Clocking and Synchronization Circuits in Multiprocessor Systems

By

Deog-Kyoon Jeong

B.S. (Seoul National University) 1981
M.S. (Seoul National University) 1984

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ENGINEERING
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA at BERKELEY

Approved: David A. Hodges .... 4/20/89 ....
Chair Date
Randy H Katz .... 4/21/89 ......
[illegible] .... 4/21/89 ......

****

CLOCKING AND SYNCHRONIZATION CIRCUITS IN MULTIPROCESSOR SYSTEMS

Ph.D. Deog-Kyoon Jeong Dept. of EECS

ABSTRACT

Microprocessors based on RISC (Reduced Instruction Set Computer) concepts have demonstrated an ability to provide more computing power at a given level of integration than conventional microprocessors. The next step is multiprocessors composed of RISC processing elements. Communication bandwidth among such microprocessors is critical in achieving efficient hardware utilization. This thesis focuses on the communication capability of VLSI circuits and presents new circuit techniques as a guide to build an interconnection network of VLSI microprocessors.

Two of the most prominent problems in a synchronous system, which most of the current computer systems are based on, have been *clock skew* and *synchronization failure*. A new concept called self-timed systems solves such problems but has not been accepted in microprocessor implementations yet because of its complex design procedure and increased overhead. With this in mind, this thesis concentrates on a system in which individual synchronous subsystems are connected asynchronously. Synchronous subsystems operate with a better control over clock skew using a phase locked loop (PLL) technique. Communication among subsystems is done asynchronously with a controlled synchronization failure rate. One advantage is that conventional VLSI design methodologies which are more efficient can still be applied.

Circuit techniques for PLL-based clock generation are described along with stability criteria. The main objective of the circuit is to realize a zero delay buffer. Experimental results show the feasibility of such circuits in VLSI. Synchronizer circuit configurations in both bipolar and MOS technology that best utilize each device, or overcome the technology limit using a bandwidth doubling technique are shown. Interface techniques including handshake mechanisms in such a system are also described.

These techniques are applied in designing a memory management unit and cache controller (MMU/CC) for a multiprocessor workstation, SPUR. A SPUR workstation is an example of a synchronous subsystems cluster with independent clocks. The MMU/CC operates between a CPU and a synchronous bus that has an independent clock frequency. The interface and communication aspect of the overall system are revealed through the description of the MMU/CC. The VLSI chip is implemented in 1.6 μm CMOS technology with 68,000 transistors.

David A. Hodges

Professor David A. Hodges
Committee Chairman

CLOCKING AND SYNCHRONIZATION CIRCUITS IN MULTIPROCESSOR SYSTEMS

Deog-Kyoon Jeong

To my parents and my wife, Hyun-Ok,

for their love and encouragement

ACKNOWLEDGEMENT

I wish to express special thanks to Professor David A. Hodges for his guidance and technical advice throughout this work. I have acquired invaluable insights and experience from him in my field. I would like to thank Professors Randy H. Katz and David A. Patterson for their considerate advice and for providing me with an opportunity to participate in the SPUR project and leading the whole group into a big success. I am thankful to Professor Charles Stone for being on my qualifying exam committee and spending his precious time in reading the manuscript.

I am very grateful to David A. Wood, Garth A. Gibson, Susan J. Eggers, Shing I. Kong, Daebum Lee, B.K. Bose, Corinna Lee, Paul Hansen, Mark D. Hill, and Ken C. Lutz in the SPUR group for technical suggestions and discussions. To work with them has been a great pleasure and has been most enjoyable.

Many thanks to my personal friends, Dev Chen, Beomsup Kim, Han-Gyoo Kim, Daebum Lee, Steve M. Hohs, Young C. Shim, and Greg A. Uvieghara, for their friendship, encouragement, and help. Daebum Lee has been a superb companion whom I can rely on.

I am thankful to my parents, sister and brother for their love and encouragement during my study, and to my wife, Hyun-Ok, for her love and patience. Last but not least, I would like to thank all the teachers and professors who, throughout my life, guided me to be what I am today.

This work was sponsored by DARPA under contract order 482427-25840, California MICRO, Texas Instruments, National Semiconductor, Cypress Semiconductor, Tektronix, and Hewlett Packard.

TABLE OF CONTENTS

Chapter 1 Introduction 1

Chapter 2 Problems and Issues 4
- 2.1 Introduction 4
- 2.2 Machine Structures 5
- 2.3 Comparison 8
- 2.4 Summary 11

Chapter 3 Clocking Strategies 13
- 3.1 Introduction 14
- 3.2 Clock Skew 15
  - 3.2.1 Clock Skew Due to Off-chip Elements 15
  - 3.2.2 Clock Skew Due to On-chip Elements 17
- 3.3 Clock Distribution Strategies 18
- 3.4 Inter-chip Clocking Strategies 21
  - 3.4.1 Conventional Schemes 23
  - 3.4.2 Direct Drive Scheme 26
  - 3.4.3 PLL Based Scheme - Delay Comparison 27
  - 3.4.4 PLL Based Scheme - Phase Comparison 30
    - 3.4.4.1 Clock Generation Circuits 31
    - 3.4.4.2 Stability Analysis of Charge Pump PLL 36
    - 3.4.4.3 Experimental Results 39
    - 3.4.4.4 Comparison with Other Clock Generation Schemes 43
- 3.5 Conclusion 45

Chapter 4 Synchronization Circuits 46
- 4.1 Background 46
- 4.2 Synchronizer Models 49
- 4.3 Synchronization Strategies 55
- 4.4 Circuit Techniques 57
  - 4.4.1 Bipolar Synchronizers 58
    - 4.4.1.1 CML and ECL Synchronizers 58
    - 4.4.1.2 Synchronizers with Bandwidth Multipliers 63
    - 4.4.1.3 Synchronizers with Darlington Pairs 66

4.4.1.4 Comparison of Bipolar Synchronizers 69
4.4.2 MOS Synchronizers 71
4.4.2.1 MOS Synchronizers using Inverters with Resistive Load 72
4.4.2.2 MOS Synchronizers using Inverters with Current Source Load 75
4.4.2.3 MOS Synchronizers using Pseudo-NMOS Inverters 77
4.4.2.4 Full CMOS Synchronizers 79
4.4.2.5 MOS Synchronizers using Bandwidth Multipliers 81
4.4.2.6 Comparison of MOS Synchronizers 82
4.5 Conclusion 84

Chapter 5 Interface Techniques 85
5.1 Introduction 85
5.2 Handshake Mechanisms 86
5.2.1 2 Phase Handshake Signaling Scheme 88
5.2.2 4 Phase Handshake Signaling Scheme 89
5.3 Hardware and Protocol Implementation 91
5.3.1 4 Phase Handshake 91
5.3.2 2 Phase Handshake 93
5.3.3 Modified 2 Phase Handshake 94
5.3.4 Data Delivery Circuits 96
5.3.5 Bidirectional Handshake Mechanisms 97
5.3.6 Reset Signal Generation 104
5.3.7 Interface with Bus Systems 105
5.4 Summary 109

Chapter 6 Example - SPUR MMU/CC 110
6.1 Introduction - SPUR Overview 110
6.2 Functional Description 114
6.2.1 The Processor Cache Controller 114
6.2.2 The Snooping Bus Controller 116
6.2.3 Asynchronous Interface 118
6.3 Implementation 119
6.4 Design Methodologies and CAD Tools 128
6.5 Status and Conclusions 131

Chapter 7 Conclusion 133

References 134

CHAPTER 1

Introduction

Advances in semiconductor processing technology have made it possible to realize the minimum feature size in a submicron range. With such advances, it has become feasible to integrate more than one million transistors on a chip. The main beneficiary has been memory chips and microprocessors. Following the pattern that computer architecture evolved in response to the advances in the underlying implementation technology, a new concept called RISC (Reduced Instruction Set Computer) emerged as a result of careful study of trade-offs between VLSI technology and software. VLSI microprocessors based on the RISC concept provide more computing power at a given level of integration than conventional processors [1.1]. RISC microprocessors benefit from pipeline techniques as well as the reduced cycle time through simplified hardware.

However, their performance tends to be limited by the instruction/data delivery time rather than by the arithmetic computation time. Thus, it is important to efficiently utilize the limited resource, pin input/output bandwidth. The chip area saved by the simple CPU architecture is used to integrate a big register file, an instruction cache, a data cache and/or a memory management unit on the same chip to reduce the traffic across pins without increasing the cycle time. Attempts to add a new dimension in computing have been directed to multiprocessors based on low-cost, high-performance microprocessors. Communication bandwidth among such microprocessors in a multiprocessor is even more critical in achieving efficient hardware utilization. Communication bandwidth, however, is dependent on many factors such as implementation technology, clock skew, signal reflections, and also by the physical limit - the speed of light.

Implementation technology limits the current drive capability of the pins, clock skew increases the cycle time, signal reflections reduce noise margin, and the speed of light adds latency to a communication channel. This thesis looks at the communication capability of the VLSI and provides new circuit techniques as a guide to build an interconnected network of such microprocessors.

Another problem in building a synchronous system is the lack of a perfect synchronizer [1.2]. An output from a synchronizer may hesitate between 0 and 1 for an unbounded time. This problem is present in almost all digital systems. However, a perfect synchronizer can be approximated if enough settling time is allowed. Although the extra latency allowed for settling is a serious performance limiting factor for some high performance systems, its effect can be minimized if the complete system relies on communication bandwidth rather than it requires a short communication latency. For example, in a bus-oriented multiprocessor system like SPUR [1.3], synchronizations are needed only on cache misses. Data transfers are done in a block mode where the initial channel set-up time including the synchronizations of request/acknowledgement handshake signals takes only a small fraction of the block data transfer time. A concept called self-timed circuits [1.4] addresses such problems but it has not been accepted in the microprocessor designs because of its complex design procedure and increased overhead.

This thesis focuses on systems in which individual synchronous subsystems are connected asynchronously. These system follow the same direction as the self-timed approach in a global scale but they give up the self-timed approach in the block/logic level and below. Synchronous subsystems can be operated in confined regions with better control over clock skew using phase locked loop (PLL) techniques [1.5]. Intercommunication among subsystems is done asynchronously with a controlled synchronization failure rate. One advantage to this structure is that conventional VLSI design techniques which are more efficient can still be applied. Circuit techniques of PLL-based clock generator will be described along with its stability criterion. The main objective of the circuit is to realize a zero delay buffer, effectively removing delays caused by a clock generator and buffer. The performance of a synchronizer, measured by the time constant with which the output grows exponentially, is mainly determined by its implementation technology. Circuit techniques which can overcome technology barriers will be investigated both in bipolar and MOS technology.

The SPUR multiprocessor system is an example of an asynchronously-tied cluster of synchronous subsystems. SPUR's VLSI chips are designed with multiple clock phases, and processor boards are connected asynchronously. Through the description of the memory management unit and cache controller (MMU/CC) which operates between a central processing unit (CPU) and a synchronous bus with an independent clock, the details and difficulties of the overall system will be revealed.

In Chapter 2, a brief outline of various problems and issues of implementing digital systems is given. First, various terminologies and machine structures are defined. Second, problems in each structure are investigated and performance comparisons are presented. Chapter 3 concentrates on the clocking problem and a new PLL-based clock generation and distribution scheme is proposed along with its experimental data. In Chapter 4, the characterizing parameters of a synchronizer are defined and various circuit techniques of building a synchronizer are investigated. In Chapter 5, several interface techniques, which can handle various handshake mechanisms working between two synchronous systems with independent clocks, are described. In Chapter 6, implementation details of the SPUR memory management and cache controller are described as an example of an asynchronously-tied cluster of synchronous subsystems. Chapter 7 concludes this thesis.

CHAPTER 2

Problems and Issues

This chapter delves into various problems and issues of building a large digital system. It describes and compares various machine structures and attempts to determine which digital system structure has a relative advantage in VLSI implementation.

2.1 Introduction

Since all electrical signals are subject to propagation delay, any information in them can only be restored with pre-determined timing. For example, a signal from a lamp switch bearing an ON-OFF information is always valid, regardless of timing. Timing information is included in the signal itself. But, if we build a finite state machine with several of those switches, problems may occur when two or more of the switches make transitions at different times during the interval during which those inputs are being processed. The next state may not be determined consistently because of its feedback from the outputs to the inputs through state bits. That is, a temporary state due to delay time mismatch may be propagated to the inputs to that determine the next state. The problem arises from the fact that information and timing are intermingled into a single electrical signal. If timing can be separated from information, such difficulties can be eliminated. For example, after holding and latching at some time point all the incoming inputs and state bits, the resulting next state can be latched and stored at another time point sufficiently late to accommodate the worst-case delay time. A special signal can be used to carry such timing information. This simplifies system design. All the signals can be treated as if they have the same propagation delay between holding and latching. The former structure where timing is in the signal itself is called an asynchronous machine. The latter structure, where a separate signal is used for carrying timing information, is called a synchronous machine. The separate signal is called a *clock*. A clock in this context can be defined as follows:

A clock is a control signal that carries timing information about the data.

For communication with the rest of the world which is inherently asynchronous, sampling and holding of the signals originating from the external world is needed. Such an operation is called *synchronization*. Synchronization in this context can be defined as follows:

Synchronization is a mechanism that conforms the signal originating from outside to the timing governed by the clock of the system.

As will be explained later, it has been shown that a perfect synchronizer cannot be built. When an input is sampled on its change, a synchronizer may delay its decision for an unbounded time. This condition is called metastability. This led to many variations to the synchronous machine structure to avoid the problem of imperfect synchronizers.

2.2 Machine Structures

Digital systems can be classified into two broad categories: asynchronous and synchronous. The *asynchronous machine* is a structure where all timing information is included in the signal itself without requiring a clock. The *synchronous machine* is a structure where all timing information is carried in a clock.

Although it is easier to build a synchronous machine than an asynchronous machine, it becomes unmanageable to include all the subsystems within a system sharing a single clock [2.1]. Typically, subsystems differ significantly from each other in their functions, and they can be physically far apart. Some systems need a slow clock and others need a fast clock. When they are physically far apart, clock skew is hard to reduce. For example, in a computer system, the clock frequency of a main processor is higher than the one used by a printer controller. Also, a printer may be located at such a distance that the electrical waveform of a clock cannot travel to it in a cycle time. Thus, the question is not whether we have to partition the whole system, but how we partition it. There are natural boundaries like the above processor-printer example due to physical distance, and functional ones that may arise between a CPU and a bus system. Each side uses the clock that is optimized locally and independent of the rest of the system. For communication among them, some two-way protocol is necessary. The next issues become 'what is the optimum partitioning?' and 'how do we handle synchronization failure?' There are no definite answers to these questions. They largely depend on the implementation technology and judgment with respect to accepting non-zero failure probability. Therefore, since there are no decisive answers, many system structures have been suggested.

The first structure, proposed by Pechoucek [2.2] and shown in Fig. 2.1(a), is a variant of a synchronous system that totally removes the probability of system malfunction due to synchronization failure. It stretches its clock arbitrarily until a synchronizer safely escapes from a metastable state. A metastable detector is a simple voltage comparator that checks if the output signal is in a metastable state. The second structure, also proposed by Pechoucek [2.2] shown in Fig. 2.1(b), removes the probability of system malfunction. However, this structure deviates more from a synchronous system and resembles a self-timed structure. It integrates a START/STOP signal with the stretchable clock generator and behaves much like a self-timed structure with a request/acknowledgement handshake. Clocks are generated on START and stop on a completion signal, STOP. These two structures have been studied in depth, resulting in more elegant structures such as an *unsynchronous system*, and an *escapement system* [2.1].

A self-timed system [2.3] is an asynchronous system where timing information for individual signals is carried by request/acknowledgement handshake control signals, allowing a speed

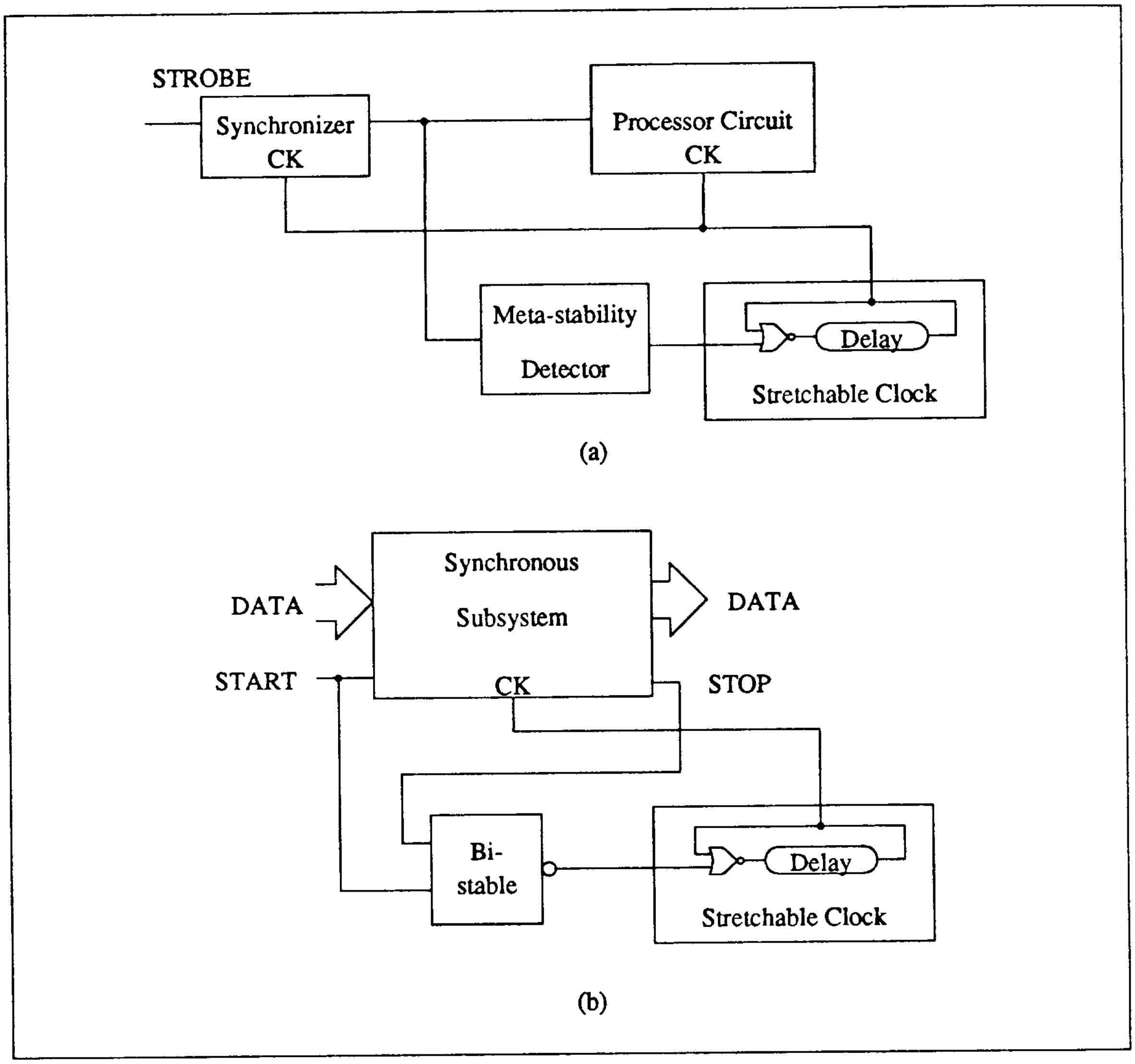

Fig. 2.1 The two machines that eliminate system errors due to synchronization failure. Modified for simplicity from [2.2]. (a) A simple scheme using a metastability detector. (b) A systematic scheme that uses a completion signal for stopping a clock.

independent operation. An escapement system and a self-timed system can be differentiated in that the core computing element of the escapement system is a synchronous machine using a stretchable clock; whereas the computing element of the latter is composed of a combinational logic with a completion signal generator.

Another variation from a synchronous system proposed by Anceau [2.4] eliminates synchronization failure while allowing different clock frequencies among subsystems. It uses a PLL technique to generate local clocks that have sub-harmonic frequencies from the master clock with a known phase relationship, thereby eliminating the possibility of the input signals being sampled at random time points. Although each subsystem cannot use a completely independent clock, it is free from synchronization failure and has the flexibility of choosing its own frequency.

The simpliest structure is a cluster of synchronous subsystems with independent clocks connected in a liberal way but with each subsystem having synchronizers with an acceptable failure rate. An acceptable failure rate may be, for example, in the same range as a device failure rate.

2.3 Comparison

There can be many top level design considerations in choosing a system level machine structure. Although a purely asynchronous system in principle has an advantage of having the fastest response, such systems are difficult to design. Currently known asynchronous design methods are very restrictive in that they allow only one input transition at a time. Or, they must use a hypothetical element called *inertial delay* to allow multiple transitions [2.5-2.8]. It was also proven that an inertial delay has the same problem as the synchronizer [2.9]. Also, the state assignment problem becomes enormous when the system complexity grows.

A purely synchronous machine has a very simple structure in which a global clock is distributed across the entire system for the timing reference. However, it is very difficult to distribute a clock without any timing error among its subsystems. This error forces the system cycle time to increase. Another annoying factor is a reliability problem associated with a synchronizer. Also, the performance of the synchronous machine is not maximized. Since the cycle time is determined by the slowest path, it under-utilizes the full potential of the hardware. The fastest hardware block has to wait for the slowest block to finish its computation. This gets worse as the system grows.

Pechoucek's machines look attractive because they are relieved of synchronization failure. However, clock stretching adds substantial complexity to clock generation circuits, and requires careful consideration of the delay in buffering and distribution circuits. For example, when the clock buffer/distribution delay is 10ns, a synchronizer has to pre-sample its input 10 ns before the main clock arrives in order not to stretch the clock for every cycle. The time wasted for pre-sampling is similar to the waiting time required for a synchronizer to settle. Pechoucek's machines still exhibit the clock skew problem of the synchronous system and it gets worse as the clock subsystem becomes more complicated.

A self-timed approach is attractive since the system does not use a global clock and is not subject to synchronization failure. Also, it can be fast since each self-timed hardware block can start the next computation regardless of other hardware blocks' status, as long as the input signals are ready and the output port is empty. This is in contrast with a synchronous system where a cycle time is determined by the worst-case delay of any blocks. For example, the carry propagation delay of a parallel adder tends to be the longest path in an ALU and thus determines the cycle time in a synchronous system. Since the worst-case carry propagation time is O(n), the cycle time is also O(n) regardless of input patterns. On the other hand, in a self-timed system, an average carry propagation delay is only O(log(n)), giving tremendous advantage in timing, especially when n is large [2.3]. However, for a self-timed system to be practical, several critical questions need to be answered. First, since its design is not conventional, a new design method should be developed that produces a design comparable to synchronous ones in terms of delay time, silicon area, etc. Second, since request/acknowledgement signals bear timing information for the individual blocks, the request/acknowledgement handshake processing delay time is a waste [2.10]. It is similar to a clock skew problem. Third, in addition to having computing elements, special blocks are needed to generate completion signals, or they have to to be processed along with data. Their overhead is very costly in terms of area and timing. Lastly, when a very complex system is designed, multiple request signals may have to be processed. This requires arbiters. An arbiter used to handle such a situation has problems similar to a synchronizer. That is, an arbiter may have an unbounded delay although overall system reliability is not a concern. Despite such problems, because of its advantages, a self-timed system is very attractive as an alternative to synchronous machines in VLSI technology.

The last, but most popular choice is an asynchronously-tied cluster of synchronous subsystems. Each subsystem has its own clock optimized according to its own function and technology. There exist many efficient design methodologies developed for synchronous systems. These make this structure the most popular one. Under-utilization of hardware resources can be overcome by adequate pipelining for each block. For communication between the synchronous subsystems with independent clocks, handshake protocols can be used. While control signals like a request and an acknowledgement need to be synchronized with respect to each clock, data signals do not need any synchronization since their timing is known after the control signal arrives. It is possible to reduce the synchronization failure by allowing sufficient waiting time to make a final decision. The probability of synchronization failure decreases exponentially as the waiting time increases. Usually, only a half or a quarter of a cycle time is enough to reduce the probability of failure to acceptable levels. Such latency added to an average of a half cycle probabilistic latency, is a penalty for crossing the asynchronous boundaries. Two of such latencies are needed for a handshake transaction. The negative impact of such communication overhead can be minimized by proper partitioning. For example, in the bus-oriented multiprocessor systems like SPUR [1.2], the communication transaction from a processor to a main memory happens predominantly in 32-byte block transfer mode in which 4 bytes can be transferred in a cycle with a maximum rate. In that case, the overhead of two cycles due to synchronization is not a serious performance degradation in view of the fact that the transaction happens only in less than 2% of the cache accesses and it takes about 20 cycles per transaction on the average including arbitration cycles, assuming the typical access delay of a memory board.

In the synchronous communication among subsystems sharing the same clock, the clock skew problem may force the cycle time to be increased without any fruitful computation. The sources of the inter-chip clock skew are the electromagnetic propagation delay, the buffer delay within the chip, and the RC delay in the distributed clock lines on the chip. As long as all delays are the same, there is no degradation in communication bandwidth. But in practice it is very difficult to maintain the same delay along all paths. For example, the buffer delay within the chip varies from chip to chip due to process variations, temperature variations and different loading capacitances. There have been many efforts to reduce the clock skew [2.11-2.12]. One approach, described in detail in Section 3.3, is to use a PLL technique to realize a zero delay buffer for clocks to minimize skew among subsystems which are mostly VLSI chips.

2.4 Summary

Traditionally, a synchronous design approach has been used predominantly. The main driving force behind this approach is the simplicity of its structure. However, there are several problems that need to be solved, i.e., the performance, the clock skew, and the failure of a synchronizer. System partitioning is required as the system complexity grows since such problems tend to get worse in larger systems.

Many alternative structures have been proposed in order to solve the problems of synchronous design. Out of the alternatives, a self-timed structure draws the most attention since it appears to solve most of the problems. With current VLSI technology, keeping synchronous design approach seems to have more advantages. A cluster of synchronous subsystems with an asynchronous interface has been and probably is the most cost-effective in building computer systems. But we must evaluate the impact of the future technology on the issues and problems of synchronous systems and self-timed systems in order to better understand the relative advantages.

CHAPTER 3

Clocking Strategies

In chapter 2, the influence of clocking strategy on a digital system's performance was described. The role of a clock has become even more important as the integration and throughput of a system grow with VLSI technology advances. Clock frequency of commercial microprocessors kept increasing from less than 1MHz to 33MHz or more.

In low performance systems, the clock cycle time is composed mostly of the arithmetic computation delay time such as the carry propagation delay in the datapath rather than of the communication delay time between subsystems. As MOS devices scale down due to technology advancements, the computation delay time scales down due to improved device performance. On the other hand, the communication delay does not decrease accordingly. Communication performance is dependent on the packaging and interconnection technology as well as the device technology. Also, the communication delay has its fundamental limit - the speed of light. Thus, the cycle time of the high performance systems based on RISC microprocessors tends to be limited by communication performance. A good clocking strategy is needed to enhance their performance since the system clock plays one of the most critical roles in handling communication between subsystems by providing a timing base for latching and driving data signals.

In this chapter, I will focus on the clocking issues for systems requiring high communication bandwidth such as the ones incorporating RISC concepts. First, in Section 3.1, I briefly raise the issues associated with clocking strategies. In Section 3.2, various sources of clock skew are described. Clock distribution problems are addressed and analyzed in Section 3.3. Various techniques to solve on-chip clocking problems are discussed in Section 3.4. Finally, a brief summary is included in Section 3.5.

3.1 Introduction

For high performance RISC systems like SPUR [1.2], the instruction delivery time is very important since the internal circuit is so fast that they consume an instruction/data stream quicker than it can be supplied through the input/output pins. Thus, it is important to provide the required communication bandwidth so that maximum performance can be achieved. Maximum attainable communication bandwidth is heavily dependent on clocking strategies - clock skew control, clock distribution, and so on. A *clock* is the source of timing reference in on-chip and off-chip communication in a synchronous system and is shared by all of its subsystems. Since all the subsystems cannot physically be in the same place, it is inevitable for a clock to be distributed with different paths from its source. As long as path delays are all the same, clock delays are hidden and all the subsystems do not see any adverse effect. The amount of relative timing difference at different clock receiving points is called *clock skew*. In some systems [3.1], a timing delay may be intentionally introduced to a clock phase to stretch a pipeline stage to accommodate unbalanced pipeline delays. In that case, clock skew is defined differently as *deviation from the intended timing*, not as the amount of stretch introduced. If the amount of clock skew is very small relative to the clock cycle time, or if system performance is not communication bound, clock skew does not raise any performance problem. Otherwise, clock skew is directly translated into system performance degradation. Clock cycle time must be increased in order to compensate for the timing loss due to clock skew.

3.2 Clock Skew

Fig. 3.1 shows various paths from a clock source to on-chip clock receiving points in a typical digital system. There are four kinds of clock delays - the clock buffer delay, the off-chip delay due to electromagnetic (EM) wave propagation through a printed circuit board, the on-chip delay due to a clock generator/driver, and the on-chip EM distributed delay due to the on-chip interconnection.

3.2.1 Clock Skew Due to Off-chip Elements

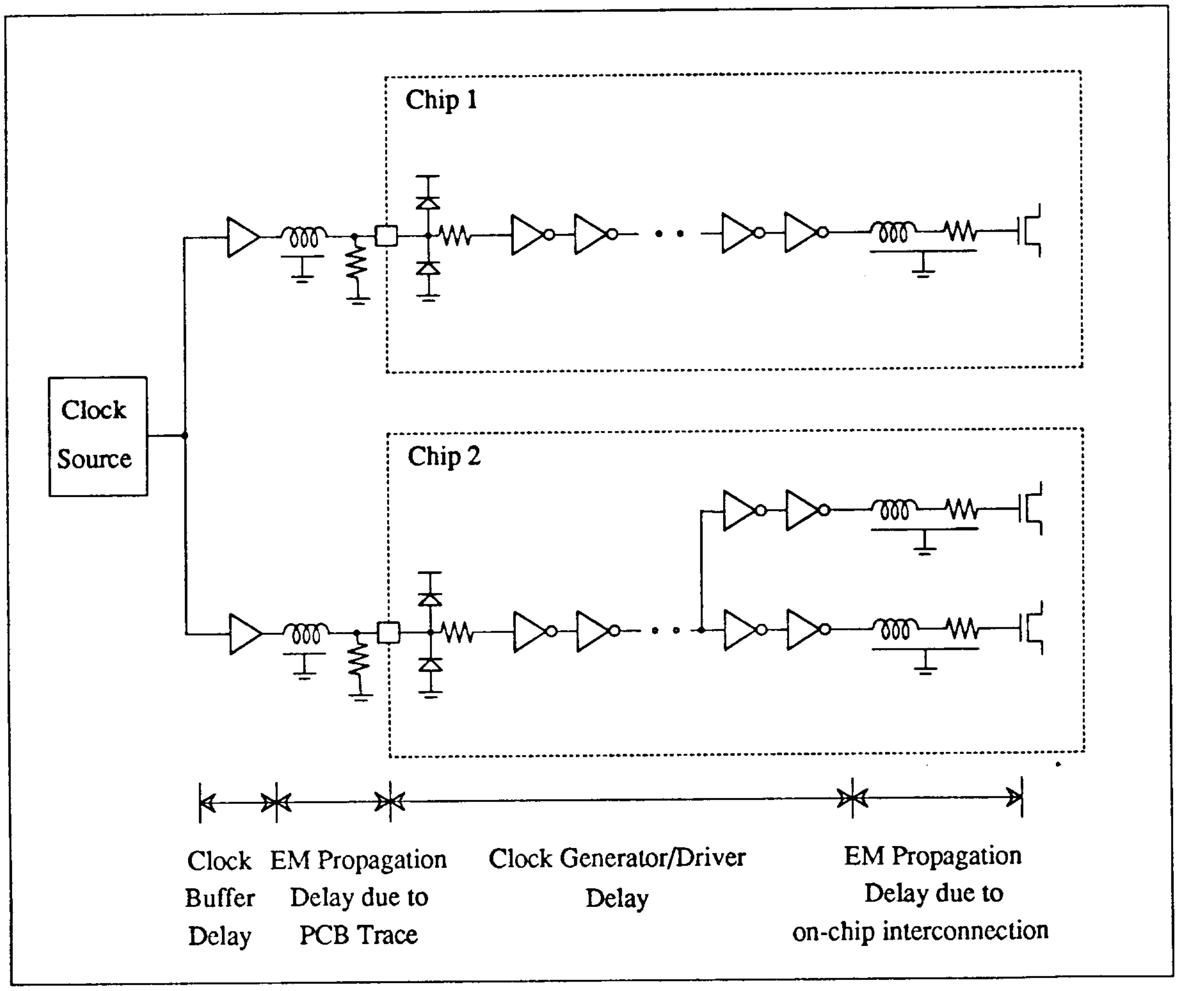
Chip 1
Clock Source
Chip 2
Clock Buffer Delay
EM Propagation Delay due to PCB Trace
Clock Generator/Driver Delay
EM Propagation Delay due to on-chip interconnection

Fig. 3.1 Three delay paths from a clock source to on-chip clock receiving points.

There are two kinds of delay elements outside clock receiving chips - off-chip clock buffers and printed circuit board (PCB) traces. Analysis of the total delay due to off-chip elements involves four factors - the inherent delay of the buffer, current drive capability of the buffer (source impedance, Z_s), interconnection line characteristics (impedance, Z_o and propagation delay, τ_d), and the load capacitance (C_L). These factors and their relations must be considered to minimize clock skew. The goal is to assure equal delays from the clock source to the clock pins of the receiver chips. The inherent delay through the off-chip clock buffers can be equalized by using a set of driver gates in a package and balancing the load. The EM delay through the PCB trace originates from electromagnetic wave propagation. Since its propagation velocity is determined by permittivity and permeability of the medium, its delay is predictable and relatively easy to control. For a PCB made of epoxy, the propagation delay is about 1.7ns per every inch of the trace. Since it has extremely low temperature variation, it is relatively easy to remove clock skew by making the clock traces have equal lengths. However, a clock line must have a proper termination resistor at the receiver to absorb all the incoming energy; otherwise, the clock signal will be seriously deformed due to the wave reflected back and forth between the source and receiver. For short traces that bear a less propagation delay than the rise/fall times of the clock, it is not necessary to provide termination resistors. The resistance should equal the characteristic impedance of the line, which ranges from 20 Ω to 100 Ω in typical PCBs. When the buffer does not have enough drive capability for terminated transmission lines, the clock signal may not cross the logic threshold, causing failure. The buffer used with unterminated transmission lines will cause the clock waveform to appear as a staircase with each time step equaling the round-trip delay, making it hard to predict the exact delay time. When the load capacitance is large, it not only slow down the clock edge with the time constant, Z_oC_L, contributing a significant delay but it also degrades the clock waveform due to imperfect termination. Thus, three requirements must be satisfied for high quality off-chip clock distribution - a clock buffer should have a source impedance low enough to drive the transmission line ($Z_s \ll Z_o$), each clock line should be terminated with the characteristic impedance of the line ($Z_T = Z_o$), and the receiver should have an input capacitance low enough not to disrupt the termination ($jwC_L \cdot Z_o \ll 1$). When the three conditions are not satisfied, unpredictable clock skew may be introduced in the system.

3.2.2 Clock Skew Due to On-chip Elements

When a clock signal is brought inside the chip, it normally propagates through an input protection circuit, a clock generation circuit and clock driver. The input protection circuit, typically composed of two diodes and a resistor, prevents damage from electro-static discharge (ESD). Although it adds an RC delay, the absolute amount is very small and its effect can be ignored. A clock generation circuit derives internal clock phases from an externally provided clock. It, for example, converts a single phase clock to a 2 phase non-overlapping clock. Normally, in a typical VLSI digital circuit, the total capacitance attached to a clock exceeds 10 pF. Driving such capacitance requires an inverter chain with gradually increasing size [3.2]. The number of intermediate logic gates including the inverter chain required for a clock to propagate from a pad to clock receiving points is usually more than 5 for the 2 phase non-overlapping clock. Clock skew occurs when chips have different clock delays. Thus, when several VLSI chips must communicate with each other at a high clock frequency, the chips must be designed to have the same delay so that its actual clock receiving points see the clock at the same time. However, because of the temperature and process variations, it is impossible to guarantee the same delays across the chips. Assuming 30% variations in device performance, clock skew among chips can easily be equivalent to 60 % of the total clock delay. On-chip clock skew due to process variations can be minimized using the circuit techniques described in [3.3]. They utilize device matching characteristics to equalize the clock delays among different paths within a chip. However, the idea cannot be applied to minimizing clock skew between chips.

3.3 Clock Distribution Strategies

Fig. 3.2 shows various ways to distribute a clock to various places. Fig. 3.2 (a) shows a scheme that a clock branches in the middle of a trace. Since it is impossible to match impedances in this scheme, wave reflections at the junction cause deformation of the clock signal. This scheme is not preferred in high performance systems. The scheme shown in Fig. 3.2 (b) does not

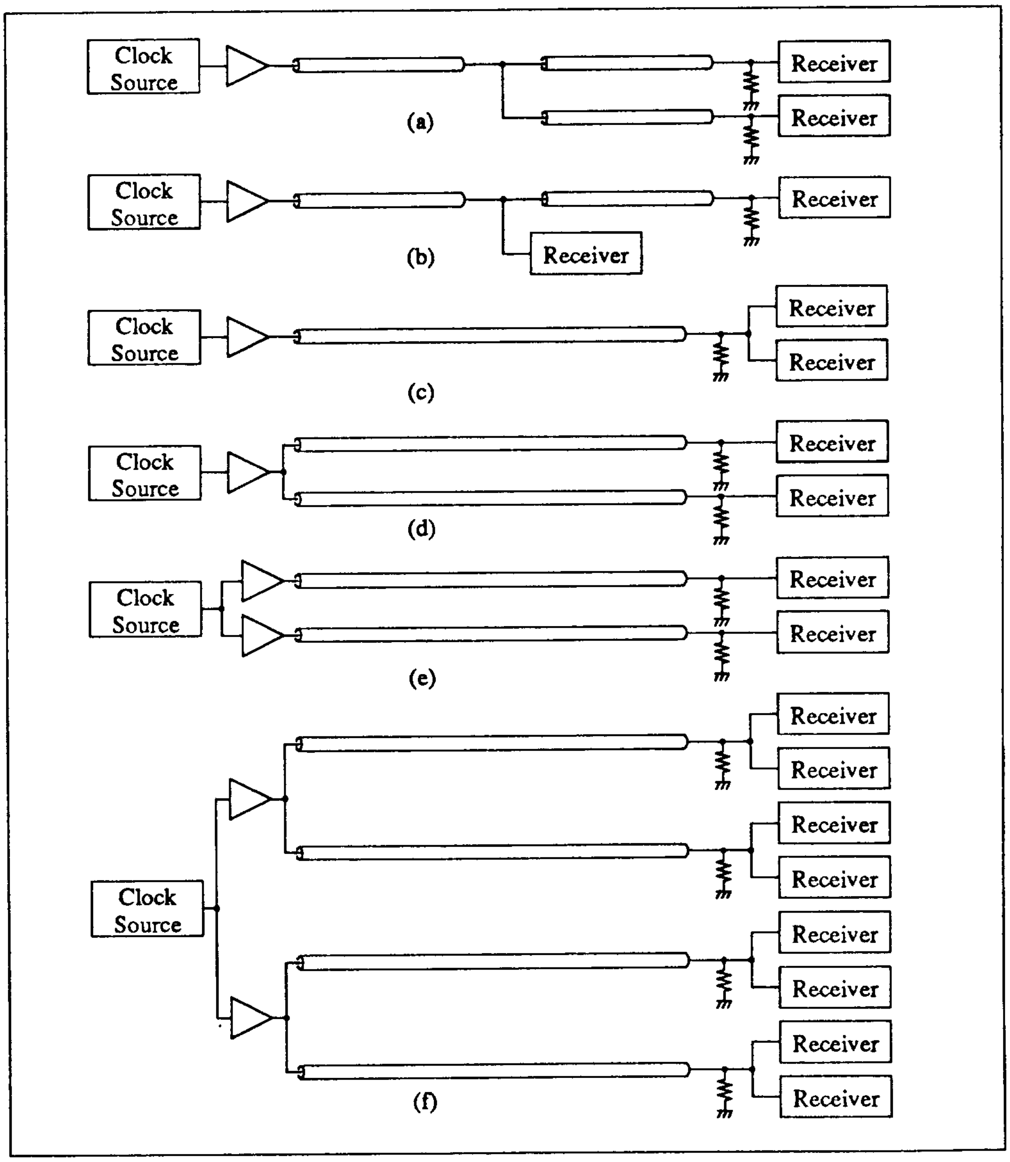

Fig. 3.2 Clock distribution strategies.

have an impedance mis-matching problem as long as the loading effect of the input capacitance is negligible. However, the scheme has inherent clock skew of the amount equaling the propagation delay of the second line. Third scheme, shown in Fig. 3.2 (c), works well when a group of chips are physically close enough so as not to cause any significant reflections. It will, however, degrade the rising/falling edge of the clock and cause reflections unless the sum of the input capacitances is negligible. If the time constant of the total load capacitance and the characteristic impedance of the line is comparable to the transmission line delay, a clock edge in the receiver will suffer much distortion as well as having a very slow transition. The fourth scheme shown in Fig. 3.2 (d), where each clock line branches from the buffer and drives only one receiver, has a definite advantage of having minimum capacitive loading, being less vulnerable to reflections. However, the buffer sees the impedances of the lines in parallel. Assuming a line impedance of $Z_o\,\Omega$, a clock buffer at the source must be able to drive $\frac{Z_o}{n}\,\Omega$, where n is the number of receivers. When $Z_o = 100\,\Omega$ and n=10, the clock driver must have a source impedance of less than 10Ω. This requires a special design technique. Although the scheme shown in Fig. 3.2 (e) is robust, its cost is high, since an off-chip buffer must be assigned for every clock receiver. Instead of the brute-force approach, a compromise can be made as shown in Fig. 3.2 (f). Groups of nearby receivers with a similar total capacitance are formed. For each group, only one termination resistor is used; each clock line connects one termination resistor for each group; and a clock buffer drives a small number of lines within its drive capability. In that compromised scheme, a driver chip which contains six identical buffers, can possibly cover all the receivers in a board.

Fig. 3.3 shows simulation results that reveal the effect of loading capacitance on the clock waveform in the terminated line. Note that the waveforms of the rising and falling edges are exponential with the time constant of $\frac{Z_o}{2}C$. Assuming a $100\,\Omega$ line with 5 ns delay and 10pF input capacitance, the distortion is not severe enough to introduce more than 1 ns skew from a

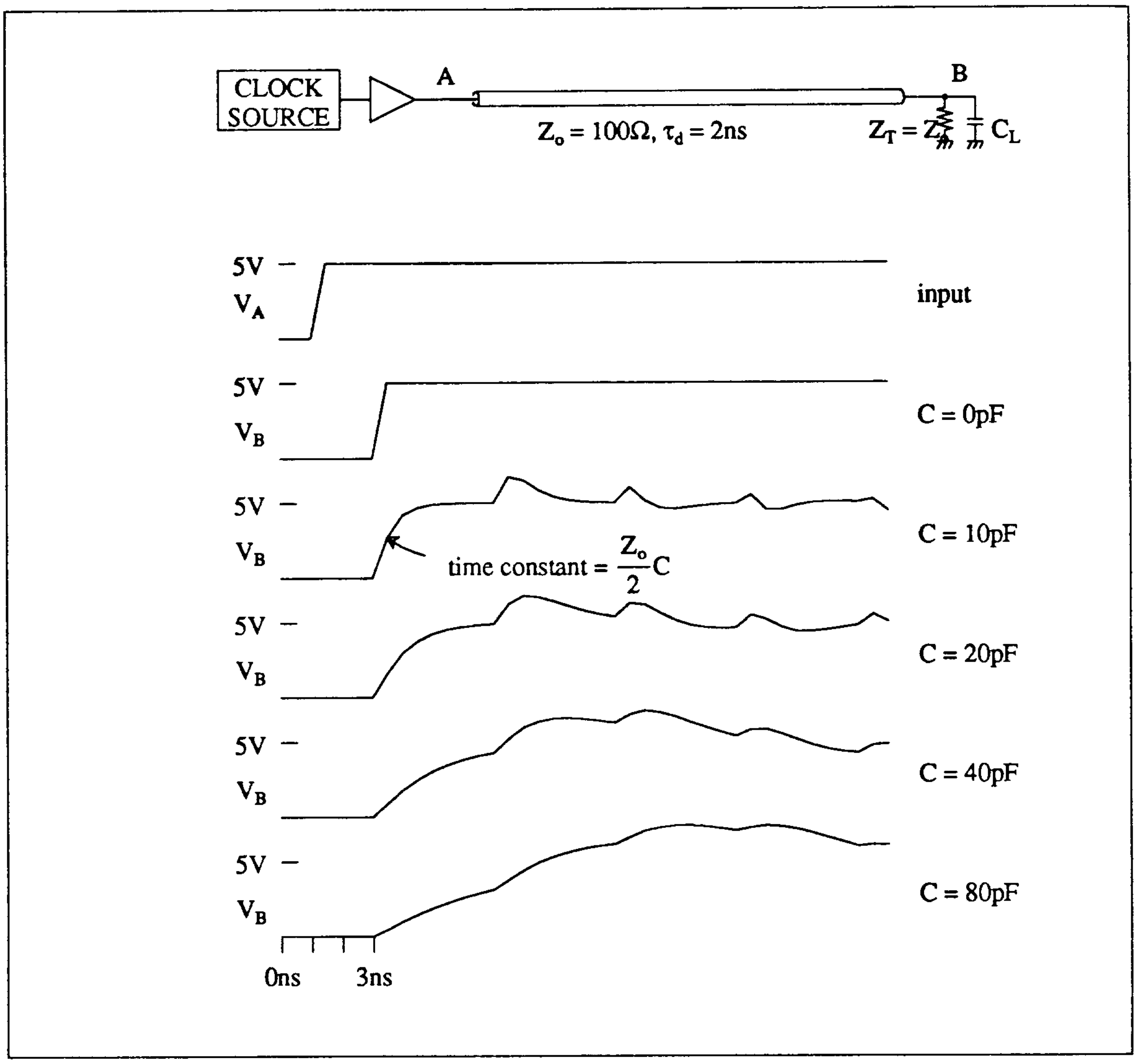

Fig. 3.3 Effect of load capacitance.

reflection free waveform. On the other hand, a 100 pF load capacitance will slow down clock transitions as well as causing excessive reflections.

In a synchronous standard microcomputer backplane bus like NuBus [3.4], a system-wide bus clock is distributed as in Fig. 3.2 (a). It does not make any effort to reduce clock skew between boards. NuBus treats the clock signal just as any other control and address/data signals. Considering that NuBus backplane propagation delay is 8.5ns, performance loss suffered due to clock skew amounts to 8.5% with 10 MHz clock frequency. The main reason it uses the bus-like trace is to save the backplane PCB area by fitting into the pitch of the pin connectors. However, since the pin position of the clock in a NuBus is located in the far side of the 96 pin connector, the backplane PCB has empty space on the side. Using the empty space, multiple clock traces can be distributed with a star-like structure instead of the bus-like structure. A better distribution scheme would be the configuration shown in Fig. 3.4 with an increased PCB area and an additional chip.

When we are dealing with clock distribution within a board, it is not very difficult to cope with the clock skew due to PCB traces. However, a system-wide clock distribution across many boards with possibly varying technology is a problem that needs special attention. Since the number of clock receivers far exceeds the capability of a set of drivers housed in a package and the receivers are physically widely separated, it is necessary to have more than a single level of buffering. For example, the VAX 8800 series mainframe computers use a hierarchical clock distribution scheme with many levels [3.5]. About 17 % of its cycle time is the tolerance for clock skew. Without the systematic methodology, it would have caused a lot more degradation.

3.4 Inter-chip Clocking Strategies

Inter-chip clock skew is defined as a timing error between two internal clock points in different chips as opposed to on-chip clock skew which is defined as a timing error between two internal clock points in the same chip. Off-chip clock skew is due to the sources outside the chips, mostly due to clock distribution. Thus, inter-chip clock skew is the sum of off-chip clock skew and the difference between the maximum and minimum on-chip clock delays among chips rather than the simple sum of off-chip and on-chip clock skew. In this section, I will concentrate on the method of reducing inter-chip clock skew. As explained in the previous section, most of

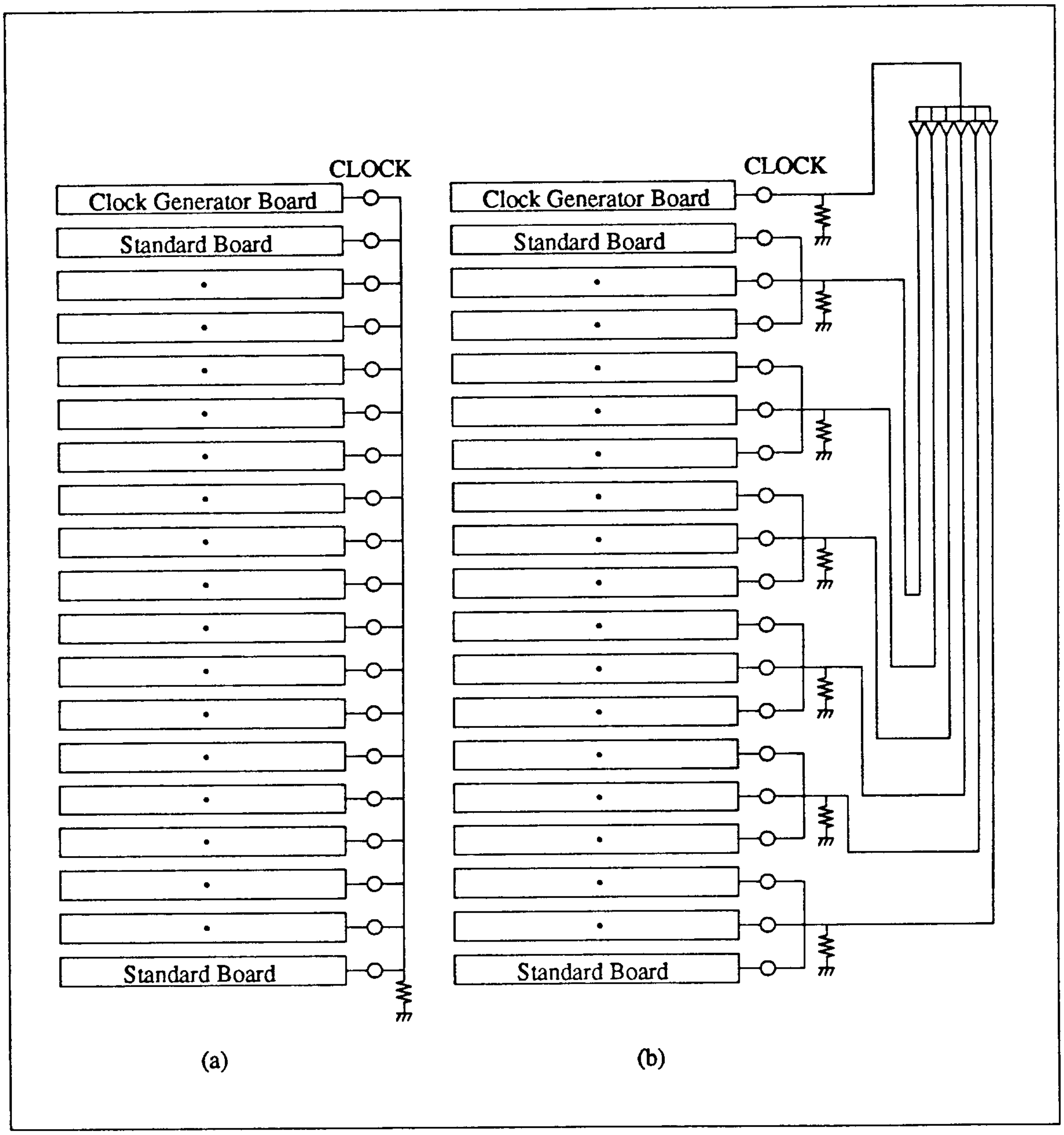

Fig. 3.4 Clock distribution in a standard microprocessor board. (a) Distribution by a bus-like trace. (b) Distribution by equal length traces.

the problem is around the internal clock generation logic and its drivers. Since the clock skew is due to delay variations, an ideal solution would be to have zero-delay buffers. A circuit technique that simulates this effect will be described.

3.4.1 Conventional Schemes

A conventional 2 phase non-overlapping clock generation scheme is shown in Fig. 3.5.

This simple scheme uses both positive and negative edges to generate two phases. The delay elements, τ_d, inserts a non-overlapping time. Since the amount of the non-overlap time is small, equivalent to a delay of a few inverters, the delay elements can be a part of the clock driver in order to reduce the total clock delay. The clock driver, which is a cascade of inverters, has only to provide a tap which has an appropriate delay from the input to be used as a non-overlap time. One problem with this approach is that an exact duty ratio must be maintained for the external clock as well as a correct frequency. This is difficult to meet practically because the same high-to-low and low-to-high delays and the same rise and fall times are not guaranteed in off-clock clock buffers and on-chip gates. One advantage of this scheme is that it is very simple to implement and requires only one clock to be distributed. Motorola microprocessors use this

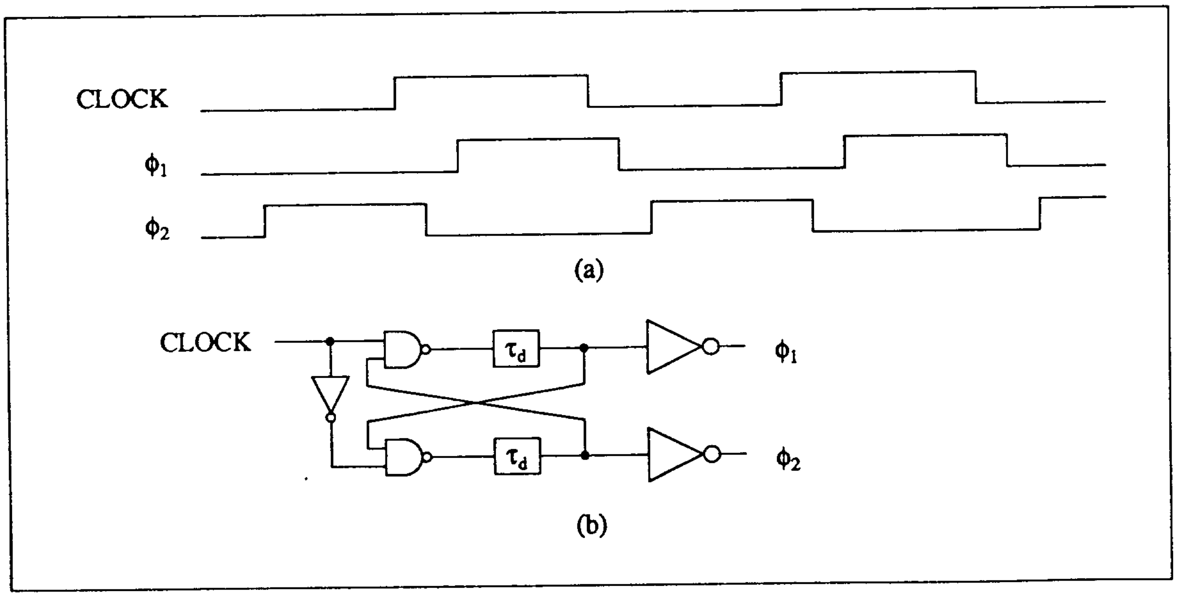

Fig. 3.5 A conventional 2 phase clock generator/driver. (a) Waveform. (b) Logic diagram.

scheme [3.6].

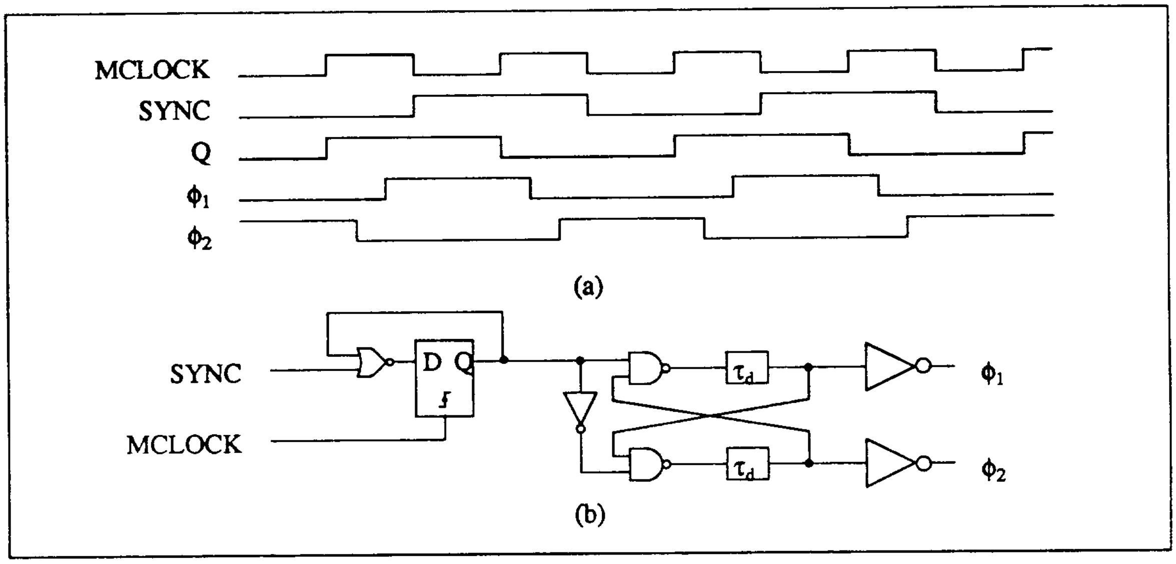

Fig. 3.6 A conventional 2 phase clock generator/driver. (a) Waveform. (b) Logic diagram.

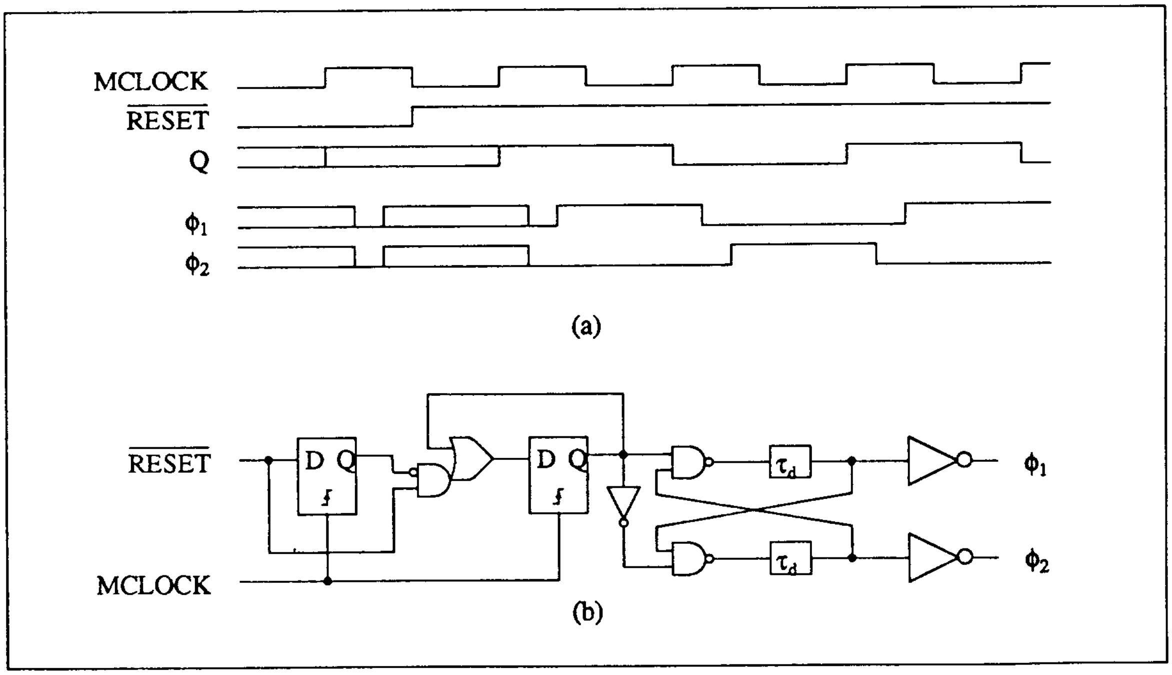

Fig. 3.7 A conventional 2 phase clock generator/driver with $\overline{\text{RESET}}$ as SYNC. (a) Waveform. (b) Logic diagram.

Another similar scheme that uses only one edge is shown in Fig. 3.6. This scheme uses a frequency divider to generate a symmetrical waveform. Since the flip-flop changes its output only on the rising edges, the output maintains 50 % duty ratio as long as the clock input maintains constant frequency. Since any clock edge can be used as either ϕ_1 or ϕ_2, an extra synchronization signal, called SYNC, is needed. Without SYNC, the two systems sharing a clock, called MCLOCK, may be out of synchronization. For example, one generates ϕ_1 out of even numbered edges, while the other uses odd numbered edges for ϕ_1, depending on the initial state of the flip-flop. A reset signal may play the same role as SYNC with added complexity in clock generation/driver circuits.

The circuit shown in Fig. 3.7 assures correct clock synchronization with a reset signal when coming out of reset. The moment the reset signal is disasserted, a synchronization pulse is generated and delivered to the frequency divider to set the flip-flop in the desired state. The reset signal must be synchronized to the MCLOCK by some other mechanism when it comes out of reset. Intel Corp. uses this scheme on its 32 bit microprocessor family [3.7].

A 4 phase non-overlapping clock generation scheme is shown in Fig. 3.8. Since there are 4 phases, we need 2 bits to encode all the states. The purpose of the last 4 latches is to remove the effect of the decoder delay so that the effective path from the external clock input to the driver can be reduced. The idea of using a reset signal replacing SYNC can also be applied here to reduce the number of clocks to distribute.

All the conventional schemes have a common drawback of an external clock passing through many stages of logic gates including the ones for the non-overlap time. The number of the logic gates to pass through is around 5-10 depending on the amount of capacitance it drives. Considering that a pipeline stage is composed of 20-50 equivalent logic gates, the delay time variation of the clock generator/driver due to process and temperature variations has significant effect. The worst-case clock skew between chips doubles because each side can vary in opposite

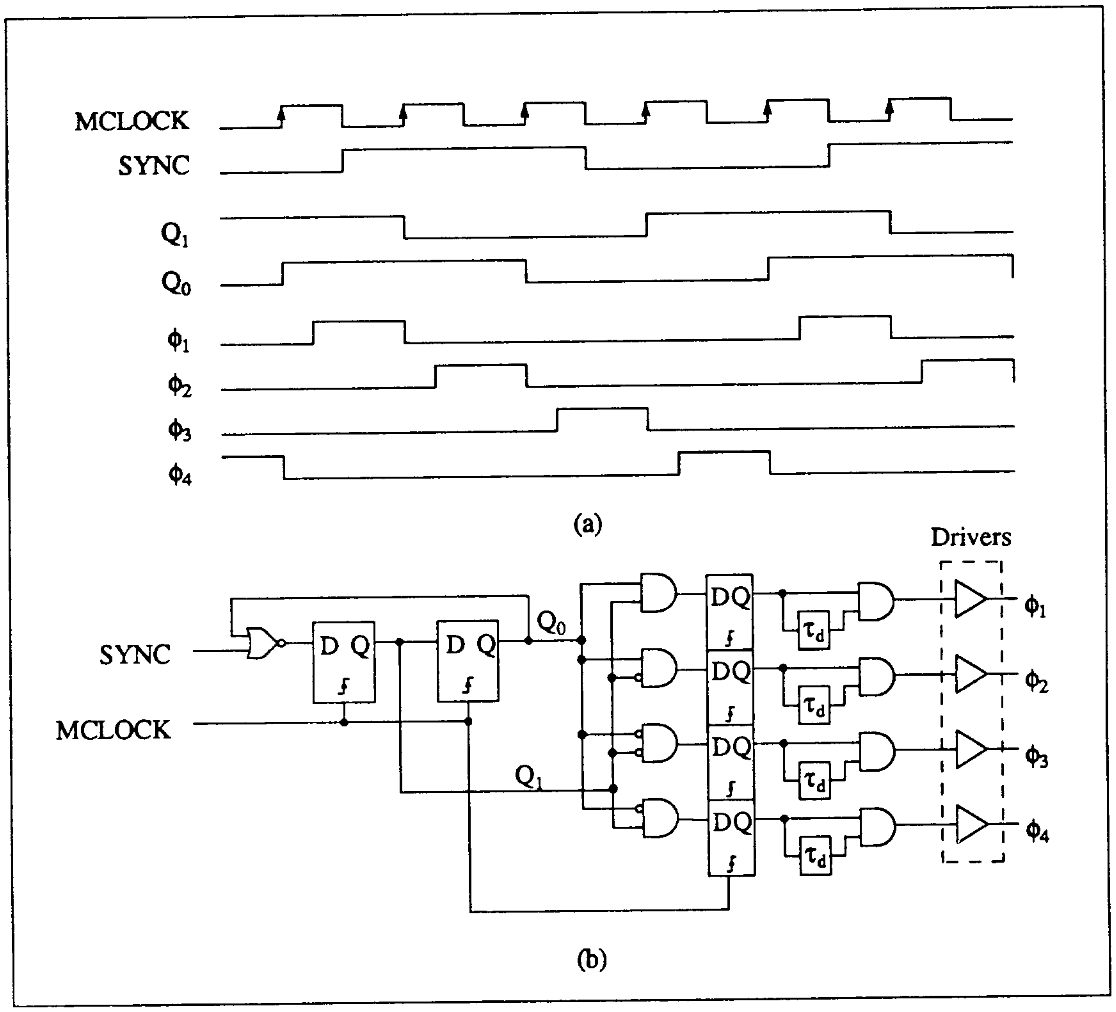

Fig. 3.8 A conventional 4 phase clock generator/driver. (a) Waveform. (b) Logic diagram.

directions. Assuming 30% variation in delay times, a delay equivalent to 3-5 logic gates is added to the pipeline with 6-25% performance degradation.

3.4.2 Direct Drive Scheme

A brute-force way of reducing clock skew is not to use buffers at all. An external clock generator has a huge drive capability of driving all the clock load capacitances without using any on-chip buffers. Since all the clock nodes are connected by a single wire, the only clock skew is due to the electromagnetic propagation delay. This is tolerable because the delay is very small and controllable. However, there are several implementation problems associated with this method. First, only one clocking scheme directed by a clock generator is allowed. Each chip does not have any flexibility of varying clock phases internally. Since we cannot afford to distribute many clock phases, the clocking has to be one phase or two phase non-overlapping. Also there is an technical problem of bringing a large current carrying signal onto the chip without adding any detrimental ringing to the signal. Since the total on-chip capacitance can easily exceed 20 pF, even a small series inductance due to a bonding wire will form a high Q resonant circuit. Several bonding wires should be connected in parallel to reduce the inductance. This scheme is adopted by the Clipper microprocessor family using a single phase clocking strategy [3.8].

3.4.3 PLL-Based Scheme - Delay Comparison

A phase locked loop (PLL) technique can be used for clock generation that can reduce inter-chip clock skew. First, I will describe a PLL-based clocking scheme that is very similar to a conventional PLL but has a voltage controlled delay line (VCDL) rather than a voltage controlled oscillator (VCO). It compares delay times rather than phases. Thus, I will refer to this as a delay locked loop (DLL) rather than as a PLL.

Fig. 3.9 shows a block diagram of a clock generator based on a DLL. Besides conventional clocks, a special clock named REF is distributed as a timing reference. In each chip, all clock inputs, other than REF, have controllable delay lines in series. Their delay times are adjusted so that an edge of the internal buffered clock phase is aligned with the edge of the REF clock. The self-adjustment mechanism with feedback is called a DLL. The range of the delay provided by the controllable delay should be determined by considering the difference between the maximum and minimum on-chip clock delay of the chips. Also, the phase lag of the REF clock from the original clocks should be large enough, so that on-chip clock generation can be completed during

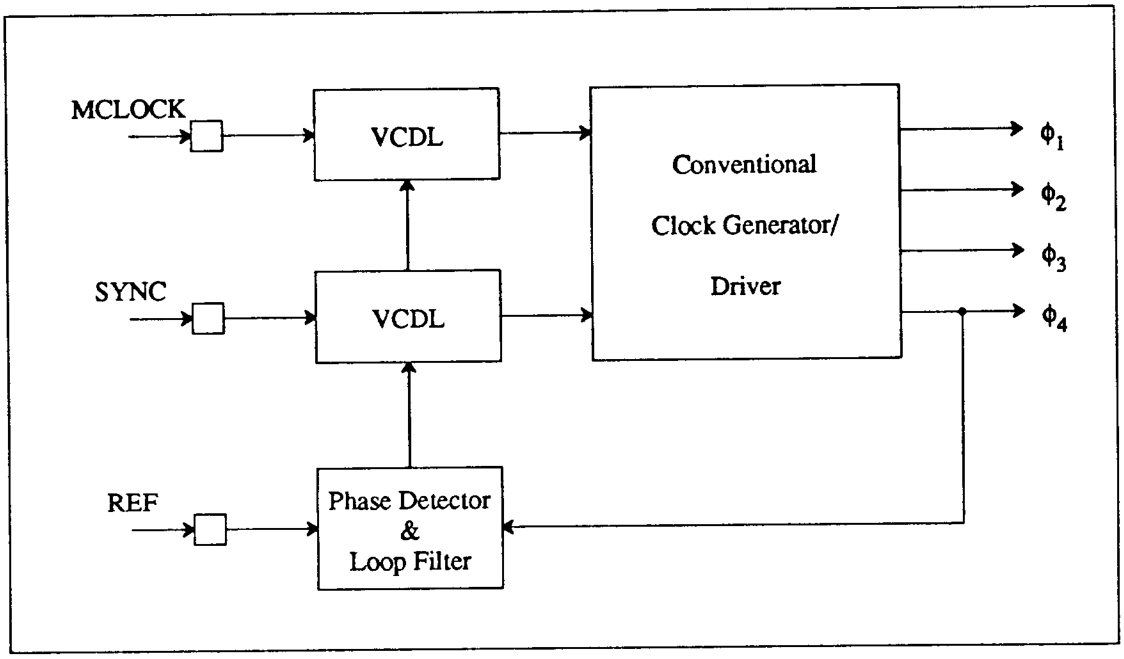
MCLOCK
SYNC
REF
VCDL
VCDL
Phase Detector & Loop Filter
Conventional Clock Generator/ Driver
ϕ_1
ϕ_2
ϕ_3
ϕ_4

Fig. 3.9 A DLL-based clock generator/driver.

the time. Another consideration that should be made is that the adjustable delay should allow pipelining: i.e., it should be able to contain many clock phases of MCLOCK. This can easily be done by using multiple delay elements in series rather than using a single element for the delay.

The primary advantage of using a DLL-based clocking scheme over conventional schemes is that this scheme removes clock skew among chips using a REF clock as a timing reference and forcing every clock edge to align with REF. It achieves the effect of a zero delay buffer among chips implementing this scheme.

However, a stability consideration has to be made since the DLL is a feedback control system. With a simple phase detector and loop filter combined with a charge pump, it is easy to formulate a stability criterion. The loop is a first-order system; its stability criterion is easy to satisfy. Its stability criterion can be found in [3.9]. Since the DLL requires a pull-in time from power-up before it settles into a steady state, it is preferable for the adjustable delay to have an upper bound which is less than a cycle time of the system.

There have been a few microprocessors that adopted this approach. Hewlett Packard's Spectrum microprocessors use this idea for 2 phase clock generation [3.10] and the MIPS chip set uses similar ideas to achieve coprocessor synchronization [3.11].

It is a challenging problem to integrate a DLL on a VLSI digital chip, since the DLL contains many analog components and careful considerations have to be made in circuit layout to reduce parasitic resistance and capacitance. Also, it is very difficult to keep the power supply clean from the noise generated from the digital part. Thus, a safer approach is to build a separate DLL chip for general use. All that a VLSI chip should do is to bring an internal clock phase, INTERNAL, off-chip without using a buffer.

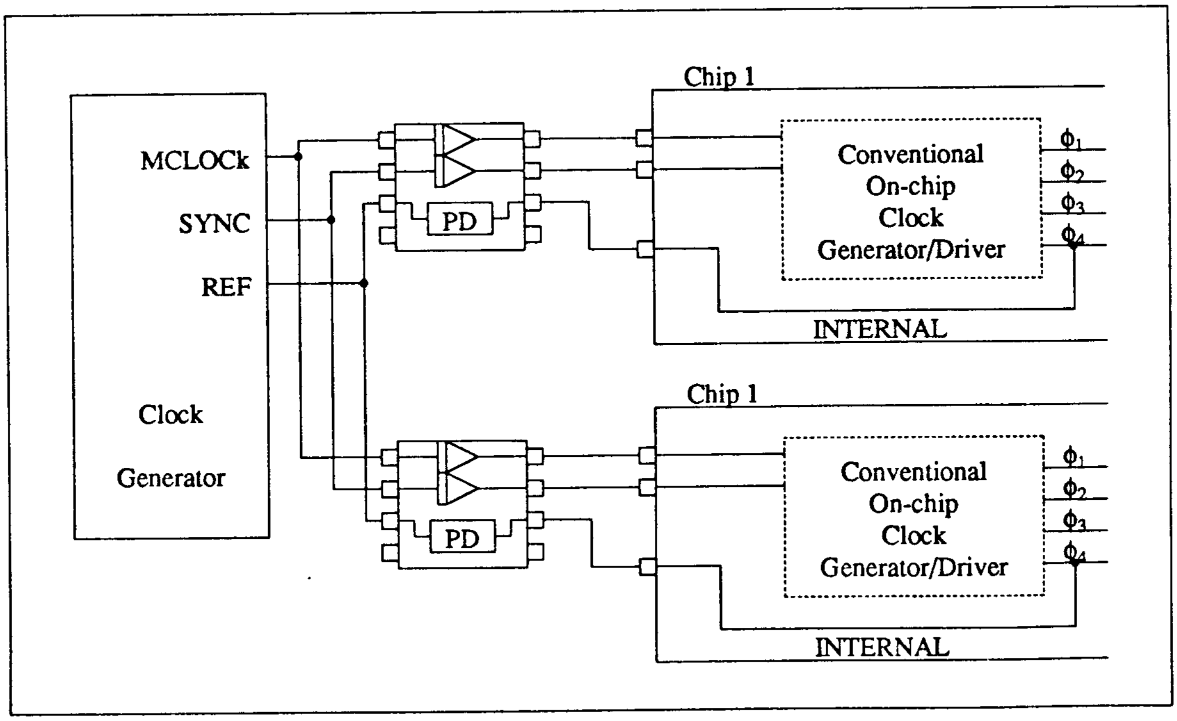

Fig. 3.10 Connection of clock synchronization chips.

The configuration of such a scheme is shown in Fig. 3.10. The DLL chip compares the phases between INTERNAL and REF, and adjusts the delay so that at steady state the two edges can be aligned. That chip's specification must include the maximum and minimum delay of the adjustable delay line, the minimum clock pulse width that can propagate through the delay line, lock-in time, maximum jitter, and so forth. Since it is separated from the VLSI, special analog design techniques can be applied using a different process technology, perhaps a double poly process to include a large capacitor.

3.4.4 PLL-Based Scheme - Phase Comparison

Instead of adjusting the delay from a reference clock, direct phase comparison can be made from the reference. By taking advantage of the extremely accurate phase tracking capability of charge pump PLLs [3.12-3.13], an edge of the internal clock is accurately aligned to an edge of the external clock (Fig. 3.11). This is accomplished by directly comparing the two phases through a sequential phase/frequency detector.

Correct synchronization between chips is achieved regardless of the clock generator/driver delay and its process and temperature variations. All the sensitive circuit elements, including clock driver, are within a negative feedback loop. The effect of the variations is tracked and removed by the PLL. The VCO is composed of a multi-stage tapped delay line that is automatically calibrated to a precise delay per stage. The generation of arbitrary multi-phase clocks is possible with proper decoding of the signals from the delay line taps.

This section describes the design principle and circuit techniques of a 4 phase non-overlapping clock generator.

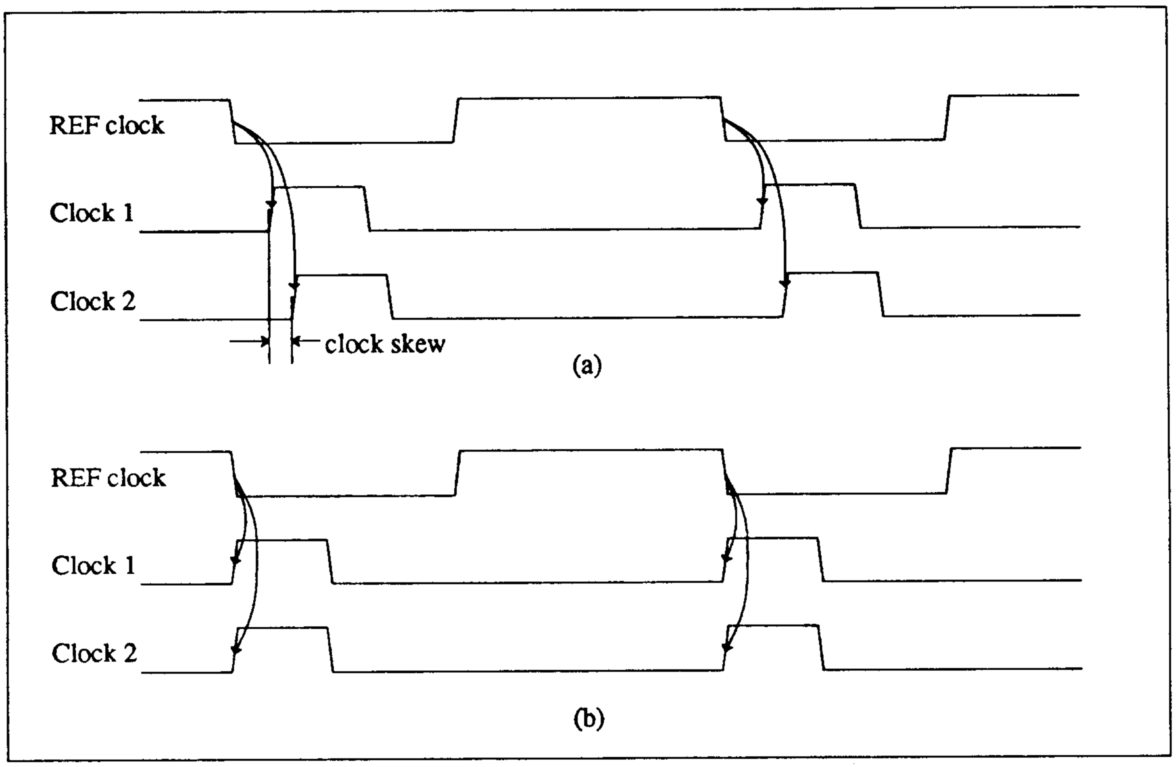

Fig. 3.11 Clock generation from an external reference clock. (a) Conventional scheme. (b) PLL-based scheme.

3.4.4.1 Clock Generation Circuits

A charge pump phase locked loop is used to derive an on-chip 4-phase non-overlapping clock from an off-chip clock. In addition to precisely determining timing relationships between internal clock phases, it is also used to eliminate clock skew between the reference (REF) clock and the internal clock. The rising edge of ϕ_1 is aligned to the falling edge of the REF clock to ensure correct synchronization throughout the chip set, i.e., all of the internal clocks in the different chips are in phase with respect to the falling edge of the REF clock.

Due to the unavoidable process- and temperature-dependent delays of the clock buffers used to drive large on-chip capacitive loads, the above requirement is hard to meet without using a charge pump PLL One important advantage of a charge pump PLL is that it is capable of

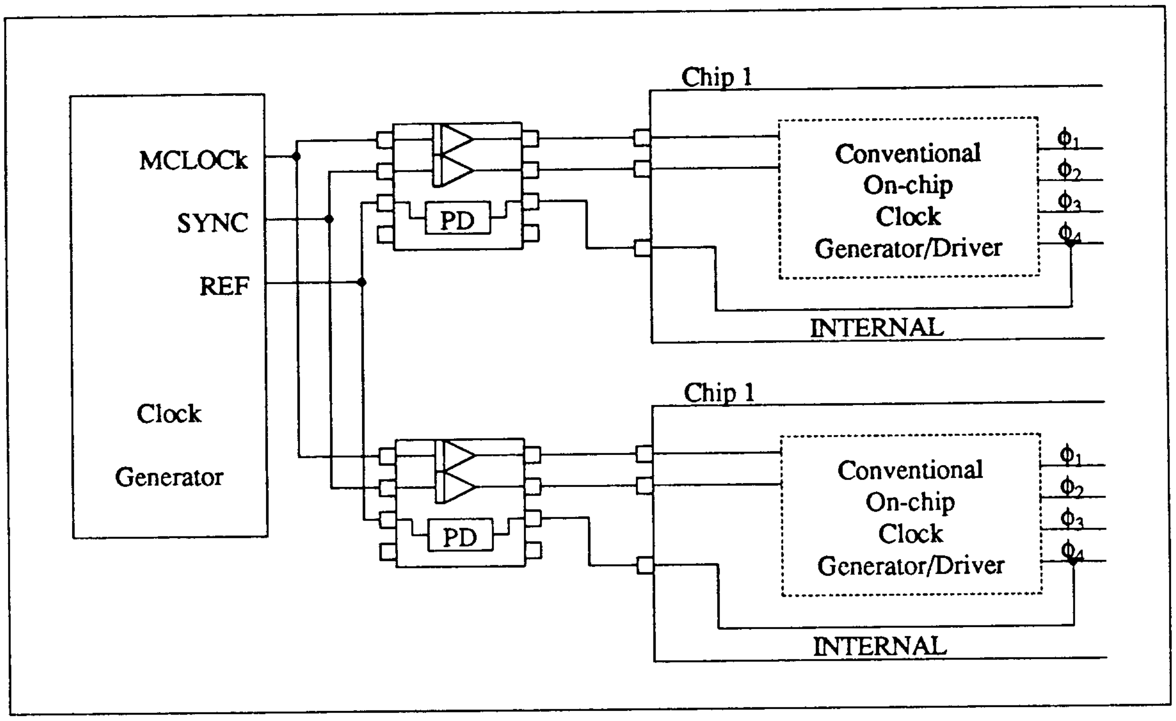

Fig. 3.10 Connection of clock synchronization chips.

extremely accurate phase tracking with a passive RC loop filter. Nominal phase error can be practically zero, regardless of its input frequency. On the other hand, conventional PLLs (for example, linear PLLs using an analog multiplier as a phase detector) have a finite phase error which is a function of input frequency [3.14]. The charge-pump PLL system shown in Fig. 3.12 eliminates clock skew due to the clock buffer simply by ensuring the same inverter delay between the ϕ_1 - phase/frequency detector (PFD) path and the REF clock - PFD path.

The remaining sources of clock skew are the mismatch of the inverter delay paths and the jitter of the PLL. Differences in the delay through the inverter paths can be minimized by using the same layout and orientation for the circuitry of both paths. The main causes of the jitter are leakage current, and noise in the VCO. Since MOSFET devices are used to realize the VCO, infinite input resistance can be assumed. the finite input resistance can be ignored. Junction leak-

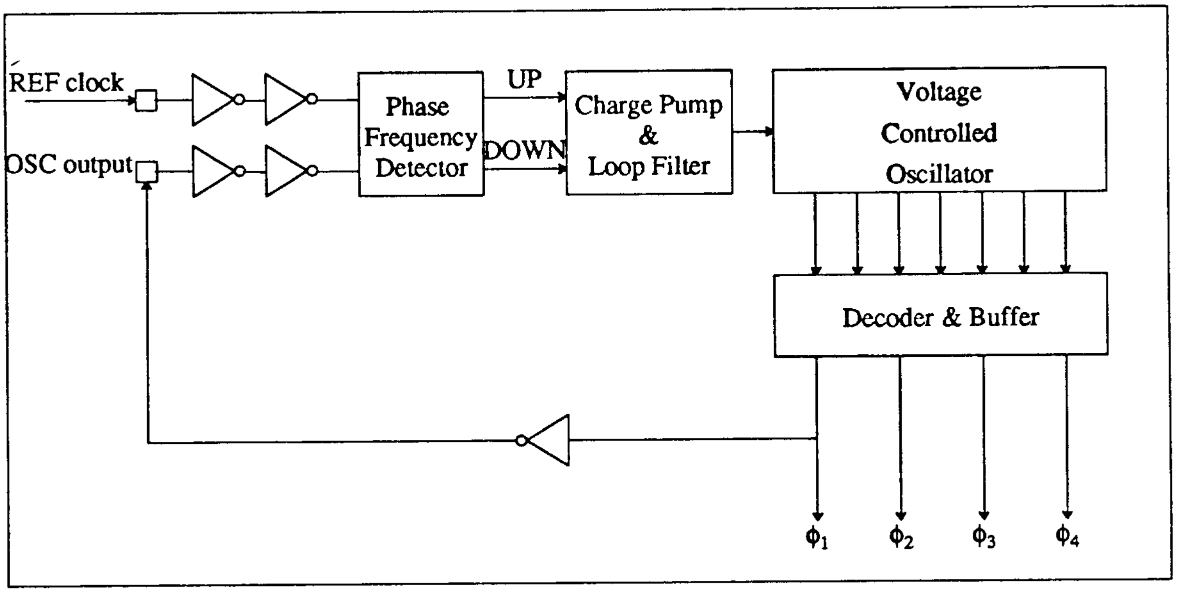

Fig. 3.12 Block diagram of a new clock generator/driver based on the charge pump PLL.

age current in the pico-ampere range makes phase jitter negligible. Also, VCO noise due to the 1/f noise of the MOS transistors becomes negligible at the high clock frequencies typically used in high-performance digital circuits. Thus, by careful design of circuits and layout, clock skew between the REF clock and the internal clock can be made nominally zero.

A well-known sequential-logic PFD shown in Fig. 3.13(a) is used for phase/frequency detection [3.14-15]. Since it has a memory to compare frequency as well as phase, the PFD is free from false locking to the second or third harmonics. Its outputs are UP and DOWN signals. When the falling edge of the REF clock leads the falling edge of the VCO output (OSC), UP is activated to low level until the falling edge of OSC arrives. Similarly, DOWN is activated when OSC leads REF. Both UP and DOWN are deactivated to high level when the loop is in a perfectly locked state. In no case are both of the signals activated. UP and DOWN are connected to the charge pump and loop filter of Fig. 3.13(b). These are comprised of three inverters, MOSFET switches, and a passive RC lowpass filter. The passive RC filter is provided off-chip so that a

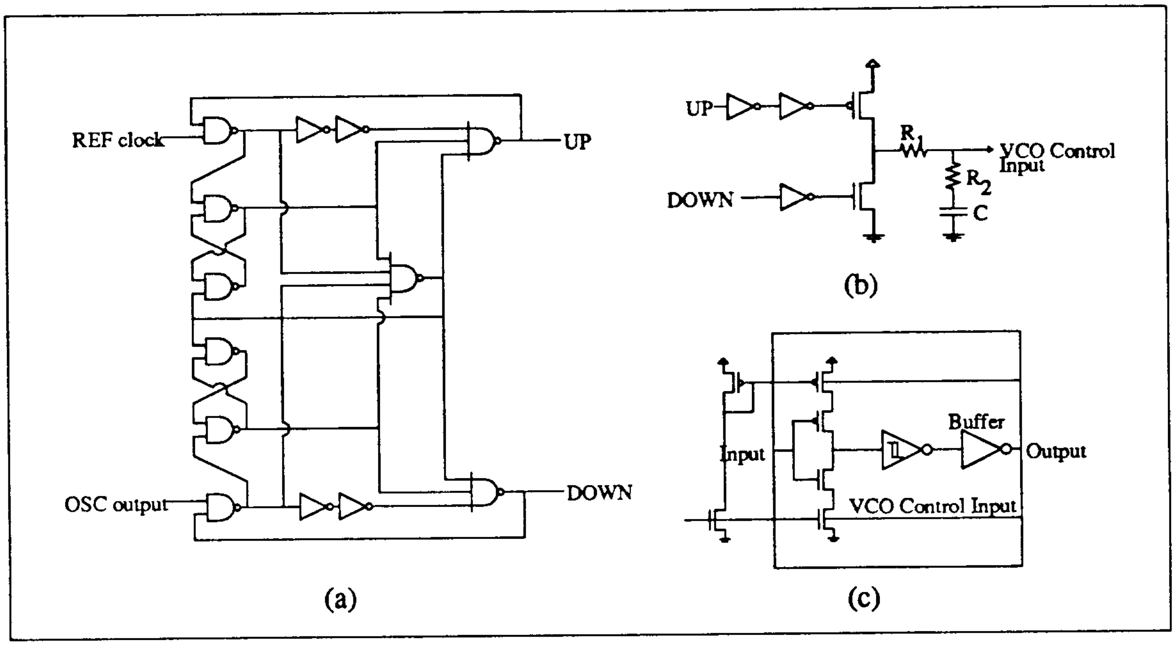

Fig. 3.13 Basic circuit elements. (a) Sequential logic phase/frequency detector. (b) Charge pump and loop filter. (c) Delay cell.

large capacitor can be used to guarantee stable operation over a wide frequency range and to provide flexibility in choosing the loop filter parameters.

The VCO is implemented as a simple ring oscillator. A series connection of delay cells forms a tapped delay line the outputs of which are used to derive the 4-phase non-overlapping clock. Its oscillating frequency is determined by the delay time of the basic cell (Fig. 3.13(c)) and the number of stages involved. The delay time of each cell is determined by the amount of current supplied through the current source, the input capacitance and the threshold of the Schmitt trigger. The two symmetric current sources are controlled by the VCO control voltage (i.e. the voltage across the loop filter capacitor in the steady state). To compensate for the asymmetry due to the difference in mobility of electrons and holes, a width ratio of 2:1 is maintained in all of the clock generator circuitry. In applications where the timing accuracy is even more critical, a pair of delay cells can be used to obtain perfectly symmetric delays. The Schmitt trigger is included to achieve fast rising and falling outputs at low frequency. These signals become the inputs to the clock derivation circuitry.

The clock derivation scheme is depicted in Fig. 3.14. To get an odd number of inversions, one extra inverter is attached to the last stage. The amount of extra delay time due to the inverter is small compared to the total delay of the cells. Output waveforms of each of the delay cells can be assumed to be symmetric (50% duty cycle). ϕ_1 is derived by ANDing the outputs from the delay cell 1 and 4, ϕ_2 from delay cell 5 and 8, and so on. The ratio of clock high time to non-overlapping time is 3:1 throughout the entire operating frequency, regardless of the variations of process and temperature. Clock buffers consisting of cascaded inverters are used to drive on-chip capacitive loads of up to 3 pF with a rise/fall time of less than 2ns. A separately buffered OSC output signal is derived to be fed back to the phase detector. By comparing the phases of the buf-

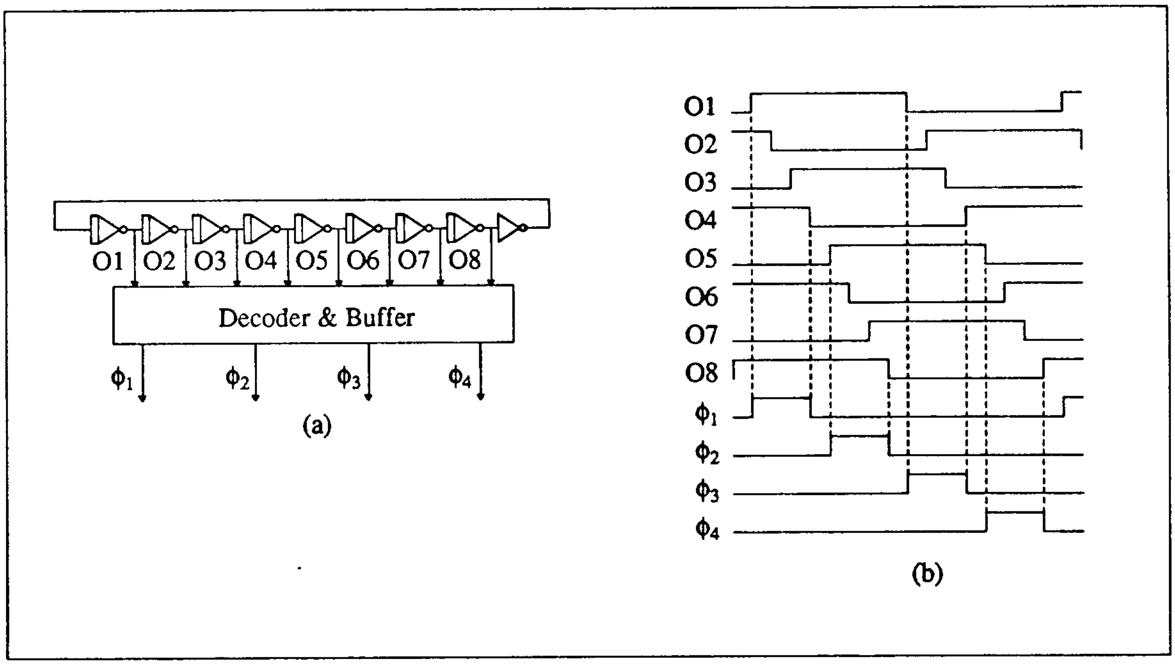

Fig. 3.14 Derivation of 4 phase non-overlapping clock. (a) Block diagram. (b) Waveform generation.

fered clock with the REF clock rather than the original derived clock, internal clock edges are accurately aligned to the input REF clock edge regardless of the buffer delay time. This achieves correct synchronization across the chip set regardless of parameter variations as long as the time constant of the PLL is small enough to track the changes.

3.4.4.2 Stability Analysis of a Charge Pump PLL

A complete analytical stability criterion for this particular type of PLL is difficult to derive because it has both linear and nonlinear elements and operates in the time-varying sampled-data domain. A simplified stability analysis for the second and third order PLL is presented in both the s- and z- domain in the literature [3.11]. The following analysis is, therefore, an extension of those in the literature. When we include the effect of logic delay in the simplified analysis of the second order loop filter in the z-domain, the stable operating condition becomes

$$K\tau_2 < \frac{1}{\frac{\pi}{\omega_i\tau_2}\left[\frac{\pi}{\omega_i\tau_2}+1-\frac{t_d}{\tau_2}\right]},$$

where $K = K_oR_2I_p$, K_o = VCO gain in MHz/V, $I_p = \mathrm{Max}\left[\frac{V_{cc}-V_x}{R_2}, \frac{V_x}{R_2}\right]$, maximum pumping current in A, V_x = average VCO input voltage, ω_i = input frequency in rad/s, $\tau_2 = R_2C$, and t_d = logic delay time.

In the continuous time domain, which assumes average time-continuous behavior, the phase margin is calculated by

$$PM = \tan^{-1}\left[\frac{K\tau_2}{\sqrt{2}}\left\{1+\left[1+\left(\frac{2}{K\tau_2}\right)^2\right]^{0.5}\right\}^{0.5}\right] - \frac{360^\circ}{2\pi}\frac{K\tau_2}{\sqrt{2}}\left\{1+\left[1+\left(\frac{2}{K\tau_2}\right)^2\right]^{0.5}\right\}^{0.5}\frac{t_d}{\tau_2}.$$

Note that the last equation is not a function of input frequency. Since the first inequality is based on the z-domain analysis of transfer functions, including the effect of a time-varying sampled-data characteristic, it is stricter than the s-domain criterion. Therefore, the second criterion is valid only when the first condition is met. Derivation of the first relation is based on the z-plane pole-zero diagram shown in Fig. 3.15. When z_1 is less than 0, the loop becomes unconditionally *unstable*. Since z_1 is given by

$$\frac{\tau_2 - t_d}{T + \tau_2 + t_d},$$

where T = period of the input frequency, another necessary condition, $\tau_2 > t_d$, must be added for stability. The above criteria for stability are drawn in Fig. 3.16.

Some important points are noticeable in the figure. The loop goes to a safer region as we increase the input frequency as long as the other parameters remain the same. Also, when we increase the phase margin by further increasing the loop gain, K, the loop may go into an unstable region in the z-domain. The most critical component affecting stability is R_2. As is shown in Fig. 3.17, when $R_2 = 0$, or ∞, the loop becomes unstable. There is a limited range of values for R_2 for stable operation. As we increase C, the loop goes into a more stable region as well as increasing

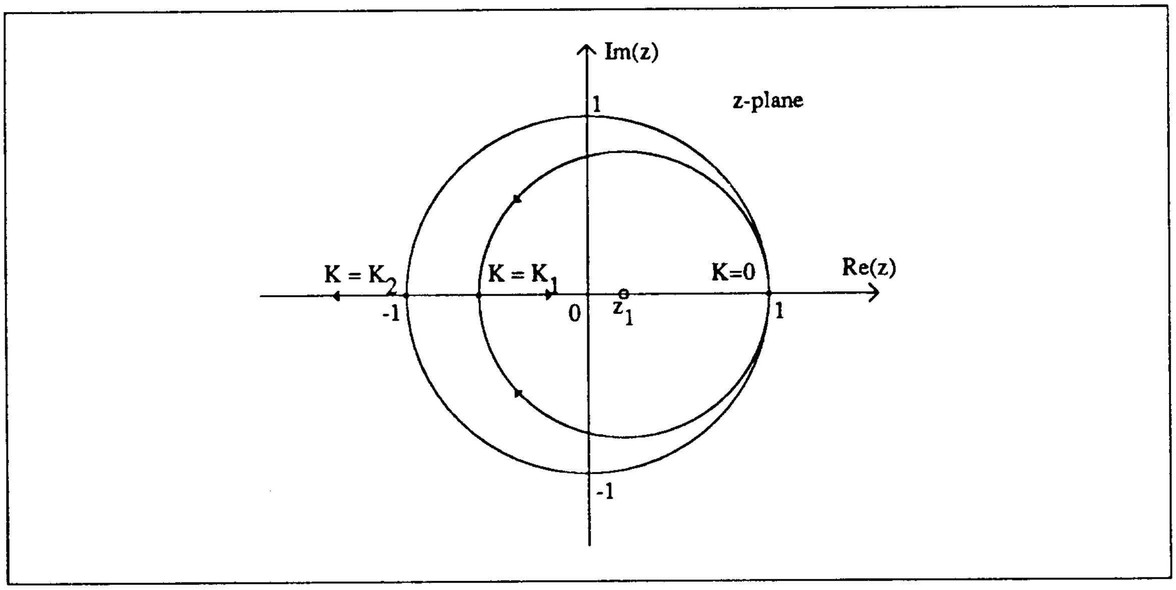

Fig. 3.15 Root locus of the second order PLL in z-domain. Loop stability conditions are $z_1 > 0$ and $K < K_2$.

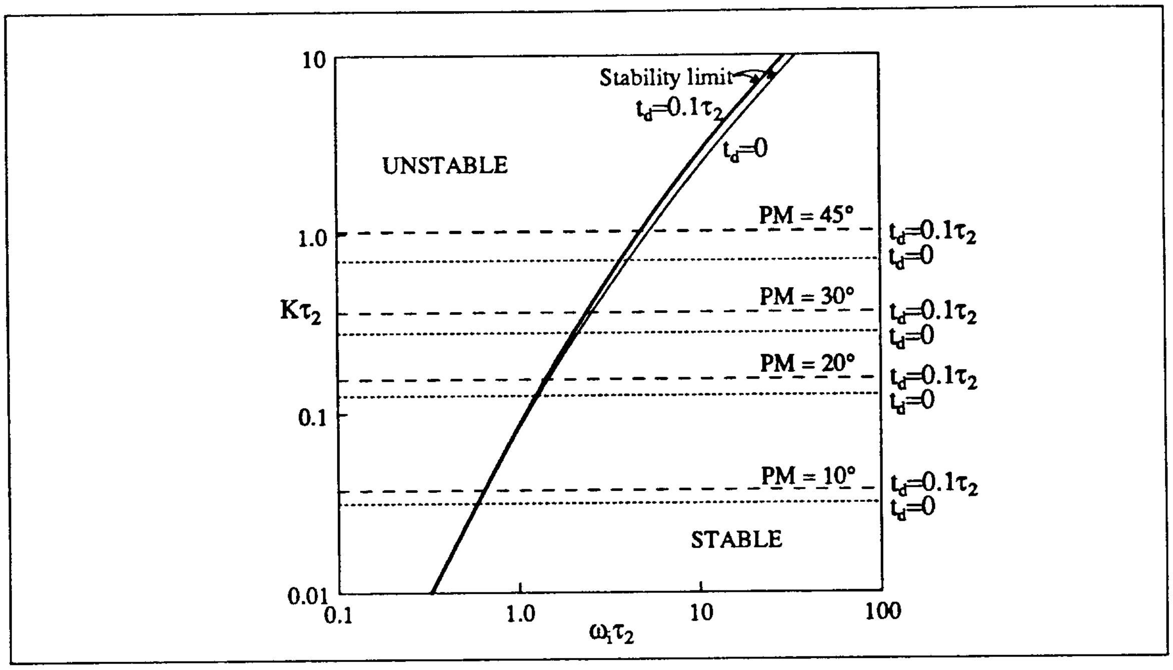

Fig. 3.16 Stability limit and phase margin.

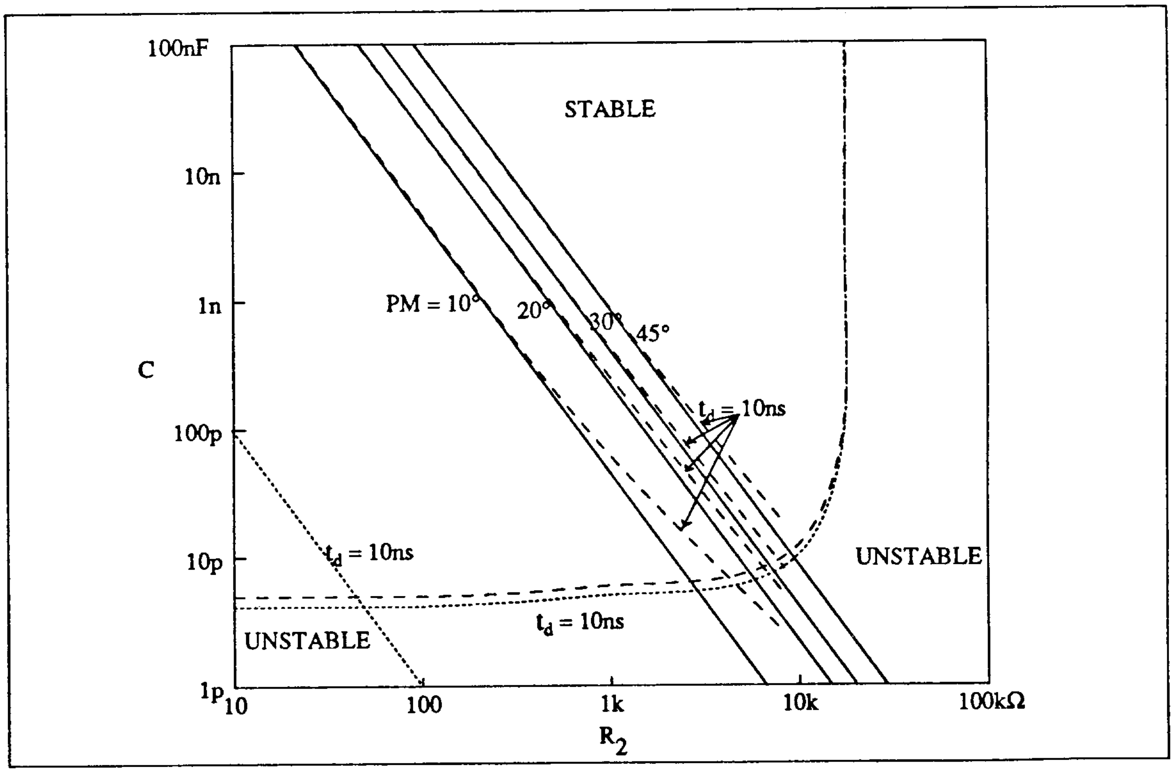

Fig. 3.17 Stability limit and phase margin as a function of R_2 and C. ($\omega_i = 2\pi$ 6.7MHz, I_p = 70μA, K_o = 12MHz)

the noise margin, at the expense of an increased start-up time. Logic delay time in the order of tens of nanoseconds does not degrade the overall stability much as long as $t_d \ll 0.1\tau_2$, which can be easily satisfied.

In our application, efforts were made to get as wide an operating frequency range as possible so as to enable low frequency testing of the chip. The capacitor was not integrated on chip so that a large value for C could be provided externally. Since the above analysis ignores many parasitic effects such as the input capacitance of the VCO, VCO gain non-linearity, an asymmetric charge pump, and so on. With this in mind, we have to include a considerable margin for safe operation. The chosen loop filter parameters are $R_1 = 50\,k\Omega$, $R_2 = 100\Omega$, and $C = 0.1\mu F$. With this large capacitance, we get only about 48° of phase margin, assuming $K_o = 12MHz/V$, $V_x = 1.5V$, $\omega_i = 2\pi 6.7MHz$. A simulation program was written to plot the transient behavior and check the stability conditions. Two simulation results are shown in Figs. 3.18 and 3.19. With the above parameters, it takes about 16,000 cycles (2.4 ms) for the PLL to start up into the steady-state of less than 1° of phase error at 6.7MHz.

3.4.4.3 Experimental Results

The entire PLL system, except for the loop filter, was designed and fabricated in 2 μm n-well CMOS technology.

Fig. 3.20 shows the chip microphotograph. The active area is about 0.4 mm^2, excluding pads. Fig. 3.21 shows the measured VCO characteristic. At room temperature, the maximum oscillating frequency is 24MHz with $V_{dd} = 5.0V$. The VCO control voltage, V_x is 1.5V at 6.7MHz and its gain is about 12 MHz/V at this operating point. With the chosen loop filter parameters, it operates from 15 kHz up to 18 MHz. Fig. 3.22(a),(b) shows the clock waveform operating at the maximum and minimum frequency. We may extend the maximum operating frequency by reducing the number of stages or eliminating the Schmitt trigger in the delay cell, at the expense of

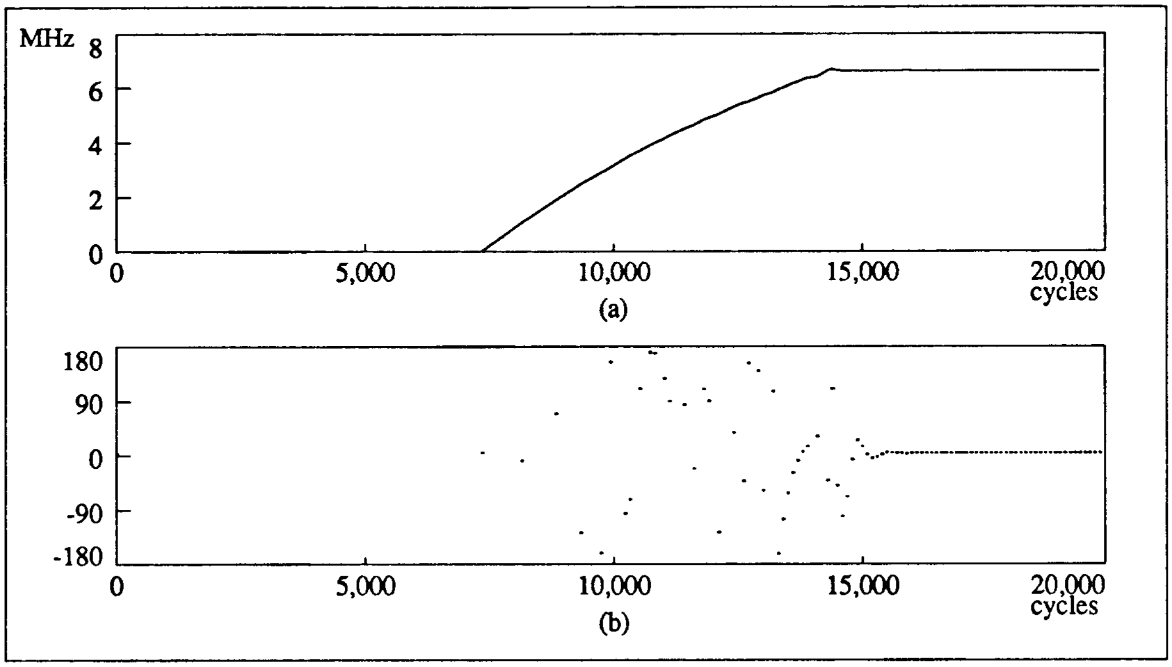

Fig. 3.18 Simulated response ($\omega_i = 2\pi\ 6.7$MHz, $R_1 = 50k\Omega$, $R_2 = 100\Omega$, $C = 0.1\ \mu F$, $K_o = 10$ MHz/V @ $V_x = 1.5$ V). (a) Frequency response. (b) Phase response.

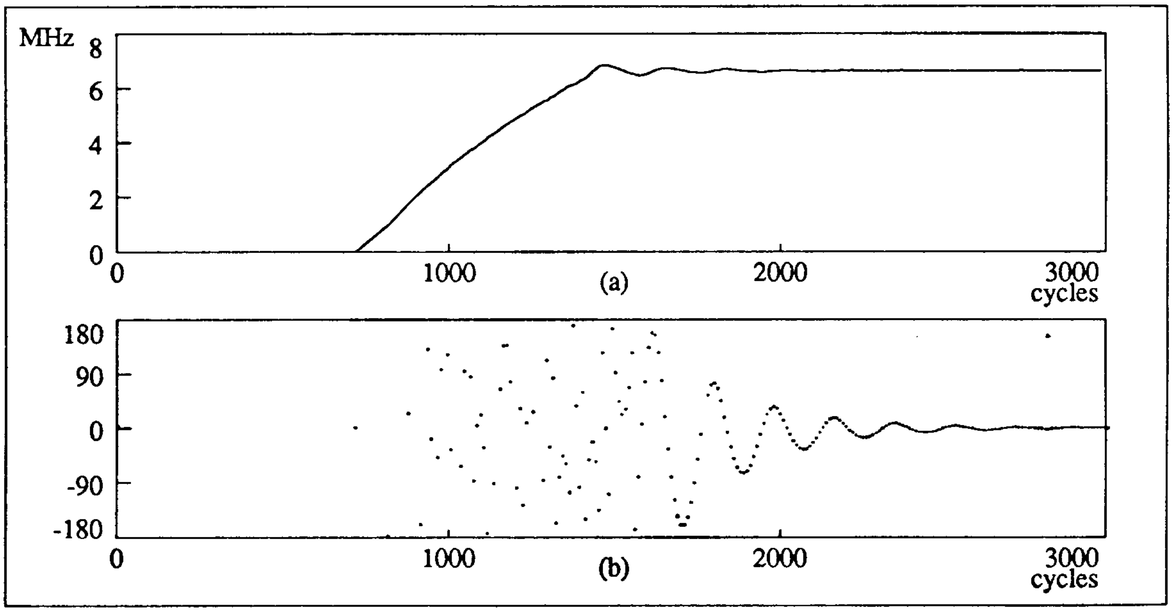

Fig. 3.19 Simulated response ($\omega_i = 2\pi\ 6.7$MHz, $R_1 = 50k\Omega$, $R_2 = 100\Omega$, $C = 0.01\ \mu F$, $K_o = 10$ MHz/V @ $V_x = 1.5$ V). (a) Frequency response. (b) Phase response.

degrading timing accuracy. At the minimum frequency and up to 100kHz, non-negligible jitter (less than 5°) was observed. Jitter is caused by VCO noise and loss of charge in the capacitor because of the leakage current of the junctions during the longer clock period. Also, at low frequency, it was found that ϕ_3 does not appear exactly in the middle of the clock period. This is believed to be due to inexact delay time of the delay cells at an extremely small current level. This effect can be reduced by including a capacitor at the input of the Schmitt trigger, thereby increasing the overall current level at the expense of decreasing the maximum operating

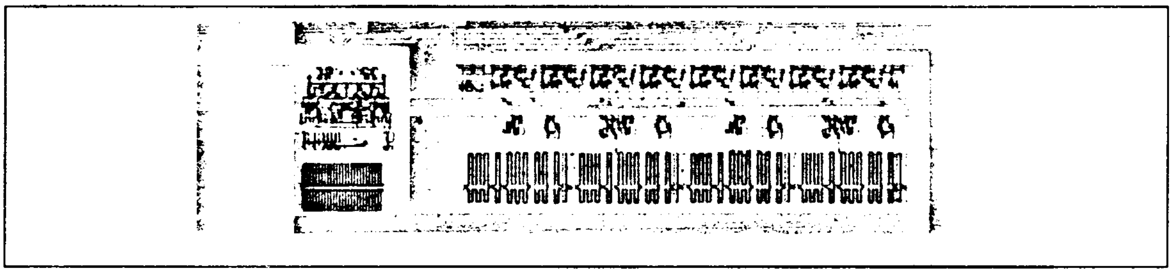

Fig. 3.20 Chip microphotograph.

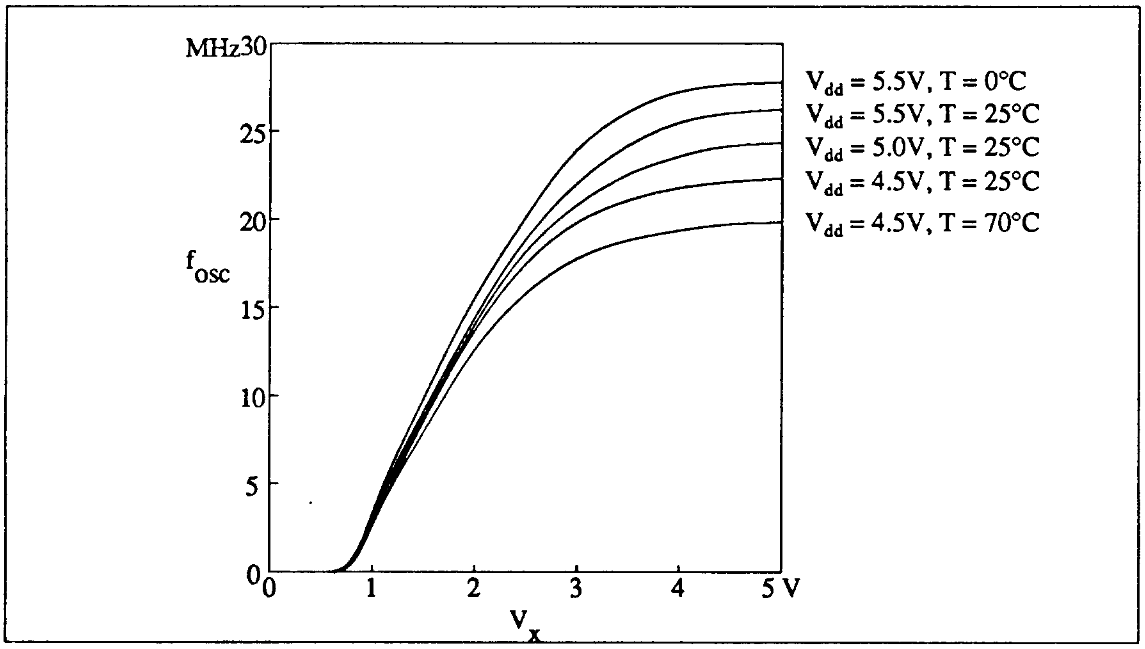

Fig. 3.21 Measured VCO Characteristic.

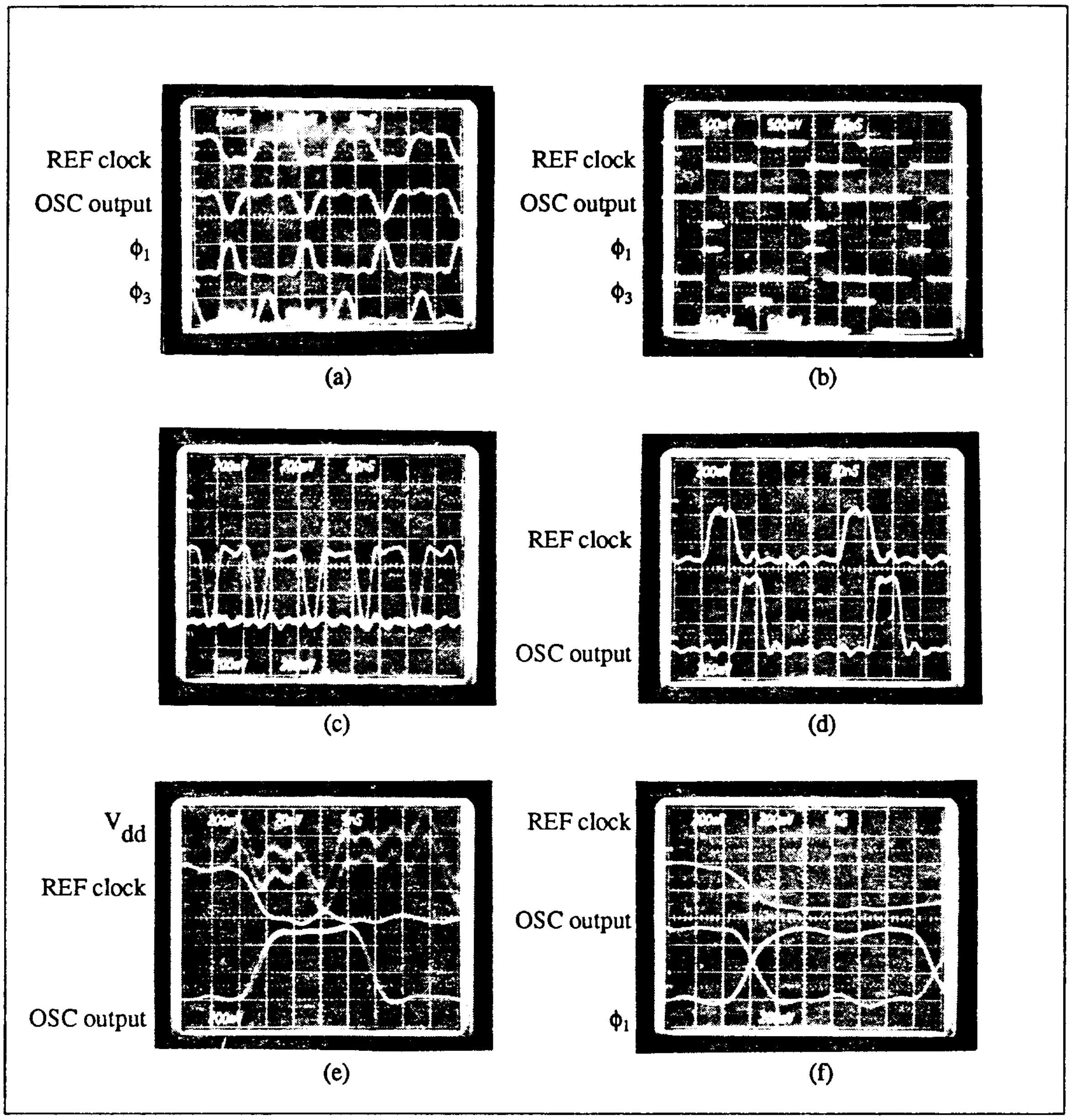

Fig. 3.22 Various waveforms of the PLL-based clock generator/driver. (a) Output waveform at maximum operating frequency (18 MHz). (b) Output waveform at minimum operating frequency (15 kHz). (c) Four phase non-overlapping clock at 6.7 MHz. (d) Effect of a skewed input pulse. For an input pulse skewed by 20 ns, the resulting output pulse skew is less than 1 ns. (e) Effect of power line fluctuation. Upper trace shows the power line fluctuation due to external and internal noise sources. Resulting jitter in the output is less than 2 ns. (f) Edge alignment between external reference clock and ϕ_1.

frequency. For 1 MHz up to 18MHz, no noticeable jitter was found on the oscilloscope trace. Fig. 3.22(c) shows the generated 4 phase non-overlapping clock operating at 6.7MHz, which is the nominal operating frequency of the SPUR chip set. The ratio of clock high time to non-overlap time is maintained constant in all 4 phases. The effect of input noise can be seen in Fig. 3.22(d). One input pulse is skewed by 20ns in the input pulse train. The resulting maximum skew of the output pulses appearing in the next cycle is less than 1ns. This indirectly shows that the immunity to clock input noise is good enough for this application. Also, the effect of power supply fluctuation (for example, ripple from the switching power supply) can be seen in Fig. 3.22(e). Upper trace shows the V_{dd} fluctuation caused by an externally applied 200kHz, 200mV square wave signal and a self-induced signal. The output pulse edge contains jitter of less than 2ns. In case the power supply fluctuation is severe, a separate, quiet supply line should be provided to prevent the interaction of the PLL with the power line signal. Fig. 3.22(f) shows the edge alignment, REF clock and OSC output being the two inputs of the phase frequency detector. The rising edge of ϕ_1 is aligned to the falling edge of the REF clock by less than 2ns skew. This shows more than acceptable synchronization between the on-chip clock and the off-chip reference. Clock skew is less than 2ns from 1 MHz up to the maximum operating frequency of 18MHz. At high temperature (70°), maximum operating frequency is reduced to about 14MHz with $V_{dd} = 4.5V$. Other than that, no significant performance degradation was observed.

3.4.4.4 Comparison with Other Clock Generation Schemes

The PLL-based scheme with phase comparison has several advantages over conventional ones. First, it achieves the effect of zero-delay buffer by comparing and aligning an internal clock edge with the externally provided reference, thereby reducing clock skew. Second, it does not need to distribute many clock phases. Only one clock edge, regardless of the duty ratio, is required for distribution, while most other schemes require more than one external clock. All that is needed is to select one from its internal phases to synchronize with the other chip's phase - for example, a sampling clock phase would be preferable because it can be common to every chip. Third, it is flexible in generating many phases internally. By tapping a delay line, many internal timings are available. Also, its internal clock phases can be generated in each chip regardless of the other chips' clock phases as long as a synchronizing edge is properly selected.

One of the disadvantages of this technique is that it includes special analog circuits on the digital chip. It requires a sophisticated design technique that should take into account such factors as stability, effect of power line fluctuation, and so forth. Without special technique, the filter capacitor must be supplied externally and clean supply lines should be provided, requiring extra pins.

Even if the PLL-based scheme and the DLL-based scheme achieve the similar goal of simulating a zero-delay buffer, there are several differences. First, the PLL-based scheme requires only a single clock to be distributed across the system, where as multiple clocks are needed in the DLL-based scheme. Additional complexity is involved in distributing more than one clock in a system. However, there are a few advantages of using the DLL than the PLL. First, it is easier to achieve stability in the DLL than in the PLL. Since the DLL uses a voltage controlled delay line, no integrator is in the feedback loop, i.e., the amount of delay change is directly proportional to the control voltage change. On the other hand, in the PLL, a control voltage change in the VCO will result in the frequency change, which causes not only the current phase but its effect will double in the next cycle and triple in the third cycle and so on. That is because the comparator compares phases while the VCO controls frequency that is an integral of phase. Thus, 90 ° phase shift is involved in the PLL which makes its loop filter design non-trivial while the DLL's filter design is easy since it is a first order system. Thus, in the DLL, it is possible to integrate the loop filter on-chip. Second, it behaves quite predictably during reset on the power up. The internal clock phases generated during the power up in the PLL-based scheme do not have correct frequency or phase. However, during power up, internal clock phases from the DLL-based scheme may not have a correct phase relationship until steady state, but correct frequency is guaranteed all the time. Third, it is possible to stretch a clock phase in the DLL. Since the loop tries to equalize the delays, pulse repetition rate can change at any time. Although not a good practice, it is possible to stall a system for a few cycles by holding some of the clock phases. On the other hand, the PLL scheme does not allow its frequency to change unless its rate of change can be followed by the loop.

3.5 Conclusion

In this chapter I described various problems in clock generation and distribution, and clocking strategies to cope with such difficulties. Two approaches using a PLL technique have been described as an effective way of making a zero-delay buffer to reduce clock skew. In the DLL-based scheme, path delays are adjusted so that all chips participating in the scheme have the same delay from a clock source, thereby removing clock skew. The PLL-based clock generator with tapped delay line compares the edge of an internal buffered clock with the external reference clock edge. Its circuitry is fully described and experimental results are presented. Although the DLL is easier to implement, the PLL has the finer characteristics of requiring only one external clock and having a flexible on-chip clock generation scheme.

CHAPTER 4

Synchronization Circuits

This chapter concerns the problems, trade-offs, and circuit techniques of building a synchronizer with performance matching the system's requirement. Section 4.1 provides a summary of previous work on the theory and experimental results for synchronizers. In section 4.2, a general model is proposed for characterizing the performance of a synchronizer. Based on the model, synchronization strategies and their trade-offs are compared in section 4.3 with respect to implementation difficulty and performance. Section 4.4 discusses circuit techniques for implementation which can fully exploit CMOS and bipolar process technologies. Finally, section 4.5 concludes this chapter.

4.1 Background

A synchronizer is an element that brings an asynchronous external signal into the domain of a synchronous system. To be qualified as valid in a synchronous system, a signal has to meet two requirements: proper timing with respect to a clock and a proper voltage level. A *time-qualified* signal is a signal which does not make transitions until the evaluation of the signal is finished at the state storage nodes. A *value-qualified* signal is a signal which maintains an unambiguous voltage level until the evaluation of the signal is finished at the state storage nodes. Synchronization is a process of converting an asynchronous signal to a signal both value-qualified and time-qualified with respect to the system clock. If an asynchronous external signal is introduced into the synchronous system, without proper synchronization, the two requirements cannot be satisfied and problems occur.

Fig. 4.1 shows a finite state machine that receives an external asynchronous signal without synchronization. If the external signal changes its state in the middle of the cycle, a fast path to the state bits will propagate a new value and a slow path to the other state bits will not have

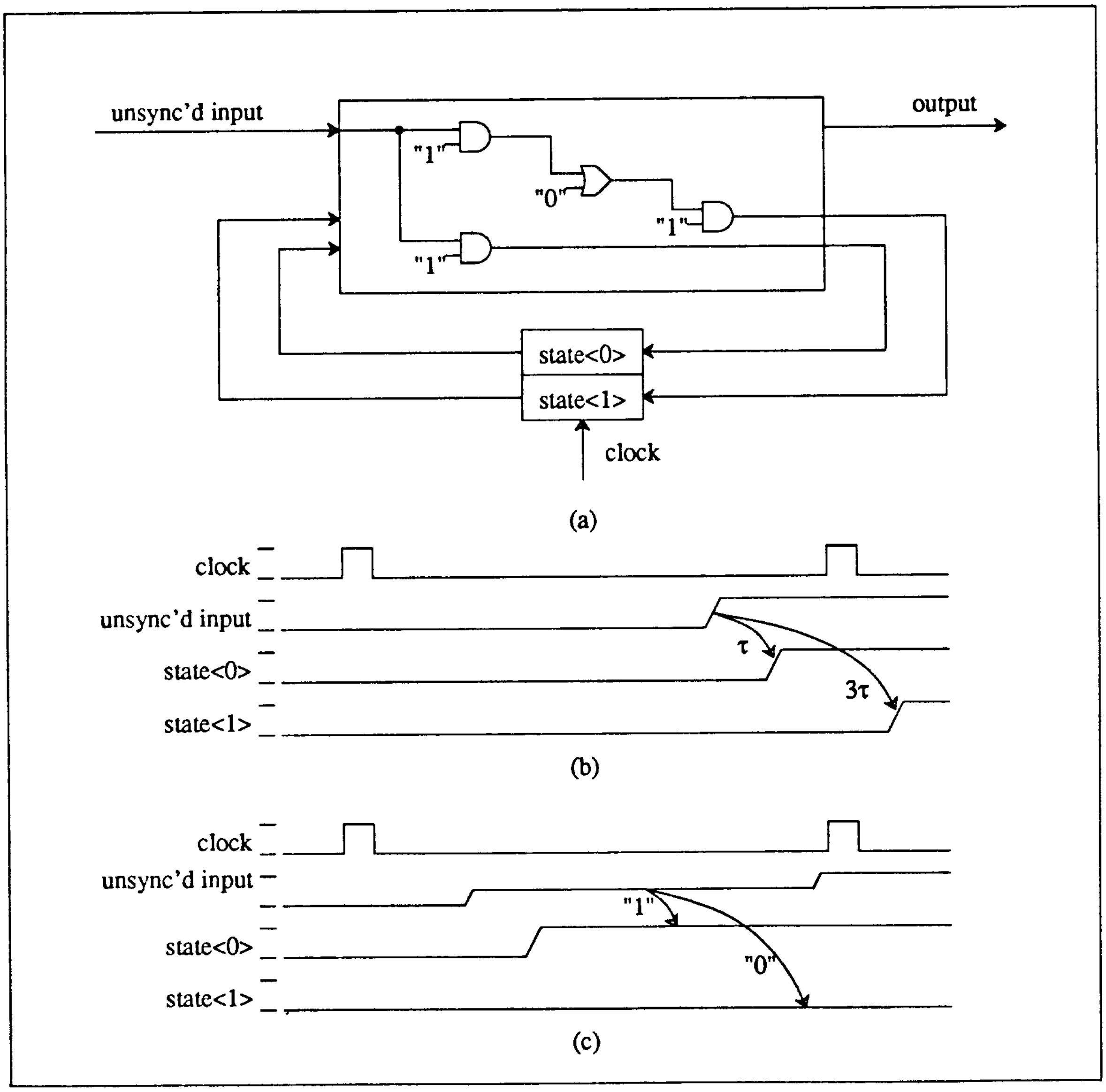

Fig. 4.1 Synchronous system with an unsynchronized input. (a) Finite state machine with an unsynchronized input. (b) Inconsistent state resulting from a time-unqualified input. (c) Inconsistent state resulting from a value-unqualified input.

enough time to propagate the new value, thus resulting in inconsistency. Similar problems arise when the input remains around the logic threshold region for more than a fraction of a cycle. Since the circuit implementations of gates do not exactly guarantee the same threshold for all the gates across the system, some part of the path may interpret the input as a logical one after amplification through subsequent stages of gates, while the other path behaves in the opposite way, resulting in inconsistency.

There have been reports of improper operation of D-FFs used as synchronizers [4.1-4.10]. Researchers found that D-FFs used as synchronizers had a longer delay time than normally required for certain combinations of input and clock transitions. A simple model and its failure analysis were developed which closely matched experimental results [4.11]. One important conclusion about all of those works was that it seemed impossible to remove the probability of a synchronizer having unbounded delay time for certain transitions of two independent signals - a clock and an asynchronous input. Logic designers found it hard to accept the fact that a simple operation like synchronization cannot be done perfectly. An attempt to build one using a Schmitt trigger failed experimentally [4.12-4.13]. It was proved later that a synchronizer, an arbiter and an inertial delay element are all equivalent in that if a perfectly reliable synchronizer can be built, a perfectly reliable arbiter and a perfectly reliable inertial delay could also be built from it, and vice versa [4.14]. One implication was that there would also be similar problems in building a perfectly reliable arbiter which is also a very important element in digital asynchronous circuits. A few years later, it was proved that it is fundamentally impossible to build a perfectly reliable synchronizer because all real electrical elements do not have true "jump" characteristics [4.15-4.16]. So, efforts were directed towards either building a synchronizer with acceptable performance or seeking a system level solution to avoid metastability. Recent work has been done to explore circuit design techniques to take full advantage of NMOS and CMOS technologies - NMOS and CMOS [4.17-4.18].

4.2 Synchronizer Models

A synchronizer has two inputs and one output - a signal input, a clock and a signal output. A D-type flip-flop will provide a logically equivalent function to a synchronizer. The reason a D-type flip-flop is used as a synchronizer is that flip-flops allow only two possible stable states - logical high and low. But this assumption is incorrect because they are implemented as cross coupled inverters, taking advantage of a positive feedback. In that implementation, a third state exists.

Fig. 4.2 shows three operating points resulting from connecting two inverters. There are three intersections at which their states can be maintained forever. However, the operating point in the middle will start to move from its state with any slight agitation, eventually settling on one

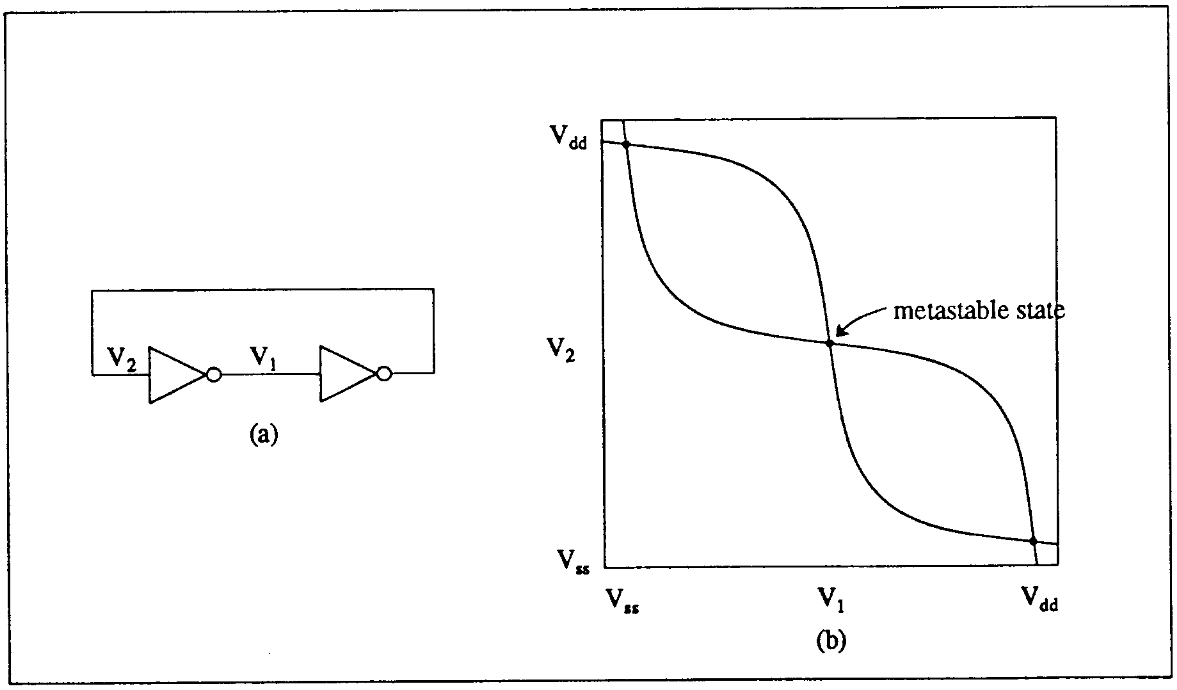

Fig. 4.2 Operation of cross coupled inverters. (a) Two inverters connected back to back. (b) Superimposed transfer curves showing three operating points - two stable ones and a metastable one. Circuits in the metastable state will decay into one of the two stable states with a slight perturbation.

of the two other states. We call this state a *metastable state.* When the cross-coupled inverter pair enters a metastable state in a hypothetical noise-free system, it will never escape from the state. The output from the flip-flop in a metastable state cannot satisfy either or both of the time-qualification and value-qualification conditions. Since metastability is unavoidable, efforts are made to speed up the escapement from metastability so that the average time of staying in the metastable zone can be reduced. An analogy can be made from the situation of a ball landing on a hill. A ball landing on the hill will eventually settle at the stable bottom. However, if the ball lands just on top of the hill, it may stay there forever. A synchronous system guarantees stability by making every ball land on the either side of the hill, while a synchronizer is subject to random landing on the hill. To reduce the chance of not settling at the bottom, we can make the hill steeper. Then a slight perturbation can easily drive the ball off the top and make it roll fast. The steepness of the hill is equivalent to the bandwidth of the positive feedback loop in a synchronizer. Although the circuit model shown in Fig. 4.3 was first explored in [4.4], it is included for completeness.

A synchronizer in the metastable state can be modeled as two transconductance amplifiers connected back to back, as shown in Fig. 4.3(a). The system is of second order and has two characteristic time constants, $\pm\frac{C}{g_m - r_o^{-1}}$. With a non-zero initial condition, the system output will grow exponentially, finally settling to one of the two stable states where the small-signal model no longer holds. The initial condition depends on the relative timing of the clock and the input signal, which we cannot control due to their independence. The most important factor is the speed with which a synchronizer escapes from the metastable state. If the initial state is so near to the metastable state that it fails to reach one of the the stable states within a time given by a synchronous system - for example, a cycle time - then logically unqualified value will propagate to the next stage. Since the system being initially in that initial condition is probabilistic, all we can do is to narrow the region so that we can reduce the probability. Fig. 4.4 shows a wavefom from

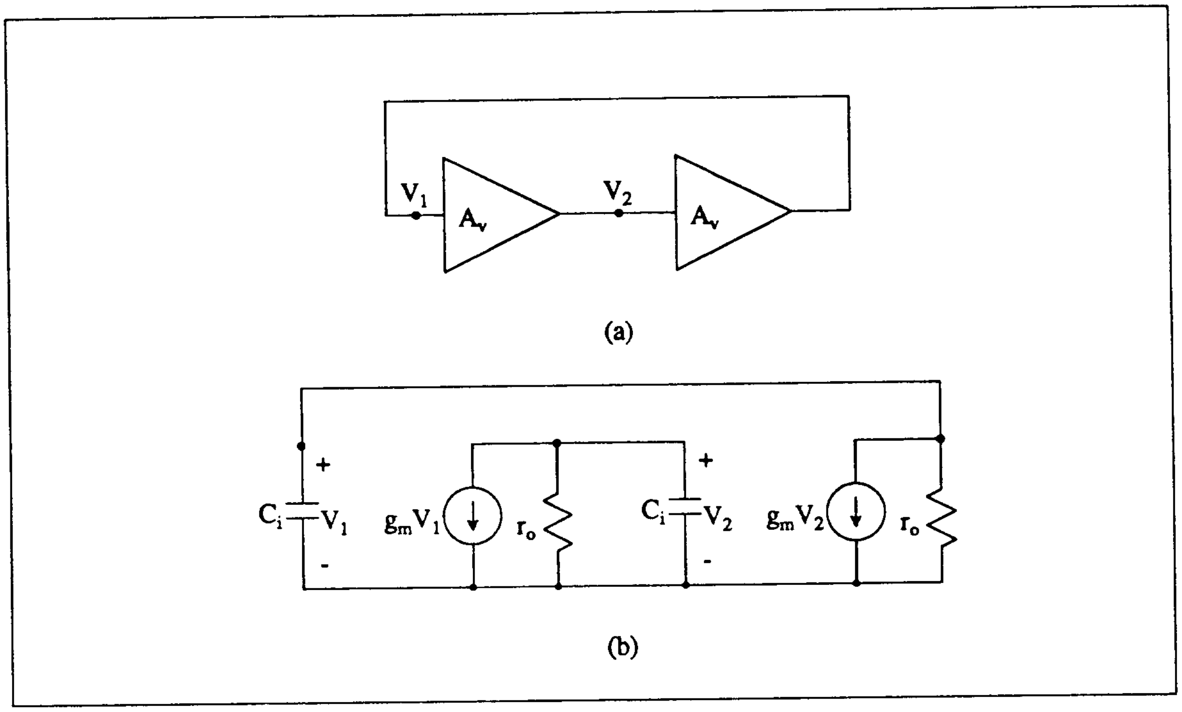

Fig. 4.3 Circuit model for a synchronizer in the metastable state. (a) Two amplifiers connected in a positive feedback loop ($A_v=-g_mr_o$). (b) Small-signal equivalent circuit.

a synchronizer settling from a metastable state. On reaching either of the supply lines, its linear model ceases to be valid and each output is clamped to a supply line.

Assuming that initial voltage difference between the two amplifier outputs is ΔV, the voltage difference after time t is $\Delta V \exp(\frac{t}{\tau})$, where $\tau = \frac{C}{g_m - r_o^{-1}}$, ignoring an exponentially decaying term. Therefore, the maximum of the initial voltage difference that causes metastability to persist longer than T is $\Delta V = V_{dd} \exp(-\frac{T}{\tau})$, assuming that the metastability appears at $\frac{1}{2} V_{dd}$ and the small-signal model still holds up to V_{dd} or V_{ss}. It should be noted that the ratio, $\frac{T}{\tau}$ is very critical for reducing ΔV. Since it is ideal to reduce the waiting time, T, it is desirable to reduce τ to keep ΔV the same. Since the bandwidth of the amplifier is $\frac{1}{2\pi\tau}$, bandwidth maximization of the amplifiers is the goal of the synchronizer design.

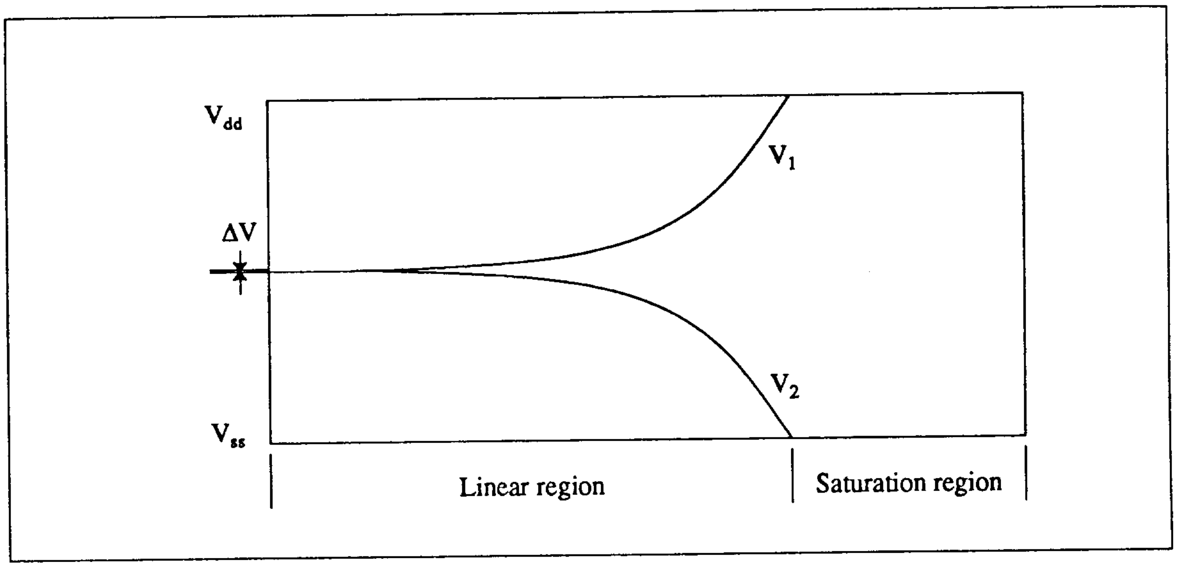

Fig. 4.4 Signal waveform of a synchronizer coming out of metastability. The linear region can be approximated as an exponential curve, whereas the saturation regions represent two stable steady-states.

One remaining modeling problem is in finding the initial condition of the synchronizer from the timing difference between the clock and input. It is not easy to derive an accurate analytical relation since it is non-linear and dependent on the implementation and the input waveform. First, it is assumed that there are no internal input and clock delays, and later, the effect of those delays will be considered. Denoting V_{in} as the input voltage when sampled by the synchronization clock, initial condition ΔV is $2 \cdot V_{in} - \frac{V_{dd}}{2}$. Note that $\Delta V = \pm V_{dd}$ when the synchronizer input is at $\pm V_{dd}$. The above model is conservative because the effect of the voltage gain of an input buffer circuit, normally included in a synchronizer, is ignored. There can be other more accurate and complex models; however, the extra precision in this area does not have dramatic effect in characterizing a synchronizer.

Fig. 4.5 shows an equivalent circuit of a synchronizer from a logic designer's point of view. Usually, a synchronizer includes buffer circuits in the inputs and the output. They contribute

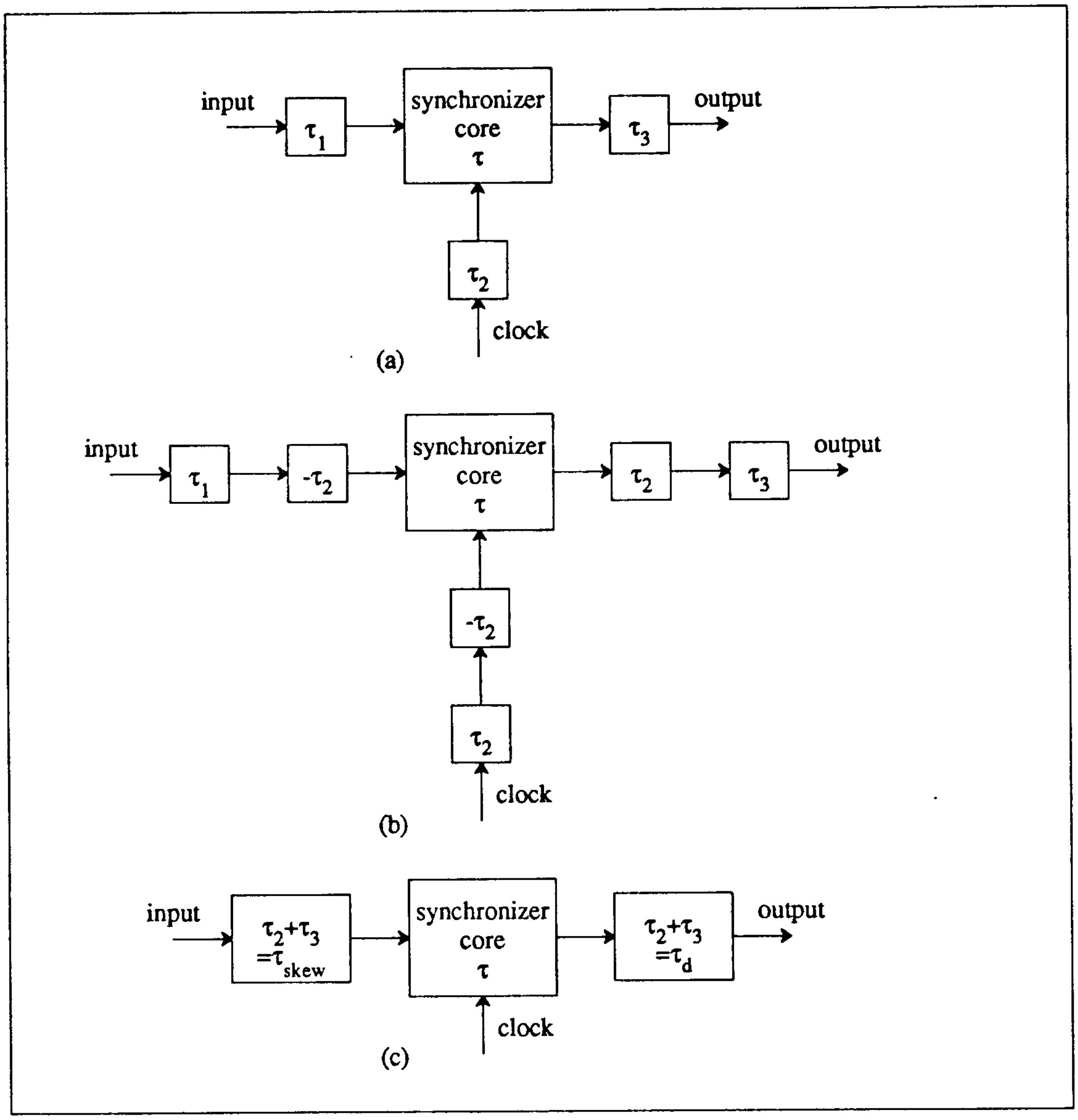

Fig. 4.5 Logical equivalent circuit of a D-type latch used as a synchronizer. (a) Synchronizer core with three delays from each terminal. (b) Transformation of delays. (c) Final observable form.

logic delays inside a synchronizer. Thus, the original model shown in Fig. 4.5(a) includes 3 delay elements. These three delay elements can be transformed into two by introducing negative delays in the inputs and a positive delay in the output. The amount of hypothetical delay equals the clock delay, so that the clock path delay can be canceled (Fig. 4.5 (b)). Only the two delays which are derived from the original ones are observable and meaningful. One is the skew delay, τ_{skew}, between the input and the clock with respect to the clock timing. The other is the output delay, τ_d from the clock to the output. τ_{skew} can be either positive or negative depending on the delays of the buffers. However, τ_d is always positive since it is the sum of the clock buffer delay and the output buffer delay.

The above mentioned delays are deterministic. However, to model a non-deterministic or probabilistic delay of a synchronizer, a third parameter, τ, is needed. The term 'probabilistic' is used because the amount of delay is unbounded and can not be determined in advance. It varies with the initial condition of the synchronizer that is probabilistically distributed depending on the input voltage sampled haphazardly with the synchronization clock. Thus, the minimum set of parameters required to characterize a synchronizer are τ_{skew}, τ_d and τ.

When this type of a D-FF is used as a latch in a pipeline stage in a synchronous system, τ is not an important parameter. The input signal is always out of the metastable region because the inputs are guaranteed to be settled well before the clock edge, not causing metastability. Instead, τ_{skew} is an important parameter. Since all the integrated circuits have parameter variations, variations of τ_{skew} must be given. Usually, the upper bound and lower bound of τ_{skew} are called set-up time and hold time respectively.

Based on the delay model described, a graph shown in Fig. 4.6 represents the two delay components - deterministic delay, τ_d, and probabilistic delay due to metastability, $T_d - \tau_d$, where T_d is a total delay. Assuming the input waveform is a ramp function with a rise time of t_R, the following equations hold.

$$\Delta V = 2\,V_{in} - V_{dd} = V_{dd}\,\frac{2\,(t - t_{skew})}{t_R} \tag{4.1}$$

$$V_{dd} = \Delta V \exp\left(\frac{T_d - t_d}{\tau}\right) \tag{4.2}$$

From these two equations, we derive the relation between t and T_d.

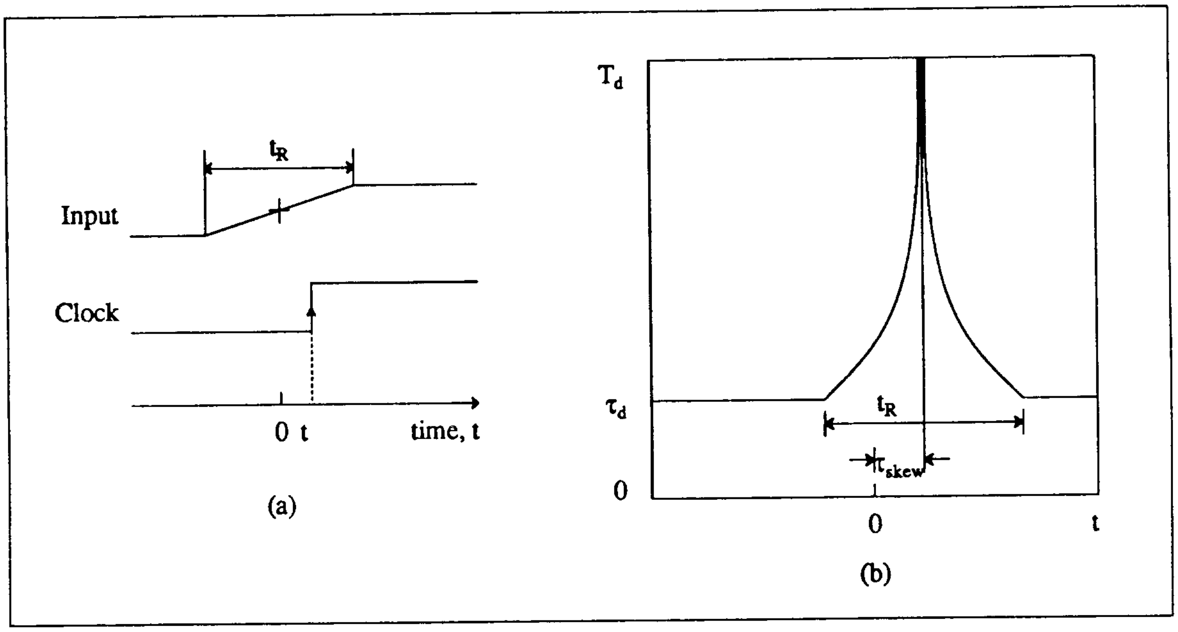

Fig. 4.6 Graph showing the total delay of a synchronizer. (a) Waveforms assumed. (b) Total delay as a function of sampling time, t.

$$T_d = \tau_d + \tau \ln \left[\frac{t_R}{2\,(t - \tau_{skew})} \right] \tag{4.3}$$

Equation (4.3) holds for $0 < (t - \tau_{skew}) < \frac{t_R}{2}$. We can see that $T_d \rightarrow \infty$, as $t \rightarrow t_{skew}$.

4.3 Synchronization Strategies

Although it is not possible to build a perfect synchronizer, it would be equally acceptable if we can reduce the probability of error to the level of the probability of hardware failure or circuit malfunction due to thermal noise. The second standard of reducing synchronization error down to the failure rate due to the thermal noise effect is called the *Mead Criterion* [4.19]. The easiest way of reducing the probability of error is to allow a synchronizer more time to settle. This necessarily introduces latency to the signal and it is not acceptable for some applications. If the time allowed to settle is less than a cycle time, a pre-sampling may be done before actual sampling is made. Also, a cascade of synchronizers can be used when more than a cycle time should be allowed for settling. In telecommunications applications where throughput is more important than latency, using a cascade of synchronizers is a good strategy that does not require multiphase clocking. It was argued in [4.20] that the subsequent stage should have a different threshold so that even if the previous stage is in a metastable region, the next stage can have the stable value. The above failed to note that the voltage at the output of the previous stage has uniform distribution as shown in [4.9]. By having a different threshold, a subsequent stage may resolve the metastable voltage from the previous stage into a stable output voltage correctly. Unfortunately it may also drive itself with equal probability into a metastable state that would have been settled otherwise. Thus, the output may change in the middle of a cycle when the output from the previous stage makes a transition from the metastable state to the stable state. Therefore, it is a useless effort to include any circuit between stages in order to convert a metastable output to a stable logic level temporarily. It only converts value-unqualified signal to time unqualified signal, whose effect to a FSM is similar. Thus, when n pipeline stages of synchronizers are used, the net time allowed for synchronizers to settle is $(n-1)(T-\tau_d-\tau_{skew})$, where T is a cycle time. Even though all n cycles are not used for synchronization, implementation of such a synchronizer is simple, and fits nicely to a pipeline paradigm.

Where latency should be minimized, the only option is to optimize the circuit so that its speed of escapement can be increased, in other words, to minimize τ. Assuming that the cycle time is composed of 50-200 unit inverter delays in a typical computer system, a synchronizer with characteristic time constant equaling a half of the inverter delay or less would be acceptable as a synchronizer settling in a cycle. If a synchronizer with characteristic time constant being a quarter of a inverter delay time is available, half a cycle latency will be more than enough for its synchronization reliability. For example, for a synchronizer with τ= 0.5ns, ΔV= 9.6×10^{-22} V with a 25 ns settling time. Under the assumption that input changes with 10 MHz frequency with rise/fall time of 2ns, the probability of unresolved synchronization is 3.8×10^{-24} per event. The Mean Time Between Failure (MTBF) with a 10 MHz sampling rate is 2.6×10^{16} seconds or 8.3×10^{8} years. However, a poorly designed synchronizer with $\tau = 2.0$ns will result in MTBF of only 0.15 second. A factor of 4 difference in τ makes a difference of 17 orders of magnitude in MTBF.

Note that ΔV that causes metastability is much smaller than thermal noise. With such a small initial condition, the synchronizer will be affected more from thermal noise. However, the noise introduced does not affect the probability of synchronization failure [4.4]. The effects of noise and the initial condition are 'orthogonal'. The noise that helped to drive the synchronizer out of the metastable state may also drive the synchronizer back to the metastable state that may have settled otherwise. Noise tends to make broader the range of the initial conditions of the metastability. However, all synchronizers in those initial conditions do not always cause metastability under noise.

4.4 Circuit Techniques

Since a synchronizer is logically equivalent to a D-type latch, the simpliest way of implementing it is to use cross-coupled NAND or NOR gates. Its primary design goal is to minimize τ as opposed to a normal design principle to minimize the input to output delay. If the D-type latch is used within a synchronous system, τ is not a concern at all. Logic delay is the only performance-related factor. The two different goals cannot be satisfied simultaneously, because the delay is basically a large signal parameter and τ is from a small signal analysis. For example, when a synchronizer has to drive a huge load capacitance, it wouldn't be a good idea to increase the size of the NAND or NOR gate and directly connect it to the load, although that is completely justifiable in a synchronous environment. The load capacitance will be part of the positive feedback loop and increase τ. A better strategy is to isolate the load capacitance from the core of the synchronizer by inserting a buffer so that the core maintains its optimum circuit configuration regardless of the load capacitance. In this section, several circuit design techniques will be shown and discussed in two different technologies that can be used in real applications that require a low-error rate synchronizer. First, I will cover the technique in bipolar technology since its concept is easier to understand. Later, I will extend the idea to CMOS technology. Generally, a synchronizer implemented in bipolar technology has better performance than the one in CMOS in similar level of technology. Thus, a bipolar synchronizer could be a choice in the BiCMOS technology.

4.4.1 Bipolar Synchronizers

Since the primary interest in this thesis is on high performance VLSI, I will concentrate on circuits in current steering logic families including Current Mode Logic (CML) and Emitter Coupled Logic (ECL), excluding TTL and I^2L.

4.4.1.1 CML and ECL Synchronizers

A bipolar transistor is modeled with two intrinsic components, base input capacitance, C_π and transconductance, g_m, along with many parasitic resistances and capacitances. All the parameters except g_m are strongly dependent on its processing technology. τ of the synchronizer can be obtained by measurement or by SPICE simulation from its circuit configuration and device parameters. Measuring τ experimentally is a complex task involving very sophisticated techniques which can be an independent subject [4.21]. The following observations are from simulation results. Out of the three main factors affecting the performance of a synchronizer - process technology, circuit design technique, and layout technique, circuit technique will be focused in this thesis.

The first order relation of an intrinsic transistor model is as follows:

$$g_m = \frac{q\, I_C}{k\, T} \tag{4.4}$$

$$C_\pi = g_m\, t_F, \tag{4.5}$$

where q is elementary charge, I_C, collector bias current, k, Boltzmann constant, T, absolute temperature, t_F, forward base transit time of the transistor. The first-order model for transistor behavior shows that t_F is proportional to the square of the base width.

$$t_F = \frac{W^2}{2\, D_n}, \tag{4.6}$$

where W is base width, D_n, electron diffusion coefficient.

Two simple small-signal models of a bipolar transistor are shown in Fig. 4.7. A complete model can be found in [4.22]. When two intrinsic transistors are connected in a positive feedback loop without biasing circuits, τ is given as follows.

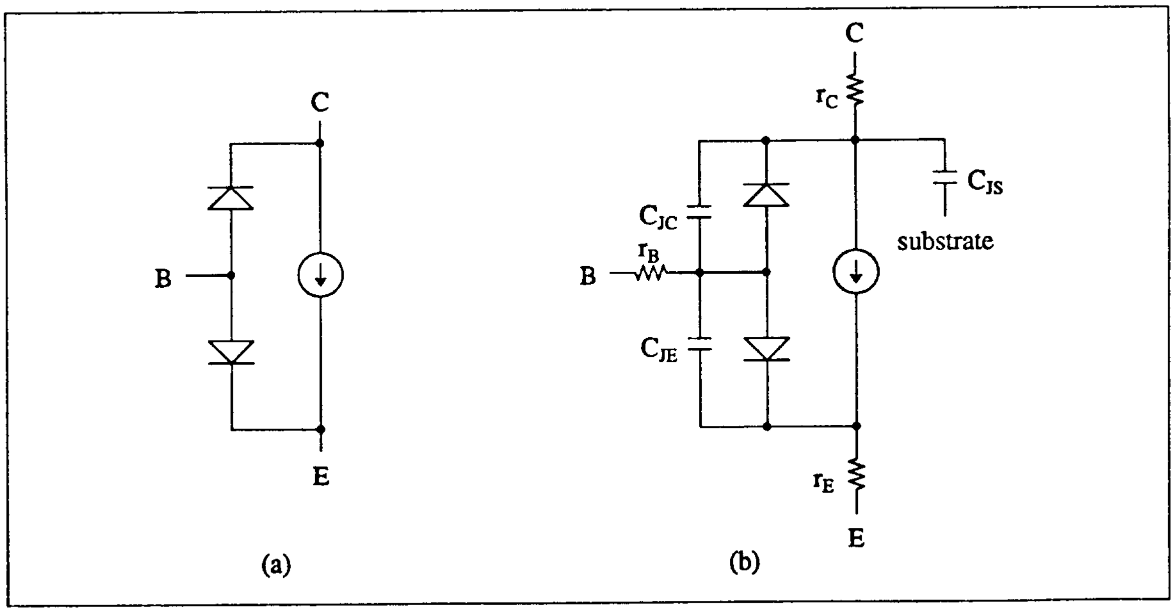

Fig. 4.7 Two bipolar transistor models. (a) Intrinsic transistor. (b) Intrinsic transistor with parasitic resistors and capacitors.

$$\tau = t_F . \tag{4.7}$$

When the circuits include biasing elements and the devices have non-negligible parasitics, τ will be greater than the value expected from eq. (4.7). High frequency characteristics of the bipolar transistors tend to be dominated by the effect of parasitics rather than by the intrinsic characteristics. However, high-performance bipolar transistors fabricated from advanced processes using oxide isolation, polysilicon emitter and sidewall base contact, have their high frequency characteristics dominated by the base transit time instead of their parasitics [4.23]. The technology trend is to decrease parasitics more rapidly than t_F. Thus, in the future, τ will be composed mostly of the base transit time.

A common implementation of a D-type flip-flop in CML is shown in Fig. 4.8(a). It uses stacked gates for efficient transistor utilization. The lower level transistors steer source current between the input stage and the storage stage. The storage stage is more important for its perfor-

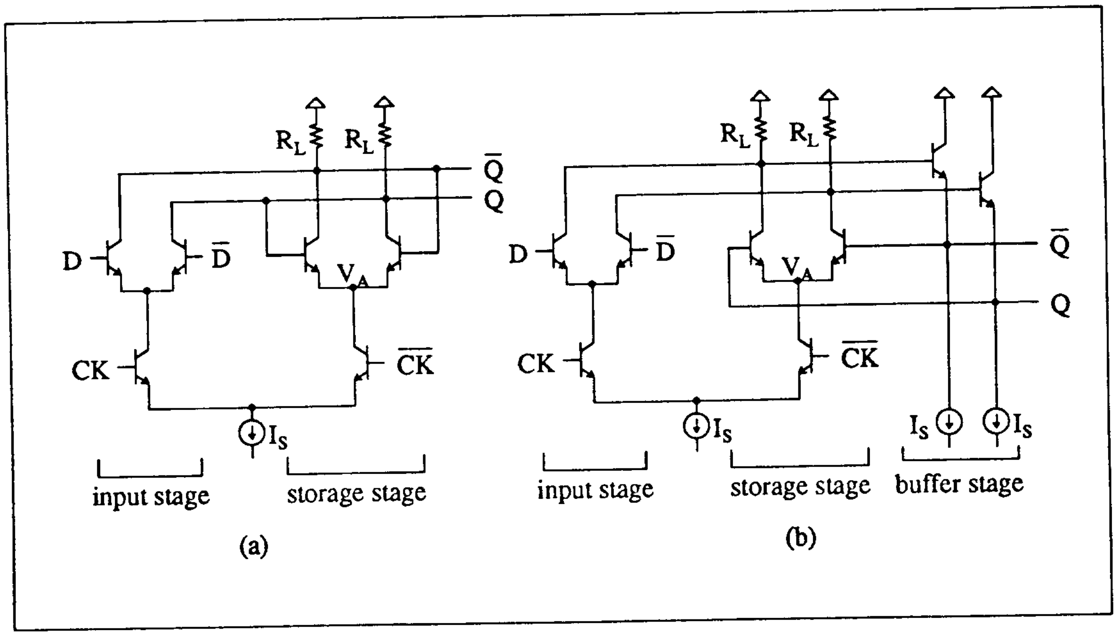

Fig. 4.8 Two common D-type flip-flops. (a) D-FF in CML and (b) in ECL.

mance because it determines its settling behavior. The ECL implementation, shown in Fig. 4.8(b), has a similar circuit configuration except that it has extra buffer stage (emitter follower) to provide low impedance drive to capacitive loads.

The bias current of the D-FF should be large enough to dominate the junction parasitic capacitance, while it should not be so high as to cause device performance degradation due to high-level injection or Kirk effect. High base input capacitance due to large current will also result in a large time constant with parasitic resistance, for example, with base resistance. In short, parasitic capacitance is the performance limiting factor at low current level while the effect of parasitic resistance is dominant at high current level.

Since Q and $\bar{Q}$ of both circuits move in the opposite direction, V_A is the AC ground. Thus, in the CML implementation, the overall small-signal equivalent circuit is the same as the one in Fig. 4.3(b). τ of this circuit can be found as follows.

$$\tau = \frac{C_\pi}{g_m - R_L^{-1}} = \frac{\tau_F}{1 - \frac{1}{g_m R_L}} = \frac{\tau_F}{1 - \frac{V_T}{\frac{1}{2} I_S R_L}} = \frac{\tau_F}{1 - \frac{2 V_T}{I_S R_L}}$$

$$= \frac{\tau_F}{1 - \frac{2 V_T}{V_{SWING}}}. \tag{4.8}$$

At room temperature, $V_T = 26$ mV, and $V_{SWING} = 400$ mV. τ of the CML with intrinsic transistors is 1.15 τ_F. The performance loss of 15% is from the loading effect of R_L.

In ECL, the outputs from the storage stage are connected back to its inputs through an emitter follower stage. The emitter follower stage does not provide any amplification but it provides low impedance drive to the output. Therefore, it is expected that the ECL version has better characteristics. Its equivalent half circuit is shown in Fig. 4.9.

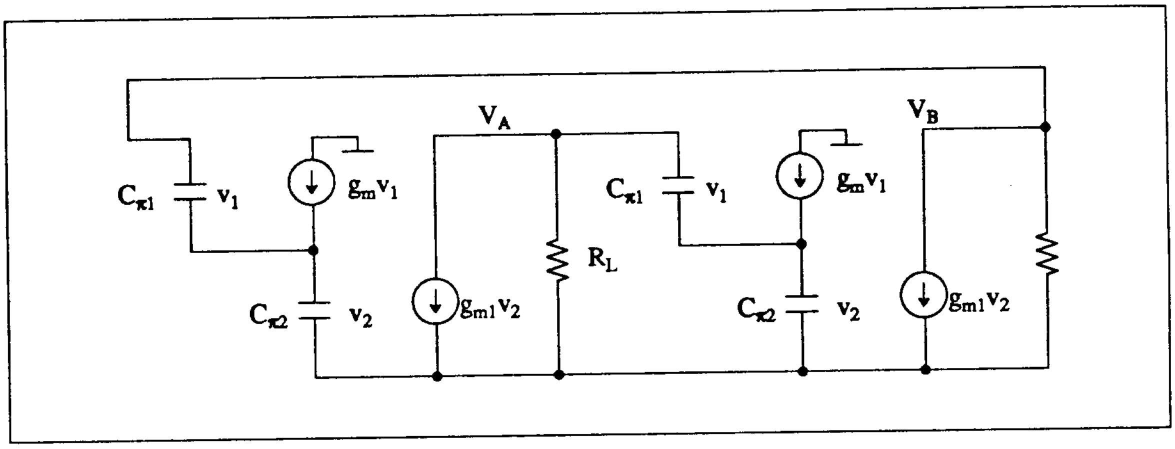

Fig. 4.9 Small-signal equivalent circuit operating in the metastable region in ECL.

From the equivalent circuit excluding R_L, the following equations hold.

$$[\, g_{m2} (s C_{\pi 1} + g_{m1}) \,] V_B = - s^2 C_{\pi 1} C_{\pi 2} V_A. \tag{4.9}$$

$$[\, g_{m2} (s C_{\pi 1} + g_{m1}) \,] V_A = - s^2 C_{\pi 1} C_{\pi 2} V_B. \tag{4.10}$$

Solving the two equations, we have the following quadratic equation,

$$s^4 \left(\frac{C_{\pi 2}}{g_{m2}} \right)^2 \left(\frac{C_{\pi 1}}{g_{m1}} \right)^2 - \left(s \frac{C_{\pi 1}}{g_{m1}} + 1 \right)^2 = 0. \tag{4.11}$$

$$s^4 + \left(\frac{1}{t_F} \right)^2 s^2 - \left(\frac{1}{t_F} \right)^3 s + \left(\frac{1}{t_F} \right)^4 = 0. \tag{4.12}$$

The above equations show that the bias current does not affect the frequency characteristics. Using MACSYMA [4.24], we obtain 4 solutions, $s = \frac{-1 \pm \sqrt{3}i}{2} \left(\frac{1}{t_F} \right)$, $\frac{1 \pm \sqrt{5}}{2} \left(\frac{1}{t_F} \right)$. The most interesting one is $s = \frac{1+\sqrt{5}}{2} \left(\frac{1}{t_F} \right) = 1.62 \left(\frac{1}{t_F} \right)$. The effective bandwidth increased by 62%. The expected τ is 0.62 t_F. If the effect of R_L is included, τ will increase by less than 20%. Thus, for synchronization, an ECL D-FF is a better circuit than a CML D-FF.

The steering current, I_S, should be selected in such a manner that it is large enough to dominate parasitic capacitance through low g_m, but should not be so large as to yield a big time constant with parasitic resistors and C_π. Load resistors are needed to provide collector bias current. The resistances are selected such that the voltage swing across them does not drive the switching transistors in saturation region and should be large enough to provide sufficient noise margin. Usually, CML has a 400 mV voltage swing across the resistor. ECL has a voltage swing between 400mV and 900mV. It is interesting to note that digital operations prefer a low voltage swing for speed, while a high voltage swing is preferable in synchronizer operations. CML in digital operation works faster with low voltage swing because the amount of charge it has to provide is reduced. However, the bandwidth of the CML when used as a synchronizer decreases with reducing the voltage swing because, with low load resistances, more signal current is shunted to ground instead of going to the base capacitances. We have two conflicting design criteria between digital and analog domain. The circuit simulation results will be shown later in Table 4.1 along with other results.

4.4.1.2 Synchronizers with Bandwidth Multipliers

To overcome the technology-bound limit, a special circuit concept called f_T doubler can be used for the storage stage [4.25]. f_T represents a maximum bandwidth a device can amplify. f_T is inversely proportional to t_F for intrinsic devices.

$$f_T = \frac{1}{2\pi t_F}. \tag{4.13}$$

This technique has been used in high bandwidth analog circuit designs. However, there is some difference between these two applications. In synchronizer applications, the frequency domain characteristics, such as phase margin and noise figure, are not as important. The principal idea is to reduce C_π relative to g_m by connecting their input devices in series and outputs in parallel. Its idea is shown in Fig. 4.10.

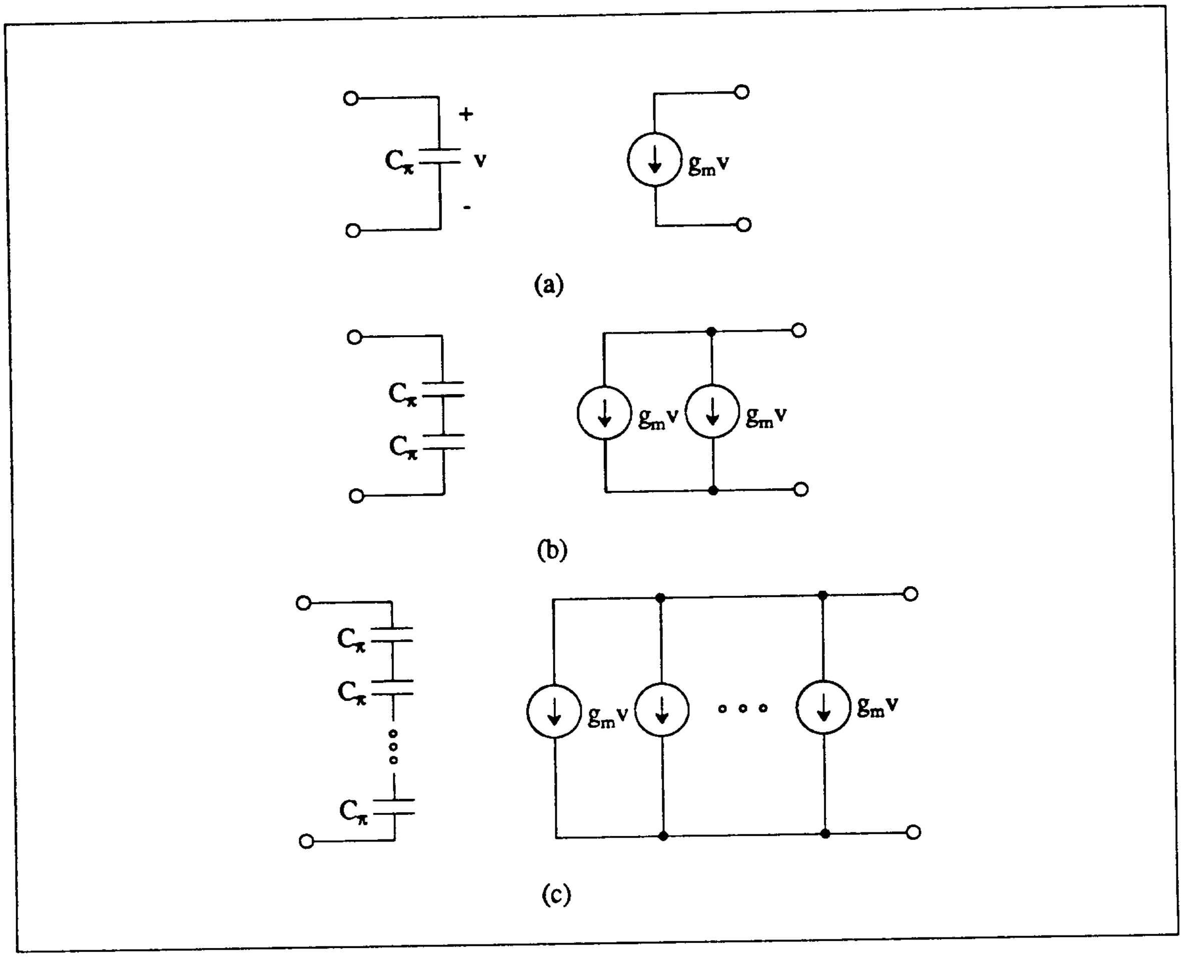

Fig. 4.10 Principle of bandwidth multiplication. (a) Single device. (b) Bandwidth doubler. (c) Bandwidth multiplier.

Most of the electronic devices, including bipolar and MOS devices, have the basic model that is similar to Fig. 4.10 (a), ignoring parasitic capacitances and resistances. The bandwidth, f_T of a single device is $\frac{g_m}{2\pi \cdot C_\pi}$. The bandwidth doubler, shown in Fig. 4.10 (b), has half as much input capacitance as a single device since the two capacitances are connected in series. The signal voltages appearing across the two capacitors also decrease to a half. However, the two dependent sources are connected in parallel, adding two currents together, and generating equal amount of overall signal current. Thus, net effect is the reduced input capacitance by half without changing overall transconductance, achieving bandwidth doubling. A similar concept applies to further multiply bandwidth as shown in Fig. 4.10 (c).

To implement this concept in real circuits, several problems arise. One problem is that a real device has only three terminals with the result that the input and output terminals are not completely isolated. It should be noted that clever circuit configurations with proper biasing are needed.

One such configuration is shown in Fig. 4.11. In this configuration, although not simple to figure out, there are four transistors which form a bandwidth doubler in differential mode. The bandwidth of this circuit can be compared with that of the CML D-FF. The outputs Q and $\overline{Q}$ will see four base-emitter junction capacitors in series, giving only half as much input capacitance as that of the CML D-FF. The overall transconductance does not change since all four switching

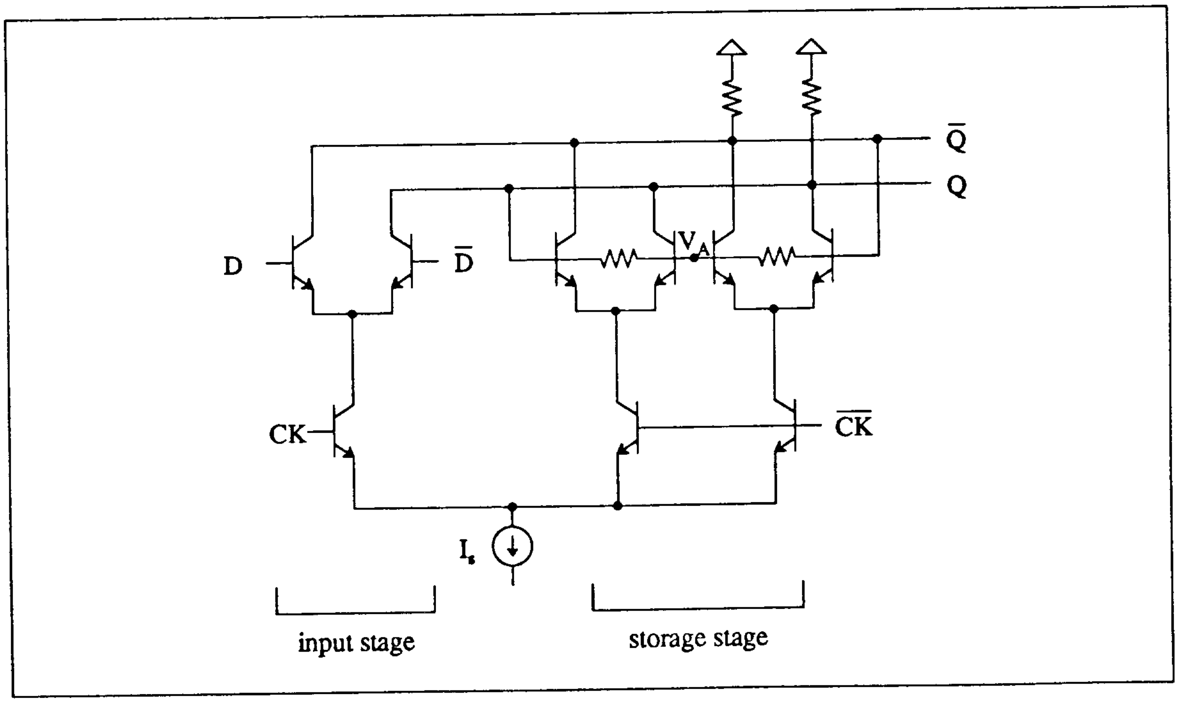

Fig. 4.11. Fast settling D-type flip-flop in CML (a) Circuit Configuration (b) Transformation of equivalent circuits.

transistors in the bandwidth doubler are involved in signal generation unlike the ECL D-FF in which emitter followers act as buffers. Therefore, the net effect is the increase of f_T by the factor of 2.

A proper biasing circuit must be designed for the node, V_A, which provides base current to the switching transistors. A plan should be made to design a biasing circuit for the node, V_A that connects two base terminals. Since the node maintains AC ground for the differential inputs, two resistors connected as a resistive voltage divider, shown in Fig. 4.11, can provide bias current. However, their values should be chosen not to shunt too much input current and should not introduce too much parasitic capacitance. This idea can be extended to many stages to further increase f_T. But it will not increase linearly due to the parasitics and secondary effect.

4.4.1.3 Synchronizers with Darlington Pairs

Another circuit that performs f_T multiplication is a Darlington pair shown in Fig. 4.12.

From the small signal equivalent circuit for a resistor-biased Darlington pair, the input and output currents are related as follows:

$$i_{out} = \frac{g_{m1}\left(\frac{1}{R} + s\,C_{\pi 1}\right) + g_{m2}\left(g_{m1} + s\,C_{\pi 1}\right)}{\frac{1}{R} + g_{m1} + s\,C_{\pi 1} + s\,C_{\pi 1}}\,v_{in}\,. \tag{4.14}$$

$$i_{in} = \frac{s\,C_{\pi 1}\left(\frac{1}{R} + s\,C_{\pi 2}\right)}{\frac{1}{R} + g_{m1} + s\,C_{\pi 1} + s\,C_{\pi 1}}\,v_{in}\,. \tag{4.15}$$

When the two conditions, $R = \frac{1}{g_{m1}} = \frac{1}{g_{m2}} = \frac{1}{g_m}$ and $C_{\pi 1} = C_{\pi 2} = C_\pi$ are satisfied,

$$i_{out} = g_m\,v_{in}\,. \tag{4.16}$$

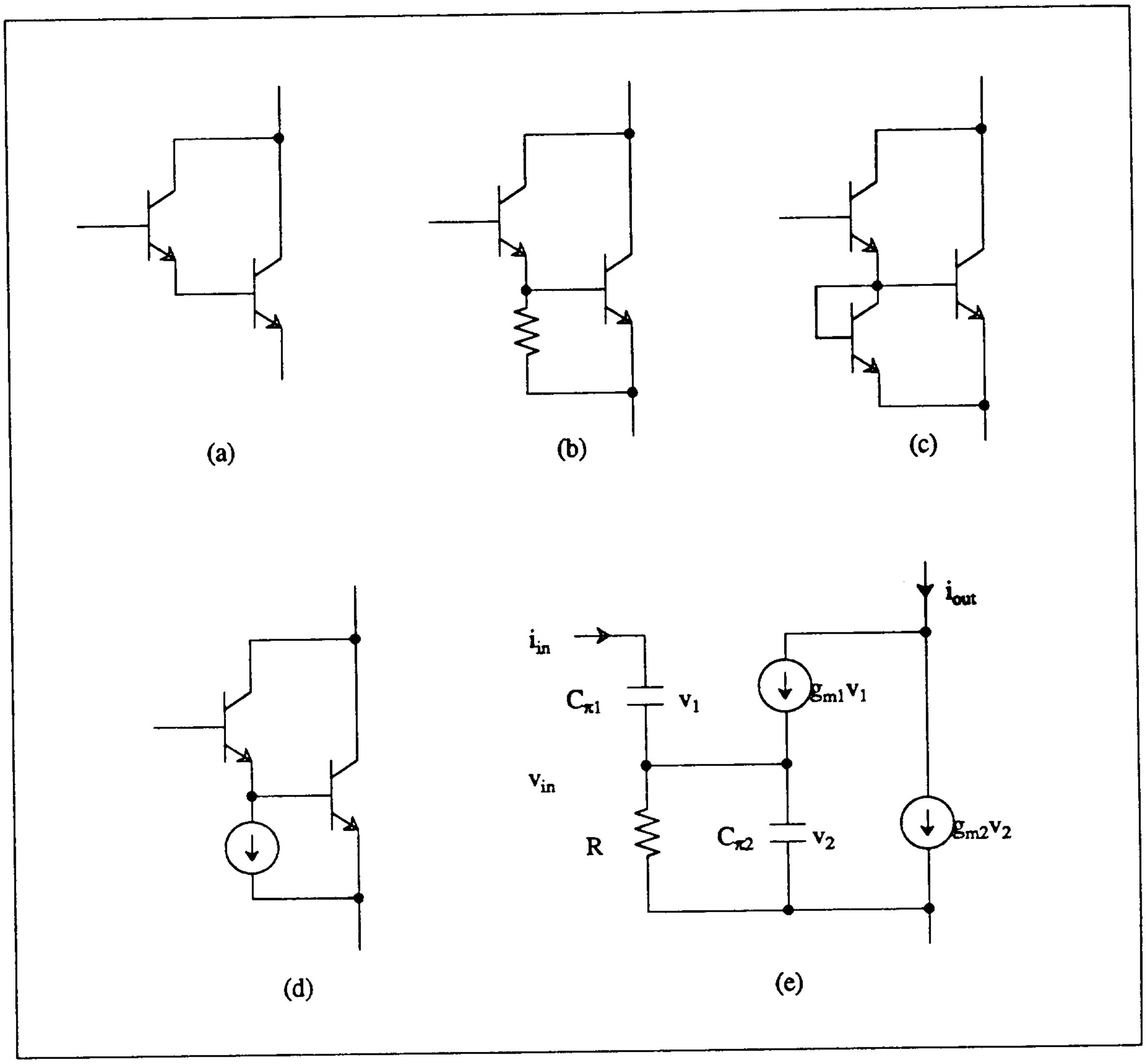

Fig. 4.12 Bandwidth multiplication using a Darlington Pair (a) An unbiased Darlington Pair. (b) Resistor Bias (c) Current Mirror Bias (d) Current Source Bias (e) Small-signal equivalent circuit for (b)

$$i_{in} = \frac{s\,C_\pi}{2}\,v_{in}\,. \tag{4.17}$$

From Eqs. (4.5) and (4.6), a bandwidth doubling effect is evident. However, satisfying $R = \frac{1}{g_m}$ is not possible because the resistor must satisfy another condition, $R = \frac{V_{BE}}{I_C}$ for proper DC biasing. A circuit shown in Fig. 4.12(c) can satisfy the two relations using a a diode-connected transistor. However, the diode-connected transistor introduces a capacitance, C_π, as well as providing resistance, which fails to double its bandwidth.

The analysis of the circuit with the current source biased Darlington pair shown in Fig. 4.12(d) follows. Since the current source has infinite impedance, $R \to \infty$.

$$i_{out} = \frac{g_{m1}(s C_{\pi1}) + g_{m2}(g_{m1} + s C_{\pi1})}{g_{m1} + s C_{\pi1} + s C_{\pi1}} v_{in} . \tag{4.18}$$

$$i_{in} = \frac{s^2 C_{\pi1} C_{\pi2}}{g_{m1} + s C_{\pi1} + s C_{\pi1}} v_{in} . \tag{4.19}$$

When a loop is connected as shown in Fig. 4.13, the following equations hold.

$$[\, g_{m1}(s C_{\pi2} + g_{m2}(g_{m1} + s C_{\pi1}))\,]\ v_A = -s C_{\pi1}\, s C_{\pi2}\ v_B . \tag{4.20}$$

$$[\, g_{m1}(s C_{\pi2} + g_{m2}(g_{m1} + s C_{\pi1}))\,]\ v_B = -s C_{\pi1}\, s C_{\pi2}\ v_A . \tag{4.21}$$

From these two equations, we obtain four poles, $(1\pm\sqrt{2})\left(\frac{1}{t_F}\right)$, $(-1)\left(\frac{1}{t_F}\right)$ (double pole). The ratio of the bandwidth to f_T is $1+\sqrt{2}$, which shows more than double the bandwidth of what is obtainable from a single device. Its complete circuit diagram is shown in Fig. 4.14.

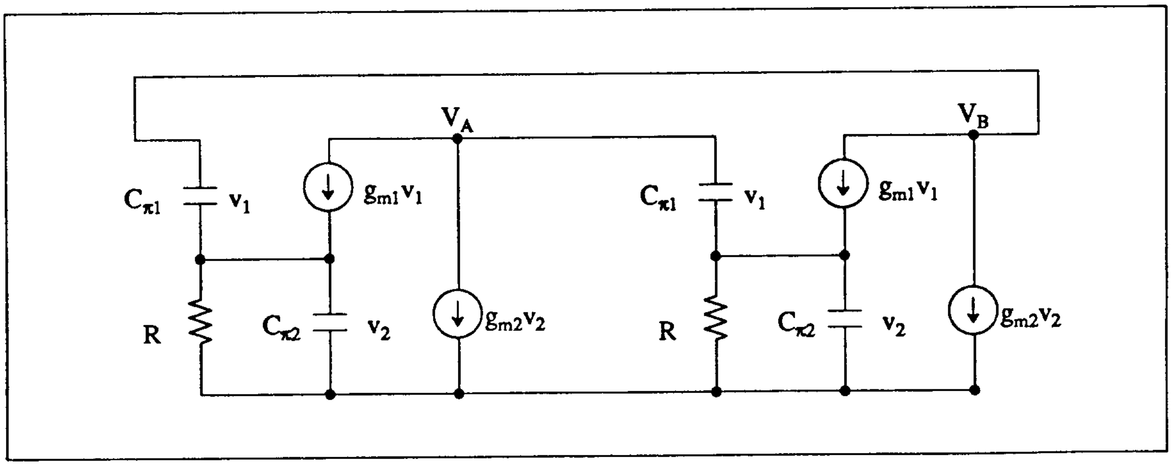

Fig. 4.13 Two Darlington devices connected as a loop.

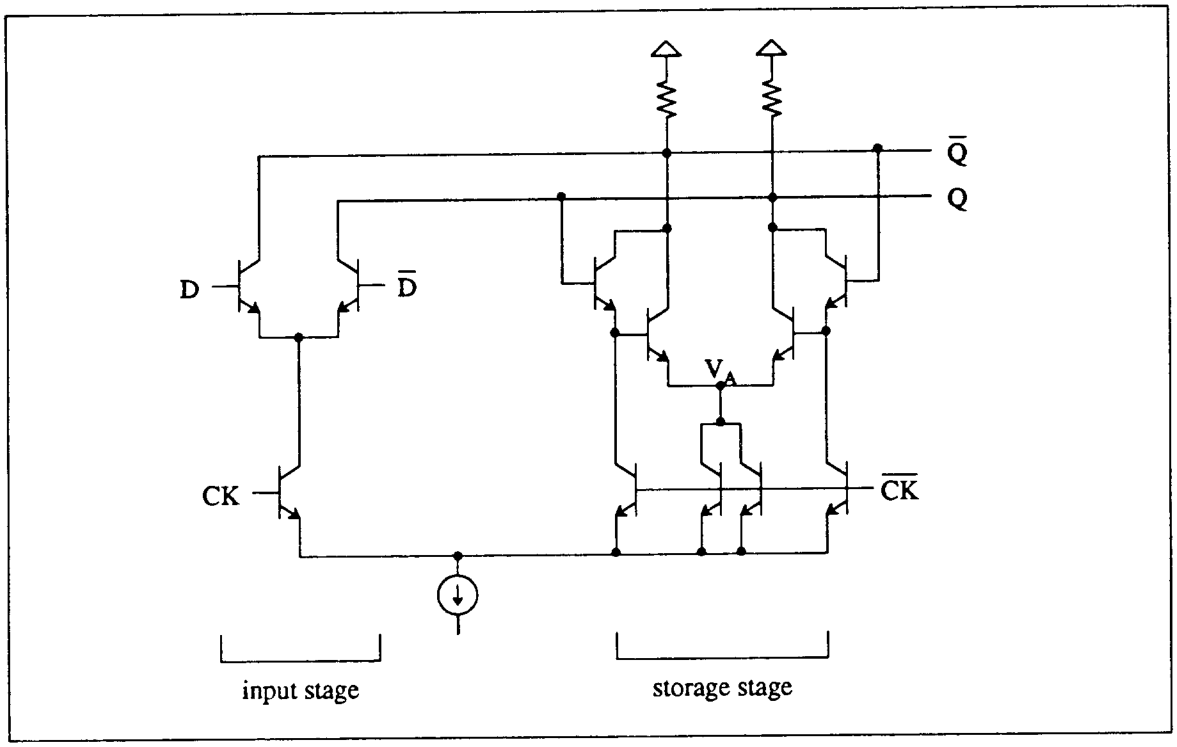

Fig. 4.14 Complete circuit diagram of the D-FF using Darlington pairs with current source bias.

4.4.1.4 Comparison of Bipolar Synchronizers

These circuits have been simulated with SPICE, using an intrinsic circuit model and also using a full model with scaled polysilicon emitter process [4.22]. Simulation results are shown in Table 4.1 and SPICE model parameters are shown in Table 4.2.

As is expected, a Darlington circuit with current source bias performs best for intrinsic transistors. However, with a full model including parasitics, all the circuits show substantial degradation from ideal performance, although the values of the Darlington circuits and bandwidth doubler are very conservative, because each transistor is assumed to have the worst-case, minimum sized device layout. The size and shape of the transistors can be designed for optimum performance. For example, the two collectors of the Darlington pair can be merged into a single island, reducing CJS. Also, since the parasitics play a major role, a finger-like device structure can be used to reduce parasitic resistance. Although the bandwidth multiplying circuits failed to perform better than conventional circuits in CML and ECL, future process technology that adds less parasitics to its intrinsic devices along with optimum layout will make the concept realizable.

Table 4.1 Simulated results.

Circuit	Schematics	τ,ideal†	τ,intrinsic	τ,full
CML	Fig. 4.7(a)	1.0 t_F	6.36ps	18.71ps
ECL (400 mV SWING)	Fig. 4.7(b)	0.62 t_F	4.01ps	16.18ps
ECL (800 mV SWING)	Fig. 4.7(b)	0.62 t_F	3.69ps	11.19ps
f_T doubler	Fig. 4.11	0.5 t_F	4.76ps	24.99ps
Darlington with CS bias	Fig. 4.14	0.41 t_F	3.17ps	26.64ps
Darlington with diode bias	Fig. 4.12(c)	-	4.52ps	23.78ps

† Effect of R_L was not considered.

Table 4.2 SPICE model parameters used.

Parameter	Intrinsic model	Full model
IS	1.0e-16 A	1.0e-16 A
BF	200	200
TF	5.7 ps	5.7 ps
RB	0	123
RC	0	43
RE	0	70
CJE	0	4.0 fF
CJC	0	3.0 fF
CJS	0	2.74 fF

4.4.2 MOS Synchronizers

Although design objectives are the same, different circuit techniques are applied in MOS circuits. MOS devices behave quite differently than bipolar junction transistors. Transconductance and input capacitance of an intrinsic bipolar transistor are proportional to its collector current and have no direct relation with its device size. On the other hand, the input capacitance of an MOS transistor is a function of its device active area and is not dependent on drain current as long as it is in saturation region. Transconductance is a function of its device geometry and its drain current. Thus, both device geometry and bias current are primary design factors in MOS circuits while biasing is the only prime concern in bipolar circuit design.

The following is a review of MOS device characteristics operating in the saturation region in terms of bandwidth maximization.

$$g_m = \mu\, C_{ox}\, (\frac{W}{L})\, (V_{GS} - V_T) \tag{4.22}$$

$$C_{gs} = \frac{2}{3}\, W\, L\, C_{ox} \tag{4.23}$$

where μ is carrier mobility, C_{ox}, gate oxide capacitance per unit area, W, device width, L, channel length, I_D, drain current, V_{GS}, gate bias voltage, and V_T, threshold voltage.

Thus, τ is calculated as follows.

$$\tau = \frac{C_{gs}}{g_m} = \frac{2}{3}\, \frac{L^2}{\mu}\, \frac{1}{(V_{GS} - V_T)} \tag{4.24}$$

From these equations, it is observed that gate bias voltage V_{GS} should be increased in order to minimize τ. μ, minimum of L, and V_T are determined by the process, not by design. Since V_{GS} cannot be greater than V_{dd} in normal designs,

$$\tau \geq \frac{2}{3}\, \frac{L^2}{\mu}\, \frac{1}{(V_{dd} - V_T)} \tag{4.25}$$

Thus, it is desired to build a bias circuit to set V_{GS} as close to V_{dd} as possible.

4.4.2.1 MOS Synchronizers using Inverters with Resistive Load

Let's consider an inverter with a resistive load as shown in Fig. 4.15(a). When the two inverters are connected back to back, it has been shown that

$$\tau = \frac{\frac{2}{3} C_{ox} W L}{g_m - \frac{1}{R}} \tag{4.26}$$

metastable output voltage V_m changes according to the value of the resistors. If we try to increase V_m to maximize its device performance as directed by Eq. (4.24), R should be decreased

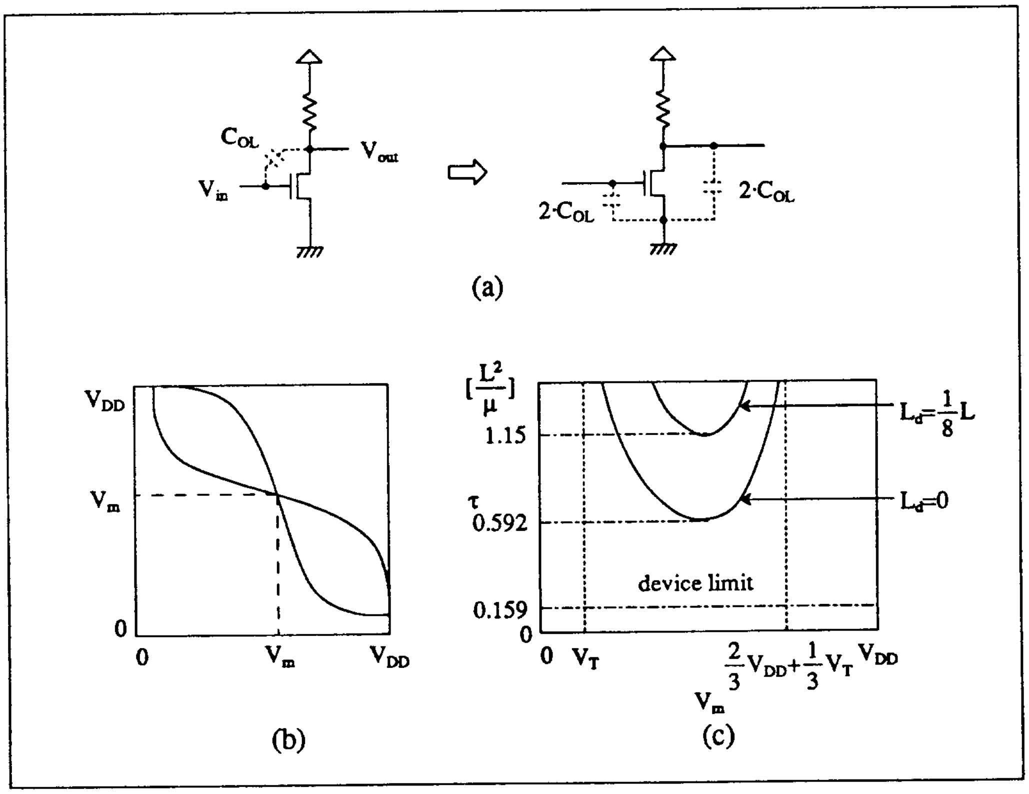

Fig. 4.15 MOS inverter with a resistive load. (a) Circuit. (b) Transfer curve. (c) V_m vs τ from the ideal device with no gate overlap capacitance.

to be able to source more current. If we decrease R too much, shunt effect of R shown in Eq. (4.26) will be dominant, resulting in decreased τ. There is a lower bound of R where the voltage gain of this inverter becomes 1. Beyond that point, regenerative feedback is broken and the circuit will ceases to work properly as a flip-flop. There is an optimum point that results in minimum τ. At optimum V_m and R, the following equations should hold.

$$I_D = \frac{1}{2}\mu C_{ox}\left(\frac{W}{L}\right)(V_m - V_T)^2 = \frac{V_{dd} - V_m}{R} \tag{4.27}$$

$$g_m R \geq 1 \tag{4.28}$$

$$\frac{d\tau}{dV_m} = 0 \tag{4.29}$$

Solving the above equations yields the following relations.

$$V_{m,opt} = V_{dd} - \frac{1}{\sqrt{3}}(V_{dd} - V_T) \tag{4.30}$$

$$\tau_{min} = \frac{(\sqrt{3}+1)^2}{3}\frac{L^2}{\mu}\frac{1}{V_{dd} - V_T} = 0.592\frac{L^2}{\mu} \tag{4.31}$$

The resistive inverter shows amplification only for $V_T \leq V_m \leq \frac{2}{3}V_{dd} + \frac{1}{3}V_T$.

τ as a function of V_m is shown in Fig. 4.14(c). Assuming $V_{dd} = 5.0V$, $V_T = 0.8V$, τ_{min} is 3.73 times larger than the τ achievable from a pure device.

Since its optimal V_m lies approximately in the middle of the supply voltage, it is trivial to design a buffer stage to restore the output voltage level and drive the external load. The input stage is also easy to implement since two series-connected MOS transistors, with twice the width of the core transistors, can properly change the state of the cell. The effect of the gate overlap capacitance, C_{OL}, has to be considered. Unless a special process is used, C_{OL} accounts for more than 10% of the total gate capacitance and the ratio tends to increase as the device scales down. Since the two nodes in the feedback loop have symmetric waveforms, the overlap capacitance can be split into two capacitors with 2 times as much as C_{OL}, due to Miller effect. Also, the effect of

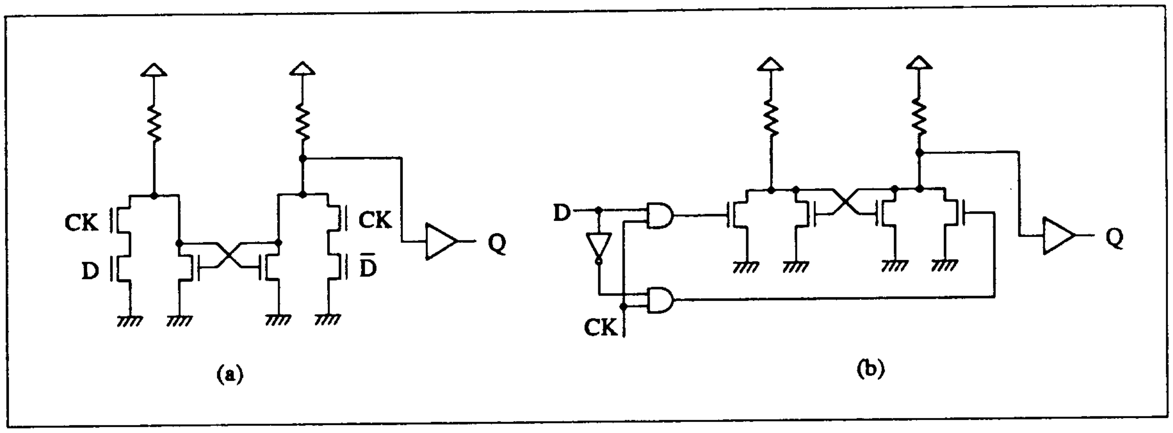

Fig. 4.16 Synchronizer with resistive load inverter. (a) Circuit with a simple input stage. (b) Circuit with a complex input stage for performance.

overlap capacitors due to the input stage, although they are not showing Miller effect, is significant because of the increased channel width. Fig. 1.16 (b) shows a better implementation in that respect: it suffers from increased delay time and added complexity, but it gains in reducing τ. Therefore, eq. (4.26) should be modified to include the effect of extra overlap capacitance of the input stage. The following is a new expression for τ for the circuit in Fig. 4.16 (b).

$$\tau = \frac{\frac{2}{3} C_{ox} W L + 5 C_{ox} W L_d}{g_m - \frac{1}{R}} \qquad (4.32)$$

where L_d is the length of lateral diffusion underneath the gate oxide. Thus, with $\frac{L}{L_d} = 0.125$, τ increases by 94% from the one expected from eq. (4.26). Revised optimal τ is $1.15 \frac{L^2}{\mu}$. The curve shown in Fig. 4.15 (c) should be modified to include a 62.5% penalty. Simulated results from 1.6μm CMOS process are shown later in Table 4.3.

4.4.2.2 MOS Synchronizers using Inverters with Current Source Load

For the inverter shown in Fig. 4.17(a), better performance in a synchronizer is expected since the pull-up provides bias current without shunting any signal current. τ from ideal device parameters free from parasitics is expected to be the same value as the one from pure device. Thus, the closer V_m is to V_{dd}, the less τ is expected according to Eq. (4.24). However, since the size of the PMOS transistors used in the current source tends to be larger than the size of the

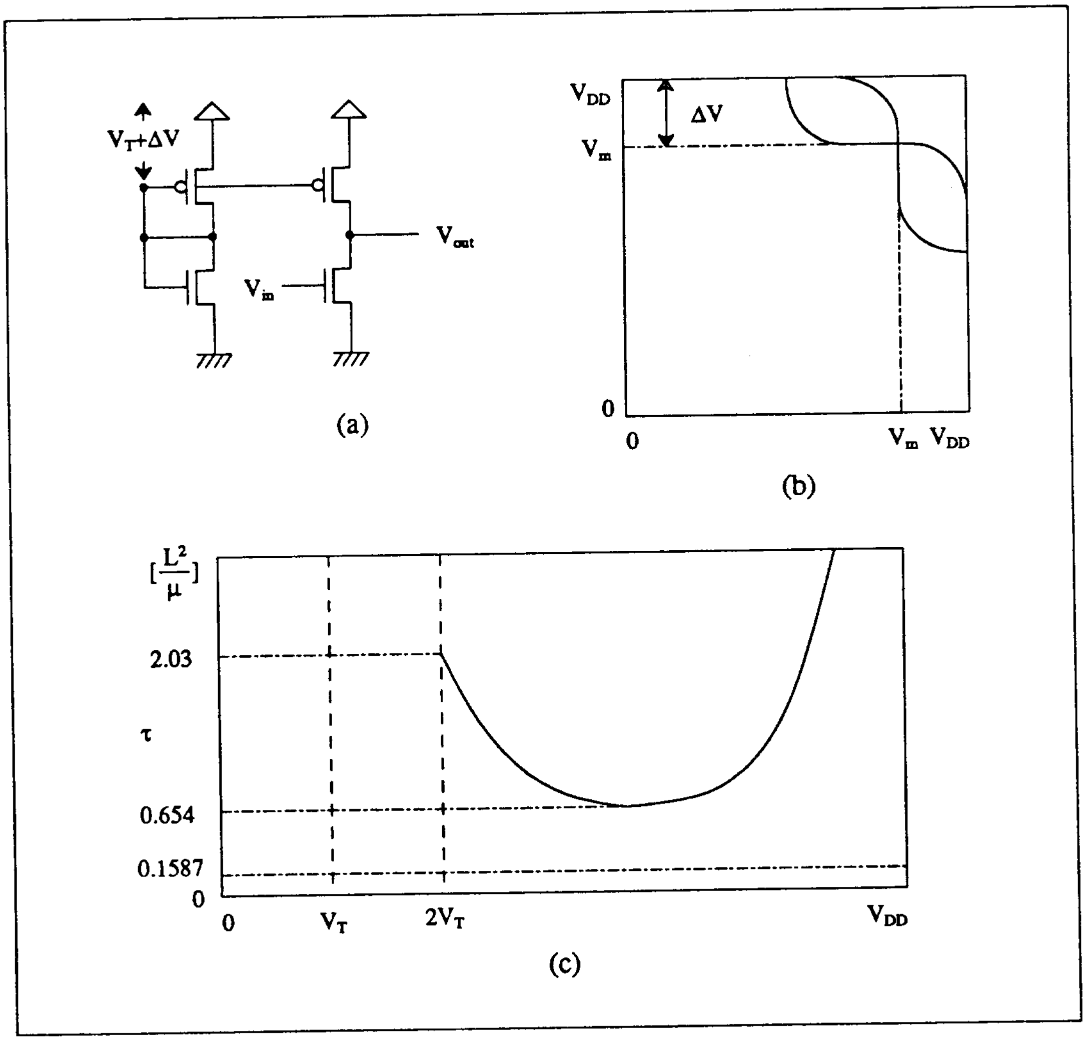

Fig. 4.17 Inverter with a current source load. (a) Circuit. (b) Transfer curve. (c) V_m vs τ.

NMOS, an effect of gate overlap capacitance should also be considered. If a too wide PMOS transistor is used, gate overlap capacitance of the PMOS will have more adverse effect on τ than the gain obtained by increasing V_m. However, if a too small PMOS transistor is used, V_m will be too low.

In Fig. 4.18, the reference, V_G, is generated by using a PMOS and an NMOS transistor. The PMOS transistor has the same size as the one in the core. However, the NMOS transistor is slightly smaller than the one in the core. By doing the above, V_m will be located at $V_G + |V_{T,P}|$, letting the PMOS be just on the edge of saturation, thereby giving extra gate drive, $|V_{T,P}|$, for the NMOS transistor. Thus, $V_m = V_G + |V_{T,P}|$. The following equations are derived to find an optimum width ratio of a PMOS and an NMOS transistor, considering all overlap capacitances. It is assumed that $V_T = V_{T,N} = -V_{T,P}$, and $L_d = L_{d,N} = L_{d,P}$.

$$\tau = \frac{\frac{2}{3} C_{ox} W_N L + 5 C_{ox} W_N L_d + C_{ox} W_P L_d}{\mu_N C_{ox} \left(\frac{W_N}{L} \right) (V_m + V_T)}$$

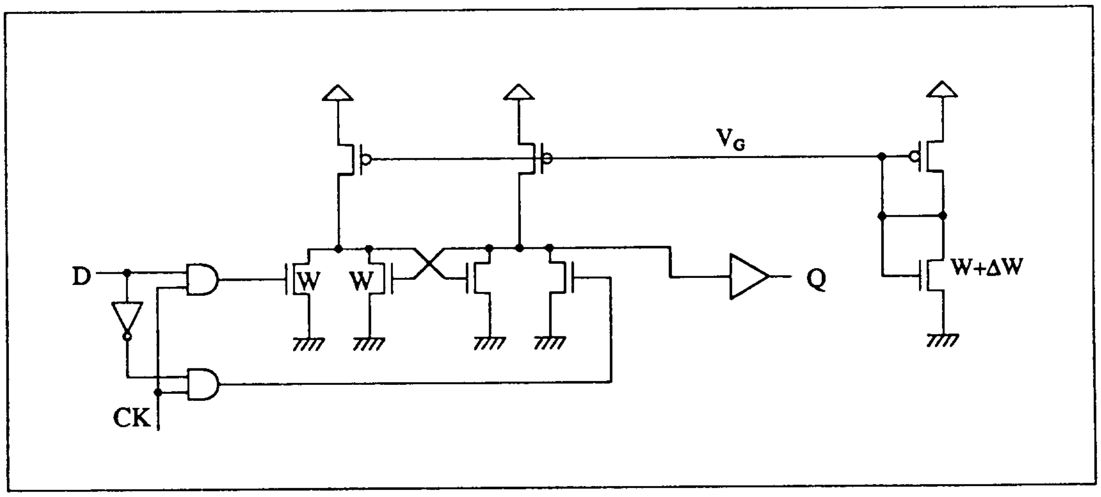

Fig. 4.18 A synchronizer with current source load.

$$= \frac{L^2}{\mu_N} \cdot \frac{[\frac{2}{3} + 5(\frac{L_d}{L}) + (\frac{W_P}{W_N})(\frac{L_d}{L})](1 + \sqrt{\frac{\mu_N W_N}{\mu_P W_P}})}{V_{dd} - V_T + \sqrt{\frac{\mu_N W_N}{\mu_P W_P}} V_T} \tag{4.33}$$

From this equation, as $W_P \to \infty$, $\tau \to \infty$ due to the dominant effect of the gate overlap capacitance of the PMOS transistor. However, as $W_P \to 0$, $\tau = 2.03 \frac{L^2}{\mu_N}$. Using $V_T = 0.8$ V, $\sqrt{\frac{\mu_N}{\mu_P}} = 1.68$, $\frac{L_d}{L} = 0.125$, optimum ratio of $\frac{W_N}{W_P} = 0.452$ or $W_P = 2.21\, W_N$. In optimum design, $\tau = 0.654 \frac{L^2}{\mu_N}$ and $V_m = 3.19$ V. An input stage can be designed similar to Fig. 4.16 (a) or (b).

4.4.2.3 MOS Synchronizers using Pseudo-NMOS Inverters

A pseudo-NMOS circuit shown in Fig. 4.19 has behavior like that of the previous two circuits. The load characteristic curve lies between that of a current source and a linear resistor. Its analytical analysis is very complicated since the pull-up PMOS is in linear region. Also, the capacitance seen from drain is a function of drain voltage. Assuming $C_{dg} = \frac{1}{2} C_g$, $V_T = V_{T,N} = -V_{T,P}$, and $L_d = L_{d,N} = L_{d,P}$, the following analysis produces a conservative estimate.

Using $V_T = 0.8$ V, $\sqrt{\frac{\mu_N}{\mu_P}} = 1.68$, $\frac{L_d}{L} = 0.125$, optimum ratio of $\frac{W_N}{W_P}$ is 2.50, or W_P is 0.40 W_N at $V_m = 2.64$V. Minimum τ is 1.21 $\frac{L^2}{\mu_N}$. Note that the optimum width of the PMOS transistor is approximately one fifth of the width of the PMOS transistor in the synchronizer with current source loads. The reason for this is that the pseudo-NMOS pull-up shunts more signal current, has 2 times more gate drive voltage, and contributes more overlap capacitance per unit width.

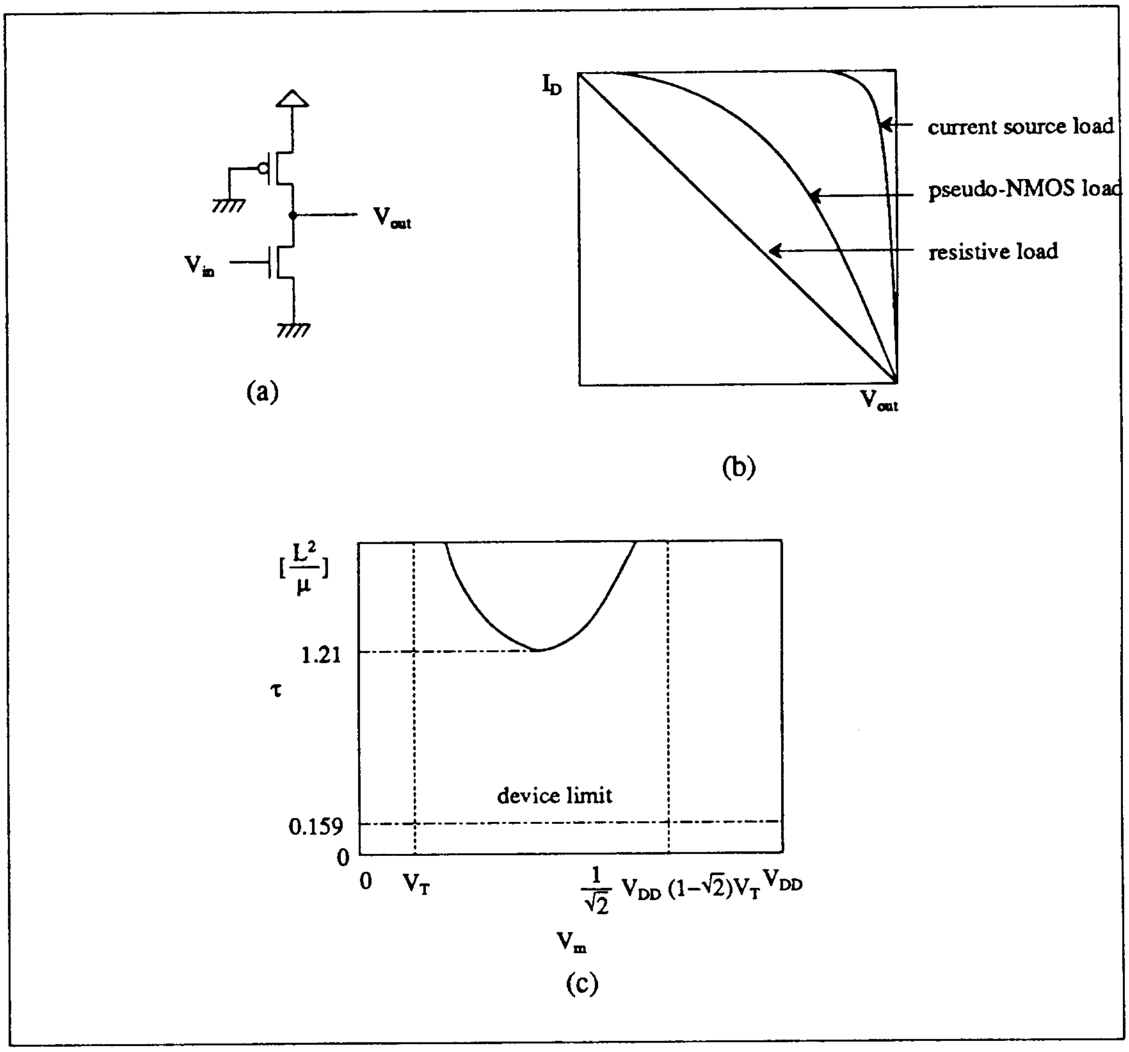

Fig. 4.19 (a) Pseudo-NMOS inverter. (b) Load characteristics. (c) V_m vs τ.

$$V_m = V_T + V_{dd} - V_T \frac{1}{\sqrt{1 + \frac{\mu_N W_N}{\mu_P W_P}}} \tag{4.34}$$

$$g_{m,N} = \frac{k_N}{\sqrt{1 + \frac{\mu_N W_N}{\mu_P W_P}}} (V_{dd} - V_T) \tag{4.35}$$

$$r_{o,P} = \frac{k_P}{\sqrt{1 + \frac{\mu_N W_N}{\mu_P W_P}}} \tag{4.36}$$

$$\tau = \frac{\frac{2}{3} C_{ox} W_N L + 5 C_{ox} W_N L_d + C_{ox} W_P L_d + \frac{1}{2} C_{ox} W_P L}{g_m - r_{o,P}} \tag{4.37}$$

$$= \frac{L^2}{\mu_N} [\frac{2}{3} + 5 (\frac{L_d}{L}) + (\frac{W_P}{W_N}) + \frac{1}{2} (\frac{W_P}{W_N})] \frac{\sqrt{1 + \frac{\mu_N W_N}{\mu_P W_P}}}{(V_{dd} - V_T) (1 - \frac{\mu_N W_N}{\mu_P W_P})} \tag{4.38}$$

4.4.2.4 Full CMOS Synchronizers

For applications where static power dissipation should be avoided, a full CMOS synchronizer can be used. Since PMOS transistors, which have inferior frequency characteristics, must be in the signal path, full CMOS synchronizers are not expected to have better performance than the synchronizer with current source load.

Two possible circuit configurations are shown in Fig. 4.20. For the circuit with two inverters, the only design parameter to consider is the width ratio of PMOS and NMOS transistors. With $W_p = W_n = W$ and $V_{T,n} = -V_{T,p} = V_T$, the following equations hold.

The condition of minimum τ is $W_P = W_N$. With optimum design, τ is 1.14 $\frac{L^2}{\mu_N}$. The two circuits in Fig. 4.20 (a) and (b) have similar performance but the one with NAND gates needs attention when selecting a data input and a feedback input. Node A should be the feedback input since it has less capacitance in the loop than node B.

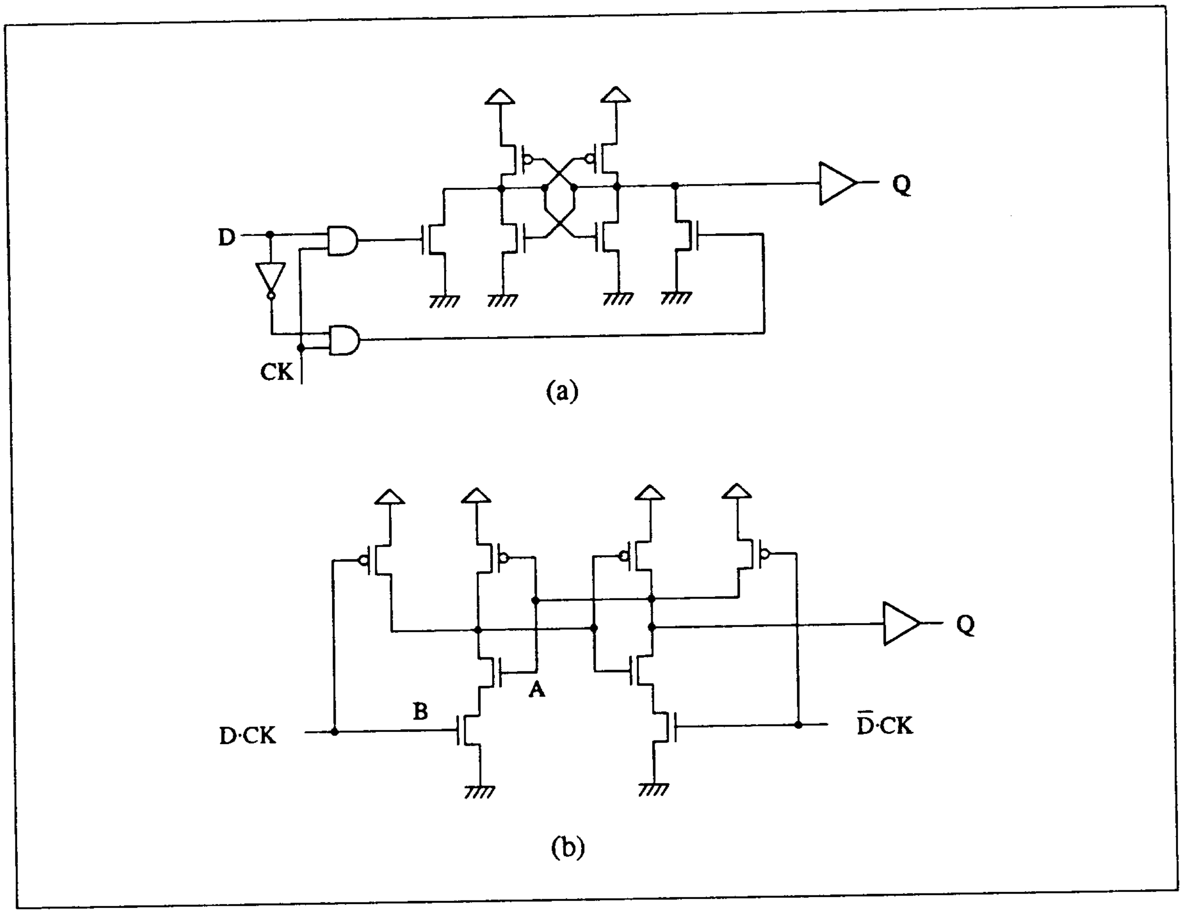

Fig. 4.20 Two synchronizer circuits in full CMOS. (a) With an inverter core. (b) With a NAND core.

$$V_m = \frac{(V_{dd} - V_T) + \sqrt{\frac{\mu_N}{\mu_P}}\, V_T}{1 + \sqrt{\frac{\mu_N}{\mu_P}}} \qquad (4.39)$$

$$\tau = \frac{\frac{2}{3} C_{ox} (W_N + W_P) L + 5\, C_{ox} (W_N + W_P) L_d}{g_{m,N} + g_{m,P}}$$

$$= \frac{L^2}{\mu_N} \frac{1}{V_{dd} - 2\,V_T} \sqrt{\frac{\mu_N}{\mu_P}} \left[\frac{2}{3} + 5\left(\frac{L_d}{L}\right) \right] \left(\sqrt{\frac{W_N}{W_P}} + \sqrt{\frac{W_P}{W_N}} \right) \qquad (4.40)$$

4.4.2.5 MOS Synchronizers using Bandwidth Multipliers

Darlington circuits with MOS devices cannot achieve the same performance as in bipolar technology. From Fig. 4.21(a), V_{GS} of the two transistors cannot be more than half of the supply

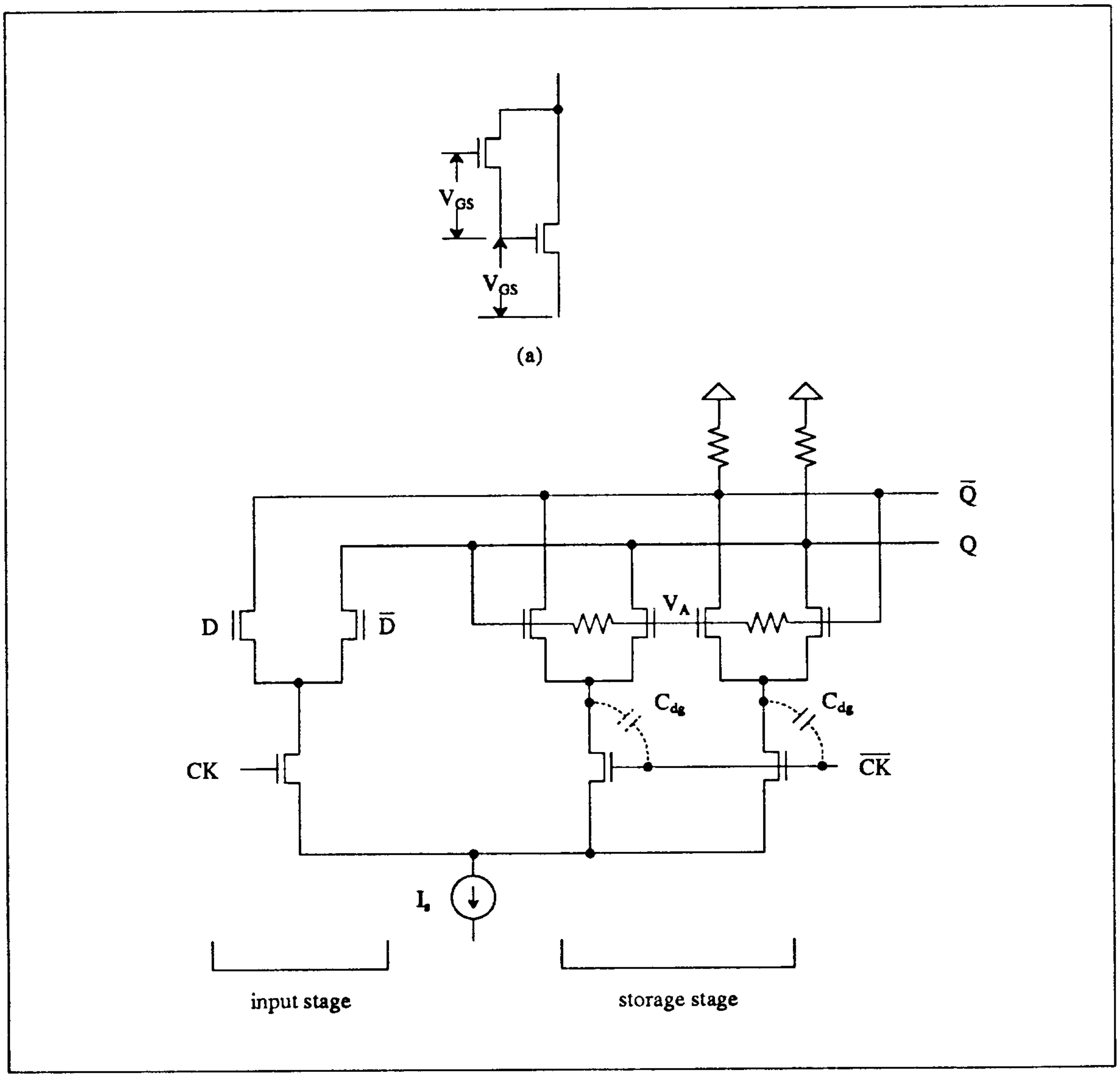

Fig. 4.21 Bandwidth multiplication in MOS technology. (a) Darlington pair in MOS. (b) f_T doubler.

voltage. In this case, the performance of each transistor is worse than a single transistor operating at high V_{GS} close to V_{dd}. Taking into account that Darlington circuits have increased parasitics, it will not be possible to achieve single device performance in any case.

The f_T doubler circuit shown in Fig. 4.21 (b) does not have the same problem. It can utilize maximum V_{GS}. However, C_{OL}'s of the current steering transistors will degrade the circuit performance significantly. Performance gain will not be achievable unless the drain to gate to drain overlap capacitance C_{OL} decreases dramatically from the one of current process technology. C_{OL} amounts to typically more than 10% of that gate capacitance, which make this circuit practically useless.

4.4.2.6 Comparison of MOS Synchronizers

The results of circuit simulation are shown in Table 4.3. The simulation results should be interpreted carefully since many effects are not considered. For example, the drain and source junction parasitic capacitance are not considered since they are heavily dependent of the layout technique. Parasitic effects can be minimized by using large ring-shaped transistors. Also, the AC model of the MOS device in the circuit simulation is not based on a more accurate model based on transcapacitance [4.24]. In this model, $C_{dg} \neq C_{gd}$. Rather than being zero, $C_{dg} \approx 0.3\ C_g$ in the saturation region. However, the results could be used to determine which circuit has the best performance. Most significant effect is due to velocity saturation of channel carriers [4.18]. It causes g_m to be proportional to $(V_{GS} - V_T)^n$, $1 < n < 2$, deviating from eq. 4.22. With significant velocity saturation effect, $n = 1$, NMOS transistors have the same g_m, regardless of its bias. Therefore, minimizing parasitic capacitance and reducing current shunt effect are the only design goals. In this case, layout technique can also play an important role. In Table 4.3, device model parameters are from Hewlett Packard 1.6μm CMOS process.

Table 4.3 Simulated results.

Circuit	Figure	τ, ideal	τ, pure †	τ, full †	τ, optimal †
Resistive Load	Fig. 4. 15	$1.15\times\frac{L^2}{\mu_N}$	24.4ps (100,800)	107ps (100,800)	73.8ps (100,2k)
Current Source Load	Fig. 4. 17	$0.65\times\frac{L^2}{\mu_N}$	16.7ps (100,221)	99.0ps (100,221)	68.5ps (100,120)
Pseudo-NMOS Load	Fig. 4. 19	$1.21\times\frac{L^2}{\mu_N}$	24.9ps (100,40)	74.1ps (100,40)	65.4ps (100,20)
Full CMOS	Fig. 4. 20 (a)	$1.14\times\frac{L^2}{\mu_N}$	22.8ps (100,100)	115ps (100,100)	88.9ps (100,70)
Full CMOS	Fig. 4. 20 (b)	$1.14\times\frac{L^2}{\mu_N}$	21.9ps (100,100)	130ps (100,100)	105ps (100,70)
F_T doubler	Fig. 4. 21	-	63.9ps	151.7ps	-

† (W_N , W_P or Resistance)

Table 4.4 SPICE model parameters used.

Parameter	NMOS, pure	NMOS, full	PMOS, pure	PMOS, full
LEVEL	1	2	1	2
VTO	0.75	0.75	0.75	0.75
KP	76	76	27	27
GAMMA	0.40	0.40	0.50	0.50
LAMBDA	0.025	0.025	0.045	0.045
TOX	25N	25N	25N	25N
NSUB	4E16	4E16	2.0E16	2.0E16
LD	0.2U	0.2U	0.2U	0.2U
UEXP	0.16	0.16	0.15	0.15
VMAX	5.5E4	5.5E4	9.0E4	9.0E4
XQC	0.0	0.4	0.0	0.4

4.5 Conclusion

Since it is impossible to eliminate metastability in a synchronizer, efforts should be directed toward decreasing the probability of metastability. Time constants for the normal exponential escape from metastability are compared among various implementations of synchronizers. Although τ is primarily determined by device technology, a circuit technique called bandwidth multiplication can be used to overcome this limitations in bipolar technology. However, with current device technology, bandwidth multiplication concept is not effective since its device bandwidth is dominated by parasitics not by intrinsic element. In current technology, ECL performs best. For MOS circuits, gate to drain overlap capacitance has the dominant effect on limiting τ other than intrinsic transistors. Bandwidth doubling will not be possible since the overlap capacitance along with junction capacitance will have worse effect as device scales. Among conventional circuits, synchronizers built with pseudo-NMOS inverters performed best even though they consume static power.

CHAPTER 5

Interface Techniques

This chapter concerns the problem of interfacing two systems for information exchange. Since most structurally simple systems do not cause any serious problem in this area, I will concentrate on the specific case of interfacing two synchronous systems with independent clocks, which poses a challenging problem in real systems. Concepts defined here can be extended to any less complex systems including asynchronous-synchronous systems interfaces. First, I briefly introduce a background of the issues in section 5.1. I classify various handshake mechanisms and their associated controllers in section 5.2 and a real implementation structure is shown in section 5.3. Section 5.4 summarizes this chapter.

5.1 Introduction

For the purposes considered here, an interface is a hardware/software mechanism that handles communication between two systems that do not have a complete knowledge of each other's timing behavior. Thus, a communication channel through a data bus in a typical computer system cannot be called an interface, because the main controller has a complete knowledge about the timing of the sender and receiver. Every digital system has interfaces to external world. They may be to a video terminal, a printer, or another digital system. Even within an apparently isolated system, interfaces can be placed between its subsystems. For example, many subsystems within a computer often share a standard backplane bus as a communication channel. Since there is no common control structure governing all of the subsystems except the bus protocol, the whole system can be thought of as a collection of a common bus and subsystems having interfaces to it.

Three performance factors must be considered to design an interface - throughput, latency and reliability. They are traded off with each other considering systems requirement and implementation difficulty. Throughput, or communication bandwidth, is a main concern for most high speed computer systems. It determines the volume of instructions and data delivered in a given time, and is directly related with performance. Another consideration is latency. When a piece of information needs to be delivered to another place, a delay is inevitably involved in setting up a communication path, delivering the information, and receiving it. Reliability issues are raised whenever a synchronization event occurs. Most of the interfaces other than fully synchronous and asynchronous ones entail important reliability considerations.

In case of a bus-based multiprocessor system like the SPUR, most of the transactions across the interface between the CPU and the bus occur in block mode transfers on cache misses, and take approximately 20 cycles to finish including arbitration cycles and assuming the typical access time of a memory board. Throughput is a bigger concern in block mode transactions than latency. For emphasis of latency, a word may be transmitted in a single transfer mode. In this case, only a few cycles are needed to finish a transaction. However, the overhead of bus arbitration and access preparation in the memory board does not amortize and decreases the overall throughput. whereas

5.2 Handshake Mechanisms

A handshake mechanism is a protocol to exchange timing information for synchronization [5.1]. It is needed when complete timing information is not known and speed-independent operation is desired. Most interfaces adopt a handshake mechanism and providc basic hardware support. Handshake mechanisms assure either mutual exclusion or concurrency synchronization. Mutual exclusion must be sustained when one should await the other for completion or there is a competition for a shared resource. Concurrency synchronization is needed when two operations need to join after concurrent operations. The difference is shown in Fig. 5.1.

In Fig. 5.1(b), it is noted that only one side is in active operation while both sides can be in productive operations as shown in 5.1(c). The relation between a CPU and a memory system is

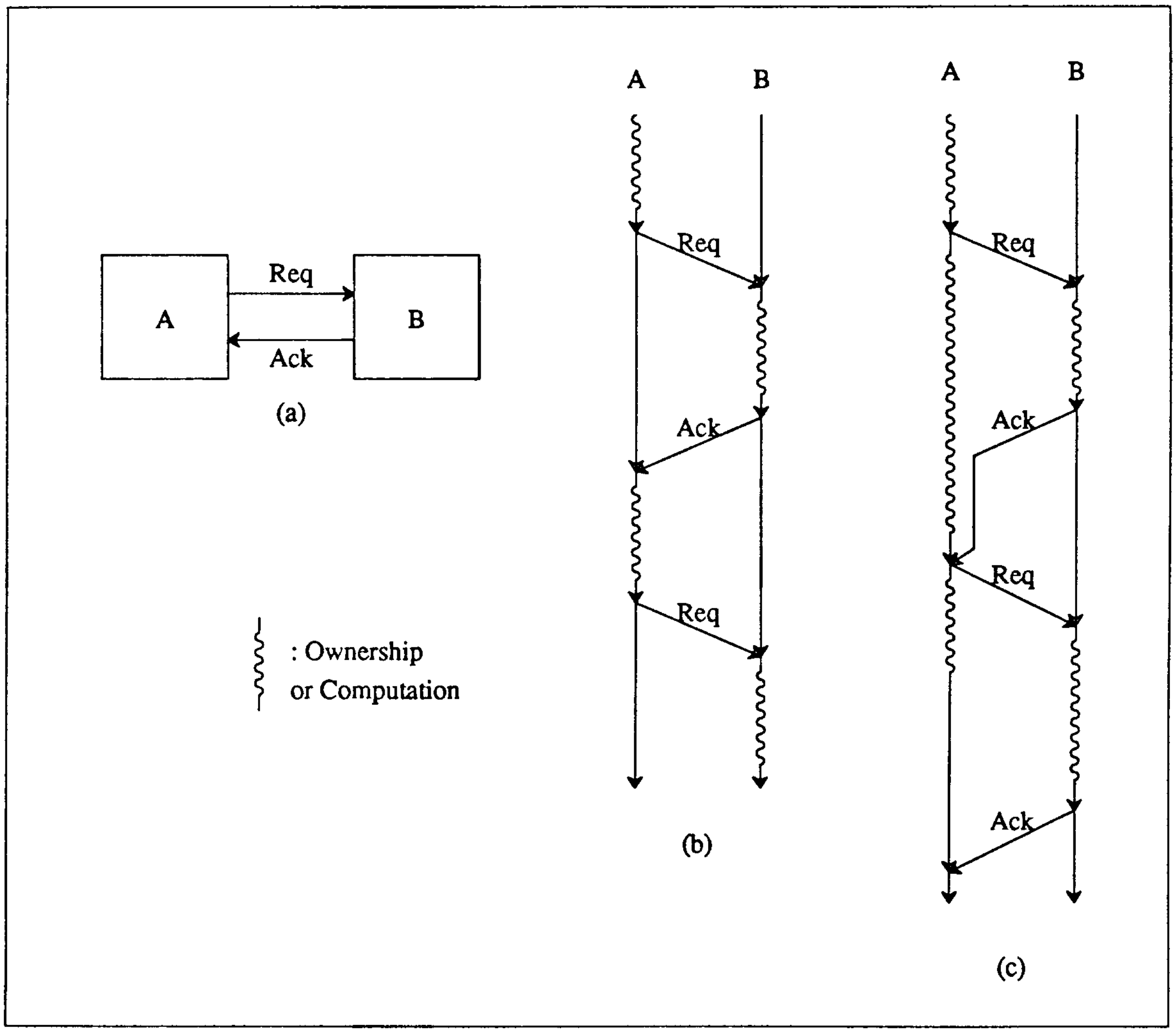

Fig. 5.1 Two uses for a handshake mechanism. (a) Two systems with a handshake channel. (b) Mutual exclusion. (b) Concurrency synchronization.

an example of these operations. The CPU issues memory addresses with VALID signal and the memory system provides data with READY signal when it finished preparing to send data. For systems built around conventional microprocessors, the CPU becomes idle until memory requests are finished. There is no overlap. On the other hand, for a CPU that generates prefetching addresses, concurrent operation is possible. The memory system prepares the next instruction and the CPU executes the current instruction, stalling only when the memory system fails to provide the next instruction in time. Note that the memory system cannot initiate a request to the CPU. A simpler example showing contrasts is the relation between a processor and a printer. The processor initiates a request to the printer to print a character 'A'. There is a design choice as to whether the processor will wait for the printer to finish or will be allowed to continue next sequences until another print request is issued. In the first scheme, the CPU is idle whenever the printer is active. In the second, concurrent operation of the CPU and the printer is possible unless the CPU issues a second request before the printer's completion of the first operation. In both examples, the two sides have a master-slave relationship. Side A initiates transactions and side B, on completion, relinquishes its operation or ownership. There is no way of initiating a transaction from side B. A handshake mechanism that allows a bidirectional channel will be discussed later.

5.2.1 2 Phase Handshake Signaling Scheme

There are two kinds of signaling schemes in a handshake mechanism - 4 phase and 2 phase. The 2 phase handshake mechanism shown in Figs. 5.2 (a) and (c) relies on edges for signaling. When side A has a request to side B, a transition on the request line is made, from low to high or reverse depending on the current level. On completion, the receiver sends an acknowledgement to the sender by making the same type of edge. A similar transaction can be made for concurrency synchronization as shown in Figs. 5.2(b) and (d).

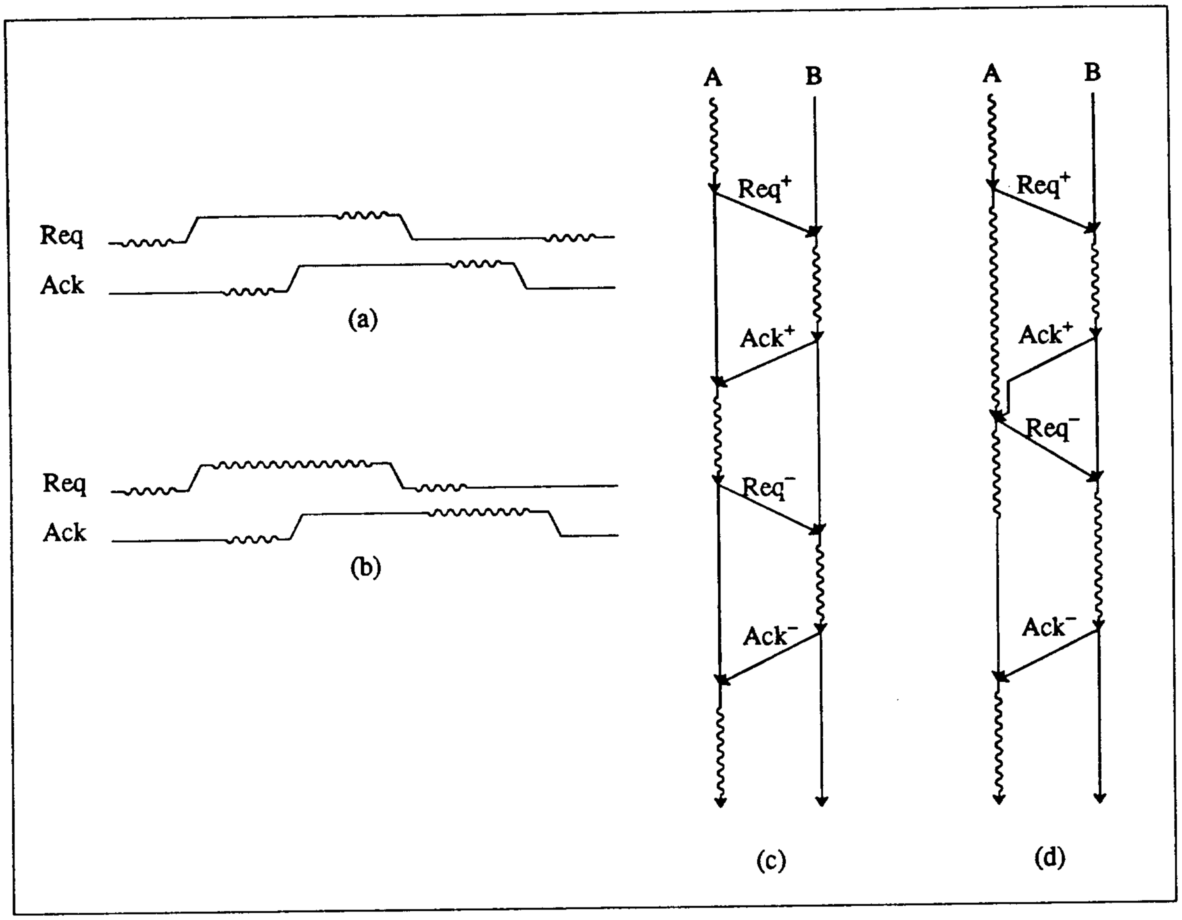

Fig. 5.2 2 phase signaling scheme. (a) Signal transitions for mutual exclusion. (b) Signal transitions for concurrency synchronization. (c) Wave diagram for (a). (d) Wave diagram for (b).

5.2.2 4 Phase Handshake Signaling Scheme

In a 4 phase handshake protocol for mutual exclusion, a request line is asserted, telling the sender's intention to make an request. On receiving the request from the sender, the receiver sends an acknowledgement after it completes its transactions. The sender, on receiving the acknowledgement, disasserts the request. The receiver disasserts the acknowledgement on receiving a disasserted request signal. Another initiation can be made only after the sender receives a disasserted acknowledgement from the receiver. In this way, a fully interlocked operation is performed without losing any data associated with the request/acknowledgement lines. Since 4

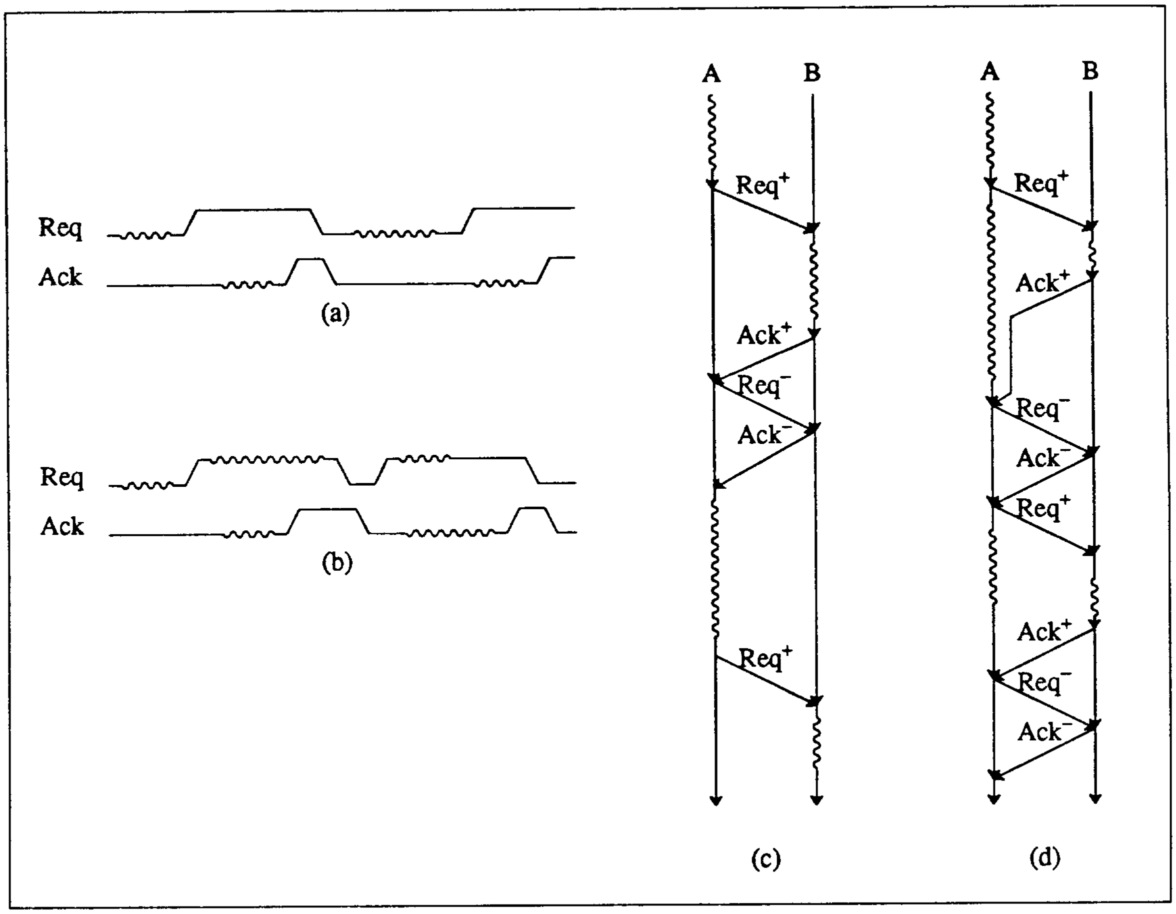

Fig. 5.3 4 phase signaling scheme. (a) Signal transitions for mutual exclusion. (b) Signal transitions for concurrency synchronization. (c) Wave diagram for (a). (d) Wave diagram for (b).

edges, thus 4 transitions, are involved in a transaction, we call this a 4 phase handshake signaling.

The 2 phase scheme has an advantage over the 4 phase handshake in that the former needs only two trips to finish a transaction. For a simple supplier-consumer relation, the 2 phase scheme is superior to the 4 cycle scheme in terms of bandwidth. One of the standard microprocessor buses, Futurebus, uses a 2 cycle handshake in block mode transfers to increase the bandwidth [5.2]. However, when implemented, it must carry a state information as to which of the two types of transitions are expected.

5.3 Hardware and Protocol Implementation

5.3.1 4 Phase Handshake

Two special circuits are needed when both sides are synchronous systems with independent clocks. A synchronizer are required in each path since the incoming signal from the other side is not guaranteed to be value- and time-qualified. Deglitchers may be needed when the output from the finite state machine (FSM) is not guaranteed to be glitch-free. A glitch in the request/acknowledge lines may cause a deadlock because the lines are sampled and monitored by the receiver continually. Deglitching is be done by sampling the output from the FSM at the valid edge and holding it for the rest of the cycle. The simpliest method is to use a D-type latch. The FSMs should implement a handshake protocol. A pseudo code for the FSMs with a 4 phase handshake mechanism is shown in Fig. 5.5.

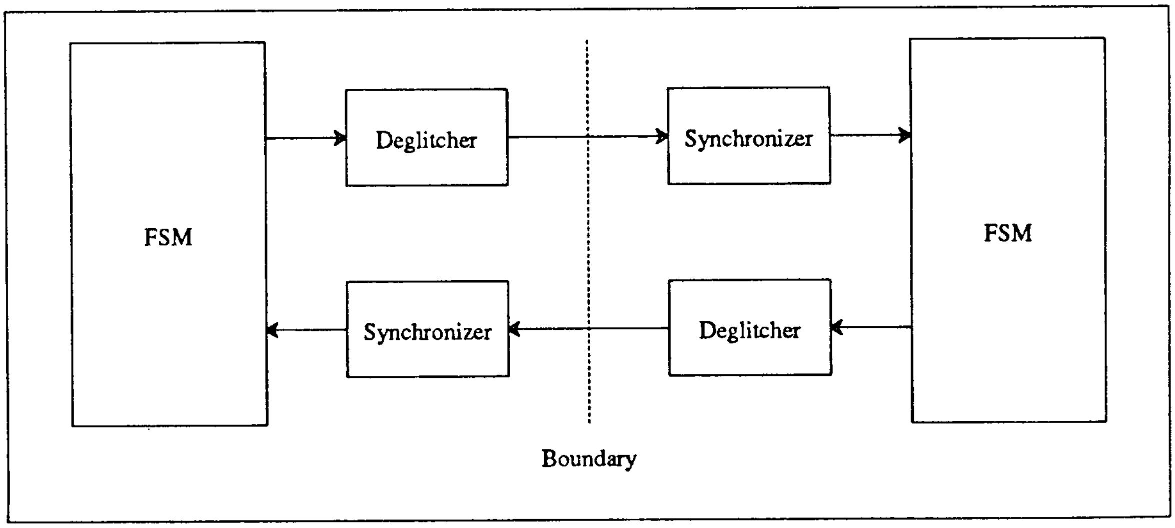

Fig. 5.4 Simple interface between two synchronous systems.

```
master()
input inputs, ack, sampled_req, reset;
output outputs, req;
state states;
{
if (reset == 1) { req = 0; states = RESET_STATE; }
else if (sampled_req == 0 and ack == 0) /* MASTER'S HOME STATE */
        {
        switch (inputs, states)
                {
                (INPUT_A, STATE_A): { req = 0; states = NEXT_STATE; }
                --------- enumeration of all transitions ----------
                (INPUT_Z, STATE_Z): { req = 1; states = WAIT_STATE; }
                }
        }
else if (sampled_req == 1 and ack == 0) { req = 1; states = WAIT_STATE; }
else if (sampled_req == 1 and ack == 1) { req = 0; states = WAIT_STATE; }
else if (sampled_req == 0 and ack == 1) { req = 0; states = WAIT_STATE; }
else ;
}

slave()
input inputs, req, sampled_ack, reset;
output outputs, ack;
state states;
{
if (reset == 1) { ack = 0; states = RESET_STATE; }
else if (sampled_ack == 0 and req == 0) { ack = 0; states = WAIT_STATE; }
else if (sampled_ack == 0 and req == 1) /* SLAVE'S SERVICE TO MASTER'S REQUEST */
        {
        switch (inputs, states)
                {
                (INPUT_A, STATE_A): { ack = 0; states = NEXT_STATE; }
                --------- enumeration of all transitions ----------
                (INPUT_Z, STATE_Z): { ack = 1; states = WAIT_STATE; }
                }
        }
else if (sampled_ack == 1 and req == 1) { ack = 1; states = WAIT_STATE; }
else if (sampled_ack == 1 and req == 0) { ack = 0; states = WAIT_STATE; }
else ;
}
```

Fig. 5.5 Pseudo code of the FSMs for a 4 phase handshake implementation of mutual exclusion. In WAIT_STATE, no operation is performed in the FSM; previous states are retained.

5.3.2 2 Phase Handshake

A pseudo code for FSMs with the 2 phase handshake mechanism is shown in Fig. 5.6. Sampled_req and sampled_ack remember the previous levels so that the FSM can detect edges. The 2 phase handshake has better performance: the master does not have to stall any cycle while a master in the 4 phase mechanism has to stall at least 2 cycles before it sends another request.

```
master()
input inputs, acknowledge, sampled_request, reset;
output outputs, request;
state states;
{
if (reset == 1) { req = 0; states = RESET_STATE; }
else if (sampled_req == 0 and ack == 0) /* MASTER' FIRST HOME STATE */
        {
        switch (inputs, states)
                {
                (INPUT_A, STATE_A): { req = 0; states = NEXT_STATE; }
                --------- enumeration of all transitions ----------
                (INPUT_Z, STATE_Z): { req = 1; states = WAIT_STATE; }
                }
        }
else if (sampled_req == 1 and ack == 0) { req = 1; states = WAIT_STATE; }
else if (sampled_req == 1 and ack == 1) /* MASTER' SECOND HOME STATE */
        {
        switch (inputs, states)
                {
                (INPUT_A, STATE_A): { req = 1; states = NEXT_STATE; }
                --------- enumeration of all transitions ----------
                (INPUT_Z, STATE_Z): { req = 0; states = WAIT_STATE; }
                }
        }
else if (sampled_req == 0 and ack == 1) { req = 0; states = WAIT_STATE; }
else ;
}

slave()
input inputs, req, sampled_ack, reset;
output outputs, ack;
state states;
{
if (reset == 1) { ack = 0; states = RESET_STATE; }
else if (sampled_ack == 0 and req == 1) /* SLAVE'S SERVICE TO MASTER'S REQUEST */
        {
            switch (inputs, states)
                {
                (INPUT_A, STATE_A): { ack = 0; states = NEXT_STATE; }
                ---------- enumeration of all transitions ----------
                (INPUT_Z, STATE_Z): { ack = 1; states = WAIT_STATE; }
                }
            }
    else if (sampled_ack == 1 and req == 1) { ack = 1; states = WAIT_STATE; }
    else if (sampled_ack == 1 and req == 0) /* SLAVE'S SERVICE TO MASTER'S REQUEST */
            {
            switch (inputs, states)
                {
                (INPUT_A, STATE_A): { ack = 1; states = NEXT_STATE; }
                ---------- enumeration of all transitions ----------
                (INPUT_Z, STATE_Z): { ack = 0; states = WAIT_STATE; }
                }
            }
    else if (sampled_ack == 0 and req == 0) { ack = 0; states = WAIT_STATE; }
    else ;
    }
```

Fig. 5.6 Pseudo code of the FSMs for the 2 phase handshake implementation of mutual exclusion. In WAIT_STATE, no operation is performed in the FSM; previous states are retained.

5.3.3 Modified 2 Phase Handshake

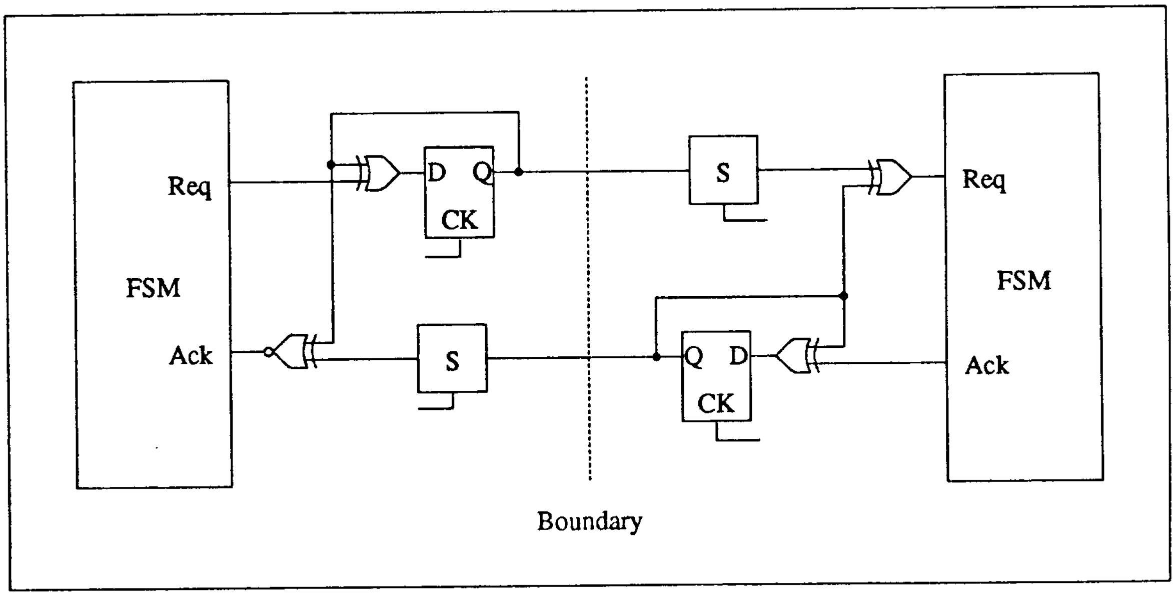

Fig. 5.7 External interface logic to simplify the FSM description of a 2 phase handshake mechanism.

With external hardware support shown in Fig. 5.7, a FSM description and associated hardware can be simplified, but the performance of a 2 phase handshake can still be retained. It contains 4 exclusive OR gates and latches. The simplified FSM description is shown in Fig. 5.8. An interface logic allows a request line to be asserted for only one cycle to log the request to the slave, without requiring the master to hold the request line until an acknowledgement arrives. Similarly, a one-cycle acknowledgement from the slave informs the master of the completion of the transaction. The master FSM looks at the acknowledge to see if there is any pending request. "1" means there is no pending request, and "0" means the slave is still working on the master's request. Similar mechanisms are applicable to the slave side.

```
master()
input inputs, ack, reset;
output outputs, req;
state states;
{
if (reset == 1) { req = 0; states = RESET_STATE; }
else if (ack == 1) /* MASTER'S HOME STATE */
        {
        switch (inputs, states)
                {
                (INPUT_A, STATE_A): { req = 0; states = NEXT_STATE; }
                --------- enumeration of all transitions ----------
                (INPUT_Z, STATE_Z): { req = 1; states = WAIT_STATE; }
                }
        }
else if (ack == 0) { req = 0; states = WAIT_STATE; }
else ;
}

slave()
input inputs, req, reset;
output outputs, ack;
state states;
{
if (reset == 1) { ack = 0; state = RESET_STATE; }
else if (req == 1 ) /* SLAVE'S HOME STATE */
        {
        switch (inputs, states)
                {
                (INPUT_A, STATE_A): { ack = 0; states = NEXT_STATE; }
```

```
                    --------- enumeration of all transitions ----------
                    (INPUT_Z, STATE_Z): { ack = 1; states = WAIT_STATE; }
                    }
          }
    else if (req == 0 ) { ack = 0; states = WAIT_STATE; }
    else ;
    }
```

Fig. 5.8 Pseudo code of the FSMs for a 2 phase handshake FSM with external hardware support. In WAIT_STATE, no operation is performed in the FSM; previous states are retained.

5.3.4 Data Delivery Circuits

Request/acknowledge lines deal only with timing information of the data. They don't convey any real data. Their associated data should also be delivered along with request/acknowledge signals. Since the timing information is already contained in the request/acknowledge signals, there is no need for synchronization or deglitching circuits for data, under the following conditions:

(1) The request signal must be asserted no earlier than its data when crossing the boundary.

(2) The sender holds its data until it receives acknowledge.

(3) The receiver looks at the incoming data only when request is asserted.

Condition (1) is needed to prevent the possibility of the request reaching the receiver before its associated data arrives. The receiver may look at old data erroneously. Although the sender FSM asserts request and data simultaneously, a small amount of skew introduced during the transmission may cause the request to be sampled before its data arrives. A delay circuit may have to be introduced to prevent such error conditions violating condition (1). Condition (2) is needed since the sender does not know exactly when the data is read by the receiver. Condition

(3) has to be satisfied by the receiver. Since data is not value- or time-qualified unless the request is asserted, it should be ignored during that time. An example circuit is shown in Fig. 5.9.

5.3.5 Bidirectional Handshake Mechanism

Sometimes an interface needs to handle requests from both sides. For example, in a shared memory multiprocessor, private caches should be accessed from the common bus as well as from the CPU to implement a snooping cache coherency protocol. The interface must handle the request from the bus to update cache tag entries, as well as the request from the CPU for fetching instructions and data from the main memory through the bus. The single sided communication protocol explained so far can be extended to cover bidirectional communication under certain restrictions. When the slave side wants to initiate a transaction, it can assert the acknowledge first to tell the master its intention. The acknowledge in this case serves as a request signal. The master can differentiate this from the conventional acknowledge since the acknowledge arrived without

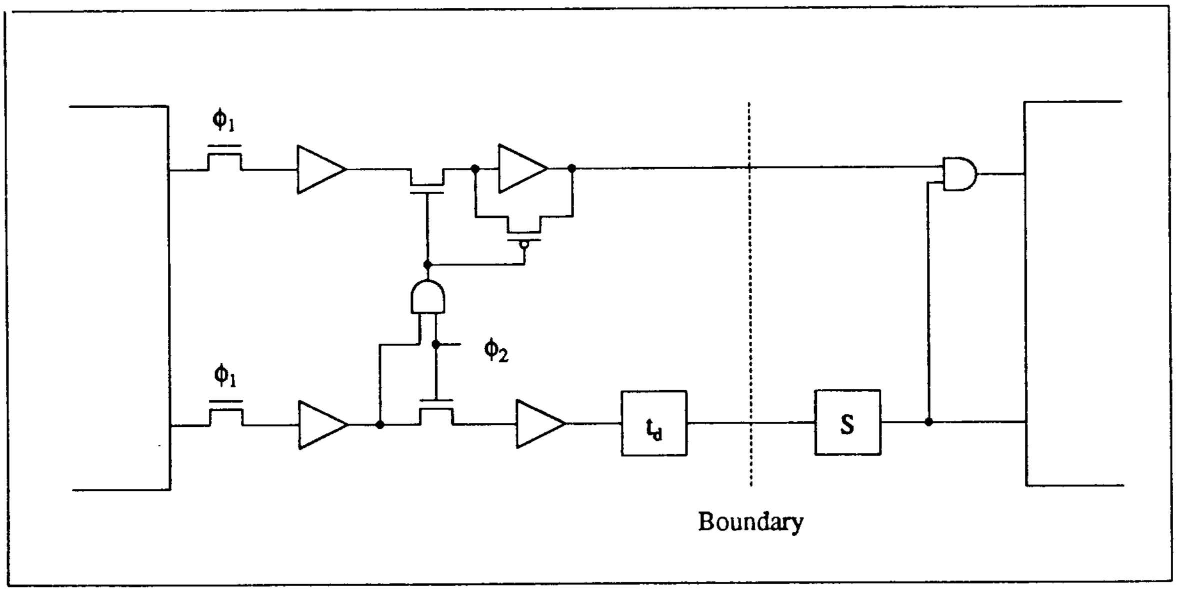

Fig. 5.9 Complete one-sided handshake circuit.

the request. Problems occur when both sides try to initiate a transaction at the same time. The acknowledge from the slave may reach the master just after the request is issued by the master. The master may mistake the acknowledge as the completion of the request although it is a request from the slave. The same thing can happen at the slave side. This error can be prevented if both side look at the incoming request or acknowledge only after a maximum path delay elapses. On detecting collision, the master may retry the request, while the slave withdraws its initiation. This scheme has many disadvantages. The two sides are no longer independent. When the clock frequency of one side changes, it affects the other side. Also, the implementation of the protocol is very complicated.

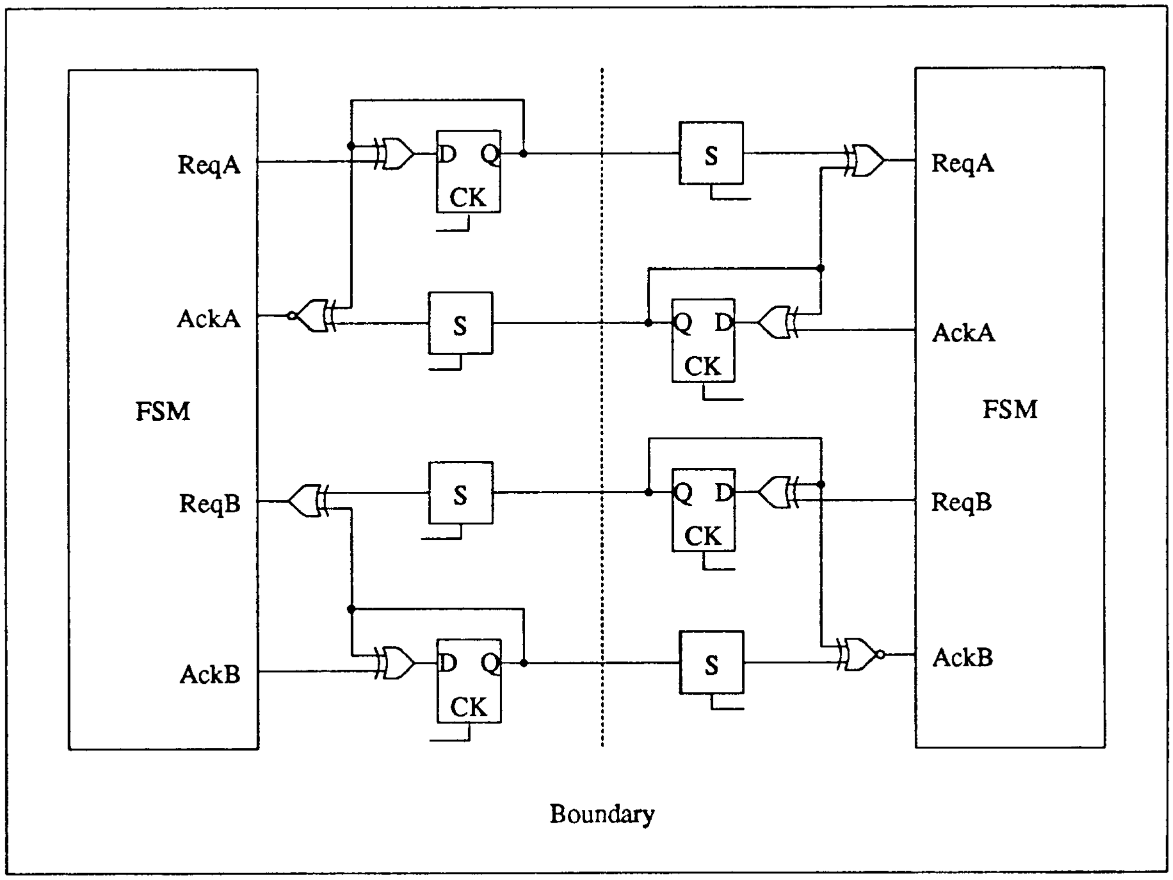

Fig. 5.10 Bidirectional handshake interface.

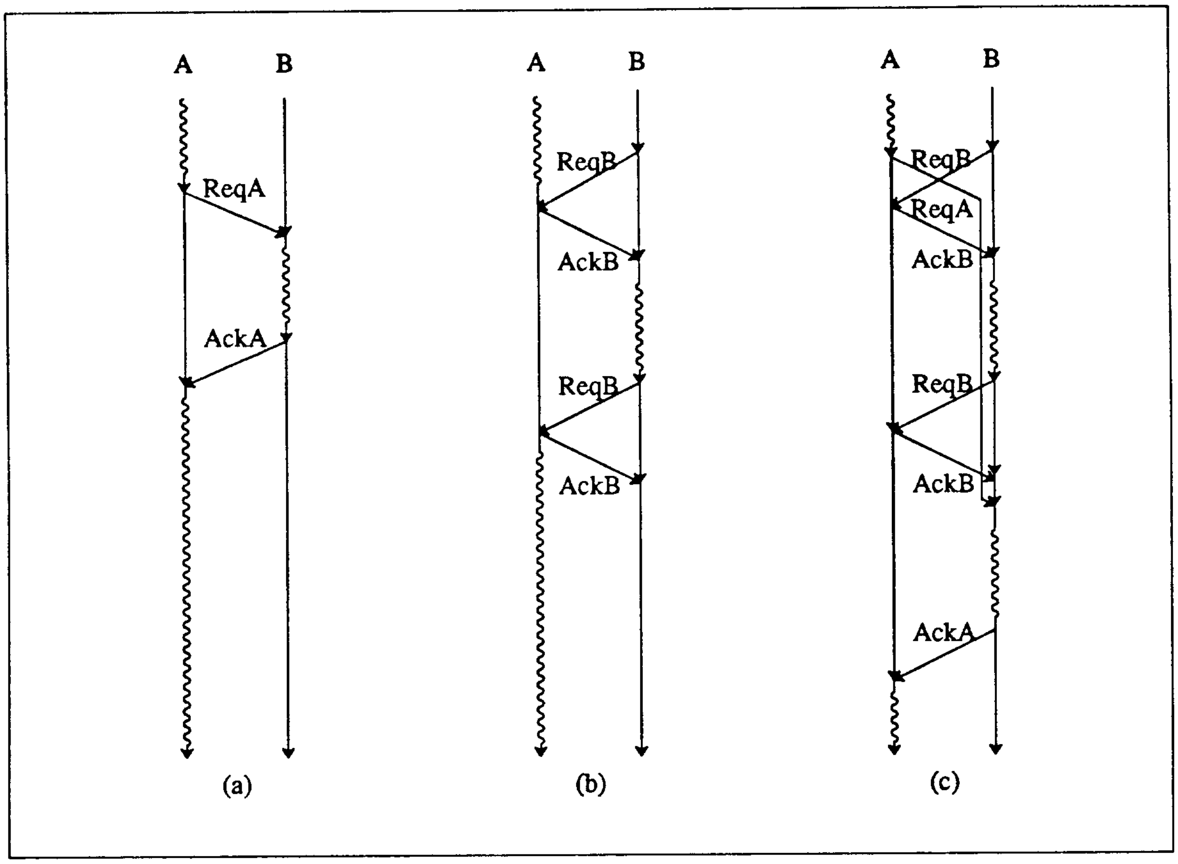

Fig. 5.11 Request/acknowledge signal flow in a bidirectional handshake interface. When a collision arises, the request from the side B always overrides request from the side A.

A better and natural approach is to add a new request/acknowledge channel in the opposite direction as shown in Fig. 5.10. Fig. 5.11 shows the diagram of request/acknowledge flow and the resolution of a collision. When the two side simultaneously send requests to the other side, the request from the side B always overrides the request from the side A. Even though this mechanism allows bidirectional communication, a master/slave relation is still retained to resolve a collision. A pseudo code for the FSMs with a 4 phase handshake mechanism is shown in Fig. 5.12.

```
master()
input inputs, ackA, reqB, reset;
output outputs, reqA, ackB;
state states;
{
if (reset == 1) { reqA = 0; ackB = 0; state = RESET_STATE; }

else if ( states == NON_INTERRUPTABLE_STATE )
        {
        switch (inputs, states)
                {
                (INPUT_A, STATE_A): { reqA = 0; ackB = 0; states = NEXT_STATE_A; }
                ---------- enumeration of all transitions ----------
                (INPUT_A, STATE_A):
                        { reqA = 1, ackB = 0; states = NEXT_INTERRUPTABLE_STATE_Z; }
                }
        }

else if ( states == INTERRUPTABLE_STATE and reqB == 1 )
                                        /* RESPOND TO SLAVE'S REQUEST WITH PRIORITY */
        { reqA = 0; ackB = 1; states = NEXT_INTERRUPTABLE_STATE; }

else if ( states == INTERRUPTABLE_STATE and ackA == 0 and reqB == 0 )
                                        /* WAIT FOR SLAVE'S OWNERSHIP RELINQUISHMENT */
        { reqA = 0; ackB = 1; states = WAIT_STATE; }

else if ( states == INTERRUPTABLE_STATE and ackA == 1 and reqB == 0 )
                                        /* MASTER'S HOME STATE */
        {
        switch (inputs, states)
                {
                (INPUT_A, STATE_A): { reqA = 0; ackB = 0; states = NEXT_STATE_A; }
                ---------- enumeration of all transitions ----------
                (INPUT_A, STATE_A):
                        { reqA = 1, ackB = 0; states = NEXT_INTERRUPTABLE_STATE_Z; }
                }
        }
}

slave()
input inputs, reqA, ackB, reset;
output outputs, ackA, reqB;
state states;
{
if (reset == 1) { ackA = 0; reqB = 0; state = RESET_STATE; }
else if ( states == INTERRUPTABLE_STATE and reqA == 0 and ackB == 1 )
                                                /* SLAVE'S HOME STATE */
        {
        switch (inputs, states)
                {
                (INPUT_A, STATE_A):
                        { ackA = 0, reqB = 0; states = NEXT_STATE_A; }
                ---------- enumeration of all transitions ----------
                (INPUT_A, STATE_A):
                    { ackA = 0, reqB = 1; states = NEXT_SLAVE_STATE_Z; }
                }
            }
    else if ( states == INTERRUPTABLE_STATE and reqA == 0 and ackB == 0 )
                                                    /* ILLEGAL STATE */
            { states = ERROR_STATE; }

    else if ( states == INTERRUPTABLE_STATE and reqA == 1 )
                                                    /* MASTER'S REQUEST DETECTED */
            {
            switch (inputs, states)
                { ackA = 0; reqB =0; states = NEXT_MASTER_STATE; }
            }

    else if ( states == MASTER_STATE )          /* SERVICE MASTER'S REQUEST */
            {
            switch (inputs, states)
                {
                (INPUT_A, STATE_A):
                    { ackA = 0; reqB =0; states = NEXT_MASTER_STATE_A; }
                --------- enumeration of all transitions -----------
                (INPUT_Z, STATE_Z):
                    { ackA = 1; reqB =0; states = NEXT_INTERRUPABLE_STATE_Z; }
                }
            }

    else if ( states == SLAVE_STATE and ackB == 0 ) /* WAIT FOR MASTER'S RELEASE */
            { ackA = 0; reqB = 0; states = SLAVE_WAIT_STATE; }

    else if ( states == SLAVE_STATE and ackB == 1 )
            { if ( count of SLAVE_REQ == EVEN ) /* RELINQUISH SLAVE'S OWNERSHIP */
                { ackA = 0; reqB = 0; states = NEXT_INTERRUPABLE_STATE; }
            else {
            switch (inputs, states)
                {
                (INPUT_A, STATE_A):
                    { ackA = 0; reqB = 0; states = NEXT_SLAVE_STATE_A; }
                --------- enumeration of all transitions -----------
                (INPUT_Z, STATE_Z):
                    { ackA = 0; reqB = 1; states = NEXT_SLAVE_STATE_Z; }
                }
              }
            }
    else ;
    }
```

Fig. 5.12 Pseudo code of the FSMs for the bidirectional handshake with external hardware support. A relation, "states = STATE", in the *if* and *case* statements means "states belong to the group of states of STATE".

In the description of the FSMs, any service arising from the request is initiated only when the receiver is in a benign, interruptable state so that the FSMs do not have any burden to store any previous state to restore when the service ends. To reduce the unnecessary latency, it would be nice to have a stack to save the current state and jump into a service routine. Thus, a stack machine is a preferable structure to implement the master and slave. Fig. 5.13 shows a block diagram of a bidirectional handshake mechanism with stack machines. The possible stack operations are push, pop, replace, and reset. The top of the stack represents the current state of the machine. If the stack operations are composed of replaces only, it would degenerate into a finite state machine structure. Fig. 5.14 shows a pseudo code for the FSMs with a 4 phase handshake mechanism.

```
master()
input inputs, ackA, reqB, reset, topStack;
output outputs, reqA, ackB, stackOp;
{
if (reset == 1) { reqA = 0; ackB = 0; stackOp = replace (RESET_STATE); }

else if ( topStack == NON_INTERRUPTABLE_STATE )
                { reqA = 0; ackB = 0; stackOp = replace (NEXT_STATE); }

else if ( topStack == INTERRUPTABLE_STATE and reqB == 1
                and count_of_SLAVE_REQ == ODD )
                { reqA = 0; ackB = 1; stackOp = push (CURRENT_STATE); }

else if ( topStack == INTERRUPTABLE_STATE and reqB == 1
                and count_of_SLAVE_REQ == EVEN )
                { reqA = 0; ackB = 1; stackOp = pop (CURRENT_STATE); }

else if ( topStack == INTERRUPTABLE_STATE and ackA == 0 and reqB == 0 )
                                /* WAIT FOR SLAVE'S OWNERSHIP RELINQUISHMENT */
                { reqA = 0; ackB = 0; stackOp = replace (WAIT_STATE); }

else if ( topStack == INTERRUPTABLE_STATE and ackA == 1 and reqB == 0 )
                                /* MASTER'S HOME STATE */
                { reqA = 0 or 1; ackB = 0; stackOp = replace (NEXT_STATE); }
else ;
}

slave()
```

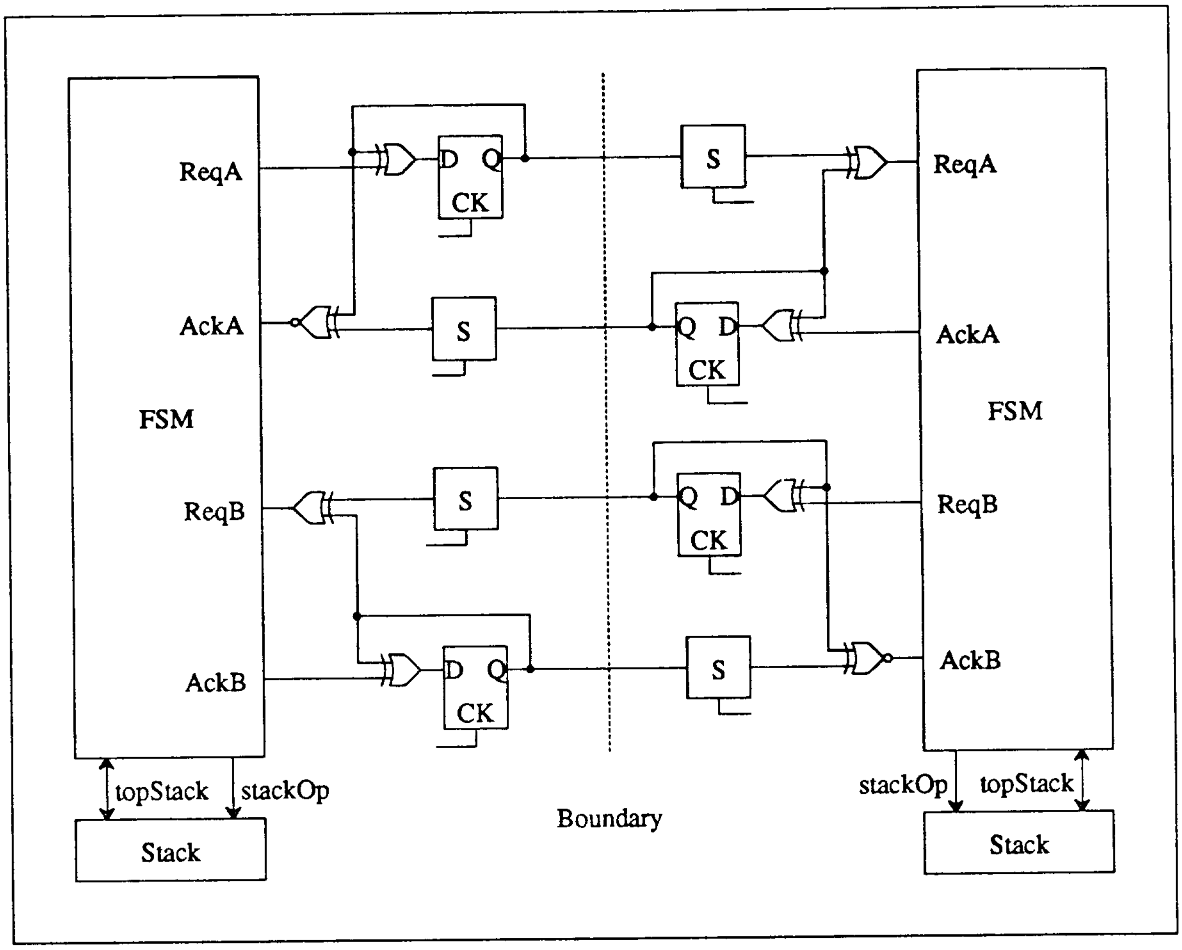

Fig. 5.13 Bidirectional handshake interface.

```
input inputs, reqA, ackB, reset;
output outputs, ackA, reqB;
state states;
{
if (reset == 1) { ackA = 0; reqB = 0; state = RESET_STATE; }

else if ( topStack == NON_INTERRUPTABLE_HOME_STATE )
          { ackA = 0; reqB = 0; stackOp = replace (NEXT_STATE); }

else if ( topStack == SLAVE_STATE and ackA == 0 )
          { ackA = 0; reqB = 0; stackOp = replace (SLAVE_WAIT_STATE); }

else if ( topStack == SLAVE_STATE and ackA == 1 and count_of_SLAVE_REQ == ODD )
          { reqA = 0; ackB = 1; stackOp = replace (NEXT_STATE); }

else if ( topStack == SLAVE_STATE and ackA == 1 and count_of_SLAVE_REQ == EVEN )
          { reqA = 0; ackB = 1; stackOp = replace (NEXT_INTERRUPTABLE_STATE); }
else if ( topStack == INTERRUPTABLE_STATE and reqA == 1 and ackB == 0 )
                                        /* WAIT FOR SLAVE'S OWNERSHIP RELINQUISHMENT */
            { reqA = 0; ackB = 1; stackOp = replace (WAIT_STATE); }

else if ( topStack == INTERRUPTABLE_STATE and reqA == 1 and ackB == 1 )
                                        /* WAIT FOR SLAVE'S OWNERSHIP RELINQUISHMENT */
      { reqA = 0; ackB = 1; stackOp = replace (NEXT_STATE); }

else if ( topStack == INTERRUPTABLE_STATE and ackA == 1 and reqB == 0 )
                                        /* MASTER'S HOME STATE */
      { reqA = 0; ackB = 0; stackOp = replace (NEXT_STATE); } or
      { reqA = 1; ackB = 0; stackOp = replace (NEXT_SLAVE_STATE); }
else ;
}
```

Fig. 5.14 Pseudo code of the stack machines for a bidirectional handshake with external interface hardware support. A relation, "states = STATE", in the *if* and *case* statements means "states belong to the group of states of STATE".

5.3.6 Reset Signal Generation

A reset signal is essential in any system. It is even more important for systems with handshake interfaces. Without a system-wide reset, inconsistent state may result, causing deadlock. There are two requirements for a successful reset. First, the duration of a reset signal should be long enough that it resets the entire system into a known state. Second, a system should be synchronized on its clock at the end of reset. Entering into the reset state can be asynchronous since any system can be driven into the reset state in a few cycles. However, asynchronous exit from reset may result in an inconsistent state since the reset input to the synchronous system is not guaranteed to be time- and value-qualified. Simple connections to the synchronizer are not reliable since we cannot rely on clock signals on power-up. Clocks may not be available immediately. During that time, an illegal system state may result in permanent hardware damage caused by, for example, bus drive collision. Every system should be designed to be in a benign, dormant state during reset even when clock signals are not available. A reset circuit that meets the requirement is shown in Fig. 5.15. It guarantees asynchronous entering into a reset and synchronous exiting from the reset during power-up.

5.3.7 Interface with Bus Systems

Many standard bus protocols are currently in use when more than 2 independent synchronous systems should be integrated with a handshake protocol. These bus protocols can be classified in two categories: synchronous and asynchronous, depending on whether they use clocks or not. In synchronous bus protocols like NuBus or Multibus II, clocks are used to synchronize all the transactions. On the other hand, asynchronous bus protocols rely entirely on a request/acknowledge handshake protocol without using a clock. Various comparisons can be made on standard microcomputer bus protocols [5.3-4]. First, performance depends on the implementation technology. For example, Fastbus, which relies on ECL technology has more bandwidth than Multibus II or NuBus which uses TTL technology. With similar technology, an asynchronous bus has better performance than a synchronous bus, since the path delay should be rounded up as the multiple of clock cycles in a synchronous bus protocol. However, a synchro-

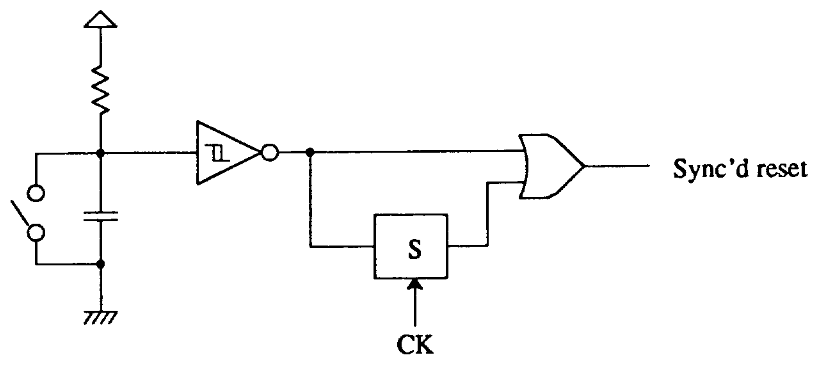

Fig. 5.15 Synchronization of a reset signal.

nous bus protocol is easier to implement.

In a typical bus-based multiprocessor system, the CPU boards are connected through a system bus. When a synchronous bus is used, the clock frequency of the CPU board is usually different from that of the bus clock. A unidirection path between two boards include two synchronization events. For each synchronization, an average of a half cycle of the receiver clock is introduced due to the random distribution of clock edges with respect to handshake signals. Synchronizer settling time is introduced as well in order to increase synchronization reliability. Thus, an average one-way overhead due to the request/acknowledge handshake is,

$$\text{latency} = \frac{1}{2}\, T_{BOARD} + T_{s,BOARD} + \frac{1}{2}\, T_{BUS} + T_{s,BUS}, \tag{5.1}$$

where T_{BOARD} is the cycle time of a receiver board, $T_{s,BOARD}$, synchronization latency in the board, T_{BUS}, bus cycle time, and $T_{s,BUS}$, synchronization latency in the bus. Typically, about a half cycle is enough for the synchronizer settling time. Therefore, an average overhead in terms of CPU cycles is,

$$\text{latency} = 1 + \frac{T_{BUS} + T_{s,BUS}}{T_{BOARD}} \text{ [cycles].} \tag{5.2}$$

Although T_{BOARD} decreases as the technology advances, T_{BUS} is fixed because it is determined by other physical considerations like bus impedance, trace length, and so forth. However, $T_{s,BUS}$ tends to be tied to the CPU cycle time since it depends on the device technology. Therefore, as T_{BOARD} decreases, the effect of latency increases.

On the other hand, when an asynchronous bus is used for a common bus, an average one-way overhead due to the request/acknowledge handshake is,

$$\text{latency} = \frac{1}{2}\, T_{BOARD} + T_{s,BOARD} \tag{5.3}$$

There is no need for synchronization on the bus clock. A request from the sender is directly sent to the receiver. The only synchronization event happens at the receiver. Thus, latency is 1 CPU cycle, assuming a half cycle synchronizer settling time.

While a synchronous bus protocol is relatively easy to implement, an asynchronous bus design requires careful precautions. Even if an asynchronous protocol alleviates a clock skew problem, similar problems arise for request and acknowledge signals. It should be guaranteed that request or acknowledge arrive at the destination before the data. Even if the pair is launched at the same time, different path delay may cause the data arrive later than request or acknowledge, which results in sampling old data which was left by previous transaction. Different path delay results from different loadings at the backplane. The source of skew is similar to the one of clock skew in a synchronous system even though its associated area is local. To guarantee error free operation, there are two choices: path trimming, which is very costly, and allowing timing margin at the cost of performance degradation. This is a similar situation with a synchronous system where clock cycle is extended to cope with the worst-case clock skew.

Asynchronous design methodologies developed in [5.5-6] describe an interface specification language and an automatic synthesis procedure to make interface circuit and logic design easier. However, the tool mentioned does not pay special attention to the synchronization failure problem which is ubiquitous in the interface design. A synchronizer with a metastability detector could be a powerful addition of the circuits used in the interface designs. The following is an example of applying such a device to an implementation of an error-free handshake mechanism. When data bits are transferred, those bits should not be allowed to change during the transfer even if synchronization is done for the data as well. It may be tempting to transfer a single word without stopping the host. If synchronization is done just at the time of change, some bits may hold old values and other bits, updated ones, resulting in inconsistency. However, synchronizing a single datum used as a flag is completely legal as long as the datum is glitch-free. Whatever direction the synchronizer settles, it is consistent, even when the flag changes at the

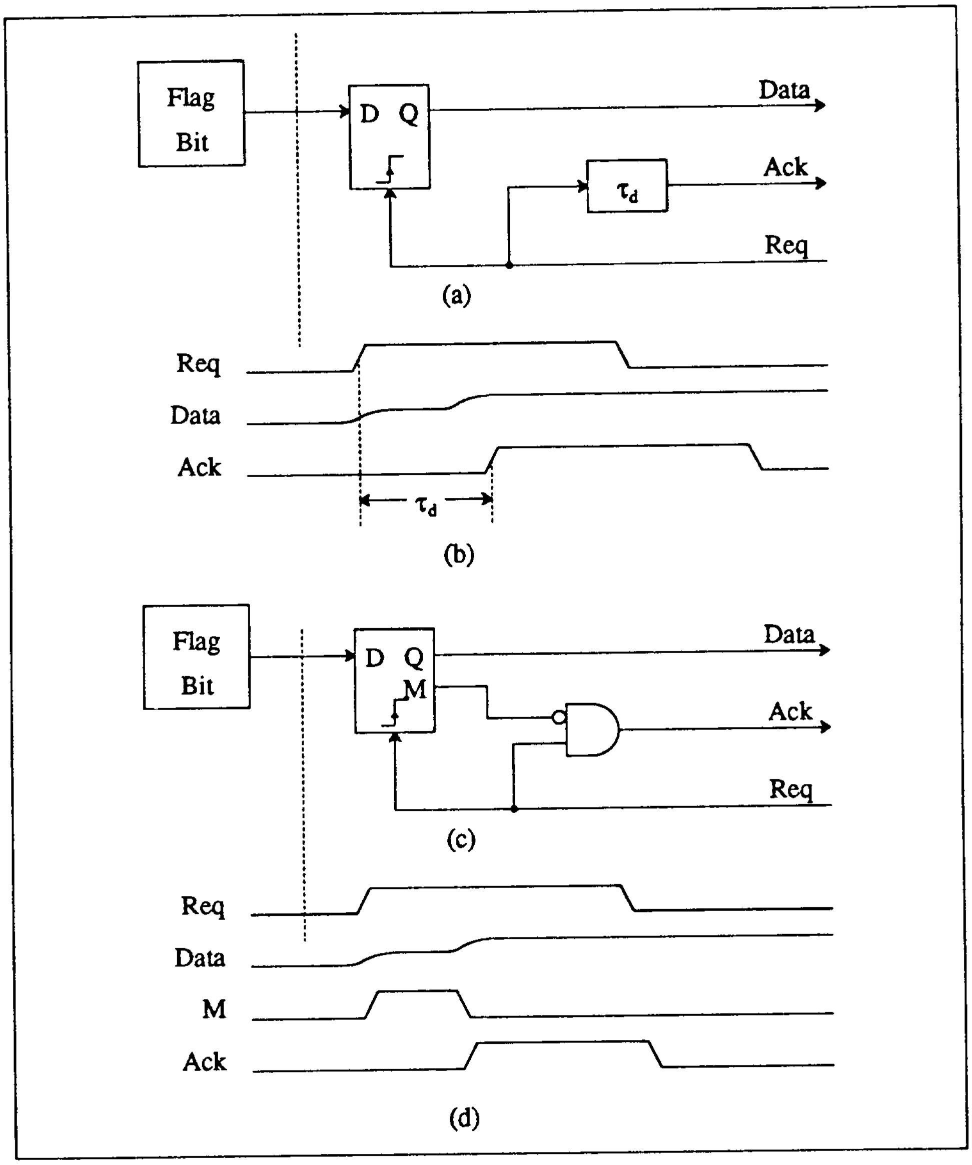

Fig. 5.16 Direct access to a variable across synchronous-asynchronous boundary.

time of sampling. When those sampling is done, metastability is again a problem. For example, a read request to the flag bit may cause metastability. The easiest solution is to allow a waiting time for the synchronizer to settle. This will increase the latency. A better solution is to use a synchronizer with metastability detector. The synchronizer with a metastability detector has extra output, M which is asserted whenever the output Q is in a metastable state. In this new scheme shown in Fig. 5.16, acknowledge signal is asserted just after its data is settle instead of waiting a worst-case settling time.

5.4 Summary

Interfaces between systems with independent clocks require careful attention. Not only should request/acknowledge signals be delivered glitch-free, they must undergo synchronization before they are used in synchronous systems. Bidirectional handshake can be implemented with a machine with a stack or equivalents to resolve conflicts. Latency is inevitably introduced whenever request/acknowledge signals cross asynchronous boundary. A half or a quarter cycle latency is needed for synchronization reliability in addition to a probabilistic, half a cycle latency. Standard bus systems work as as interface between synchronous sub-systems with independent clocks.

CHAPTER 6

Example - SPUR MMU/CC

This chapter describes the design and implementation of the SPUR memory management unit and cache controller (MMU/CC). The SPUR workstation is an excellent example of a synchronous sub-systems cluster with independent clocks. Since the SPUR MMU/CC handles almost all the board level communications, its description and implementation will reveal the details and difficulties of the overall system. The SPUR MMU/CC in silicon is fully functional and runs an operating system in a multiprocessor configuration. The design and implementation of the SPUR MMU/CC has been done jointly with David A Wood, Garth A. Gibson, and Susan J. Eggers.

6.1 Introduction - SPUR Overview

Computer architecture evolves by responding to advances in the underlying implementation technology. Current CMOS VLSI technology makes it possible to integrate a very powerful processor on to a single chip. Computer architects respond with RISC (Reduced Instruction Set Computer) concepts as a result of careful study of trade-offs between VLSI technology and compiler technology [6.1]. RISC-based microprocessors have demonstrated an ability to provide more computing power at a given level of integration than conventional microprocessors [6.2-3]. The next logical step is multiprocessors composed of RISC processing elements. We have designed and built such a multiprocessor called SPUR [6.4]. SPUR (Symbolic Processing Using RISC's) is a bus-oriented shared memory multiprocessor developed at U.C. Berkeley to explore

RISC applicability to symbolic programming languages such as LISP, parallel processing with shared memory, and floating point processing. Design started in 1983, implementation in 1985, and the hardware has been operational since 1988. Fig. 6.1 shows a block diagram of the SPUR system. Multiple identical processor boards reside in each workstation, providing an aggregate performance far exceeding uniprocessor performance. Each processor board contains a cache memory and three custom VLSI chips - a Central Processing Unit (CPU), a floating point unit (FPU), and a memory management unit and cache controller (MMU/CC). A 128 Kbyte direct-mapped cache is essential for reducing multiprocessor memory traffic. Processor boards are con-

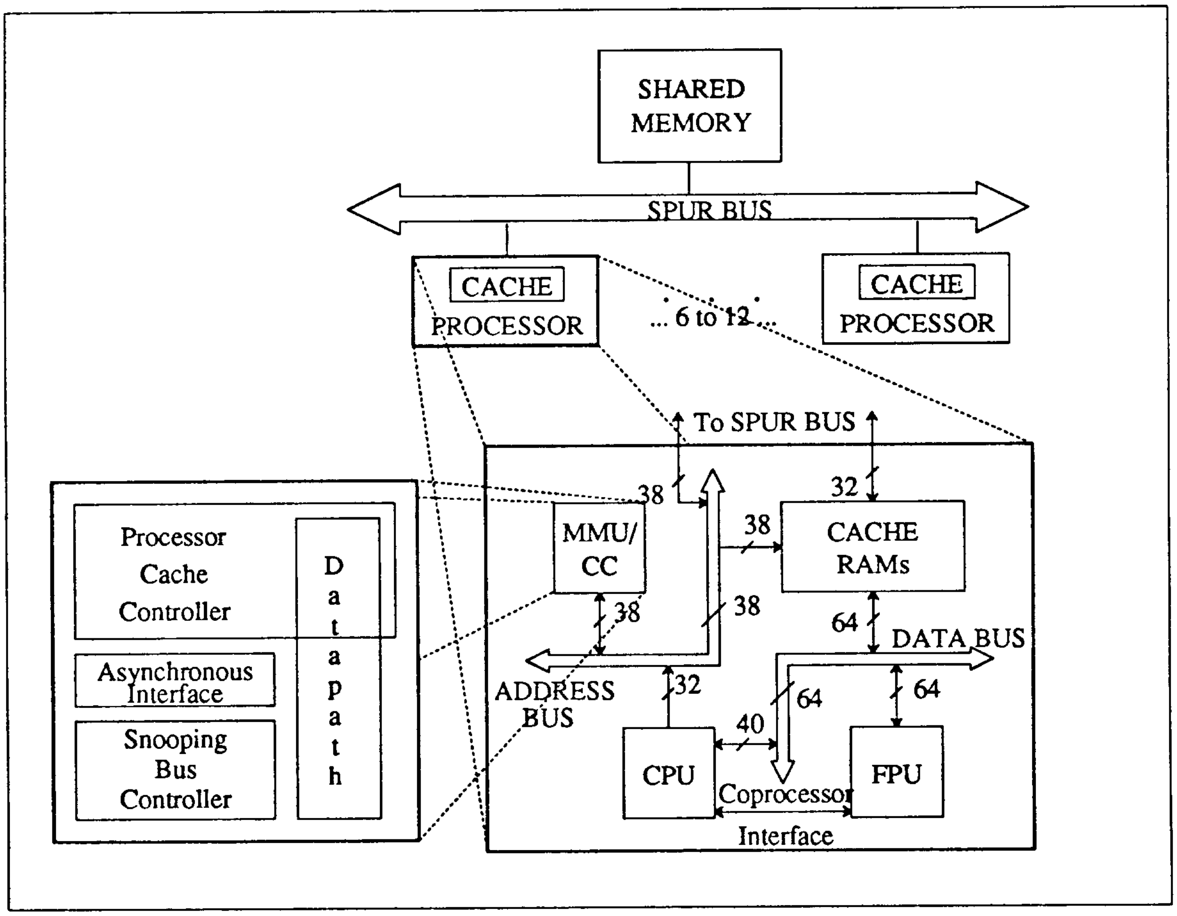

Fig. 6.1 SPUR block diagrams: (a) system block diagram, and (b) processor board block diagram.

nected to each other and shared memory by a common bus called the SpurBus, an extension of Texas Instrument's *NuBus* that incorporates a snooping bus protocol [6.5].

One of the most challenging problems in building multiprocessors with private write-back caches is to successfully design, verify and implement cache coherency protocols. This paper describes the MMU/CC that implements such a protocol. The innovative functions of the MMU/CC are a virtual memory management mechanism via in-cache address translation [6.6] and a write-invalidate cache coherency protocol [6.7]. The MMU/CC controls access to the cache and generates all control signals for the board and backplane. An interval timer, an interrupt controller and performance counters are also integrated onto the MMU/CC.

The MMU/CC has been fabricated in a double metal 1.6μm N-well CMOS process at Hewlett Packard through MOSIS. The chip integrates 68,400 transistors on a die measuring 11.5 × 11.5 mm^2 and consumes 0.7 W. Fig. 6.2 shows the chip microphotograph. The chip statistics are summarized in Table 1.

TABLE 1 - Chip Summary.

Number of Transistors	68,395
Number of circuit nodes	30,285
Total gate capacitance	1280 pF
Total wire/junction capacitance	2630 pF
Number of PLAs	19
Total number of PLA inputs	262
Total number of PLA outputs	277
Total number of PLA product terms	707
Die Size	11.5mm x 11.5mm
Package	208-pin PGA
Power Dissipation	0.7W @ 5V, 10MHz
Process	Double-metal 1.6um CMOS

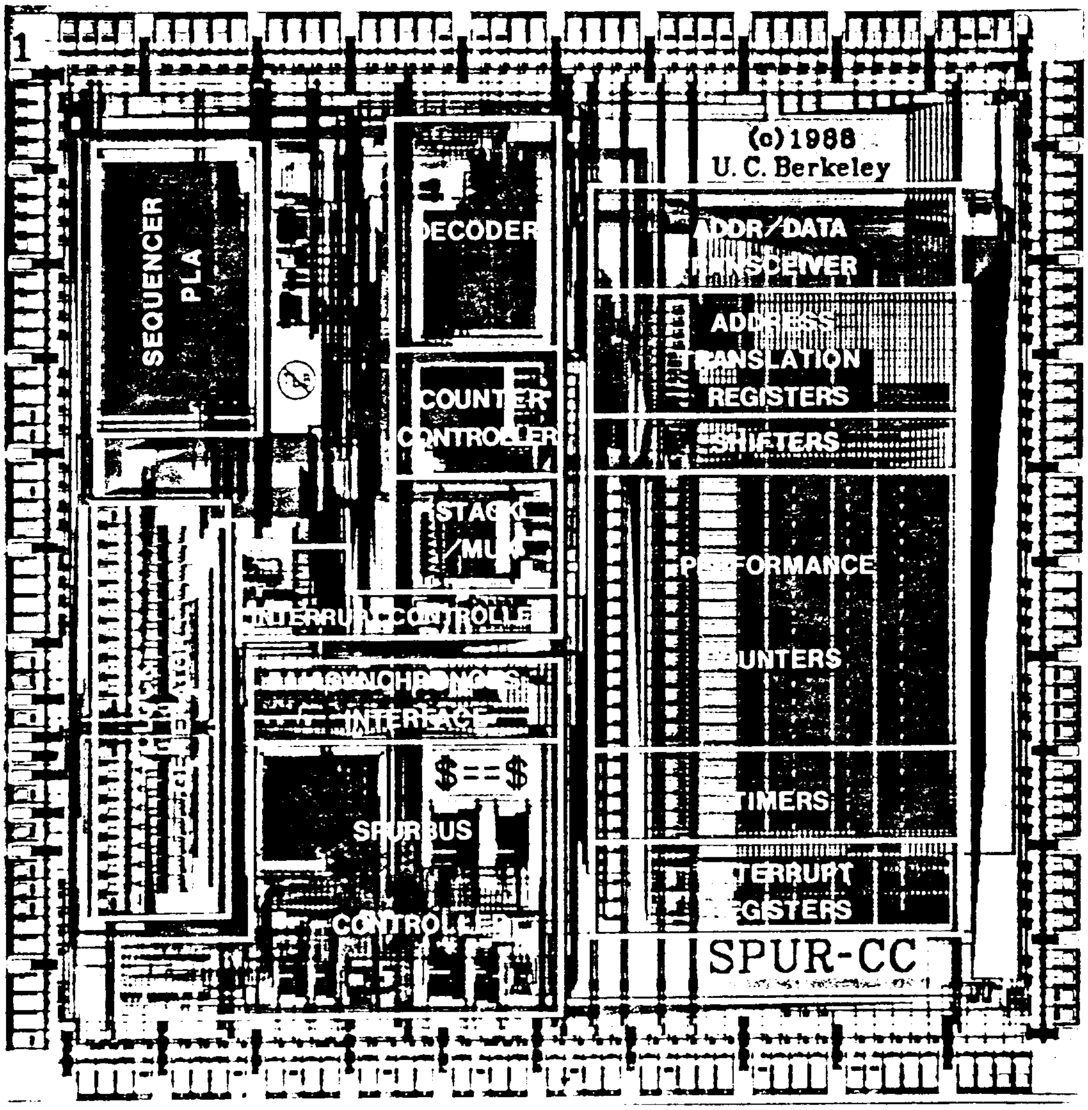

Fig. 6.2 Chip microphotograph.

In the next section, we present a functional description of the MMU/CC. In Section III, implementation details and circuit techniques are described. Section IV explains VLSI design methodologies including test vector generation, layout generation, layout verification, and so forth. Finally, status and conclusions are given in Section V.

6.2 Functional Description

The MMU/CC combines two control units - a processor cache controller (PCC) and a "snooping" bus controller (SBC). The PCC handles memory references for the CPU, while the SBC interacts with the backplane bus. The PCC and SBC run independently unless an event that requires the other's attention occurs.

6.2.1 The Processor Cache Controller

The PCC's memory management scheme is based on an in-cache address translation mechanism [6.6]. Virtual, rather than physical, addresses are used for the cache index and tag. The advantage of the virtually addressed and virtually tagged caches is that address translation is needed only on cache misses. Because the cache is quite large, misses occur infrequently, and a high speed translation mechanism is not needed. The virtual to physical address translation map is located in the virtual address space and the data cache serves as a translation look-aside buffer (TLB), reducing the complexity of translation and eliminating the need for a separate unit for this function.

On each CPU reference, the PCC first transforms the 32-bit virtual processor address into a 38-bit global virtual address. It maps the most significant two bits of the virtual address into an 8 bit segment number by selecting one of 4 global segment number registers in the datapath. The mapping is done in parallel with the cache tag and data RAM access, since the high order bits are

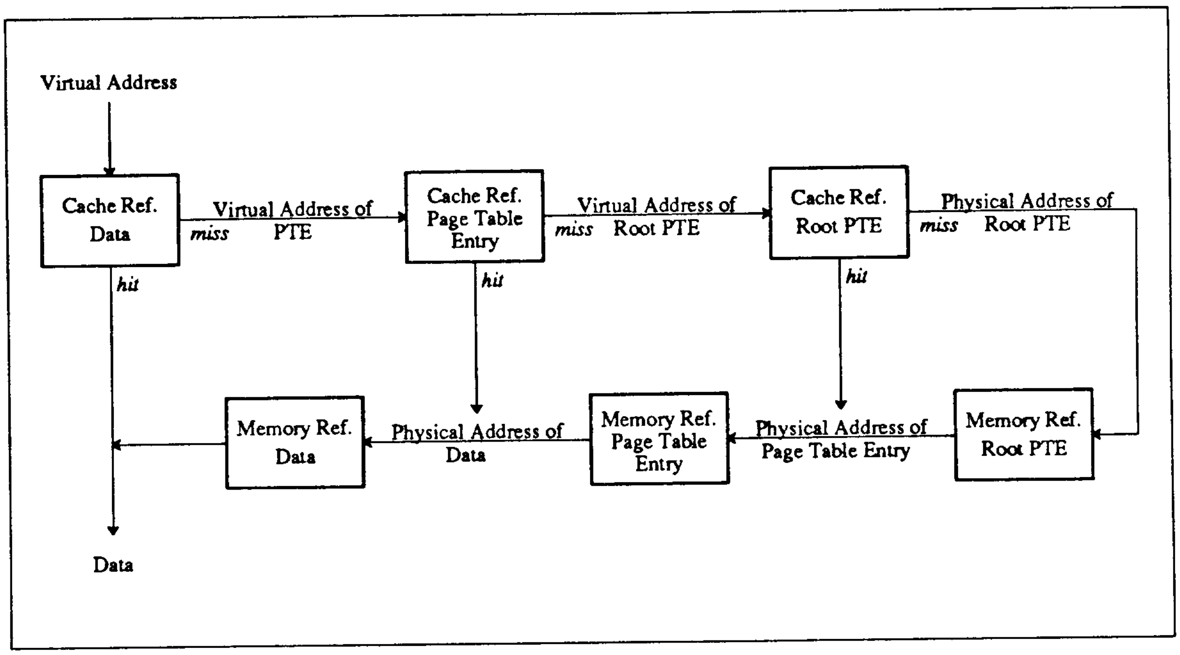

Fig. 6.3 In-cache address translation mechanism - cache/memory access flow.

part of the tag and this simple map is faster than the cache tag store access. If the address tag matches the cache tag, the data is returned and the MMU/CC takes no further action. On a cache miss, the PCC generates the virtual address of the page table entry (PTE) using a page table base register and a special shifter. It uses the PTE virtual address to access the cache in the next cycle. If this access finds the PTE, the physical address of the data is extracted and a main memory reference is made to fetch the desired cache block. On a PTE miss, a third cache reference is needed to access a root page table entry (RPTE) with the assistance of a root page table base register and another shifter. The desired cache block is fetched in a recursive manner. This recursive process ends if the third cache reference misses in the cache. Instead of going to a deeper level, it uses a root page table entry map register that contains the base address of the root page table entry in the physical address space. Because of the size of the cache and locality in accesses, PTE misses and especially RPTE misses are extremely infrequent [6.6]. The translation mechanism is shown in Fig. 6.3 and Fig. 6.4.

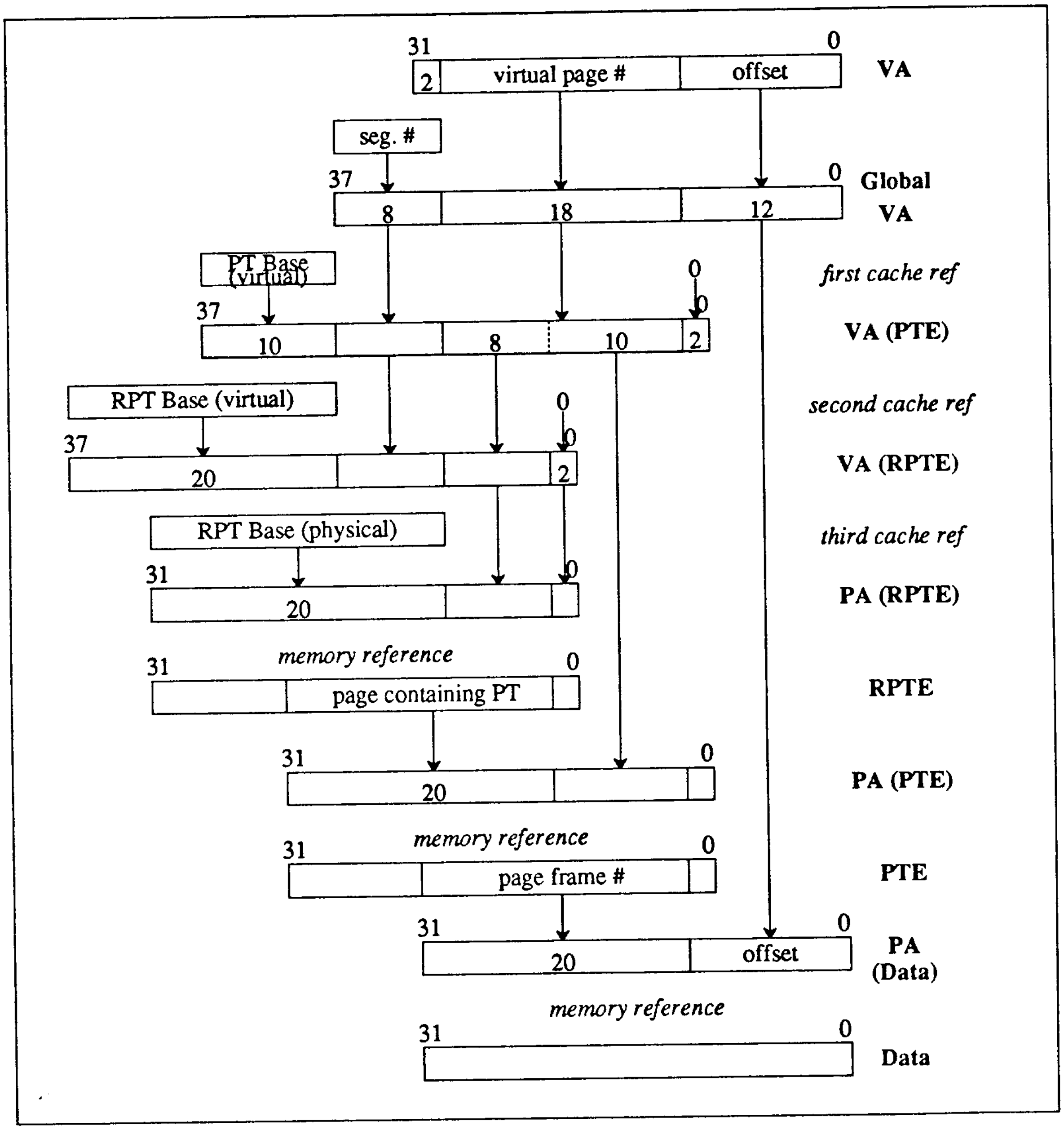

Fig. 6.4 In-cache address translation mechanism - address mapping.

6.2.2 The Snooping Bus Controller

The controller provides special hardware support for maintaining coherency among private caches. When a memory block is shared by more than two caches, a local write into one of the shared cache blocks should be reflected in the other cache's corresponding block, so that the system maintains a consistent, single-level view of memory. The Snooping Bus Controller (SBC) implements a distributed cache coherency mechanism, called Berkeley Ownership [6.7]. The SBC not only initiates its own bus transactions on behalf of the processor, but also "snoops" on other bus activities to detect when one of the local cache blocks is involved.

The protocol works as follows. A cache block is in one of four coherency states: *Invalid, UnOwned, OwnShared* and *OwnPrivate.* Also, four kinds of bus transactions are possible: *ReadShared, ReadForOwnership, Write,* and *WriteForInvalidation.* Owning a cache block means that the owning SBC has the responsibility to provide an up-to-date copy on a read request from the bus and to write the cache block to memory if it needs to replace the block in the cache. There is at most one SBC that owns a memory block. Main memory is the implicit owner of the block if it is not cached by any processor. On a write to a valid, non-*OwnPrivate* cache block, the PCC stalls the processor while the SBC initiates a *WriteForInvalidation* to invalidate corresponding cache blocks in other caches. The state of the local cache block becomes *OwnPrivate.* Until another SBC takes ownership, subsequent writes can be made locally without informing others because it is a unique copy. On receiving a *ReadShared* bus request, the state of the OwnPrivate cache block is changed to *OwnShared.* In the *OwnShared* state, the next local write should accompany *WriteForInvalidation* because there are other copies of the same block. A special processor request, *ReadPrivate,* is included for improving performance under software direction. Ownership can be obtained immediately when reading a non-shared block, instead of waiting until a processor write operation. In this way, an unnecessary bus transaction, *WriteForInvalidation* can be avoided on "private" data. Fig. 6.5 shows a state transition diagram of the protocol.

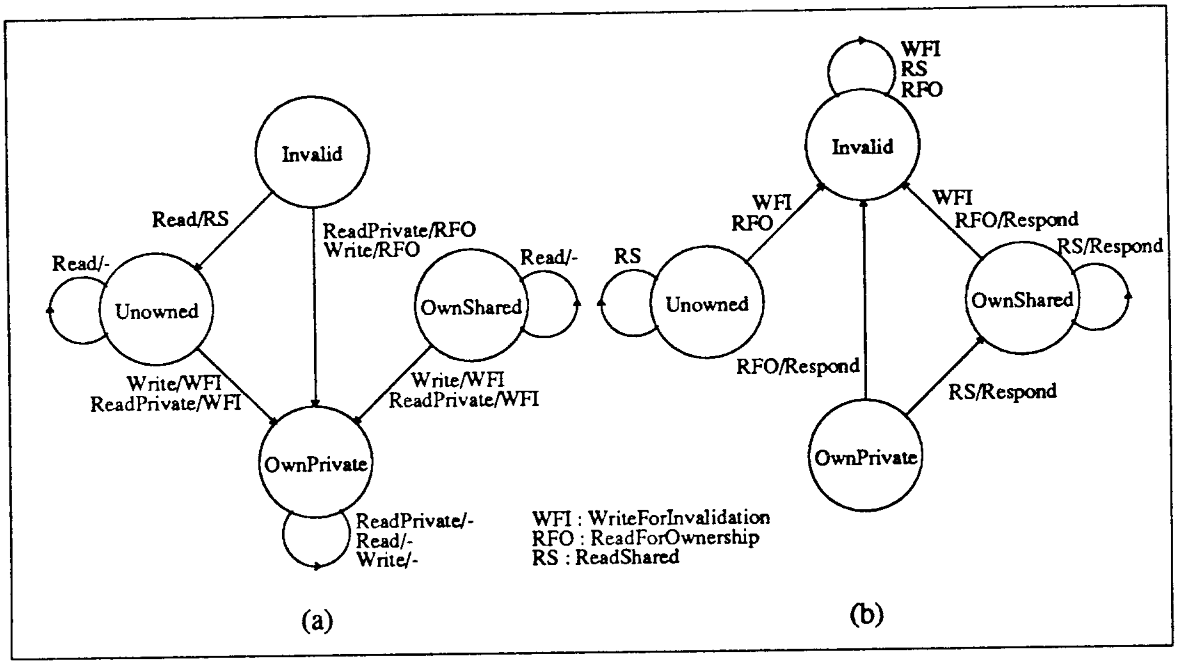

Fig. 6.5 Cache block state transition diagram: (a) state transition due to CPU operations, and (b) state transition due to snoop operations. A label in an arc represents (Request Received)/(Bus Action). Processor FLUSH operations are omitted for simplicity.

We have extended a standard microcomputer bus, Texas Instrument's *NuBus*, to incorporate the Berkeley Ownership protocol [6.5]. We wanted to use existing commercial memory and I/O boards. Since the devices did not participate in the Berkeley Ownership protocol, we used a separate set of backplane lines for inter-cache transactions.

6.2.3 Asynchronous Interface

While the SBC derives its clock from the SpurBus (10 MHz fixed frequency), the PCC shares the processor clock which is asynchronous with the bus. The frequency of the processor clock can, therefore, be set according to its implementation technology, regardless of the bus clock and other processors. We chose to do this to provide more flexibility during testing and integration with memory and I/O devices. An asynchronous interface handles a request/acknowledgement handshake for communication between the PCC and SBC.

Other system functions are integrated onto the MMU/CC such as performance counters, interval timers, interrupt controller. Performance counters monitor various system activities to measure performance metrics. The purpose of including these counters is to aid in the performance analysis of the working multiprocessor without perturbing the system. They count 32 kinds of coherency-related and cache access events in user and/or kernel mode such as *Read-ForOwnership* bus operations, cycles spent by the PCC waiting for a bus transfer to finish, instruction fetches, and so on.

6.3 Implementation

Fig. 6.6 shows a detailed block diagram of the MMU/CC internals. The chip consists of the PCC, the SBC, an asynchronous interface and other system functions. A complete specification for the MMU/CC is in [6.8].

A sequencer implementing the PCC consists of a programmable logic array (PLA), a stack and a decoder that sends low level control signals both on-chip and off-chip. It is configured as a push-down automaton rather than as a finite state machine to efficiently implement a recursive algorithm used in address translation. State information is stored in the 4 entry stack and can be pushed, popped, replaced, or flushed under the control of the sequencer. Such a machine structure is also convenient to handle SBC request servicing. When the SBC requests the PCC's attention, the PCC is able to save its current state while executing the SBC's request.

The PCC's sequencer PLA has 41 inputs, 36 outputs and 207 product terms. A small sense amplifier that fits into the pitch of the array has been designed for fast signal detection in large PLAs. It also limits the voltage swing in the highly capacitive nodes in the AND plane and the OR plane.

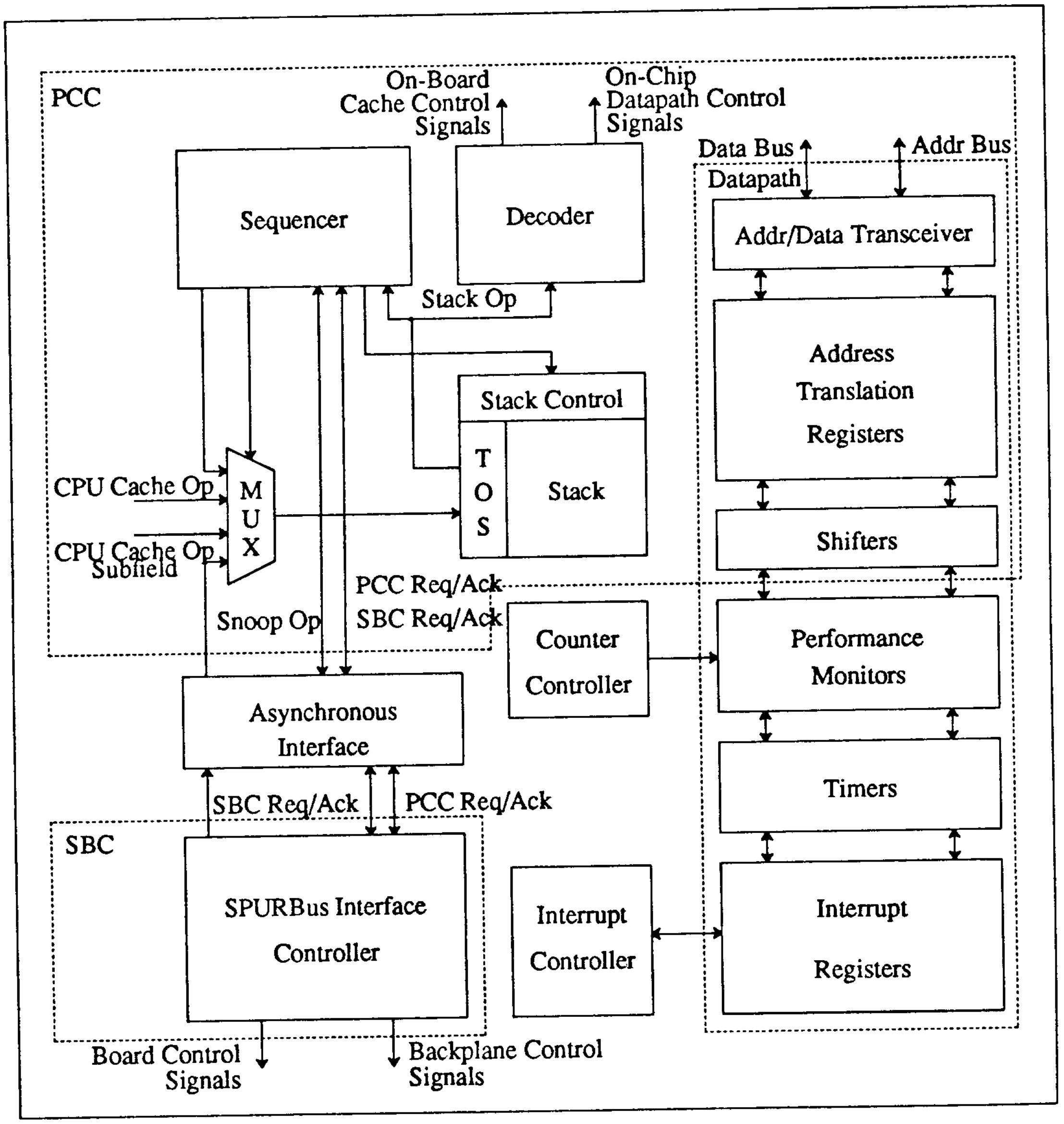

Fig. 6.6 SPUR MMU/CC block diagram.

Fig. 6.7 contains a circuit diagram and its transfer characteristics. The range of the voltage swing is set to approximately 1 V, which is determined by a reference voltage generator. The simple reference voltage generator is composed of a diode-connected transistor and a set of the pull-up and pull-down transistor with the same size and orientation as the ones in the array. It assures insensitivity to process and power line variations. Assuming only one array transistor is selected in the data line, the voltage swing is limited to ΔV_{GS} of M2. An array transistor at logic threshold sinks the same amount of current as the pull-down in the reference generator if ΔV_{GS} of the cascode transistor, M1, is half as much as that of the diode-connected transistor, M2, in the reference generator. Thus, the logic threshold voltage of the highly capacitive node is placed in the middle of the voltage swing, regardless of the variations, by making the width of M1 2.5 times larger than the width of M2.

The worst-case delay (fully loaded AND/OR plane) of a 50 input, 50 output, 200 product

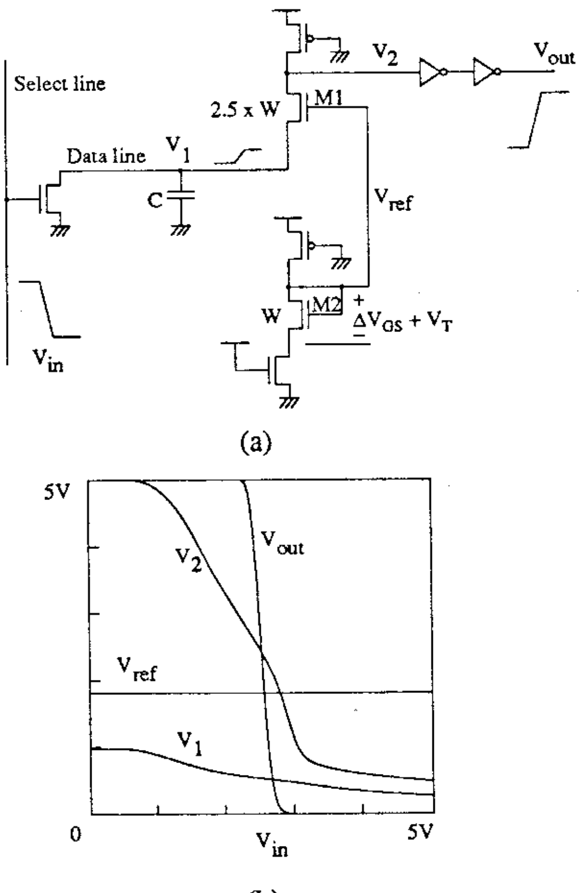

Fig. 6.7 PLA sense amplifier: (a) schematic, and (b) transfer curve.

PLA was simulated to be 31ns with a 100μA pull-up current. An equivalent PLA without the sense amplifiers would have had a 55ns delay. However, real PLAs with a sparse AND/OR plane would have significantly less delay because of reduced capacitance in the select and data lines. Actual delay of the sequencer PLA is estimated to be 18 ns.

A comparison with other PLAs is shown in Fig. 6.8. PLAs with polysilicon select lines have the longest delay because the RC delay increases quadratically with the size of the PLA. Without low ohmic silicided polysilicon lines, first level metal could be used as the select line

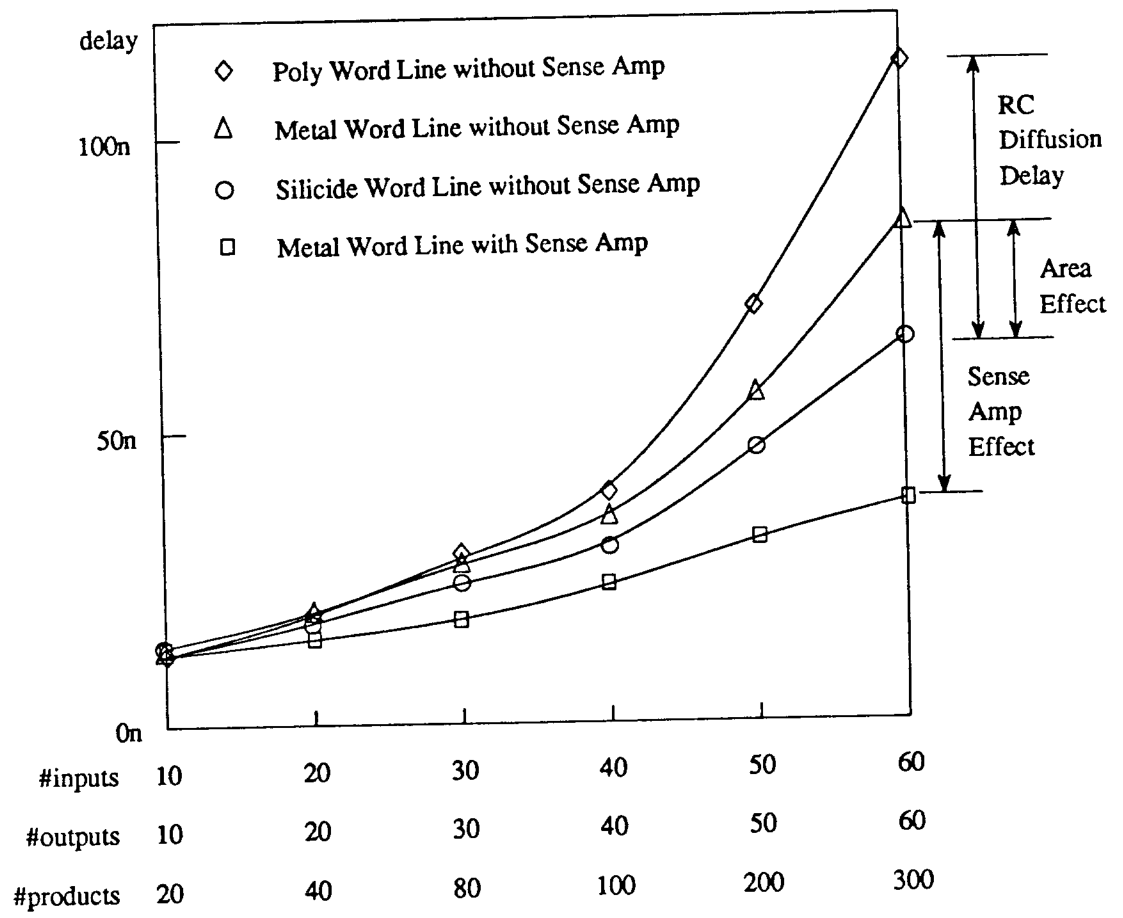

Fig. 6.8 Comparison of the worst-case delays among different PLAs.

and second level metal as the data line. Since the pitch of the metal lines and their contact sizes are larger than that of the polysilicon lines, the area of a PLA that uses metal for both lines is approximately 2 times larger than the PLAs with polysilicon or silicided polysilicon lines. Our sense amplifiers are more efficient as the size of the PLA increases. Overhead delays due to inverter stages amortize, and the total delay time reduces asymptotically to [$\frac{V_{SWING}}{V_{DD}}$] times the delay of the PLA without sense amplifiers.

The address translation datapath includes special purpose registers and shifters. It also shares its buses with other system utility functions: performance counters, interval timers, and interrupt registers. All registers and buses are implemented with fully static or pseudo-static logic. Although fully static circuits take more area, they are relatively immune to noise and tend to generate less noise.

The SBC consists of 7 PLAs, 19 OR gates, and several logic blocks with random gates and latches. Each PLA is a controller partitioned to specific functional operations, running in parallel with other PLAs. For example, a PLA named *master* generates almost all backplane control signals when it acts as a bus master, while a *slave* PLA does the similar operation responding to the bus as a bus slave. A smaller controller, *nubus*, receives interrupts and notifies the *master* about SpurBus availability. A *virmach* PLA manages data transmission and data reception on the inter-cache backplane lines. It may be triggered either by the *master* to override memory's copy of the locally requested block or by the *slave* to transmit the block requested by the inter-cache backplane. A *physrec* PLA handles the transfers from the memory to the cache and it may be requested by the *master* to release the cache RAMs in favor of the *virmach*. A *reset* PLA continuously monitors bus activities to check for reset conditions and potentially override all other PLAs. The total numbers of inputs, outputs and product terms of the SBC PLAs are 100, 123, and 236, respectively, with the largest PLA having 25 inputs, 39 outputs, and 76 product terms.

Of the total SBC area, 65% is consumed by routing. There are 209 nets among major blocks in total. Net length distribution is shown in Fig. 6.9, not including clock lines and scan path related routing.

Since the MMU/CC must generate board-level signals as well as internal datapath control signals, stringent timing relationships are required. Clock skew must be minimized for high speed synchronous communication among the PCC, CPU and FPU. Multi-phase clocks are needed to provide many different timings for enables, chip selects, and address/data drivers. Two charge pump Phase-Locked Loop's (PLL's) with tapped delay lines [6.9] provide the flexibility needed to generate multiple clock phases, in addition to maintaining accurate phase relationships with clock phases on other chips and the backplane.

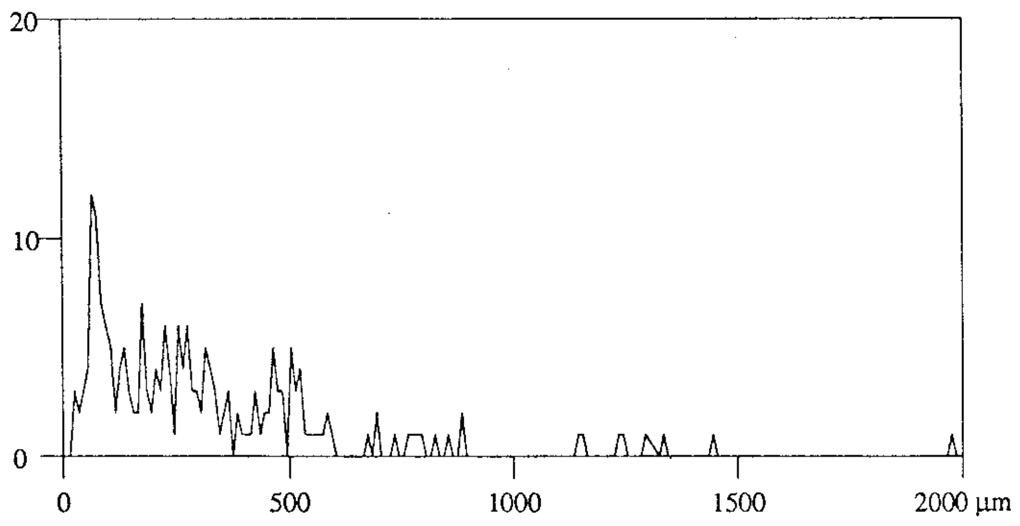

Fig. 6.9 Net length distribution of the SBC.

Fig. 6.10 shows the 16 internal clock phases. Since the internal frequency follows the external reference clock, all phases stretch or contract proportionally. It is more forgiving to critical paths since any phase including non-overlap time can be stretched by slowing down the external clock. With a 10 MHz external clock, the minimum timing quantization of any internal clock phase is 5 ns. Conventional clocking would have required a 200 MHz external clock to obtain 5 ns timing resolution. Special techniques would have been required to distribute the external clock in a printed circuit board and design an on-chip clock generator operating at such a high frequency.

As with all asynchronous interfaces, metastable states can cause system failures. To reduce the probability of these failures, we employ a two-prong design strategy: first, maximize the bandwidth of internal core amplifiers in each synchronizer, and second, allow the synchronizer half a cycle to settle. Because most interface transactions occur on infrequent cache misses, the extra latency for synchronization does not significantly degrade system performance. The core of the synchronizer is an RS flip-flop composed of two NAND gates that has been carefully laid out to reduce parasitic capacitance. Simulation shows that the characteristic time constant of the synchronizer is 0.24ns. Initial condition of the input voltage difference in the synchronizer core that cause metastability to persist longer than 20 ns is 3.2×10^{-36} V [6.10]. Assuming the initial input voltage difference caused by asynchronous inputs is uniformly distributed (conservative assumption), the probability of metastability persisting longer than 20 ns is 6.4×10^{-37}. When synchronizations happen at a 10 MHz rate, the system's Mean Time Between Failure (MTBF) due to synchronization error is more than 10^{21} years.

Two channels are needed for bidirectional communication between the PCC and SBC. One channel is responsible for delivering to the SBC the PCC's requests to fetch or write data to/from the backplane on cache misses. Acknowledgements must also be returned from the SBC to inform the status of the current transactions - whether they have been finished successfully or

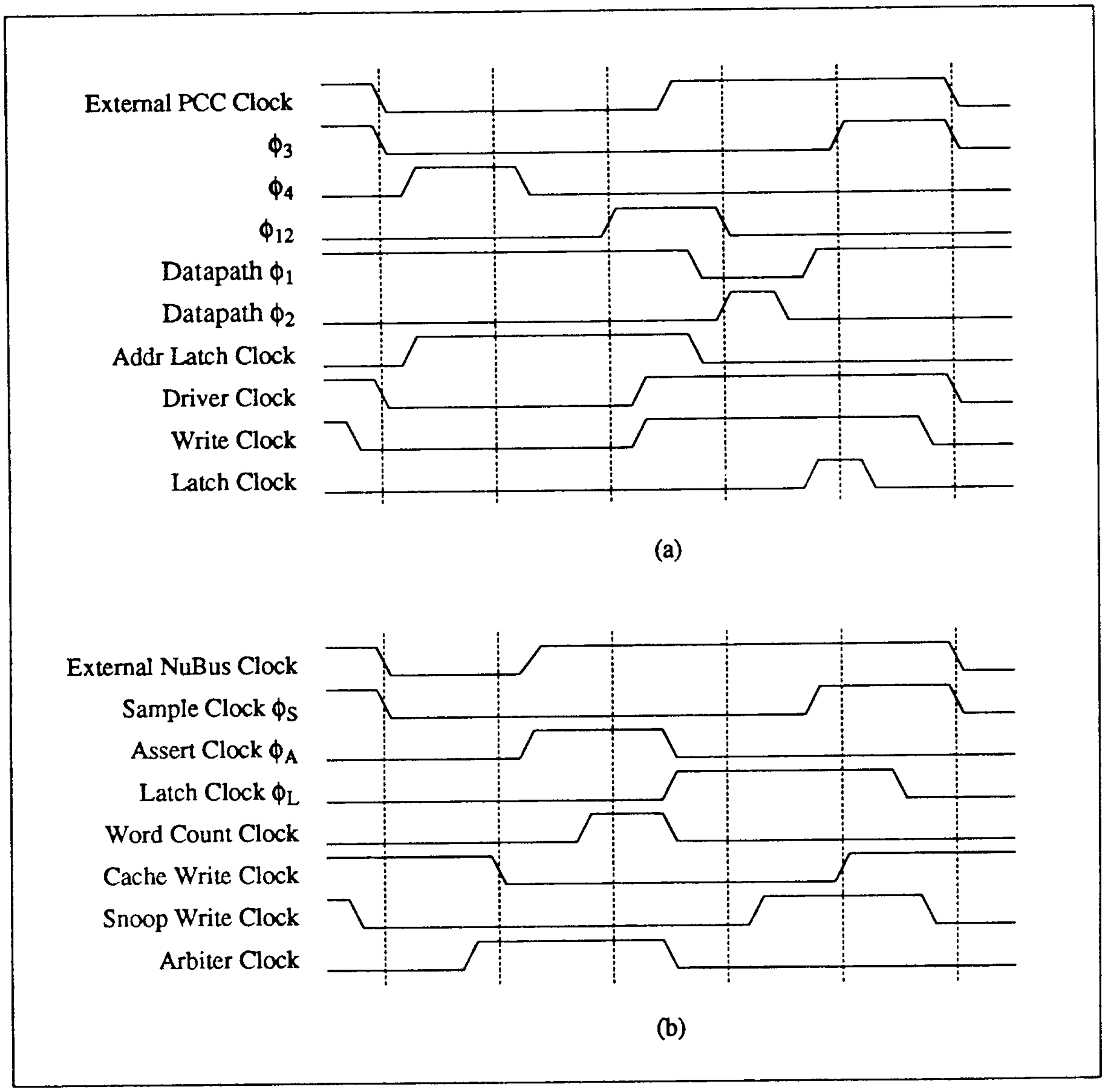

Fig. 6.10 Internal clock phases in the MMU/CC: (a) PCC clock phases, and (b) SBC clock phases.

resulted in errors. The other channel running in the opposite direction is mostly involved in snooping operations. The requests from the SBC are initiated when the SBC detects backplane transactions that require the PCC to relinquish the cache RAMs so that the SBC can invalidate, update or transmit a cache block. A logic diagram of one of the two asynchronous interface channels is shown Fig. 6.11. Instead of using a conventional 4 cycle handshake mechanism, a

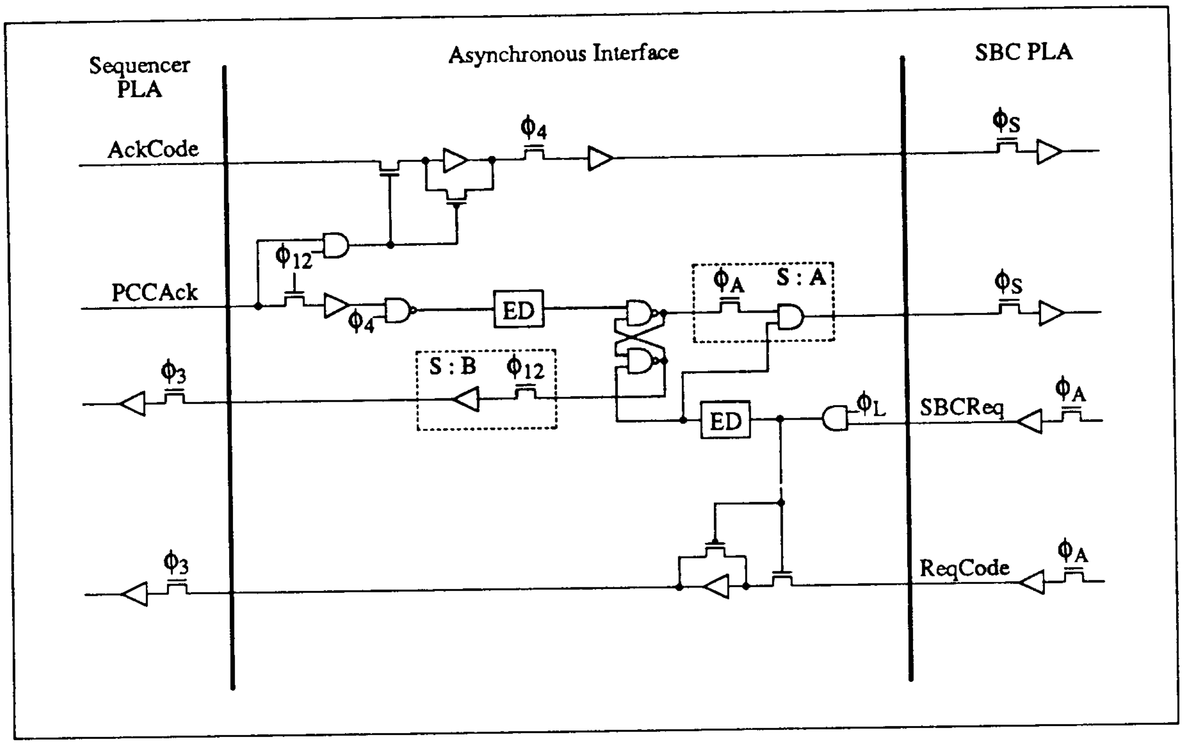

Fig. 6.11 Interface between the PCC and SBC. ϕ_{12}, ϕ_3 and ϕ_4 are PCC clock phases, and ϕ_A, ϕ_B and ϕ_L are SBC clock phases. ED is an edge detector that generates a short, negative pulse on the rising edge of the input, and S is a synchronizer with or without reset terminal. AckCode and PCCAck are valid during ϕ_{12}. Synchronizer A is allowed a half cycle of the SBC clock for settling time while synchronizer B is allowed two fifths of the PCC cycle time.

variant of 2 cycle handshake mechanism is used. Interface logic allows a request line to be asserted for only one cycle to log the request to the receiver, without requiring the sender to hold the request line all the way until an acknowledgement arrives. Similarly, a one-cycle acknowledgement informs the sender of the completion of the transaction. This mechanism in the interface reduces the complexity of implementing a handshake protocol in the PCC and SBC, as well as retaining the speed advantage of a 2 cycle handshake. Speed independent operation is achieved as long as the cycle time of each side does not exceed the pulse width of the edge detector output which is approximately 10 ns. Since all data (ReqCode and AckCode) arrives at the destination at the same time or before a request/acknowledgement is asserted, there is no need for synchronization for data. About a half cycle is allowed for synchronizing request/acknowledgement signals.

6.4 Design Methodologies and CAD Tools

For the design of a VLSI chip with as much functionality as the MMU/CC, good computer-aided design tools and adequate design methodologies are essential. Our design methodologies include design verification at the behavioral level, layout generation, layout verification, and a test suite. The way CAD tools are combined to design the SPUR MMU/CC chip is shown in Fig. 6.12.

Our highest level design tool is Endot's N.2 behavioral simulator based on a hardware description language called ISP' [6.11]. The entire SPUR system has been described in ISP', and system level design verification has been done using a set of diagnostics in behavioral level. Although the MMU/CC can be simulated as part of the whole system, it is very difficult and time-consuming to write diagnostics to test all the cases for the MMU/CC. MMU/CC events are not single cycle and much state information is concealed in protocol - so the number of possible state configurations is astronomical. A random tester was developed to verify the memory system including the MMU/CC in a multiprocessor configuration [6.12]. A *stub* module that simulates the CPU's memory reference behavior with an accurate interface between the CPU and the MMU/CC replaces the CPU to speed up verification time. The stub CPU generates memory references by randomly selecting from a set of predefined scripts. The scripts are composed of two parts, an action part that generates references to cause state changes, and a check part that checks if the correct state change is made. The actions and checks are executed at different times with other actions or checks intervening, causing complex cases to be generated. For example, one of the scripts includes an action part that writes a word to a memory address, and a check part

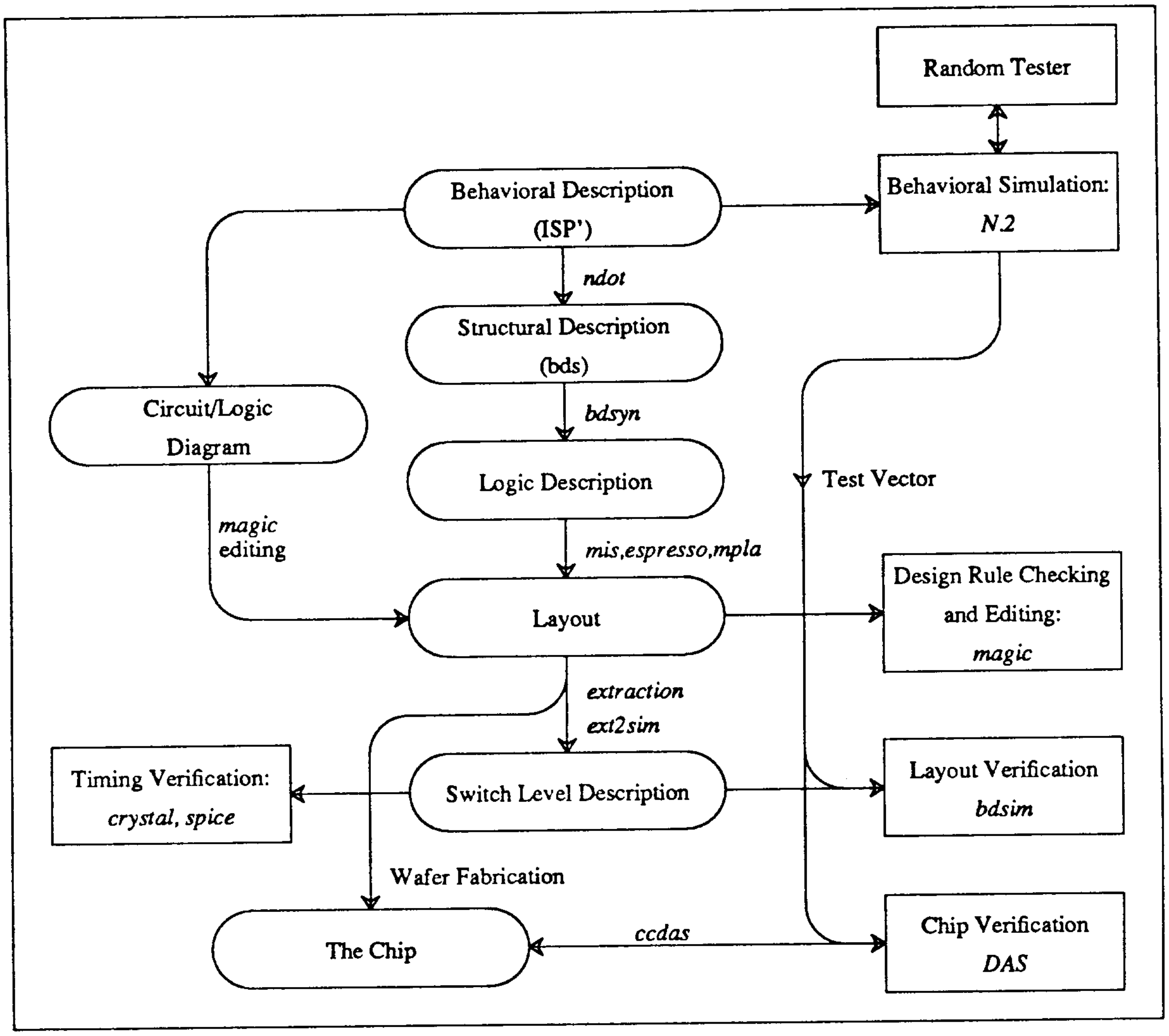

Fig. 6.12 Design flowchart and CAD tools.

that reads a word from the same address. If they do not match, an error will be signaled. Although random testing takes significant amount of computer time and memory space, a significant amount of human effort devising test vectors can be saved. The total number of simulation cycles was between 50-100 million cycles, and the simulator ran 1000 cycles per hour in a SUN-3 workstation. The random tester uncovered more than half of the functional bugs found during the simulation. The random tester alone is not powerful enough - it does not stop simulation exactly where the fault occurs. Rather, it stops later when the fault is detected. The monitor module is a passive "watcher" that stops simulation whenever the SpurBus or cache coherency protocol is broken. There is also a daemon module that generates *NuBus* I/O transactions that Spur Boards must cope with, but do not generate. These three together form the "random tester system."

Layout generation started before detailed design verification was completed. Almost all the layout design tools used had been developed at U.C. Berkeley. Limited layout generation tools were used for converting combinational ISP' descriptions into physical PLA layout. First, language translation from a ISP' description into a BLIF (Berkeley Logic Intermediate Format) description was done using the program *ndot* and *bdsyn* [6.13]. After application of a logic manipulation program *ml* and filtering through the logic minimizers *espresso* and *mis*, final PLA layouts were made using a tile-based PLA generator *mpla* [6.14]. Instead of merging all the controllers together and implement them with a standard cell approach, we decided to implement them with separate PLAs and connect them with global routing. By doing so, a behavioral description and the corresponding layout match closely and as a result a minor change in the controller description does not result in major layout revision. Only automatic regeneration of the PLA layout is involved without any change in the routing. Also, since the terminal names are preserved, it is easier to verify the layout. For random sequential logic and datapath circuits, manual conversion from ISP' descriptions to logic and circuit schematics was done. After circuit designs were made, the interactive layout tool *magic* was used to design and edit the layout [6.15]. There are several powerful features in *magic* which make the usually error-prone layout procedure more effective. Features such as a run-time, hierarchical design rule checker, multiple windows, cell hierarchy support and versatile layout selecting mechanisms are powerful enough to make a VLSI layout process manageable in a workstation environment. Macro blocks closely matching the procedures of the ISP' description were connected using an automatic router built into *magic*. Routing was driven by a netlist generated from a topology file prepared in functional level simulation. After layout, circuits were extracted along with parasitics for layout verification. The extracted hierarchical description was then converted into a flat switch level description using *ext2sim* [6.14].

After these preparation steps, final verification of the layout was done using a switch level simulator, *bdsim* [6.13]. *Bdsim* speeds up the simulation by grouping several transistors into a single construct and evaluating them as a unit. Test vectors used for *bdsim* were generated from the random tester. Final layout verification was completed by comparing the outputs of the two simulators against each other.

Timing verification was done using *crystal*. Crystal computes gate delay with a simplified MOS model and finds the critical path [6.14]. Even though the estimated delay found by *crystal* is not accurate in absolute time, it successfully finds the slowest signal path. As a final step, the circuit simulator *spice* is used to accurately estimate the critical path delay in case this path needs optimization [6.15]. *Spice* was also extensively used to verify the functionality of special circuits such as sense amplifiers and pad drivers.

For testability, "passive" scan paths were included that snapshot internal states and shift the result out under external control. Although they do not provide the ability to introduce arbitrary states, they are useful for debugging errors.

6.5 Status and Conclusions

The first version of the MMU/CC was sent out for fabrication in November 1987, and first silicon was received in February 1988. Omitting wafer probing, all chips were packaged and tested on a printed circuit board specially designed to connect to the Tektronix Digital Acquisition System (DAS). After downloading test vectors from a SUN workstation, the DAS exercised a chip and acquired result vectors. Result vectors were compared against the expected results using the SPUR specific tool, *ccdas*. Simple, short test vectors were used at this stage of testing. Although some chips passed all functional testing, we have experienced occasional errors. The errors were traced using scan paths and we discovered that some stack entries occasionally changed from 0 to 1 due to floating wells. In our methodology, instead of drawing wells explicitly, we relied on the *magic* layout editor for generating wells automatically. A few PMOS transistors were more than 12 λ's away from well contacts, so their wells were not properly biased. An ad hoc electrical rule checker was developed by changing the technology file of *magic*, and used for the next version of the chip. A second version arrived in September, 1988 and is fully functional. The SPUR CPU, MMU/CC and processor board now run the Sprite operating system at the intended clock frequency, 10 MHz. A three processor system is reliably running parallel processes. The working prototype of SPUR MMU/CC demonstrates the manageability of complexity in implementing both address translation mechanism and cache coherency in a full custom VLSI chip.

CHAPTER 7

Conclusion

This thesis examines various problems and issues around a digital system implementation. A particularly considered model is an asynchronously tied synchronous subsystems cluster. Since the synchronous subsystems are operating with their own frequencies, conventional synchronous VLSI design techniques can be applied regardless of the rest of the systems. Within such subsystems, an improved clocking scheme using an on-chip PLL-based clock generator can be used which minimizes inter-chip clock skew by achieving the effect of a zero-delay buffer. Experimental results show its applicability to VLSI systems although its circuits are composed of very sensitive analog devices. For communication among subsystems, synchronizers with low failure rate are required. Several circuit techniques of implementing a synchronizer that overcomes the technology limit are investigated. Their ideas are verified using parasitic-free devices through circuit simulation. However, it is concluded that these techniques are not effective with the current technology where device performance is dominated by parasitics. Since the device technology is directed to reduce parasitic components more rapidly than improving intrinsic devices, the circuit techniques can be applied in the future technology. Hardware implementation of handshake mechanisms are investigated which can be applied to an asynchronous interface between two synchronous sub-systems with independent clocks. As a design example, implementation details of a memory management unit and cache controller for a multiprocessor workstation are described which incorporated many of the circuit techniques explored in this thesis.

REFERENCES

Chapter 1

[1.1] Katevenis, M., et al., "Reduced Instruction Set Computer Architectures for VLSI," Ph.D. Thesis, U.C. Berkeley, Oct. 1983.

[1.2] Hill, M.D., et al., "Design Decisions in SPUR," *IEEE Computer*, vol.19, no. 10, pp. 8-24, Nov. 1986.

[1.3] Pechoucek, M., "Anomalous Response Times of Input Synchronizers," *IEEE Trans. Comput.*, vol. C-25, no. 2, Feb. 1976.

[1.4] Chapiro, D., "Globally-Asynchronous Locally-Synchronous Systems," Ph.D. Thesis, Department of Electrical Engineering, Stanford University, Oct. 1984.

[1.5] Jeong, D.-K., et al., "Design of PLL-Based Clock Generation Circuits," *IEEE J. Solid-State Circuits*, vol. SC-22, no. 2, pp. 255-261, April 1987.

Chapter 2

[2.1] Chapiro, D., "Globally-Asynchronous Locally-Synchronous Systems," Ph.D. Thesis, Department of Electrical Engineering, Stanford University, Oct. 1984.

[2.2] Pechoucek, M., "Anomalous Response Times of Input Synchronizers," *IEEE Trans. Comput.*, vol. C-25, no. 2, Feb. 1976.

[2.3] Seitz, C., "System Timing," Chapter 7 of *Introduction to VLSI Systems* by Mead, C., et al., pp. 218-262, Addison Wesley, 1980.

[2.4] Anceau, F., "A Synchronous Approach for Clocking VLSI Systems," *IEEE J. Solid-State Circuits*, vol. SC-17, Feb. 1982.

[2.5] Unger, S. H., "Hazards and Delays in Asynchronous Sequential Switching Circuits," *IEEE Trans. Comput.*, vol. C-6, pp. 12-25, March 1959.

[2.6] Friedman, A. D., "Synthesis of Asynchronous Sequential Circuits with Multiple-Input Changes," *IEEE Trans. Comput.*, vol. C-17, no. 6, pp. 559-566, June 1968.

[2.7] Unger, S. H., "Asynchronous Sequential Switching Circuits with Unrestricted Input Changes," *IEEE Trans. Comput.*, vol. C-20, no. 12, pp. 1437-1444, Dec. 1971.

[2.8] Chu, T.-A., "A Design Methodology for VLSI Self-timed Systems," Ph.D. Thesis, Department of EECS, MIT, Sept. 1986.

[2.9] Barros, J., et al., "Equivalence of the Arbiter, the Synchronizer, the Latch, and the Inertial Delay," *IEEE Trans. Comput.*, vol. C-32, No. 7. July 1983.

[2.10] Komori, S., et al., "An Elastic Pipeline Mechanism by Self-Timed Circuits," *IEEE J. Solid-State Circuits*, vol. SC-23, no. 1, Feb. 1988.

[2.11] Bazes, M., "A Novel Precision MOS Synchronous Delay Lines," *IEEE J. Solid-State Circuits*, vol. SC-20, No. 6, pp. 1265-1271, Dec. 1985.

[2.12] Johnson, M., et al., "A Variable Delay Line PLL for CPU-Coprocessor Synchronization," *IEEE J. Solid-State Circuits*, vol. SC-23, no. 5, pp. 1218-1223, Oct. 1988.

Chapter 3

[3.1] Fishburn, J. P., "Clock Skew Optimization," AT&T Work Project No. 311305-0199, Aug. 1987.

[3.2] Mohsen, A. M., "Delay-Time Optimization for Driving and Sensing of Signals on High-Capacitance Paths of VLSI Systems," *IEEE J. Solid-State Circuits*, vol. SC-14, pp. 462-470, April 1979.

[3.3] Shoji, M., "Elimination of Process-Dependent Clock Skew in CMOS VLSI," *IEEE J. Solid-State Circuits*, vol. SC-21, no. 5, pp. 875-880, Oct. 1986.

[3.4] *NuBus Specification*, Texas instruments, 1983.

[3.5] Samaras, W., "The CPU Clock System in the VAX 8800 Family," *Digital Technical Journal*, No. 4, pp. 34-40, Feb. 1987.

[3.6] *MC68020 32-bit Microprocessor User's manual*, Motorola Inc., 1985.

[3.7] *iAPX286 Hardware Reference Manual*, Tech. Report 210760, Intel Corp., Santa Clara, Calif., 1983.

[3.8] *CLIPPER Module Product Description*, Fairchild Corp., 1985.

[3.9] Best, R., *Phase-Locked Loops: Theory, Design, and Applications*, McGraw-Hill, New York, 1984.

[3.10] Forsythe, M., et al., "A 32-bit VLSI CPU with 15-MIPS Peak Performance," *IEEE J. Solid-State Circuits*, vol. SC-22, no. 5, pp. 768-775, Oct. 1987.

[3.11] Johnson, M., et al., "A Variable Delay Line PLL for CPU-Coprocessor Synchronization," *IEEE J. Solid-State Circuits*, vol. SC-23, no. 5, pp. 1218-1223, Oct. 1988.

[3.12] Gardner, F., "Charge-Pump Phase-Locked Loops," *IEEE Trans. Commu.*, vol. COM-28, pp. 1849-1858, Nov. 1980.

[3.13] Gardner, F., "Phase Accuracy of Charge Pump PLL's," *IEEE Trans. Commu.* vol. COM-30, pp. 2362-2363, Oct. 1982.

[3.14] Best, R., *Phase-Locked Loops: Theory, Design, and Applications*, McGraw-Hill, New York, 1984.

[3.15] Ebenhoech, H., "Make IC digital frequency comparators," *Electronic Design*, vol. 15, No. 14, pp. 62-64, July 5, 1967.

Chapter 4

[4.1] Chaney, T., et al., "The glitch phenomenon," Computer Systems Lab., Washington University, Tech Memo 10, Dec. 1966.

[4.2] Chaney, T., et al., "Beware the synchronizer," *Proc. COMPCON*, pp. 317-319, Sept. 1972.

[4.3] Chaney, T., et al., "Anomalous behavior of synchronizers and arbiter circuits," *IEEE Trans. Comput.*, vol. C-22, pp. 421-422, Apr. 1973.

[4.4] Courantz, G., et al., "Theoretical and Experimental Behavior of Synchronizers Operating in the Metastable Region," *IEEE Trans. Comput.*, vol. C-24, no. 6, June 1975.

[4.5] Pechoucek, M., "Anomalous Response Times of Input Synchronizers," *IEEE Trans. Comput.*, vol. C-25, no. 2, Feb. 1976.

[4.6] Chaney, T., et al., "Characterization and Scaling of MOS Flip-Flop Performance in Synchronizer Applications," Caltech Conference on VLSI, Jan. 1979.

[4.7] Stucki, M., et al., "Synchronization Strategies," Caltech Conference on VLSI, Jan. 1979.

[4.8] Seitz, C., "Ideas About Arbiters," *Lambda*, 1st Quarter, 1980.

[4.9] Veendrick, H., "The behavior of Flip-Flops Used as Synchronizers and Prediction of their Failure Rate," *IEEE J. Solid-State Circuits*, vol. SC-15, No. 2, April 1980.

[4.10] Chaney, T., "Measured Flip-Flop Responses to Marginal Triggering," *IEEE Trans. Comput.*, vol. C-32, no. 12, Dec. 1983.

[4.11] Kacprzak, T., " Analysis of Oscillatory Metastable Operation of an RS Flip-Flop," *IEEE J. Solid-State Circuits*, vol. SC-23, no. 1, Feb. 1988, pp. 260-266.

[4.12] Wormald, E., "A Note on Synchronizer or Interlock Maloperations," *IEEE Trans. Comput.*, March 1977.

[4.13] Chaney, T., "Comments on 'A note on Synchronizer or Interlock Maloperation'," *IEEE Trans. Comput.*, vol. C-18, no. 10, Oct. 1979.

[4.14] Barros, J., et al., "Equivalence of the Arbiter, the Synchronizer, the Latch, and the Inertial Delay," *IEEE Trans. Comput.*, vol. C-32, No. 7. July 1983.

[4.15] Marino, L., "The Effect of Asynchronous Inputs on Sequential Network Reliability," *IEEE Trans. Comput.*, vol. C-26, no. 11, pp. 1082-1977, Nov. 1977.

[4.16] Marino, L., "General Theory of Metastable operation," *IEEE Trans. Comput.*, vol. C-30, no. 2, Feb. 1981.

[4.17] Flannagan, S., "Synchronization reliability in CMOS technology", *IEEE J. Solid-State Circuits*, vol. SC-20, No. 2, pp. 880-882, April 1980.

[4.18] Sakurai, T., "Optimization of CMOS Arbiter and Synchronizer Circuits with Submicrometer MOSFET's," *IEEE J. Solid-State Circuits*, vol. SC-23, no. 4, Aug. 1988, pp. 901-906.

[4.19] Mead, C., et al., *Introduction to VLSI Systems*, Addison Wesley, pp. 236-242, 1980.

[4.20] Cordell, R., "A 45-Mbit/s CMOS VLSI Digital Phase Aligner," *IEEE J. Solid-State Circuits*, vol. SC-23, No. 2, pp. 323-328, April 1988.

[4.21] Rosenberger, F., et al., "Flip-Flop Resolving Time Test Circuit," *IEEE J. Solod-State Circuits*, vol. SC-17, no. 4, Aug. 1982.

[4.22] Getreu, I., *Modeling the Bipolar Transistor*, Amsterdam, Elsevier, 1976.

[4.23] Chor, E., et al., "A Propagation -Delay Expression and its Application th the Optimization of Polysilicon Emitter ECL Processes," *IEEE J. Solid-State Circuits*, vol. SC-23, no. 1, Feb. 1988, pp. 251-259.

[4.24] *MACSYMA Reference Manual*, The Mathlab Group, Laboratory for Computer Science, MIT, Jan. 1983.

[4.25] Glasser, L. A., "Frequency Limitations in Circuits Composed of Linear Devices," *IEEE Trans. Circuits Syst.*, vol. CAS-35, no. 10, pp. 1299-1307, Oct. 1988.

[4.26] Bagheri, et al., "Model for the Four-Terminal MOSFET," *IEEE Trans. Electron Devices*, vol. ED-32, no. 11, Nov. 1985.

Chapter 5

[5.1] Stone, H., *Microcomputer Interfacing*, Reading, MA, Addison-Wesley, 1982.

[5.2] Borrill, P., et al., "An Advanced Communication Protocol for the Proposed IEEE 896 Futurebus," *IEEE Micro*, pp. 42-56, Aug. 1984.

[5.3] Borrill, P., "MicroStandards Special Feature: A Comparison of 32-Bit Buses," *IEEE Micro*, pp. 71-79, Dec. 1985.

[5.4] Borrill, P., "Objective comparison of 32-bit buses," *microprocessors and microsystems*, vol. 10, no. 2, pp. 94-100, March 1986.

[5.5] Borriello, G., "A New Interface Specification Methodology and its Application to Transducer Synthesis," Computer Science Division Report, No. UCB/CSD 88/430, University of California, Berkeley, May 1988.

[5.6] Chu, T.-A., "On the Models for Designing VLSI Asynchronous Digital Systems," *Integration*, the VLSI Journal 4, 1986.

Chapter 6

[6.1] Katevenis, M., et al., "Reduced Instruction Set Computer Architectures for VLSI," Ph.D. Thesis, U.C. Berkeley, Oct. 1983.

[6.2] Moussouris, J., et al., "A CMOS RISC processor with integrated system functions," *24th Annual IEEE Computer Conference (COMPCON '86)*, March 1986.

[6.3] Garner, R., et al., "The Scalable Architecture (SPARC)," *26th Annual IEEE Computer Conference (COMPCON '88)*, March 1988.

[6.4] Hill, M. D., et al., "Design Decisions in SPUR," *IEEE Computer*, vol. 19, no. 10, pp. 8-24, Nov. 1986.

[6.5] Gibson, G., "SpurBus Specification," *Proc. of CS292i: Implementation of VLSI Systems*, R.H. Katz (Editor), University of California, Berkeley, Sept. 1985.

[6.6] Wood, D. A., et al., "An In-cache Address Translation Mechanism," Proceedings 13th Annual Symposium on Computer Architecture, Tokyo, Japan, pp. 358-365, June 1986.

[6.7] Katz, R. H., et al., "Implementing a Cache Coherency Protocol," Proceedings 12th Annual Symposium on Computer Architecture, Boston, MA, pp. 276-283, June 1985.

[6.8] Wood, D., et al., "SPUR Memory System Architecture," Computer Science Division, Report No. UCB/CSD 87/394, University of California, Berkeley, Dec. 1987.

[6.9] Jeong, D.-K., et al., "Design of PLL-Based Clock Generation Circuits," *IEEE J. Solid-*

State Circuits, vol. SC-22, no. 2, pp. 255-261, April 1987.

[6.10] Pechoucek, M., "Anomalous Response Times of Input Synchronizers," *IEEE Trans. Comput.*, vol. C-25, no. 2, Feb. 1976.

[6.11] Kong, S., et al., "Design Methodology for a VLSI Multiprocessor Workstation," *VLSI Systems Design*, pp. 44-55, Feb. 1987.

[6.12] Wood, D. A., et al., "Verifying a Multiprocessor Cache Controller Using Random Case Generation," Computer Science Division Report, No. UCB/CSD 89/490, University of California, Berkeley, Jan. 1989.

[6.13] Segal, R. B., "BDSYN: Logic Description Translator, BDSIM: Switch-level Simulator", *Electronics Research Laboratory Memorandum*, No. M87/33, University of California, Berkeley, May 1987.

[6.14] Scott, W., et al., "1986 VLSI Tools: Still More Works by the Original Artists", Computer Science Division Report, No. UCB/CSD 86/272, University of California, Berkeley, Dec. 1985.

[6.15] Ousterhout, J., et al., "The Magic VLSI Layout System", *IEEE Design & Test of Computers*, vol. 2, pp. 19-30, Feb. 1985.

[6.16] Nagel, L. W., et al., "Simulation program with integrated circuit emphasis (SPICE)," 16th Midwest Symp. Circuit Theory, Waterloo, Ont, Canada, 1973.