work noise immunity degradation caused by unlikely hazards (uncovered in this thesis) should be investigated in different technologies.

Delay hazards under the well-behaved environment and the mechanism of their generation were addressed in this thesis. However, a general solution to eliminate the delay hazard problem under the well-behaved environment has not been investigated yet. We expect the solution to be concentrated first on finding a different cover, as delay hazards under the well-behaved environment assumption are manifested as newly introduced function hazards. Although this method will not resolve all delay hazards it should assist the designer with different trade offs, such as STG reshaping, and to find an optimum cover to avoid more likely hazards. If on the other hand a speed-independent solution is sought then unresolved delay hazards may be treated as a special case of general simple gate based speed independent design methodologies, in which state holding gates and some combinational type of redundancy are normally used.

We introduced safe cells as hazard free building blocks and developed a theorem identifying some sufficient conditions to implement a node in the circuit with a safe cell. More work should be performed to identify the corresponding necessary conditions. This would result in an optimum circuit with the minimum possible number of safe cells. The problem becomes more complex if a bounded delay is assumed for inverters. Then for every delay a different arrangement for safe cells would result.

Bibliography

[1] F. Aghdasi, M. Bolton, "Self-clocked asynchronous state machine design with PAL22IP6", *Microprocessors and Microsystems*, February 1991.

[2] D. B. Armstrong, A. D. Friedmen, P. R. Menon, "Design of Asynchronous Circuits Assuming Unbounded Gate Delays", *IEEE Transactions on Computers,* Vol. C-18, No. 12, pp. 1110-1120, December 1969.

[3] P. A. Beerel and T. H-Y. Meng, "Automatic Gate-Level Synthesis of Speed Independent Circuits", *in Proceedings of the International Conference on Computer-Aided Design*, November 1992.

[4] P. A. Beerel and T. H-Y. Meng, "Gate-Level Synthesis of Speed Independent Asynchronous Control Circuits", *in Proceedings of the ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems* (TAU), March 1992.

[5] P. A. Beerel and T. H-Y. Meng, "Semimodularity and Testablity of Speed-Independent Circuits", *Integration, the VLSI Jourrnal,* 13(3), pp. 301-322, September 1992.

[6] P. A. Beerel, J. Burch and T. H-Y. Meng, "Efficient Verification of Determinate Speed-Independent Circuits", *in Proceedings of the International Conference on Computer-Aided Design*, 1993.

[7] P. A. Beerel, CAD Tools for the Synthesis, Verification, and Testability of Robust Asynchronous Circuits", Ph.D thesis, Stanford University, August 1994.

[8] J. G. Bredeson, "On Multiple Input Change Hazard-Free Combinatorial Switching Circuits Without Feedback", *in 14th Annual Symposium on Switching and Automata Theory,* pp 56-63, October 1973.

[9] J. G. Bredeson, "Synthesis of multiple input change hazard free combinational switching circuits without feedback", *International Journal of Electronics*, 39 (6), pp. 615-624, 1975.

[10] J. G. Bredeson and P. T. Hulina, "Elimination of static and dynamic hazards for multiple input changes in combinational switching circuits", *Information and*

Control, 20, pp. 114-224, 1972.

[11] B. Bernhardt, et. al., "Complementary GaAs (CGaAs^{TM}): A High Performance BiCMOS Alternative", *1995 IEEE GaAs IC Symposium Technical Digest*, pp. 18-21.

[12] E. Brunvand and R. F. Sproull, "Translating Concurrent Programs into Delay - Insensitive Circuits", *in Proceedings of the International Conference on Computer-Aided Design*, pp. 262-265, November 1989.

[13] S. M. Burns and A. J. Martin, "Syntax-directed Translation of Concurrent Programs into Self-timed Circuits", *in Proceedings of the fifth MIT Conference on Advanced Research in VLSI*, March 1988.

[14] S. M. Burns, "General Conditions for the Decomposition of State Holding Elements", *in Proceedings of the Second International Symposium on Advanced Research in Asynchronous Circuits and Systems* (Async'96), pp.48-57, Aizu, Japan, March 1996.

[15] T. J. Chaney and C. E. Molnar, "Anomalous Behavior of Synchronizers and Arbiters", *IEEE Transactions on Computers,* Vol. C-22, pp. 421-422, April 1973.

[16] T. J. Chaney, "Measured Flip-Flop Responses to Marginal Triggering", *IEEE Transactions on Computers*, Vol. C-32, pp. 1207-1209, Dec. 1983.

[17] T. A. Chu, "Synthesis of Self-timed Control Circuits from Graphs: an example", *in Proceedings of the International Conference on Computer Design*, pp. 565-571, 1986.

[18] T. A. Chu, "Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications", *Ph.D thesis*, MIT, June 1987.

[19] H. Y. H. Chuang, S. Das, "Synthesis of Multiple-Input Change Asynchronous Machines Using Controlled Excitation and Flip-Flops", *IEEE Transactions on Computers,* Vol. C-22, No. 12, 1103-1109, Dec. 1973.

[20] W. A. Clark, "Macromodular Computer Systems", *in Proceedings of the Spring Joint Computer Conference,* AFIPS, 1967.

[21] D. L. Dill, "Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits", *in Proceedings of the fifth MIT Conference on Advanced*

Research in VLSI, March 1988.

[22] B. J. Benschneider, et al., "A 300MHz 64-b Quad-Issue CMOS RISC Microprocessor", *IEEE Journal of Solid-State Circuits,* Vol. 30, No. 11, November 1995.

[23] A. Doerr and K. Levasseur "Applied Discrete Structures for Computer Science", *Science Research Associates*, 1985.

[24] K. Eshraghian. "Fundamentals of Ultra High Speed Systems", *Prentice Hall*, to be published.

[25] A. D. Friedman and P.R. Menon, "Synthesis of Asynchronous Sequential Circuits with Multiple-Input Changes", *IEEE Transactions on Computers*, Vol. C-17, No. 6, June 1968.

[26] B. Hazeltine, "Encoding of Asynchronous Sequential Circuits", *IEEE Transactions on Electronic Computers*, Vol. EC-14, pp 727-729, October 1965.

[27] C. A. R. Hoare, "Communicating Sequential Processes", *Communications of ACM* 21, 8, PP. 125-130, August 1987.

[28] D. H. K. Hoe and A. T. Salama, "GaAs Trickle Transition Dynamic Logic", *IEEE Journal of Solid State Circuits*, Vol.26, No. 10, Oct. 1991.

[29] L. A. Hollaar, "Direct implementation of asynchronous control units", *IEEE Transactions on Computers,* Vol. C-31, No. 12, pp. 1133-1141, Dec. 1982.

[30] S. Hauck, "Asynchronous Design Methodologies: An Overview", *Proceedings of the IEEE*, pp. 69-93, Vol. 83, No.1, Jan. 1995.

[31] L.G. Heller, W. R. Griffin, J. W. Davis, N. G. Thoma, "Cascode Voltage Switch Logic: A Differential CMOS Logic Family", *in Proceedings of 1984 IEEE International Solid State Circuits Conference,* IEEE Press, pp. 16-17, 1984.

[32] D. A. Huffman. "The Synthesis of Sequential Switching Circuits." *J. Franklin Institute,* 257: 161-190, 275-303, March 1954.

[33] M. Kishinevsky, A. Kondratyev, A. Taubin, V. Varshavsky, "Concurrent Hardware, The Theory and Practice of Self-Timed Design", John Wiley & Sons, 1994.

[34] L. Kleeman and A. Cantoni, "On the Unavoidabilty of Metastable Behaviour in Digital Systems", *IEEE Transactions on Computers*, Vol. C-36 No. 1, pp. 109-112, 1987.

[35] L. Kleeman and A. Cantoni, "Metastable Behaviour in Digital Systems", *IEEE Design and Test of Computers,* Vol. 4 No. 6, pp. 4-19, 1987.

[36] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, A. Yakovlev, "Basic Gate Implementation of Speed-Independent Circuits", *in Proceedings of ACM/ IEEE Design Automation Conference*, 1994.

[37] A. Kondratyev, J. Cortadella, M. Kishinevsky, L. Lavagno, A. Yakovlev, "Technology Mapping for Speed-Independent Circuits: Decomposition and Resynthesis", *in Proceedings of the Third International Symposium on Advanced Research in Asynchronous Circuits and Systems (Async'97)*, pp. 240-253, Eindhoven, The Netherlands, April 1997.

[38] M. Ladd and W. P. Birmingham, "Synthesis of multiple-input change asynchronous finite state machines" *in Proceedings of the Design Automation Conference*, 309-314, June 1991.

[39] L. Lavagno, C. W. Moon, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Solving the state assignment problem for signal transition graphs", *in Proceedings of Design Automation Conference*, pp. 303-308, 1991.

[40] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli, "Algorithms for Synthesis of Hazard free Asynchronous Circuits", *in Proceedings of Design Automation Conference*, pp. 568-572, June 1992.

[41] L. Lavagno, and A. Sangiovanni-Vincentelli, Linear Programming for Optimum Hazard Eliminaiton in Asynchronous Circuits", *in Proceedings of International Conference on Computer Design,* pp. 275-278, 1992.

[42] L. Lavagno, "Synthesis and Testing of Bounded Wire Delay Asynchronous Circuits from Signal Transition Graphs", *Ph.D Thesis*, University of California at Berkeley, 1992.

[43] Trevor W. S. Lee and Mark R. Greenstreet and Carl-Johan Seger, "Automatic Verification of Asynchronous Circuits", *IEEEDT*, Vol. 12, No. 1, pp. 24--31, 1995.

[44] K. J. Lin and C. S. Lin, "Direct Synthesis of Hazard-free Asynchronous Circuits from STGs based on Lock Relation and MG-Decomposition Approach", *in Proceedings of European Conference on Design Automation,* pp. 178-183, 1994.

[45] C. N. Liu, "A State Variable Assignment Method for Asynchronous Sequential Switching Circuits", *Journal of ACM*, Vol. 10, pp 209-216, April 1963.

[46] G. Mago, "Realization Methods for Asynchronous Sequential Circuits", *IEEE Transaction on Computers,* Vol. C-20, No. 3, March 1971.

[47] A. J. Martin, "Programming in VLSI, From Communicating Processes to Delay-Insensitive Circuits", Chapter one, pp. 1-64, *Developments in Concurrency and Communication,* Addison-Wesly 1990.

[48] A. J. Martin. "The Limitations to Delay Insensitivity in Asynchronous Circuits" *in William J. Dally, editor, Proceeding of Advanced Research in VLSI,* pages 263-278, MIT press, 1990.

[49] A. J. Martin, "The Design of a Self-timed Circuit for a Distributed Mutual Exclusion", *in Proceedings of Chapel Hill Conference on VLSI*, pp 245-260, 1985.

[50] A. J. Martin, "Compiling Communicating Processes into Delay Insensitive VLSI Circuits", *Distributed Computing*, 1, pp. 226-234, 1986.

[51] A. J. Martin, "Formal Program Transformations for VLSI Circuit Synthesis", *UT Year of Programming Institute on Formal Developments of Programs and Proofs*, ed. E. W. Dijkstra, pp. 59-80, Addison-Wesley, 1989.

[52] A. J. Martin, S. M. Burns, T. K. Lee, D. Borkovic, P. J. Hazewindus, "The Design of an Asynchronous Microprocessor", *Decennial Caltech Conference on VLSI,* C. L. Seitz, ed., pp. 351-373, MIT Press, Cambridge, Mass., 1989.

[53] A. J. Martin, "Synthesis of Asynchronous VLSI Circuits", *Formal Methods for VLSI Design*, J. Staunstrup (editor), pp. 237-283, Elsevier Science Publishers B. V. (North-Holland) 1990.

[54] Mc-Cluskey, "Fundamental mode and pulse mode sequential Circuits", *in Proceedings IFIP congress on Information Processing*, Amsterdam, North-Holland Publishing Co., pp. 725-730, 1963.

[55] T. H.-Y. Meng, R. W. Brodersen and D. G. Messerschmitt, "Automatic Synthesis

of Asynchronous Circuits from High Level Specifications", *IEEE Transactions on Computer-Aided Design*, 8(11), pp. 1185-1205, November 1989.

[56] T. H.-Y Meng, "Asynchronous Design for Digital Signal Processing Architectures", *PhD thesis,* U.C. Berkeley, November 1988.

[57] R. E. Miller, "Switching Theory", Volume 2, Chapter 10, *John Wiley and Sons*, 1965.

[58] C. E. Molnar, T. P. Fang and F. U. Rosenberger, "Synthesis of Delay Insensitive Modules", *in Proceedings of the 1985 Chapel Hill Conference on VLSI*, Computer Science Press, Princeton, USA, pp. 67-86, 1985.

[59] C. W. Moon, "Synthesis and Verification of Asynchronous Circuits from Graphical Specifications", *Ph.D Thesis*, University of California at Berkeley, 1992.

[60] C. W. Moon, P. R. Stephen and R. K. Brayton, "Synthesis of hazard free synchronous circuits from graphical specifications", *in Proceedings of International Conference on Computer Aided Design,* pp. 322-325, November 1991.

[61] T. Murata, "Petri nets: Properties, analysis and applications", *Proceedings of IEEE*, Vol. 77, No. 4, pp. 541-580, 1989.

[62] C. Myers and T. H.-Y. Meng, "Synthesis of Timed Asynchronous Circuits", *in Proceedings of International Conference on Computer Design,* pp. 279-282, 1992.

[63] C. J. Myers and T. G. Rokicki and T. H.-Y. Meng, "Automatic Synthesis of Gate-level Timed Circuits with Choice", *in Proceedings of the 16th Conference on Advanced Research in VLSI"*, pp. 42-58, 1995.

[64] S. M. Nowick and D. L. Dill, "Exact Two-Level Minimization of Hazard Free Logic with Multiple-Input Changes", *in Proceedings of the International Conference on Computer-Aided Design*, pp. 626-630, November 1992.

[65] S. M. Nowick and D. L. Dill, "Synthesis of Asynchronous State Machines using a Local Clock", *in Proceedings of the International Conference on Computer Design*, pp. 192-197, 1991.

[66] S. M. Nowick and D. L. Dill, "Automatic synthesis of Locally Clocked Asynchronous State Machines" *in Proceedings of the International Conference on*

Computer-Aided Design, pp. 318-321, 1991.

[67] S. M. Nowick, K. Y. Yun and D. L. Dill, "Practical Asynchronous Controller Design" *in Proceedings of the International Conference on Computer Design*, pp. 341-345, November 1992.

[68] M. C. Paull and S. H. Unger, "Minimizing the Number of States in Incompletely Specified Sequential Switching Circuits", *IRE Transactions on Electronic Computers*, Vol. EC-8, pp356-367, September 1959.

[69] A. Peeters and K. van Berkel, "Single-Rail Handshake Circuits", *in Proceedings of the Second Working Conference on Asynchronous Design Methodologies*, pp. 53-62, May 1995.

[70] F. U. Rosenberger, C. E. Molnar, T. J. Chaney, T. P. Fang, "Q-Modules: Internally Clocked Delay-Insensitive Modules", *IEEE Transactions on Computers*, Vol. 37, No. 9, pp 1005-1018, Sept. 1988.

[71] L. Ya. Rosenblum, A. V. Yakovlev, "Signal Graphs: from Self-Timed of Timed ones", *in Proceedings of International Workshop on Timed Petri Nets*, pp. 199-206, Torino, Italy, 1985.

[72] G. Saucier, "Encoding of Asynchronous Sequential Circuits", *IEEE Transactions on Electronic Computers*, Vol. EC-16, No. 3, pp 365-369, June 1967.

[73] C. L. Seitz, "System Timing", in C. A. Mead and L. A. Conway, "Introduction to VLSI Systems", Chapter 7, *Addison-Wesley*, 1980.

[74] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephen, R. K. Brayton, and A. Sangiovanni-Vincentelli. "SIS: A system for sequential circuits synthesis", *Technical Report UCB/ERL M92/41*, U.C. Berkeley, May 1992.

[75] I. E. Sutherland, "Micropipelines", *Communications of the ACM,* Vol. 32, No. 6, pp. 720-738, June 1989.

[76] N. Tabrizi, M. J. Liebelt and K. Eshraghian "Dynamic Hazards and Speed Independent Delay Model", *in Proceedings of the Second International Symposium on Advanced Research in Asynchronous Circuits and Systems (Async'96)*, pp. 94-103, Aizu, Japan, March 1996.

[77] N. Tabrizi, M. J. Liebelt and K. Eshraghian, "Delay Hazards in Complex Gate Based Speed Independent VLSI Circuits", *in Proceedings of the Sixth Great Lakes Symposium on VLSI (GLS-VLSI'96)*, pp 266-271, Iowa, USA, March 1996.

[78] N. Tabrizi, M. J. Liebelt and K. Eshraghian, "A Tabular Method for Guard Strengthening, Symmetrization and Operator Reduction for Martin's Asynchronous Design Methodology", accepted for publication in *the IEEE Transactions on Computers*, February 1997.

[79] N. Tabrizi, M. J. Liebelt and K. Eshraghian, "Delay Hazards in Two Level Asynchronous VLSI Circuits Synthesised from Signal Transition Graphs", accepted for presentation at *the 14th Australian Microelectronics Conference, Micro'97*, Melbourne, Australia.

[80] J. H. Tracy, "Internal state assignment for Asynchronous Sequential Machines", *IEEE Transactions on Electronic Computers*, EC-15(4), pp 551-560, August 1966.

[81] S. H. Unger, "A Row Assignment for Delay-free Realization of Flow Tables without Essential Hazards", *IEEE Transactions on Computers*, Vol. C-17, No. 2,pp 146-158, February 1968.

[82] S. H. Unger, "Asynchronous Sequential Switching Circuits with Unrestricted Input Changes", *IEEE Transactions on Computers*, Vol. C-20, No. 12, December 1971.

[83] S. H. Unger, "Asynchronous Sequential Switching Circuits", *Wiley-Interscience*, 1969.

[84] P. Vanbekbergen, B. Lin, G. Goossens, and H. de Man, " A Generalized State Assignment Theory for Transformations on Signal Transitions Graphs", *in Proceedings of International Conference on Computer Design,* pp. 112-117, November 1992.

[85] K. van Berkel and M. Rem, "VLSI Programming of Asynchronous Circuits for Low Power", *in G. Birtwistle and A. Davis editors, Asynchronous Digital Circuit Design,* workshops in computing, springer, pp. 152-210, 1995.

[86] K. van Berkel, "Beware the Isochronic Fork", *Integration, the VLSI journal*, 13 (2):103-128, June 1992.

[87] K. van Berkel, J Kessels, M. Roncken, R. Saeijs, F. Schalij, "The VLSI Programming Language Tangram and its Translation into Handshake Circuits", *in Proceedings of European Conference on Design Automation (EDAC)*, pp. 384-389, 1991.

[88] K. van Berkel, "Handshake Circuits. An Asynchronous Architecture for VLSI Programming", *International Series on Parallel Computation*, vol. 5, Cambridge University Press, 1993.

[89] K. van Berkel, F. Huberts and A. Peeters, "Stretching Quasi Delay Insensitivity by Means of Extended Isochronic Forks", *in Proceedings of the Second Working Conference on Asynchronous Design Methodologies*, pp. 99-106, May 1995.

[90] K. van Berkel and Arjan Bink, "Single-Track Handshaking Signaling with Application to Micropipelines and Handshake Circuits", *in Proceedings of the Second International Symposium on Advanced Research in Asynchronous Circuits and Systems (Async'96)*, pp. 122-133, Aizu, Japan, March 1996.

[91] V. Varshavsky, M. Kishinevsky, V. B. Marakhovsky, V. A. Peschansky, L.Y. Rosenblum, A. R. Taubin, B. S. Tzirlin, Edited by V. Varshavsky, "Self-Timed Control of Concurrent Processes", *Kluwer Academic Publishers*, 1990. (Russian edition 1986)

[92] N. H. E. Weste and K. Eshraghian, "Principles of CMOS VLSI Design", *Addison-Wesley*, 1993.

[93] S. F. Wu and D. Fisher, "Automating the Design of Asynchronous Sequential Logic Circuits", *IEEE Journal of Solid State Circuits*, Vol. 26, No. 3, March 1991.

[94] A. V. Yakovlev. "On Limitations and Extensions of STG Model for Designing Asynchronous Control Circuits", *in Proceedings of International Conference on Computer Design,* pp. 396-400, 1992.

[95] C. Ykman-Couvreur and B. Lin, "Optimised State Assignment for Asynchronous Circuits Synthesis", *in Proceedings of the Second Working Conference on Asynchronous Design Methodologies*, pp. 118-127, May 1995.

[96] K. Y. Yun, S. M. Nowick, D. H. Dill, "Synthesis of 3D Asynchronous State Machines", *in Proceedings of International Conference on Computer Design*, pp.

346-350, 1992.

[97] K. Y. Yun, D. H. Dill, Automatic Synthesis of 3D Asynchronous State Machines", *in Proceedings of International Conference on Computer Aided Design*, pp. 346-350, 1992.

Appendix A

Parallel Transitions and Distributive Lattices

A.1 Introduction

A parallel input transition to a logic circuit specifies a cube, called a *transition cube*, containing all possible input states which may occur during that transition. In this appendix an interesting feature of transition cubes is uncovered and it is shown that every transition cube in a parallel monotonic input transition to a logic circuit realizes a *distributive lattice* under the *follows* partial ordering.

This Appendix is organized as follows: In subsection A.2 basic notions and definitions are reviewed mainly from [23] with some minor changes. In subsection A.3 Lemmas and Theorems are developed to prove our claim. Section A.4 shows an example as the conclusion.

A.2 Basic notions and definitions

Definition A.1: A *relation,* $\mathcal{R}$ from set A to set B is a subset of the product of the two sets. A relation, $\mathcal{R}$ from A to A is called a relation on A. For $a, b \in A$, $a \mathcal{R} b$ means $(a, b) \in \mathcal{R}$

Definition A.2: A relation $\mathcal{R}$ on a set S is called *transitive* if whenever $a\ \mathcal{R} b$ and $b\ \mathcal{R}\ c$, then $a\ \mathcal{R} c$, for all $a, b, c \in S$.

Definition A.3: A relation $\mathcal{R}$ on a set S is called *reflexive* if $a\ \mathcal{R} a$ for all $a \in S$.

Definition A.4: A relation $\mathcal{R}$ on a set S is called *antisymmetric* if whenever $a\ \mathcal{R} b$ and $b\ \mathcal{R} a$, then $a = b$, for all $a, b \in S$.

Definition A.5: A relation is called a *partial ordering* if it is transitive, reflexive and antisymmetric.

A.3 Transition cubes and Poset theory

Having introduced the basic foundation, in this section the notion of transition cube is studied in terms of the Poset theory. We first show that every transition cube is a Poset, then it is proved that a transition cube is a lattice as well and finally we show that every transition cube, moreover, is a distributive lattice.

Definition A.6: If a partial ordering relation is defined on a set, then the set is called a *partially ordered set* or a *Poset* under that partial ordering relation.

Lemma A.1: A transition cube, T, is a Poset under the *follows*[1] partial ordering. (This partial ordering, hereafter, will be represented by the symbol $\leq$.)

Proof: We need to show that the *follows* relation is reflexive, antisymmetric and transitive.

1- Reflexive- Since there is no timing restriction on any transient input vector, each vertex may be imagined as a successor of itself, that is $x \leq x$ for all $x \in T$.

2- Antisymmetric- During a parallel input transition when a vertex, x, changes to another vertex, y, that is $x \leq y$, y cannot change back to x under monotonic input transition, that is $y \leq x$ may only hold if $x = y$. Therefore, the *follows* relation is antisymmetric.

3- Transitive- If vertex x follows vertex y, and vertex y follows vertex z, then x also follows z, that is the *follows* relation is transitive.

Therefore, the *follows* relation is a partial ordering and hence a transition cube is a Poset under that relation. ◊

1. By *follows* ordering, we mean the order in which input transitions may occur. In other words, each input route specifies one possible order.

Definition A.7: In Poset P, $x \in P$ is a *lower bound* of a, $b \in P$ if $x \leq a$ and $x \leq b$.

Definition A.8: In Poset P, $y \in P$ is an *upper bound* of a, $b \in P$ if $a \leq y$ and $b \leq y$.

Corollary 1: One of the two elements in any pair of elements in a Poset may be less or greater than the other one or they may not be comparable at all.

Corollary 2: Not every pair of elements in a Poset have necessarily a lower and/or upper bound.

Definition A.9: In a Poset if all lower bounds of a pair of elements, x and y, are comparable with at least one of the lower bounds, u, such that u is greater than all of them, then for that pair of elements u is called the *greatest lower bound* or *glb* for short and is represented as $u = y \wedge x$ (read "y meet x").

Corollary 3: In a Poset if at least one lower bound of a pair of elements is non-comparable with any other lower bound of that pair, then there is no *glb* for that pair of elements.

Definition A.10: In a Poset if all upper bounds of a pair of elements, x and y, are comparable with at least one of the upper bounds, v, such that v is less than all of them, then for that pair of elements v is called the *least upper bound* or *lub* for short and is represented as $v = y \vee x$ (read "y join x").

Corollary 4: In a Poset if at least one upper bound of a pair of elements is non-comparable with any other upper bound of that pair, then there is no *lub* for that pair of elements. ◊

It has been shown that for each pair of elements in a Poset, *lub* and *glb* are unique if they exist (see [23]).

Definition A.11: In a Poset if each pair of elements has a *lub* and a *glb*, then the Poset is called a *lattice*.

Theorem A.1: Every transition cube is a lattice under the *follows* partial ordering.

Proof: Suppose that the starting vertex in a transition cube is all *zero* and the ending vertex is all *one* for those vertices that change during the transition. This does not affect the generality of our discussion and can always be assumed. Notice the notions of *glb* and *lub* in a transition cube. The *glb* of two vertices is the first common input vertex which is reached if the corresponding Hasse diagram is traversed starting from x and y toward the least vertex, that is an ancestor vertex which is as similar to those two vertices as possible. Notice that such a ancestor necessarily exists, as it belongs to the transition cube. Also no-

tice that an ancestor cannot have any greater bit than its successors[2]. In other words, the *glb* of two vertices, x and y, is a vertex, v, such that

$$v_i = x_i \text{ if } x_i = y_i \tag{A.1}$$

$$v_i = Min.\ (x_i\ ,\ y_i) \text{ if } x_i \neq y_i \tag{A.2}$$

where a_i is the *i'th* bit of vertex *a*.

Equations (A.1) and (A.2) may be written in a more compact form as

$$v_i = x_i\ .\ y_i \tag{A.3}$$

where "." is the logic AND operator. Equation (A.3) clearly shows that for every arbitrary pair of vertices, x and y, the (unique) *glb*, v, exists.

Now consider the notion of *lub*. The *lub* is the nearest common vertex which is reached if the corresponding Hasse diagram is traversed starting from x and y toward the greatest element, that is a successor vertex which is as similar to those two vertices as possible. Notice that such a successor necessarily exists, as it belongs to the transition cube. Also notice that a successor cannot have any smaller bit than its predecessors. In other words, the *lub* of two vertices, x and y, is a vertex, u, so that

$$u_i = x_i \text{ if } x_i = y_i \tag{A.4}$$

$$u_i = Max\ (x_i\ ,\ y_i) \text{ if } x_i \neq y_i \tag{A.5}$$

or

$$u_i = x_i + y_i \tag{A.6}$$

where "+" is the logic OR operator. Equation (A.6) clearly shows that for any arbitrary pair of vertices, x and y, the (unique) *lub*, u, exists. ◊

In the last part of this Appendix we show that each transition cube is a *distributive* lattice.

Definition A.12: A lattice L is *distributive* if the meet and join operators (see Definition A.9) are distributive on each other, that is $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$, and $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$, for all $a, b, c \in L$.

Theorem A.2: Every transition cube is a distributive lattice under the *follows* partial ordering.

---

2. Notice that 1 is greater than 0, and 0 is less than 1, as the initial vertex has been assumed to be all 0.

Proof: In Theorem A.1 it was shown that the *glb* and the *lub* of every pair of vertices in a transition cube exist and may be determined by bit-wise ANDing and bit-wise ORing respectively of the two vertices. In other words, it was proved that the meet and join operators can in fact be replaced with bit-wise logic AND and OR operators respectively, which are distributive. So, the proof of Theorem A.2 is now complete. ◊

A.4 Example

The following example clarifies some of the notions used in the context of this appendix.

Figure A.1 shows the transition cube for the parallel input transition *abcd*: *1010 (s)* → *1101 (x)* where *a, b, c* and *d* are the inputs of a four input logic function. Notice that the logic function itself is not required to be specified for our purpose.

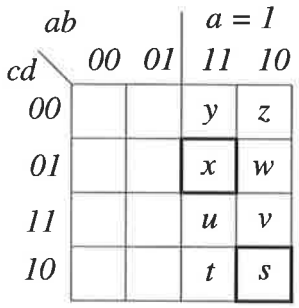


Figure A.1: Transition cube is a = 1 for transition 1010 (*s*) to 1101 (*x*).

Since only three out of four variables are to change, the transition cube is an 8 member set: T = {s, t, u, v, w, x, y, z}, each member being identified by a unique 3-bit pattern. The Hasse diagram of this Poset is shown in Figure A.2.

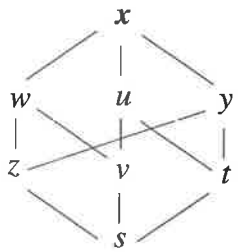


Figure A.2: Hasse diagram for the transition cube shown in Figure A.1.

The upper half and lower half (the shaded area) of Table A.1 show all upper bounds and lower bounds, respectively, of all pair of vertices in the transition cube. Notice that each pair has its *lub* and *glb* which have been shown in bold style in the Table.

As an example of distributivity consider $v \vee (z \wedge t)$ which is supposed to be equal to $(v \vee z) \wedge (v \vee t)$. Referring to Table A.1, $v \vee (z \wedge t)$ may be replaced with $v \vee s$ which is equal to v. $(v \vee z) \wedge (v \vee t)$, on the other hand, may be replaced with $w \wedge u$ which is equal to v, as well.

Table A.1 : Upper bounds, lower bounds (shaded area), lub's and glb's of all pair of vertices in the Hasse diagram of Figure A.2.

	s	*t*	*u*	*v*	*w*	*x*	*y*	*z*
s		**t**, y, u, x,	**u**, x	**v**, u, w, x	**w**, x	**x**	**y**, x	**z**, w, x, y
t	s		**u**, x	**u**, x	**x**	**x**	**y**, x	**y**, x
u	s	**t**, s		**u**, x	**x**	**x**	**x**	**x**
v	s	s	**v**, s		**w**, x	**x**	**x**	**w**, x
w	s	s	**v**, s	**v**, s		**x**	**x**	**w**, x
x	s	**t**, s	**u**, v, t, s	**v**, s	**w**, v, z, s		**x**	**x**
y	s	**t**, s	**t**, s	s	s	**y**, t, s, z		**y**, x
z	s	s	s	s	**z**, s	s	s	