DESIGN AND TESTING OF A MICROPROCESSOR COMPATIBLE 128-BIT CORRELATOR CHIP

A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND
ELECTRONICS ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCES
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Satılmış TOPÇU
July 1989

Thesis
TK
7874
T62
1989

DESIGN AND TESTING OF A MICROPROCESSOR COMPATIBLE 128-BIT CORRELATOR CHIP

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND

ELECTRONICS ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Satılmış Topçu

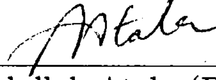
July 1989

Thesis
TK
7874
T62
1989

B1869

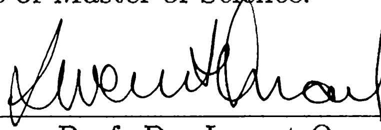
To my lovely wife and family

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



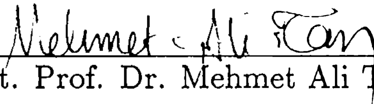
Assoc. Prof. Dr. Abdullah Atalar(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



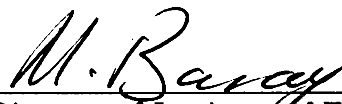
Assoc. Prof. Dr. Levent Onural

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Asst. Prof. Dr. Mehmet Ali Tan

Approved for the Institute of Engineering and Sciences:



Prof. Dr. Mehmet Baray, Director of Institute of Engineering and Sciences

ABSTRACT

DESIGN AND TESTING OF A MICROPROCESSOR COMPATIBLE 128-BIT CORRELATOR CHIP

Satılmış Topçu
M.S. in Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Abdullah Atalar
July 1989

In digital synchronous data transmission, synchronization (sync) words are used to mark the beginning of the incoming data stream. Detection of the sync word received from a noisy channel is a difficult problem. One of the optimum solutions to this problem is to use a correlator. A correlator could be implemented with SSI and MSI components on a printed circuit board with the disadvantage of bulkiness. To use it in light-weight equipment such as portable data terminals, it is designed to be implemented as a full-custom single VLSI chip. It can be used for the 128-bit sync word detection and PRBS generation. Two chips can be cascaded for 256-bit correlation as well as distributed sync words, and inverted or non-inverted sync words can be detected. It is fully programmable by a microprocessor to set the number of tolerable errors in detection and to select the bits of the 128-bit (or 256-bit) input data stream to be used in the correlation and hence, it can be directly connected to a microprocessor as a peripheral device.

In designing the correlator chip some Design For Testability methods are used to improve the testability. Especially, scan design and partitioning techniques are applied resulting in a significant decrease in the number of test patterns although these techniques involve an overhead in the overall transistor count only by 1 percent.

For functional and timing simulations **ESIM** and **RNL** simulators are used, respectively. Test patterns for the registers are generated manually

and for testing of the combinational part two programs, **gen** and **check**, are written in C programming language. The simulation programs and test pattern generation programs are run on SUN workstations under 4.3 BSD UNIX[1] operating system.

Keywords: Digital synchronous data transmission, correlator, chip, VLSI, IC testing, design for testability.

[1]UNIX is a Trademark of Bell Laboratories.

ÖZET

BİR MİKROİŞLEMCİ UYUMLU 128-BİT KORELATÖR YONGASININ TASARIMI VE TEST EDİLMESİ

Satılmış Topçu
Elektrik ve Elektronik Mühendisliği Bölümü Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Abdullah Atalar
Temmuz 1989

Eş zamanlı sayısal veri iletişiminde, senkron (sync) sözcüğü gelen verinin başlangıcını belirler. Gürültülü bir kanaldan alınan senkron sözcüğün sezilmesi zor bir problemdir ve bu problemin en uygun çözümlerinden biri korelatör kullanmaktır. Bir korelatör düşük ve orta yoğunluklu tümleşik devreler kullanılarak baskılı devre üzerinde gerçekleştirilebilir, fakat büyük bir alan kaplaması olumsuz yanıdır. Bu çalışmada, taşınabilir veri terminalleri gibi hafif donanımlarda kullanılabilecek, mikroişlemci uyumlu, çok yüksek yoğunluklu tümleşik (VLSI) 128-bit programlanabilir sayısal korelatör yongası tasarlandı. Bu korelatör yongası 128-bit uzunluğa kadar olan senkron sözcüklerini sezebilir. Ayrıca iki yonganın birbirine kademeli olarak bağlanmasıyla 256-bitlik korelatör elde edilebilmektedir. Senkron sözcük, gelen verinin düz yada tersine çevrilmiş olduğu her iki durumda da sezilebilir. Korelatör bir mikroişlemci tarafından bütünüyle programlanabilir ve bu nedenle mikroişlemciye bir çevre birimi olarak doğrudan bağlanabilir. Korelasyonda kullanılacak bitleri seçme özelliğine sahip olan korelatör, dağıtılmış senkron sözcüğü seziminde ve yalancı rasgele ikili seri (PRBS) üretiminde kullanılabilir.

Korelatörün tasarımında bazı "test edilebilirlik tasarımı" metodları kullanıldı. Özellikle tarama tasarımı (scan design) ve parçalama (partitioning) yöntemleri kullanılarak test vektör sayısı önemli ölçüde azaltıldı. Buna

rağmen test edilebilirlik tasarım metodları yonga tasarımına yalnızca % 1 ek transistör artışı getirdi.

Yonganın işlevsel ve zamanlama simülasyonları **ESIM** ve **RNL** simülasyon programları kullanılarak yapıldı. Yazmaçlar için kullanılacak olan test vektörleri doğrudan elle bilgisayarda üretilmiş olup kombine mantık devrelerinin test edilmesi için de, C programlama dilinde **gen** ve **check** adlı iki program yazıldı. Bu test vektörü üreten programlar ve simülasyon programları 4.3 BSD UNIX[2] işletim sistemi altında SUN bilgisayar sistemlerinde çalıştırıldı.

Anahtar kelimeler : Eş zamanlı sayısal veri iletişimi, korelatör, yonga, VLSI, tümleşik devre testi, test edilebilirlik tasarımı.

[2]UNIX, Bell Laboratuvarlarının ticari markasıdır.

ACKNOWLEDGEMENT

I would like to thank to Assoc. Prof. Dr. Abdullah Atalar for his supervision, guidance, suggestions, and encouragement throughout the development of this thesis. I am also indebted to the members of my thesis committee: Assoc. Prof. Dr. Levent Onural, Asst. Prof. Dr. M. Ali Tan, and Asst. Prof. Dr. Gürhan Şaplakoğlu for their advice and support.

A special note of thanks is due to the research assistants in VLSI group: İ. Enis Ungan and Mustafa Karaman for their valuable remarks, comments, and helps. Many thanks also to Şenol Toygar who is the original designer of the correlator, to Nesip Aral, Tuncay Ergün, Oğuz Şener (all from ASELSAN) who gave their time for the improvement of the correlator system design.

I am grateful to my wife for providing morale support during this study.

TABLE OF CONTENTS

1 INTRODUCTION 1

1.1 Detection of Digital Signals 1

1.2 PRBS Generation 4

1.3 The Digital Correlator 5

2 THE ARCHITECTURE DESIGN AND OPERATION OF THE CHIP 7

2.1 General IC Design Process 7

2.2 Specifications 10

2.2.1 Specification of the Functions 10

2.2.2 Performance Requirements 11

2.2.3 System Environment and Interface Definition 11

2.3 The Architecture Design 12

2.3.1 General Description 12

2.3.2 Block Representation 12

2.3.3 Pin Description 22

2.4 Modes of Operation 23

2.4.1 Correlator Mode 23

2.4.2 PRBS Generator Mode 29

3 THE LOGIC AND CIRCUIT DESIGN 31

3.1 The Logic Design 31

3.2 The Circuit Design 40

4 DESIGN FOR TESTABILITY 42

4.1 Ad Hoc DFT Methods 43

4.2 Structured DFT Approaches 45

5 SIMULATIONS AND TESTING OF THE CHIP 51

5.1 Functional Verification 51

5.2 Timing Verification 53

5.3 Test Pattern Generation (TPG) and Fault Simulation 55

6 CONCLUSION 63

REFERENCES 65

APPENDIX A 68

APPENDIX B 75

LIST OF FIGURES

1.1 Basic digital correlator. 3

1.2 Maximal-length linear feedback shift register. 5

2.1 The linear design process not including iterative loops. 8

2.2 Block diagram of the correlator in the beginning. 13

2.3 Simplified block diagram of the correlator. 15

2.4 Block diagram of the mask and reference registers. 16

2.5 Timing diagram of the write and read cycles. 17

2.6 Block diagram of the integrator. 20

2.7 Block diagram of the decision maker. 20

2.8 Pin diagram of the correlator. 22

2.9 Timing diagram of the sync detection. 25

2.10 128-bit correlation scheme. 26

2.11 256-bit correlator. 27

2.12 256-bit correlation scheme in the master chip. 28

2.13 The configuration for PRBS generator. 29

3.1 Logic diagram of the 8-bit tristate bidirectional buffer. 32

3.2 Logic diagram of the controller module. 33

3.3 Decoder circuit used in the controller. 34

3.4 Logic diagram of the status register. 34

3.5 Master-slave flip-flop. 35

3.6 Schematic of the 128-bit shift register. 36

3.7 Logic diagram of the reference and mask registers. 37

3.8 Logic diagram of the threshold register. 37

3.9 Logic diagram of the comparator. 37

3.10 Logic diagram of the 1's counter. 38

3.11 The circuit of an n-bit full-adder. 38

3.12 The 9-bit full adder circuit. 39

3.13 Transistor circuit of a compare cell. 41

4.1 Symbolic representation of a shift register latch. 46

4.2 Level sensitive scan design of the reference/mask registers. . . 47

4.3 Level sensitive scan design of the threshold register. 47

4.4 Level sensitive scan design of the status register. 48

4.5 LSSD configuration of the registers. 49

LIST OF TABLES

2.1 Function of a compare cell. 18

2.2 Function table of the controller block. 21

2.3 Selecting modes of the correlator. 23

2.4 Decision table for the sync detection. 27

2.5 Decision table for 256-bit sync detection. 29

3.1 Function table of the buffer. 32

3.2 Truth table of a carry cell. 39

5.1 Timing simulation of the reference register. 54

5.2 Timing simulation of the sync detection. 56

5.3 Test vectors for two-bit full-adders. 60

5.4 Test vectors for three-bit full-adders. 61

5.5 The number of test vectors for stages in the 1's counter. 61

1. INTRODUCTION

1.1 Detection of Digital Signals

Synchronous data transmission is more complicated than asynchronous data transmission. Because it requires a higher level of coordination between the data source and user data terminal equipment than does asynchronous data communication. In return for this inconvenience, synchronous data transfer largely eliminates the overhead of the start/stop pulses of the asynchronous method and therefore provides for more efficient data transfer.

On a synchronous data link, there are two levels of synchronization to be achieved: bit synchronization and character synchronization. Bit synchronization refers to the adjustment of the receiving data communications equipment timing so that it "knows" at what point in time to make the decision as to whether a 1 bit or a 0 bit is currently being received. Character or frame synchronization allows the receiving equipment to determine which bit of the received bit stream actually is the first bit in a received character or which bit actually begins the data in a data frame. Frame synchronization technique is useful in situations where the data transmission (bit) rate is known and invariant, where the receiver's clock is triggered by the clock of the transmitter or where the "receiver and transmitter" are part of the same system [1].

In general, a system can achieve and maintain frame synchronization only if the incoming data stream is interrupted periodically by a specific "start-of-frame" pattern. This pattern, transmitted at the beginning of each new frame, tells the receiver that a new frame will immediately enter the input register and must be handled accordingly.

In digital synchronous data transmission, frame synchronization is obtained by transmitting a synchronization (sync) word to define the beginning

of the incoming data frame. Detection of the sync word received from a noisy channel is an important problem. Because, in practice, due to the noise in the channel, some bits of the incoming data stream will be altered. That is why error detection and correction techniques are used in digital data transmission systems. Especially, the sync word, which determines the beginning of the incoming data stream, can be lost if the channel noise is high. A powerful technique for detecting the sync word is to use a correlator [2].

Correlation techniques are used widely in communications, instrumentation, computers, telemetry, sonar, radar, medical and other signal processing systems. Correlation has several desirable properties, including:

- The ability to detect a desired signal in the presence of noise or other signals,
- The ability to recognize specific patterns within analog or digital signals,
- The ability to measure time delays of known signals through various media, such as materials, the human body, RF paths, electronic circuits, etc..

Probably we use correlation daily when we compare sounds, images, or other sensations relative to other sounds, images or sensations stored in our brain. As these properties indicate, the correlation between two functions is a measure of their similarity; loosely termed, it is a comparison process. This comparison can be expressed mathematically as the correlation between two functions $v_1(t)$ and $v_2(t)$ [2]:

$$R_{12}(\tau) \equiv \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{+T/2} v_1(t)\, v_2(t+\tau)\, dt$$

Here , $R_{12}(\tau)$ refers to the correlation between two signals, v_1 and v_2. It is determined by multiplying one signal, $v_1(t)$, by the other signal shifted in time, $v_2(t+\tau)$, and then taking the integral of the product. Thus, correlation involves multiplication, time shifting (or delay) and integration. Inspection of the arguments of functions v_1 and v_2 shows that correlation handles two functions "forward" in time and it is very sensitive to their relative phase.

Whereas the functions discussed above are continuous analog representations of physical variables, digital signal processing requires functions to be

represented in discrete form, where the time scale and amplitude are quantized into discrete steps. So, in a digital system, each signal may be a series of single bit samples where the two bit values 1 and 0 are generally interpreted as positive and negative respectively and the correlation integral is changed into a finite sum. The correlation equation in discrete form becomes [2]:

$$R(n) = \sum_{k=-\infty}^{\infty} v_1(k)\, v_2(n+k)$$

Here, the indices "k" and "n" measure out the variables denoted by "t" and "τ" in the earlier discussion. In practice, the summation will cover finite range of values of "k", rather than the infinite range shown here. The range depends on the durations of the two functions and of their sampled portions.

In applications involving single bit reference and single bit data streams, multiplication is actually implemented with the exclusive-NOR (EX-NOR) function, which yields positive results (1) if the two bits (polarities) coincide and negative results (0) if they differ. If these 1's and 0's are then summed (i.e. if the number of 1's is calculated), then the result is a correlation score ranging from 0 (for perfect anticorrelation) to N (for perfect correlation, where N is the number of taps in the system).

A digital correlator can perform correlation, operating according to the discrete summation equation. The major functions of a digital correlator are shown in Fig. 1.1. A reference shift register is pre-loaded with the standard synchronization pattern, while the incoming data stream is shifted serially through the input shift register. Both shift registers are n-bit long. The respective bits of the two shift registers are connected to individual exclusive-NOR gates, whose outputs are applied to a summing circuit [2].

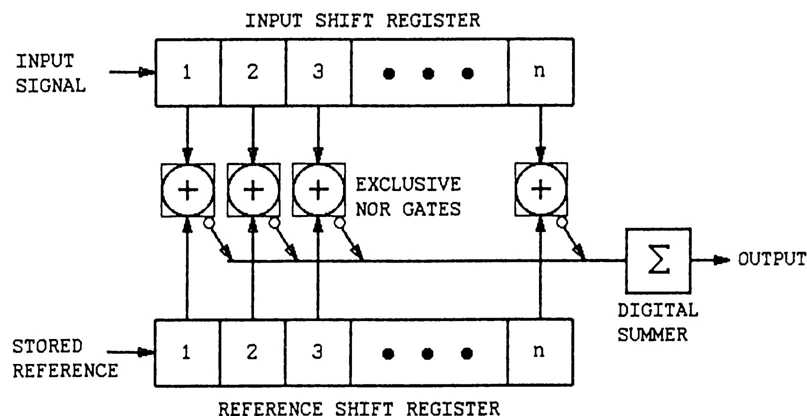


Figure 1.1: Basic digital correlator.

In operation, the correlator output is obtained by aligning the input word relative to the reference word. The respective bits in the two shift registers are compared by the exclusive-NOR gates, whose outputs are summed. The shift registers, the exclusive-NOR gates, and the summer fulfill the three functions of the correlation: time delay, multiplication, and integration, respectively. When a frame sync pattern embedded in the incoming data stream aligns exactly with the stored sync pattern, the correlator produces a sync pulse at the output, which in turn tells the receiver to start handling a new frame of data. The special start-of-frame pattern must be long and unusual to prevent false synchronization, which occurs when the correlator finds and locks onto a portion of the data stream resembling the desired synchronization pattern. The longer the sync pattern, the lower the chance of a false synchronization, but the larger the percentage of the total data stream that must be dedicated to synchronization and hence is unavailable for transmitting useful information.

The receiver-correlator system operates continuously, with the correlator producing a sync pulse as it receives each successive frame sync pattern in the incoming data stream. The rate at which these pulses are generated is the frame rate of the incoming data. The correlator mainly compares the incoming data stream with a predetermined reference data and then decides whether the sync word is received or not. But, since there is a noise in the channel, the correlator must be modified in such a way that it may tolerate some number of errors in the data stream.

1.2 PRBS Generation

A pseudo-random binary sequence (PRBS) consists of a sequence of ones and zeros that possess certain specific autocorrelation properties. Such sequences play an important role in almost all types of spread-spectrum systems. One method of producing pseudo-random binary sequence is the use of feedback shift registers. These are simple to implement, are very fast in operation, and can be made to generate statistically very good sequences, provided the registers are long enough. In general, such a shift register consists of n locations labeled from *1* on the left to n on the right, which holds a binary n-string as shown in Fig. 1.2. Here, *modulo 2* addition can be achieved by using exclusive-OR gates. The shift register is stepped on by shifting all the bits right one location and feeding back the *modulo 2* sum output into the left-hand location [3].

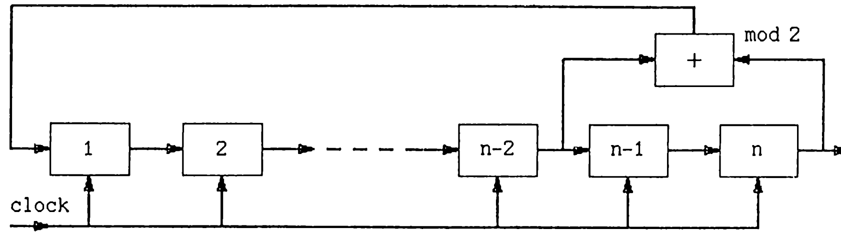


Figure 1.2: Maximal-length linear feedback shift register.

It can be shown that the output sequence must eventually repeat with a period which is at most $2^n - 1$. If the feedback connections have been chosen to give this maximal period, the output sequence depends on the initial contents only for its starting point (provided the initial contents are not the string of all zeros). Since the output sequences can be shown to have good quasi-random properties, it seems reasonable to use them as sync words, provided n is chosen large enough and provided that the feedback is chosen to give the maximal period $2^n - 1$. The key would be to specify the feedback connections and the initial contents of the feedback shift register. In this case, the output sequence is known as a maximal-length feedback shift-register sequence.

In the correlator chip, a 128-bit shift register is used for PRBS generation and a single feedback line obtained from it modulo 2 summation of the selected shift register taps is applied to the serial input on the left hand side. The PRBS output is also taken serially from this feedback line instead of the output of the last stage. Here, it n is equal to 128 and it is large enough to generate very good pseudo-random binary sequences.

1.3 The Digital Correlator

Electronic systems that perform correlation have been around for years, but they have been bulky and inefficient. The development of VLSI (Very Large Scale Integration) has changed this; now correlation can be performed efficiently with a digital correlator chip [4]. In fact, a digital correlator could be implemented with SSI and MSI components on a printed circuit board with the disadvantage of bulkiness. The continued progress in increasing the performance, speed, reliability, and the simultaneous reduction in size and cost of IC's has made the solution easier. To use it in light-weight equipment, such as portable data terminals, this correlator and PRBS generator was designed to be implemented as a full-custom VLSI chip. Actually, it is seen that as levels of integration go towards VLSI, the design time and design

effort needed for full-custom design technology is likely to grow exponentially. But, on the other hand, full-custom design technology offers great flexibility to the designer and it reduces circuit complexity per function and improves system performance. Also, this design technique may provide the designer an easy way of testing the chip.

The design of full-custom VLSI correlator chip was jointly carried out with İ. Enis Ungan. At the beginning, we had a correlator circuit at the logic level designed by M. Şenol Toygar. Then this circuit was modified to give it some important properties such as cascade connectibility, PRBS generation, etc.. One of the main improvement is the added circuitry for testability. For this purpose, special Design For Testability (DFT) methods were used. Especially, scan design technique was applied to control the states of the registers and observe the contents of them easily by forming a scan path. Another important DFT technique used is the partitioning which reduces the problem of testing into dealing with the smaller modules. As a result, testability was improved considerably. Circuit and layout designs were done by İ. Enis Ungan and so details of the circuit and layout designs can be found in [5,6].

2. THE ARCHITECTURE DESIGN AND OPERATION OF THE CHIP

2.1 General IC Design Process

The design description for an integrated circuit may be described in terms of three domains, namely: 1) the behavioral domain, 2) the structural domain, and 3) the physical domain. A good VLSI design system should provide for consistent descriptions in all description domains and at all relevant levels of abstraction. The means by which this is accomplished may be measured in various terms that differ in importance based on the application. These design parameters may be summarized in terms of [7,8,9]

- performance - speed, power, function,
- size of die,
- time to design - ease of use, and
- ease of test generation and testability.

Design is a continuous trade-off to achieve adequate results for all of the above parameters. As such, the tools and methodologies used for a particular chip will be a function of these parameters. Certain end results have to be met (i.e. the chip must conform to performance specifications), but other constraints may be a function of economics or even subjectivity [10,11].

The general IC design process includes all individual steps required for designing an integrated circuit, starting with specification and ending with the generation of production control and test data. The linear design process, not including iterative loops, is shown in Fig.2.1 [12].

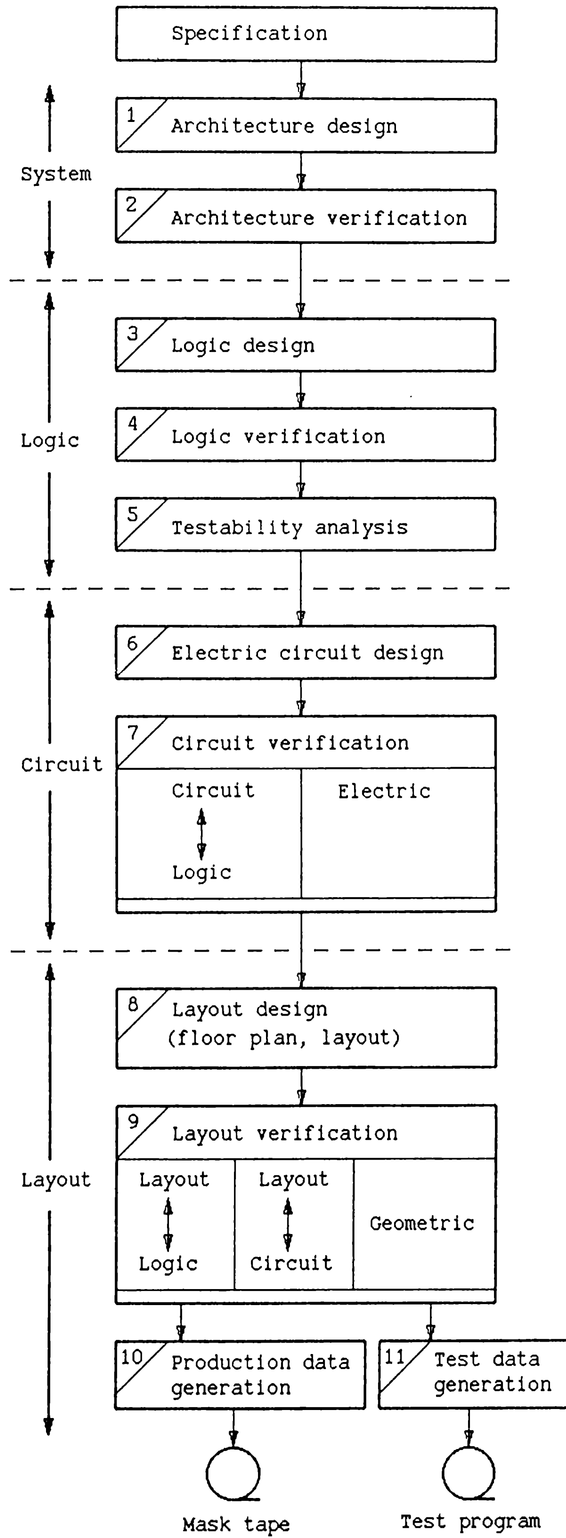


Figure 2.1: The linear design process not including iterative loops.

It is divided in four segments, namely

1. system design,
2. logic design,
3. circuit design, and
4. layout design.

The first two segments are also often termed "logical design", the last two "physical design".

The *system design* includes the architecture and module design which, in several detailing steps, leads to *logic design* on the gate level. Next come the *logic verification* and the *testability analysis*. The *electrical circuit design* initiates the physical design with realization of logic by means of transistor circuits and subsequent verification. Finally, the *layout design* includes geometric design (i.e. the design of all masks required for production of all levels), corresponding steps of verification and, in order to establish a path to production, *generation of the mask tapes*. The *generation of test data* is based upon the logic design which results in executable test programs.

This segmentation in various levels is characteristic for IC development and shows the process from the abstract representation on a conceptual level in step-by-step realization to the mask tapes which describe in detail the manufacturing process. Each higher level constitutes an abstract representation of the next lower level. Vice versa, moving from a higher to a lower level requires special design know-how corresponding to the design method and technology chosen. Thus, the circuit designer has to know, for example, a way to realize the logic function of a D-flipflop by means of transistors and their interconnection utilizing CMOS technology.

However, it is obvious that if certain rules (design rules) have to be followed, errors can be made. So, a sequence of the "implementation" and "integration and test" steps is formed which is repeated on the various design levels. The repetition of the "implementation" and "integration and test" phases is required because, for each design realization, compliance with two correctness conditions has to be examined:

- Compliance with the requirements of the previous design levels, i.e. ensuring that the given target function is met by the design result obtained to that point,

- Compliance with the design rules for the actual draft of the design object on each level of representation, across the whole of the design object.

Rules pertaining to point 1 ensure vertical consistency, and rules pertaining to point 2 ensure horizontal consistency. In the case of the "layout design" step, for example, the result has to be examined with regard to consistency with the given electrical circuit design. In addition, a check of the layout design rules must be done [12].

In designing the correlator and PRBS generator chip, the general top-down method has been used following the design steps shown in Fig. 2.1. Basically, the design process includes eleven individual steps starting with specification and ending with the generation of production and test data. In the following sections, these basic design steps are presented in detail.

2.2 Specifications

Based upon a study of the requirements, the specification or performance description is developed by applying available techniques for problem solving. The specification includes the solution concept, functional concept, performance concept and the structure concept. For the development of subsystems, their arrangement in the main system plays a crucial role. The interface between subsystem and main system is defined by criteria such as type, format and range of data and command flow. For object specification including the solution concept, the performance requirements are also defined. These are, in most cases, the determination of critical time conditions, or of maximum values, such as the maximum current supply or the size of the memory. In the specification phase for the correlator and PRBS generator chip, after the specification of functions and performance requirements, the system environment and interface definition has been done [12].

2.2.1 Specification of the Functions

The correlator and PRBS generator executes the following functions:

- It can detect sync words of length up to 128-bit.

- Two chips can be cascaded to increase the length to 256-bit.
- It allows detection of the sync regardless of the polarity of the incoming data stream.
- It can also detect distributed sync words.
- It has error tolerance utility by means of which a number of errors in the data stream can be tolerated.
- It can generate PRBS (Pseudo Random Binary Sequence).

2.2.2 Performance Requirements

Basically, the performance requirements for the correlator and PRBS generator chip have included the following:

- clock rate for correlation greater than 25 KHz,
- time to decide whether the sync is detected or not less than 20 μs,
- clock rate for PRBS generation greater than 500 KHz,
- supply current less than 20 mA, supply voltage : 5 V.

2.2.3 System Environment and Interface Definition

The correlator and PRBS chip functions as a programmable microprocessor peripheral. So, it has an 8-bit data bus and a 3-bit address bus with chip select (CS), write (WR), and read (RD) inputs controlled by microprocessor. Thus, its internal registers can be programmed directly through data bus of the microprocessor. The incoming data entering from the serial input (SIN) is taken inside the chip serially by using the clock (CLK) supplied by phase lock loop (PLL) and go out from the serial output (SOUT). When the sync is detected, the chip informs microprocessor by sending an interrupt signal, which we call the sync pulse, through sync (SYN) output. In addition, the chip has an 8-bit bidirectional bus making possible 256-bit correlation by means of cascade connection of two chips.

2.3 The Architecture Design

2.3.1 General Description

The architecture design is a first approach to meet the specification requirements. The realization or actual development begins with this design phase. As usual, a top-down method is applied. The system function is divided into hierarchical levels from top to bottom. It is divided into single functions which are placed in a static connection structure, and whose interactions are controlled by means of a dynamic cooperation structure. Connection and cooperation structures are called function structures. The cooperation structure consists in turn of two components: one for the information flow, and one for the control flow.

In the architecture design phase, far-reaching assessments are made, such as the formal representation of system functions (target functions), division into subfunctions by hierarchically layered resolution, design of interface architecture, connection structure and functional structure. For complex problems such decisions cannot, in general, be made with certainty. Wrong decisions and design errors in this phase cause especially undesirable effects. So, on the various levels of the architecture design, verification steps are performed repeatedly (architecture verification). These steps have been carried out manually for the first design outlines, as design reviews or walk-throughs [12].

2.3.2 Block Representation

In the beginning of the architecture design, the correlator circuit designed by M. Şenol Toygar was used [13]. The block diagram of this circuit is shown in Fig. 2.2. It has some unnecessary features as well as some missing and important properties. For example, it has a PWO (Power Okey) input which will put the data bus in high-Z state when the power supply of the microprocessor is down. It has also IACS and SACS bits which control the active position of INV and SYN outputs, respectively. In addition, it has an SYN output pin. These features were thought as unnecessary and they were omitted from the design. Afterwards, the design was changed to have some important properties. For this purpose, firstly, cascade connection property was added to it in order to have the ability of performing 256-bit correlation.

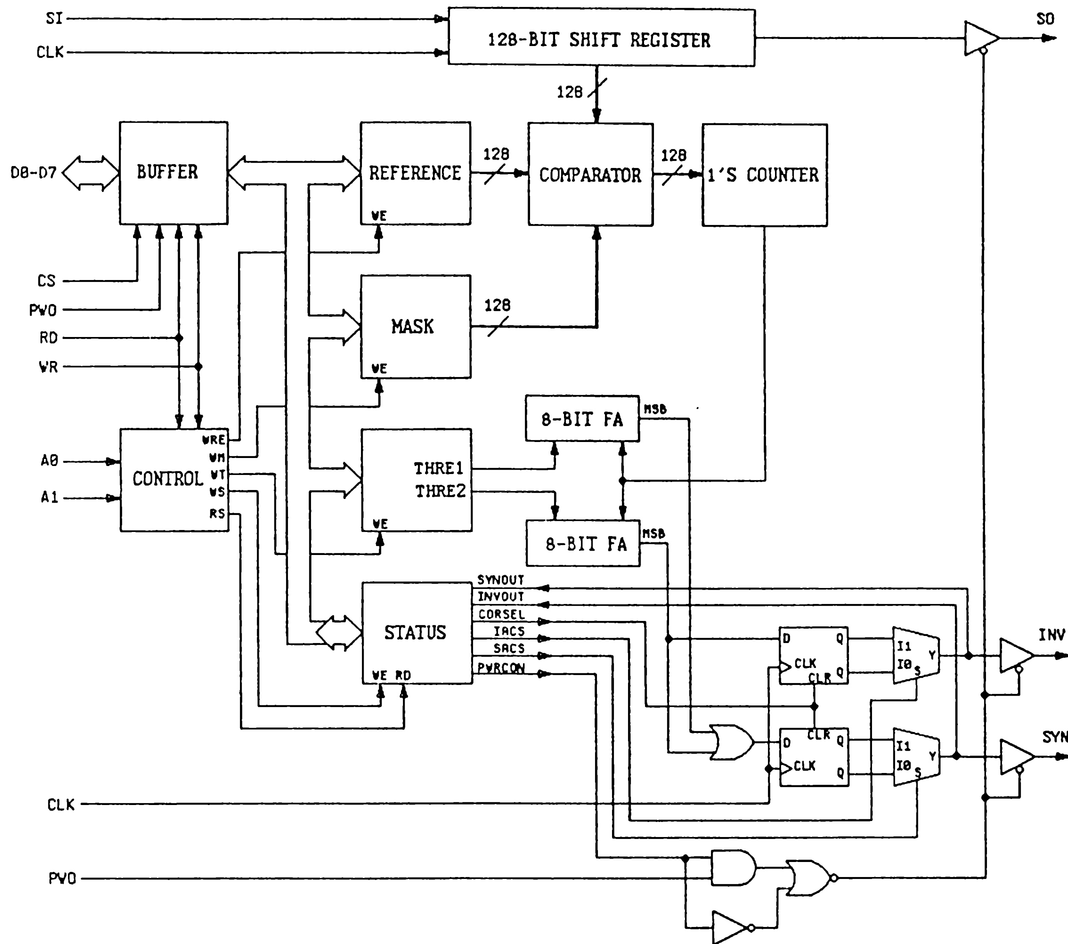


Figure 2.2: Block diagram of the correlator in the beginning.

Due to this cascade connection property, 8 pins were added to be able to transfer the intermediate 128-bit correlation score at the 1's counter output from one chip to the other. Later, PRBS generation was made possible by forming a synchronous feedback loop from the 1's counter output to the serial input of the 128-bit shift register. In addition, some other useful features were added such as reading the 1's counter output through 8-bit data bus and loading the 128-bit shift register using data bus in 16 write cycles of the microprocessor. Also, the first 8-bit of the shift register can be read again through data bus. In parallel with doing these changes, control and status blocks were modified [14].

The most important feature that the first design does not have is the testability. So, aiming to improve the testability of the chip, some design for testability techniques were used in the architecture design. First, the registers were modified to make them easily testable since they can be used to control many other signals going to the inputs of other modules. The 128-bit shift register was easily testable because it has a serial input and a serial output

which enable us to control and observe the register contents. Then comes the reference, mask, threshold, and status registers. These registers can be easily controlled, i.e. they can be loaded directly by using data bus. But, after loading, their contents cannot be observed in any way. Therefore, the scan design technique is used for these registers such that by means of a test signal, they are connected serially to form a single shift register, called the scan path. This scan path has a serial input and output, SCIN and SCOUT, respectively. Thus, the contents of these registers may be easily observed at SCOUT by shifting the bits towards the serial output. The comparator and 1's counter blocks are tested together because there is a 128-bit bus between them and it is not suitable to control and observe this bus. The inputs to the comparator block are controlled indirectly by loading the shift, reference, and mask registers. The 1's counter output can be read from the 8-bit output pins used for cascade connection. Consequently, the design was brought into a position that it is efficiently testable at the expense of minimum additional logic. Actually, details of the modifications to make the design testable are given in a subsequent chapter. A detailed block representation of the circuit after all changes is given in the following paragraphs.

This correlator chip can detect sync words of length up to 128-bits. That is, the maximum correlation length that can be done using only one chip is 128-bits. However, two chips can be cascaded in a master-slave configuration to increase the length to 256-bits. The sync word can be detected for either inverted or non-inverted input data streams. This means that the sync word can also be detected if every bit of the input data is inverted. The correlator is fully programmable by a microprocessor to set the number of tolerable errors in sync detection and to select the bits of the 128-bit (or 256-bit) data stream to be used in correlation. The latter feature makes the correlator capable for use in detection of distributed sync words and PRBS (Pseudo-Random Binary Sequence) generation.

The correlator chip is basically composed of five registers, a comparator, an integrator, a decision maker, a buffer, and a controller block. It has also an internal 8-bit data bus and a 3-bit address bus. A simplified block diagram of the correlator is shown in Fig.2.3. Buffer is an 8-bit tristate bidirectional buffer controlled by chip select (CS), read (RD), and write (WR) signals. It is bidirectional because the 8-bit data bus is used for both read and write purposes. When the CS is low, it is in the read mode if the RD signal is also low, but it is in the write mode if the WR signal is low and it goes into high impedance state when the chip is not selected (i.e. CS is high).

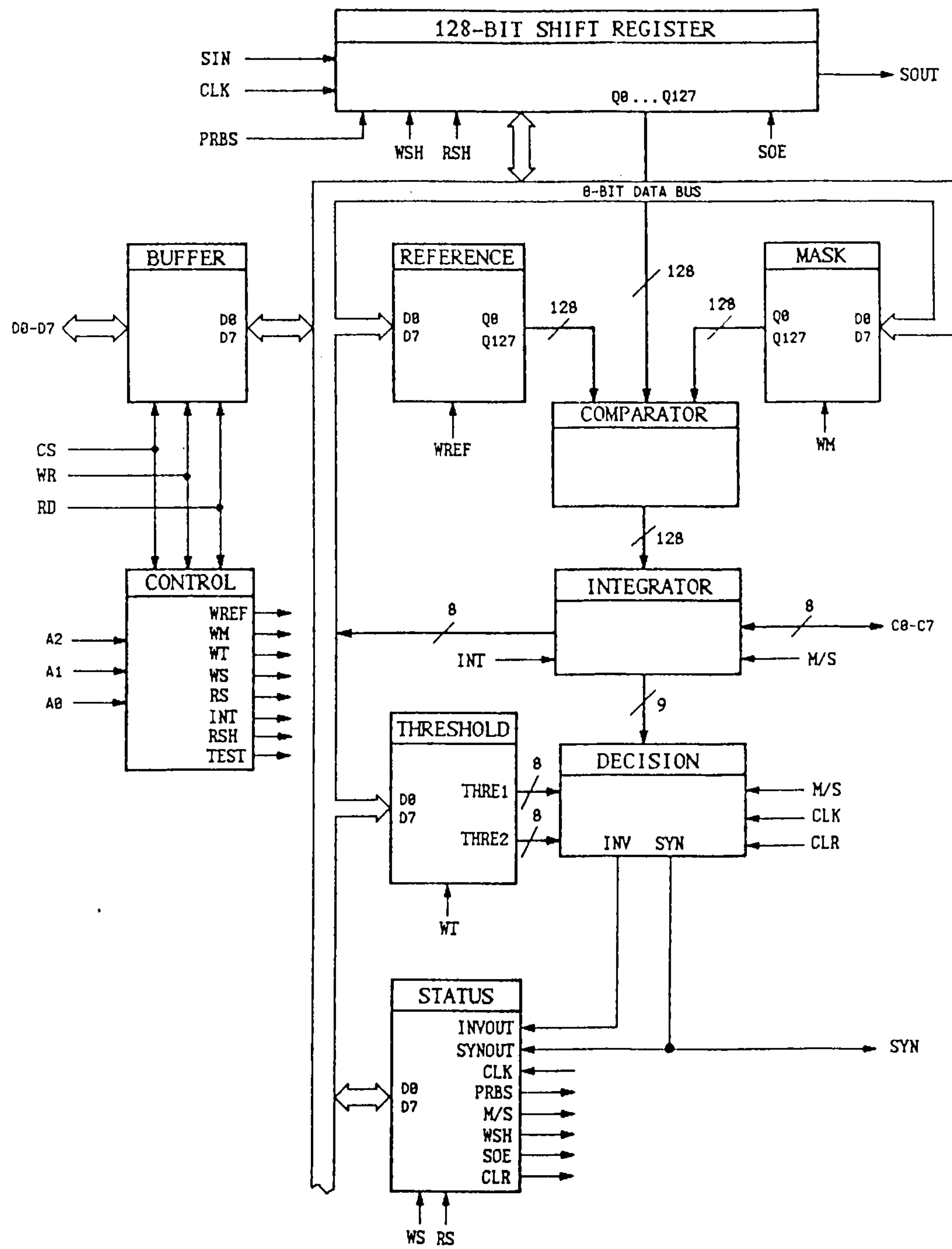


Figure 2.3: Simplified block diagram of the correlator.

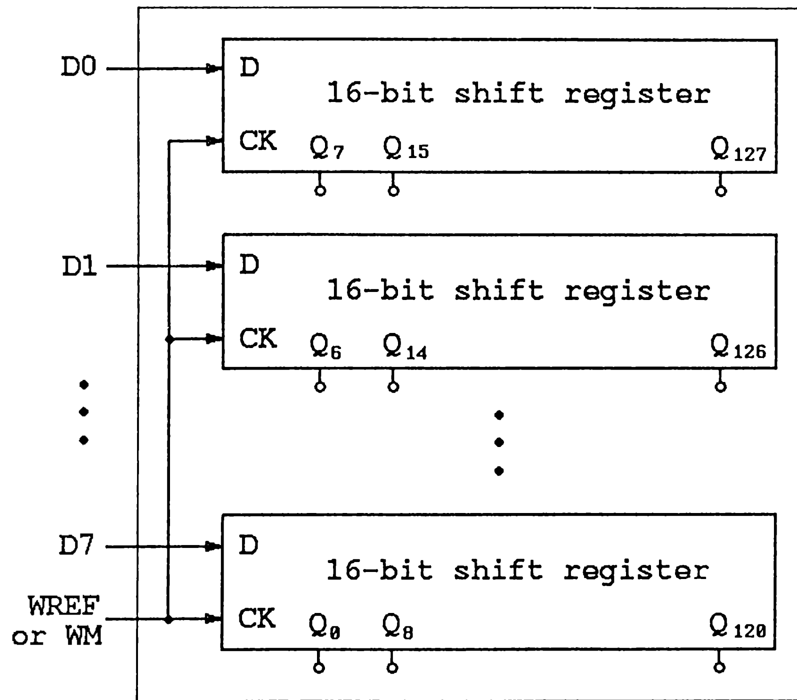


Figure 2.4: Block diagram of the mask and reference registers.

One of the five registers is the serial 128-bit shift register which has a serial input and a serial output. It passes the serial data input (SIN) to serial data output (SOUT) and it holds 128 consecutive data bits for correlation. While doing correlation, in order to ensure synchronization, its clock is supplied by a phase-locked loop (PLL) which sends the serial incoming data stream to the correlator. In addition to serial loading, it can be loaded through the data bus in 16 write cycles of the μP and the first 8-bits beginning from the serial input of it can be read by microprocessor using data bus. It is also used for holding the binary sequence in PRBS generation.

The reference and mask registers which are the 128-bit write-only registers have the same structure as shown in Fig. 2.4. Each of these registers consists of eight 16-bit shift registers and so they can be loaded through the data bus in 16 write cycles of the microprocessor. The reference register must be loaded with the sync word and the 1's in the mask register show the corresponding bits in the 128-bit shift register to be masked. The presence of the mask register allows the detection of distributed sync words and PRBS generation, because it provides the capability of selecting any bit in the 128-bit shift register to be included in the correlation. The clock signals of these registers come from the controller block during write operation. While the data and address are valid at the data and address busses respectively, data is latched at high-to-low transition of WR signal of μP as shown in Fig. 2.5.

The threshold register is very similar to the reference and mask registers in structure and it consists of two 8-bit write-only registers which are named as THRE1 and THRE2. In the same way as the reference and mask registers are loaded, the threshold register can be loaded from the data bus in two write cycles of the microprocessor. During write cycle, the clock for this register is generated by the controller block. THRE1 and THRE2 registers must be loaded with the error tolerances in order to produce sync (SYN) and inverted sync (INV) pulses respectively.

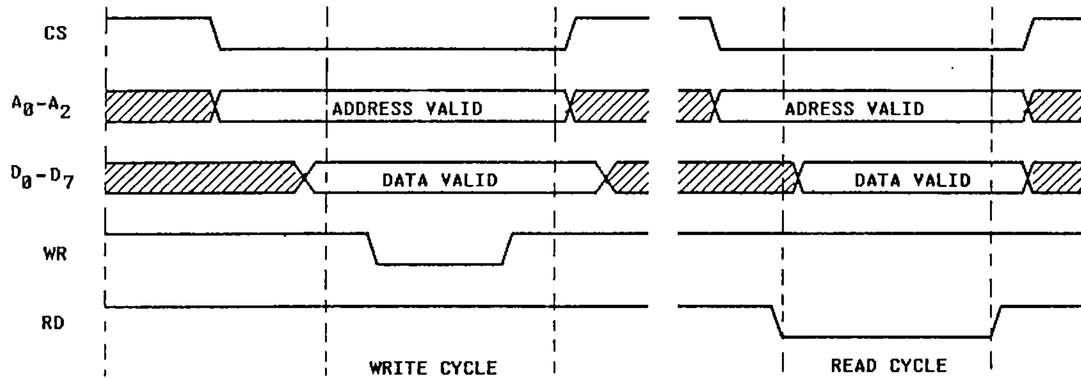


Figure 2.5: Timing diagram of the write and read cycles.

The status register has three read-only and five write-only bits. This register holds some status bits determining the operation modes of the correlator. The read-only bits are SYNOUT, INVOUT, and CLK which show the states of SYN and INV outputs and clock (CLK) signal respectively. When the address of status register is valid on the address bus (A0-A2), read-only bits are seen at the data bus after high-to-low transition of the read (RD) signal as shown in Fig. 2.5 and by this way the microprocessor can check the states of these bits. Actually the correlator chip has a sync (SYN) output pin which informs the μP by sending an interrupt signal when the sync word is detected. Thus the microprocessor has two alternatives to learn the state of SYN output. There is no output pin for the INV output and so the only way to learn the state of INV output is to read the status register. Every time a sync word is detected, the μP checks the state of INV output immediately in order to determine whether the detected sync word is inverted or not. Both of the SYN and INV outputs are activated if an inverted sync word is detected. On the other hand, for a non-inverted sync word, only the SYN output is activated. In writing to the status register, after the data and address become valid, data is latched when the write (WR) signal goes from high to low as shown in Fig. 2.5. While doing write operation, the read-only bits goes into high impedance state. Since the read and write signals for this register do not become low simultaneously, there is no contention problem.

The write-only bits of status register are PRBS, master-slave (M/S), serial data output enable (SOE), clear (CLR), and write shift register (WSH) bits. PRBS bit is used to control the operation of the chip as correlator or PRBS generator. It is set to "1", if PRBS generation is to be done, otherwise it is set to "0". That is, before starting to generate PRBS, this bit must be made high. M/S bit determines whether the chip is master or slave while doing a 256-bit correlation in cascaded configuration. If it is "1", the chip becomes master, otherwise it becomes slave. For a 128-bit correlation M/S bit is set to "0" and thus the chip behaves as slave. SOE bit is used to disable serial data output (SOUT) and it is active low. So when it is low, SOUT is in high impedance state. CLR bit is used to disable the SYN output and it is also active low. This bit is useful while doing PRBS generation, because in this mode SYN output is not used and so it is better to disable it, in the view of system environment. WSH is an active high bit which is used to load the 128-bit shift register through the data bus. In other words, this bit must be set to high and it must be remain at high state until the shift register is fully loaded . In addition, this bit is necessary in order to synchronize the clock of the shift register with the write signal of the microprocessor. Actually, when the WSH bit is not high, clock is supplied by the phase lock loop to be synchronized with the incoming data. The loading of shift register from data bus is needed in PRBS mode to increase the speed.

The comparator is a simple combinational logic consisting of 128 compare cells each of which compares the bits in the shift and reference registers and produces a "1" at the output if they are equal. But if the corresponding bit in the mask register is "1", the output of the compare cell is "1", irrespective of the state of the shift and reference register bits. The operation of a single compare cell is shown in Table 2.1.

Mask	Reference	Shift	Output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	X	X	1

Table 2.1: Function of a compare cell.

The integrator block takes the 128-bit comparator output and finds the number of matched and masked bits of the shift register. Its block diagram is shown in Fig.2.6. It contains a 1's counter block, two 8-bit tristate unidirectional buffers, an 8-bit adder, and a 9-bit 2-to-1 multiplexer. The 1's counter block is a combinational logic consisting of half-adders and full-adders in an inverse binary tree form. It simply counts the number of 1's in the 128-bit comparator output. The tristate buffers at the output of 1's counter has different purposes and different enable signals where the INT signal is produced by controller block and the M/S signal comes from the status register. The buffer having enable signal INT is used to read the 8-bit output of the 1's counter through the data bus. That is, the microprocessor can read the 1's counter output while doing correlation. This is an important property of the correlator because it provides an immediate result of the correlation to be able to make some decisions before continuing. Additionally, the output of 1's counter gives more detailed information than the SYN pulse about the correlation. In other words, by looking at the 1's counter output, one can calculate the number of matched and masked bits in the 128-bit shift register and can learn the noise level on the incoming data. The other tristate buffer with enable signal M/S is used to send the 1's counter output to the bidirectional pins C0 to C7. This is necessary when two chips are cascaded to make a 256-bit correlation. In this case, the chip operating as slave must send its 1's counter output to the one being master through 8-bit bus C0-C7. So, when M/S bit is low tristate buffer is enabled and 1's counter output is seen at the output pins and when M/S bit is high tristate buffer is disabled and its output is in high impedance state. The 8-bit adder in the integrator block is used to be able to make 256-bit correlation. It adds two 1's counter block outputs in the master and slave chips resulting in a 9-bit number at the output. Then the output of 8-bit adder is multiplexed with the 1's counter output by means of the 9-bit 2 to 1 multiplexer of which the select signal is M/S. The multiplexer output is the output of 1's counter if M/S bit is high otherwise it is the output of 8-bit adder. Thus, by using this multiplexer, desired correlation score depending on the correlation length can be sent to decision maker. Consequently, the intermediate results of the correlation are prepared by the integrator to be used in decision maker or to be read by the microprocessor.

The decision maker is the block which gets the integrator output and the error tolerances kept in the threshold registers, THRE1 and THRE2, and decides whether the sync word is detected or not. Therefore, it has two outputs which are SYN and INV. Its block diagram is shown in Fig. 2.7.

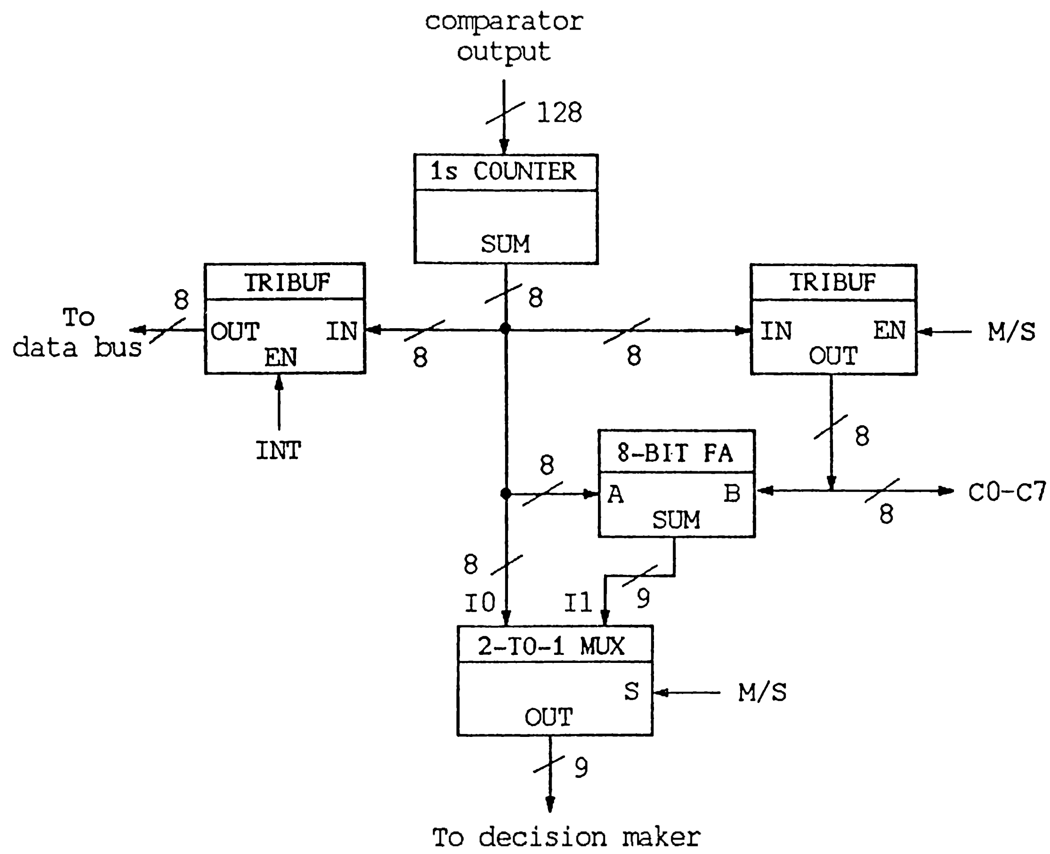


Figure 2.6: Block diagram of the integrator.

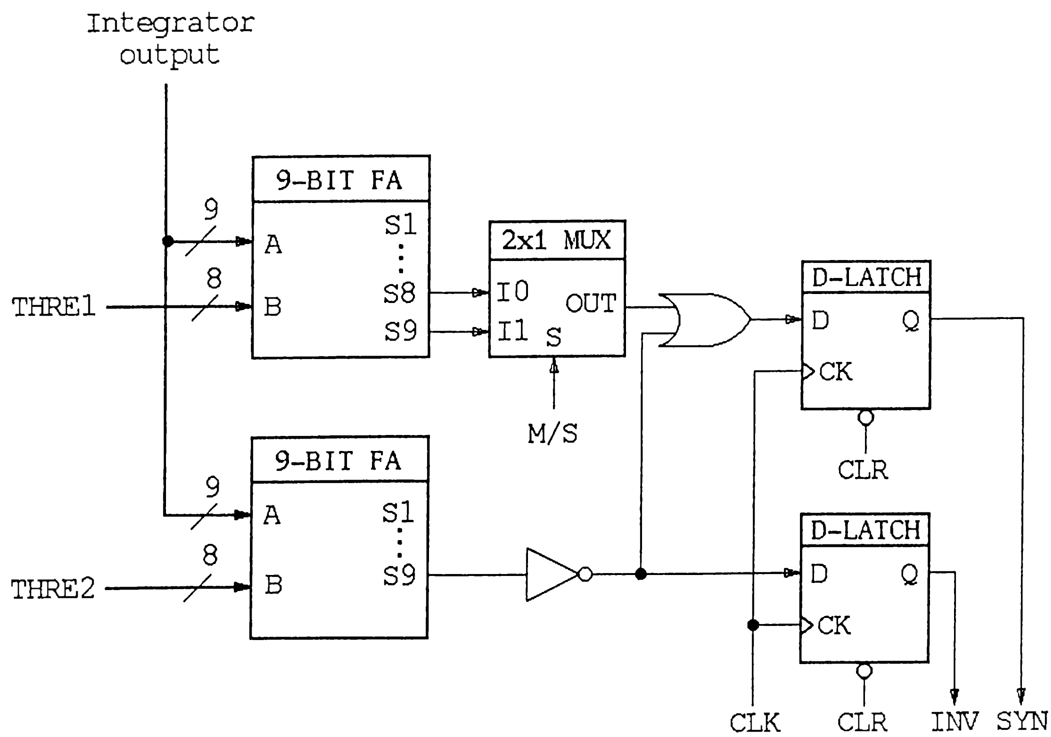


Figure 2.7: Block diagram of the decision maker.

The decision block contains mainly two 9-bit adders, a 2 to 1 multiplexer, and two D-latch with a reset. The 9-bit adders are used to add the integrator output with the error tolerances. As shown in the figure, these adders have only the most significant two sum outputs that are the only ones needed to produce SYN and INV pulses. The two input "or" gate ensures that if INV signal is high, SYN signal is also high because, as mentioned earlier, when an inverted sync word is detected SYN and INV must be both high. Finally, the values of SYN and INV outputs are latched in every low-to-high transition of the CLK signal in two master-slave flip-flops. Thus, they can be read by the microprocessor using the status register. The master-slave flip-flops have the same clock signal with the shift register and so at the beginning of every clock cycle, as a new data enters into the shift register, the SYN and INV signals produced by using the previous data are latched into flip-flops. By means of the reset of flip-flops, the SYN and INV outputs can be reset while generating PRBS where these outputs are not needed.

The controller block is used to produce timing signals for the registers and for the purpose of testing. Besides the test signal which is used in testing, it generates read and write signals for the registers by using the address signals (A0-A2), CS, RD, and WR signals. It decodes the address signals to select the registers or other blocks and produces eight different signals as shown in Table 2.2. Here the X values are used for don't care conditions. While the CS is high, all output signals become inactive, i.e. low, irrespective of all other input signals to the controller.

CS	RD	WR	A2	A1	A0	Signal	Function
1	X	X	X	X	X		All signals inactive
0	1	0	0	0	0	WREF	Write reference register
0	1	0	0	0	1	WM	Write mask register
0	1	0	0	1	0	WT	Write threshold register
0	1	0	0	1	1	WS	Write status register
0	0	1	0	1	1	RS	Read status register
0	0	1	1	0	0	RSH	Read shift register
0	X	X	1	0	1	INT	Integrator selected
0	X	X	1	1	0	TEST	Scan path formed

Table 2.2: Function table of the controller block.

2.3.3 Pin Description

The correlator has 28 pins and its pin diagram is shown in Fig.2.8. Three of them are chip select (CS) input pin, read (RD) and write (WR) signal pins which are all active low and controlled by μP. The other ones are:

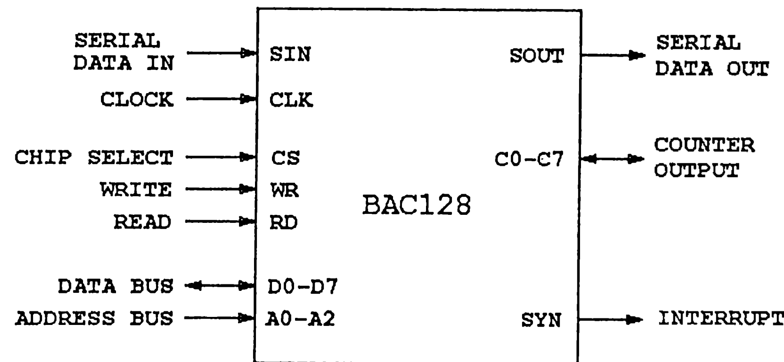


Figure 2.8: Pin diagram of the correlator.

1. D0-D7: Bidirectional, tristate 8-bit data bus. When CS is high it is in high impedance state. Otherwise it is either in the write or read mode.

2. A0-A2: 3-bit address pins in order to select the internal registers and integrator block and to produce a test signal in the chip.

3. CLK: It is the clock generated by digital phase-lock loop (DPLL) such that the incoming data can be taken into the 128-bit shift register.

4. SIN: Serial data input to the correlator. Data at this pin is latched to the 128-bit shift register at the low-to-high transition of the CLK.

5. SOUT: Serial data output from the correlator. At the low-to-high transition of the CLK, the least significant bit of the shift register is seen at this pin during a period of the CLK.

6. C0-C7: Bidirectional, 8-bit integrator input/output bus which is used to perform 256-bit correlation. That is, in cascade connection the 8-bit busses of two chips are connected to each other and the slave chip sends its 1's counter output to the master chip through this 8-bit bus.

7. SYN: Sync output pin. When a sync word is detected it makes a transition from low to high and remains at high state for one CLK cycle.

In addition to these, there are two pins for power (VDD) and ground (GND) connections.

2.4 Modes of Operation

As mentioned previously, the chip can function in two modes which are the correlation and PRBS generation modes. Selection of these modes is done as shown in Table 2.3. The operation mode of the chip is determined by the PRBS bit in the status register. In order to switch from one mode to another the state of this bit must be altered. Although the same circuitry is used in both modes, the chip performs totally different functions in two modes. In the following two subsections operation of the chip in two modes is presented in detail.

PRBS	M/S	Operation mode
0	0	128-bit correlator(SLAVE)
0	1	256-bit correlator(MASTER)
1	X	PRBS generator

Table 2.3: Selecting modes of the correlator.

2.4.1 Correlator Mode

In order to operate the chip as a correlator, the PRBS bit in status register must be set to low. In addition to PRBS bit, also M/S bit in status register must be set to a value to determine the length of the correlation which is 128-bit or 256-bit. In this mode the chip perform the basic function:

$$CORRELATION = \sum_{i=1}^{N} OR(XNOR(D_i, R_i), M_i)$$

where the D_i are the current contents of the 128-bit shift register which holds the data, the R_i are the corresponding values in the reference register, and M_i are the corresponding latched masking values. The mask function, implemented with one OR gate per bit, tells the chip to include only a specified subset of 128-bit shift register bits in the final correlation score. N is the total number of shift register taps which is equal to 128 or 256 depending on the length of the correlation.

In the correlator mode, after setting the PRBS bit in the status register to low, it is determined that whether a 128-bit or 256-bit correlation is to be done and then the M/S bit in the status register is set to low or high according to the correlation length. If a 128-bit correlation is done, this bit is set to low and we call such a chip as slave. On the other hand, if a 256-bit correlation is done, two chips are cascaded and one of them is called as master whereas the other one is called as slave. As opposed to slave, in the master chip, the M/S bit in status register is set to high and this means that the final 256-bit correlation result is to be produced by the master chip. The only difference between the functions of these two types of chips called master and slave, although they have the same hardware, is that one of them produces a 256-bit correlation score while the other produces a 128-bit correlation score. This is an important property of the correlator chip that by controlling a bit in the status register, the correlation length can be doubled. In other words, it is sufficient to change the state of M/S bit in order to determine the correlation length.

In 128-bit correlation, the reference register is loaded with the sync word and the mask register is loaded with the masking data in 16 write cycles of the μP. The sync word is chosen by the user and it can be changed at any time only by loading the reference register with the new sync word. This can be done by putting the reference register address and the data on the 3-bit address bus and 8-bit data bus for 16 write cycles of the μP. The mask register content determines the subset of the 128-bit data in the shift register to be included in the correlation. That is, the "1" s in the mask register means that the corresponding bits in the shift register are considered as "don't care". This is also an important property of the correlator chip because by having this feature the chip can detect the distributed sync words which may have data bits among the sync bits. This means that, if it is desired, sync words of length less than 128 can also be detected. These capabilities makes the correlator flexible about the sync word length and the distribution of its bits. Now, after the reference and mask registers are loaded, there comes the loading of the threshold register consisting of two 8-bit registers, THRE1 and THRE2. THRE1 is related with the noninverted sync detection and THRE2 is related with the inverted sync detection. THRE1 register must be loaded by the maximum number of errors that can be tolerated and as it is clearly understood, this register content can have values between 0 and 128. THRE2 must be loaded by 255 (all 1's) minus (maximum number of tolerable errors + number of 1's in the mask register). This is equal to the 1's complement of the binary number which is equal to the tolerable errors plus number of

masked bits. Here, this number in the register THRE2 is to be used for subtracting the number of error tolerance from the number of erroneous bits in the shift register. If the result of the subtraction becomes positive, this means that the number of erroneous bits is greater than the number of error tolerance. But if the result of the subtraction is negative then the number of erroneous bits is smaller than the number of error tolerance. Consequently, an inverted sync word is detected when this subtraction results in a negative number. Actually, by using this method, a Hamming distance between input data and reference data is calculated and if this Hamming distance is below a given threshold, it is decided that a sync word is detected.

Now, the correlation can begin with the first bit of the incoming data entering to the 128-bit shift register from serial input, SIN. Data is latched to the first master-slave flip-flop of the shift register at the high level of the clock. Since master-slave flip-flops are used in the shift register, the new content of it for calculating the correlation score becomes available after high-to-low transition of the clock. After 128 clock cycles the first bit of the incoming data appears at the serial output SOUT and if a sync word is detected, SYN and/or INV outputs become high at the 129'th pulse as shown in Fig. 2.9. As it is clear, correlation is done for every clock period and calculations start after the falling edge of the clock and finish before the next rising edge of the clock. Thus, all calculations must be done while the clock is low and it must be decided whether the sync word is detected or not before the rising edge of the clock. Therefore, the duration between the falling edge of the clock and the time at which the decision maker output becomes stable determines the speed of the correlator and frequency of the clock.

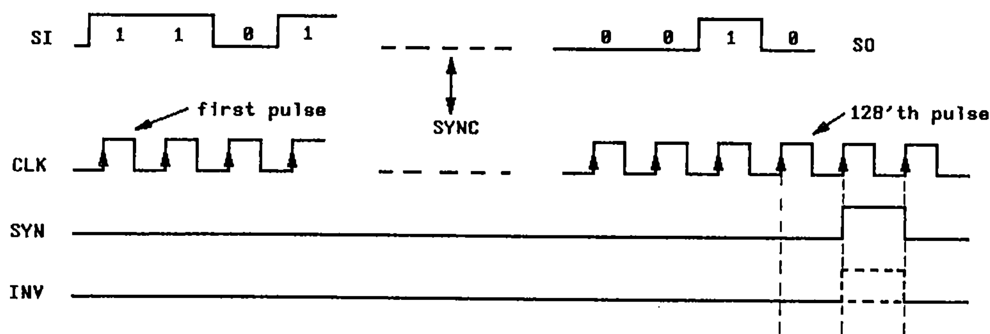


Figure 2.9: Timing diagram of the sync detection.

When the outputs of the shift register become valid, meaning that the comparator inputs are ready, comparator block checks whether the shift and reference register bits are the same or not and also looks for the mask register bits in order to understand which shift register bits are to be masked (i.e.

don't care). Then it produces a "1" at the output for the matched or masked shift register bits and produces "0" for others. The comparator block has a 128-bit output which is connected directly to input of the 1's counter in the integrator block. 1's counter finds the number of comparator output bits which are "1" and produces an 8-bit number having a value between 0 and 128. In 128-bit correlation this is the output of the integrator block which is fed into the decision maker. In decision block, this 8-bit number is summed with both threshold register outputs, THRE1 and THRE2. This configuration is shown in Fig. 2.10. In this way, the 1's counter output is summed with the error tolerances held in the threshold register.

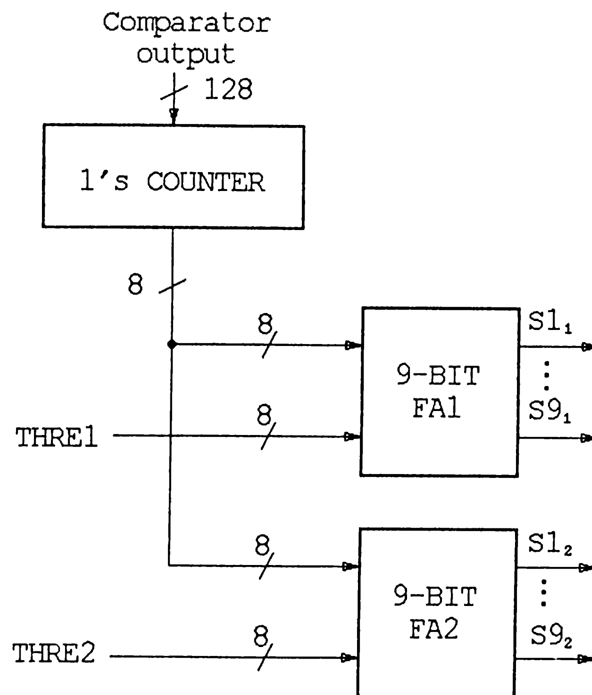


Figure 2.10: 128-bit correlation scheme.

We are interested only in the most significant output bits of the 9-bit adders in order to decide about the detection of a sync or inverted sync word as shown in Table 2.4. The output of that 9-bit adder which adds the 1's counter output with the output of THRE1 can have values between 0 and 256. If the output of this adder is between 128 and 256, this means that the number of matched and masked bits plus the tolerable erroneous bits in the shift register is greater than or equal to 128. In other words, if the 8'th bit of the 9-bit adder output is 1, it means that a sync word is detected. For deciding about the inverted sync word detection, other 9-bit adder which adds the 1's counter output with the output of THRE2 is used. If the most significant bit is "1" meaning that the output of the adder is greater than 255, there is more erroneous bits in the shift register than the error tolerance. But if the

most significant bit is "0", this shows that an inverted sync word is detected. So, when a sufficient number of bits come inverted with respect to bits in the reference register, an inverted sync word is detected. Actually, in the case of inverted sync word detection, both the sync (SYN) and inverted sync (INV) outputs of the decision maker becomes high. In 256-bit correlation,

$S8_1$	$S9_2$	SYN	INV
0	1	0	0
1	1	1	0
0	0	1	1

Table 2.4: Decision table for the sync detection.

two 128-bit correlator chips are cascaded as shown in the Fig. 2.11. The reference, mask, and threshold registers in both chips are similarly loaded as in the 128-bit correlation. But this time, the contents of THRE1 registers can have values between 0 and 255 which corresponds to the maximum number of error tolerance. In the same way, the THRE2 registers are loaded with the 1's complement of the binary number which is equal to the number of tolerable errors plus the number of masked bits.

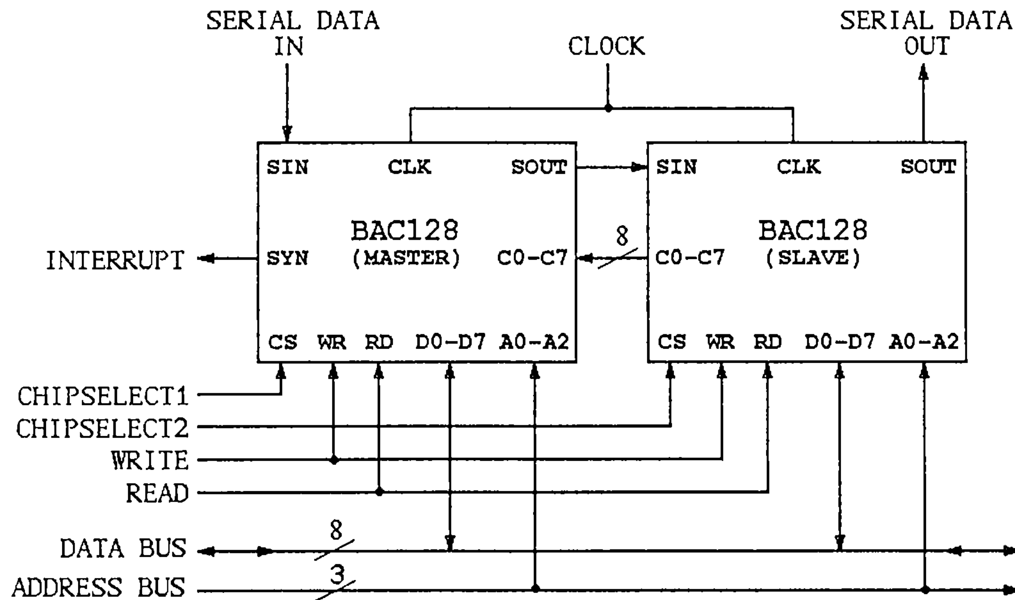


Figure 2.11: 256-bit correlator.

We call the first chip as master since it calculates the final correlation score and produces a sync pulse if a 256-bit sync word is detected. The incoming data first enters to the serial input of the master and after 128 clock cycles

it is seen at the output of this chip. Then the data enters into the slave from its serial input and at the end of the 256'th clock cycle it appears at the serial output of this chip. At the falling edge of the 256'th clock pulse, the comparison between input data and reference data is started in the chips and both chips produce their intermediate results which correspond to 128-bit correlation score of each chip. Later, the slave sends its result to the master through the 8-bit bus $C0 - C7$. Then the master sums the result coming from the slave and its result and sends the summation output to the decision maker block as shown in Fig. 2.12. As in the 128-bit correlation, additions with the THRE1 and THRE2 outputs are performed by using two 9-bit adders. The decision for the 256-bit sync or inverted sync detection is made as shown in Table 2.5. It is clear that, this decision is made only in the master chip so the SYN output of only this chip is connected to the interrupt pin of the microprocessor. In fact, also the slave chip makes some calculations and decisions and it may produce a sync pulse too. However, its SYN output pin is not connected to anywhere and therefore its decision is not taken into consideration in the 256-bit correlation.

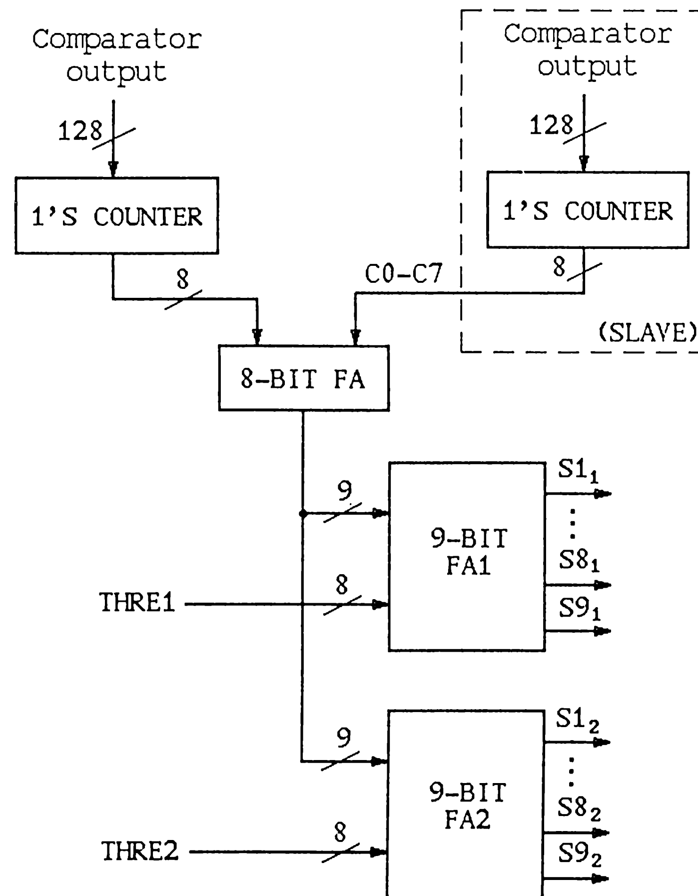


Figure 2.12: 256-bit correlation scheme in the master chip.

$S9_1$	$S9_2$	SYN	INV
0	1	0	0
1	1	1	0
0	0	1	1

Table 2.5: Decision table for 256-bit sync detection.

2.4.2 PRBS Generator Mode

The configuration for operating the chip as PRBS generator is shown in the Fig. 2.13. In fact, this scheme realizes a linear feedback shift register used in PRBS generation. Here the 128-bit shift register is loaded with an initial state chosen by the user and at every clock cycle its contents changes randomly. The mask register is used to select the necessary taps from the shift register outputs. In order to do this, the mask register bits corresponding to the selected taps are loaded with zero and all other bits are loaded with one.

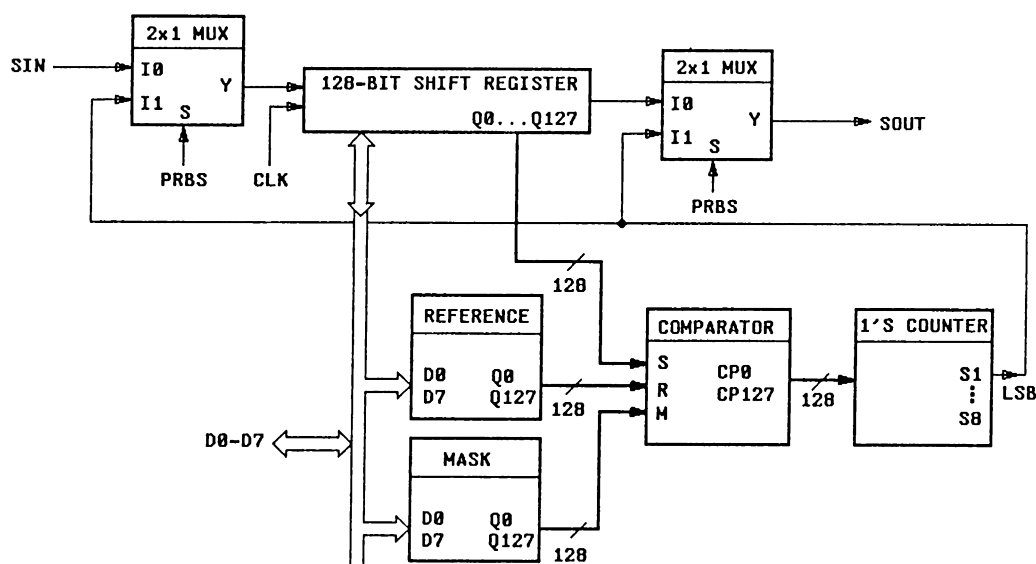


Figure 2.13: The configuration for PRBS generator.

Thus selected taps of the shift register appear at the corresponding comparator output bits while other comparator outputs are 1. Then the 1's counter in the integrator block counts the number of 1's at the comparator outputs. The 1's counter behaves as a huge EX-OR gate with 128 inputs because we are interested in only the least significant bit (LSB) of its output. That is, if there is an even number of 1's at the comparator output, LSB will

be 0 but if there is an odd number of 1's then LSB will be 1. In this way, the selected taps in the shift register would be EX-OR'ed and LSB representing the EX-OR output is fed back to the serial input of the shift register by using a multiplexer which is controlled by PRBS status bit. Consequently, a new pseudo-random bit enters to the shift register at every clock cycle and at the same time this bit can also be seen at SOUT. In other words, PRBS generator output is taken from SOUT because we don't have to wait for a random bit propagating from serial input through shift register to the serial output.

In this mode, the generated bit sequence will have a nearly random distribution if the tap points are suitably chosen, because the length of the shift register is quite large. The sequence repeats itself after a large number of clock cycles. But here the important thing is the initial state of the shift register. If the initial state is known, the state that the shift register will have after a certain number of clock cycles can be predicted. Due to these properties of PRBS, it can be used for sync word generation, i.e. a portion of this sequence can be chosen as a sync word.

3. THE LOGIC AND CIRCUIT DESIGN

In this chapter the details of the logic design are given and some important points about the circuit and layout designs are also presented. Actually, for the details of the circuit and layout designs we address the reader to [6].

3.1 The Logic Design

After an architecture plan has been made which has resulted in a block diagram of the chip and the description of functions and specifications of the blocks, interior of each block has been designed in logic level carefully so that it would be consistent with the architecture design. In other words, the function of each block and the communications among the blocks have been determined in the architecture design and the logic design of the blocks has been carried out with respect to these specifications trying to use minimum number of logic gates [7,8,15]. In addition, a testability analysis has been done manually and according to the result of this analysis some modifications with minimum additional logic have been performed which are presented in the next chapter. By the way, all input-output connections of the blocks are shown in Fig. 2.3.

Buffer is an 8-bit tristate bidirectional buffer which is controlled by CS, RD, and WR signals of the μP as shown in Fig. 3.1. It is called bidirectional because it can be used for both read and write operations. Its outputs go to the high-impedance state when the CS is high meaning that the chip is not selected and also when the RD and WR are both high. The function table of the buffer module is given in the Table 3.1. It is clear that, while the CS is low if the WR signal becomes low, the data can be written to the chip but if the RD signal becomes low then the data can be read from the chip. In addition, inserting a simple NAND gate to the logic, data collision is

prevented when both of the RD and WR signals become low simultaneously. In such a case (i.e., $CS = 0, WR = 0, RD = 0$) the WR signal appears to be dominant with respect to RD signal and thus a write operation is performed.

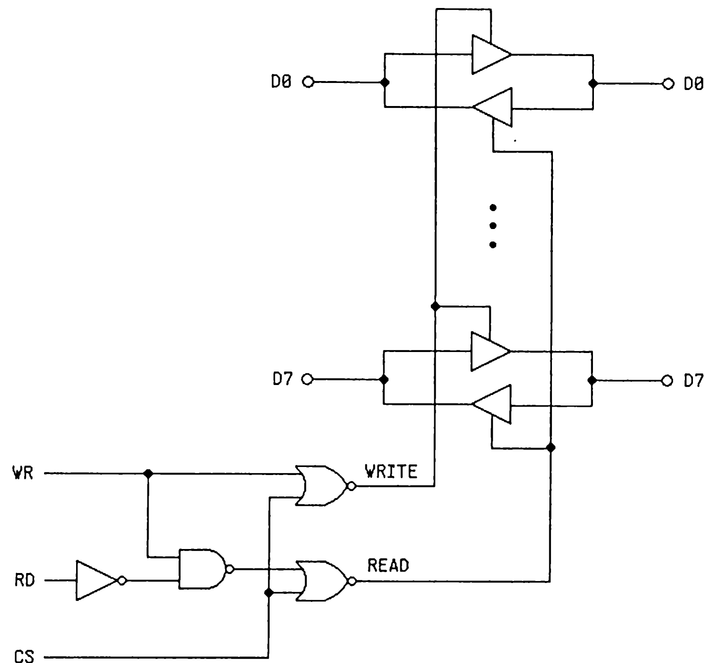


Figure 3.1: Logic diagram of the 8-bit tristate bidirectional buffer.

CS	WR	RD	WRITE	READ	FUNCTION
1	X	X	0	0	HIGH-IMPEDANCE
0	0	X	1	0	WRITE
0	1	0	0	1	READ
0	1	1	0	0	HIGH-IMPEDANCE

Table 3.1: Function table of the buffer.

The controller is an important module which produces the clocks for the registers and the enable signal for integrator as well as a test signal. Its logic diagram is shown in Fig. 3.2. Note that the address signals entering to the controller come inverted because we use inverting input pads for these pins on the chip. The pads are the layout blocks connected to pins through which the chip communicates with the outside world. As it is seen, the controller decodes the address bits in order to generate its output signals to be used in writing into the reference, mask, threshold, and status registers as well as in reading the status register, shift register, and integrator output.

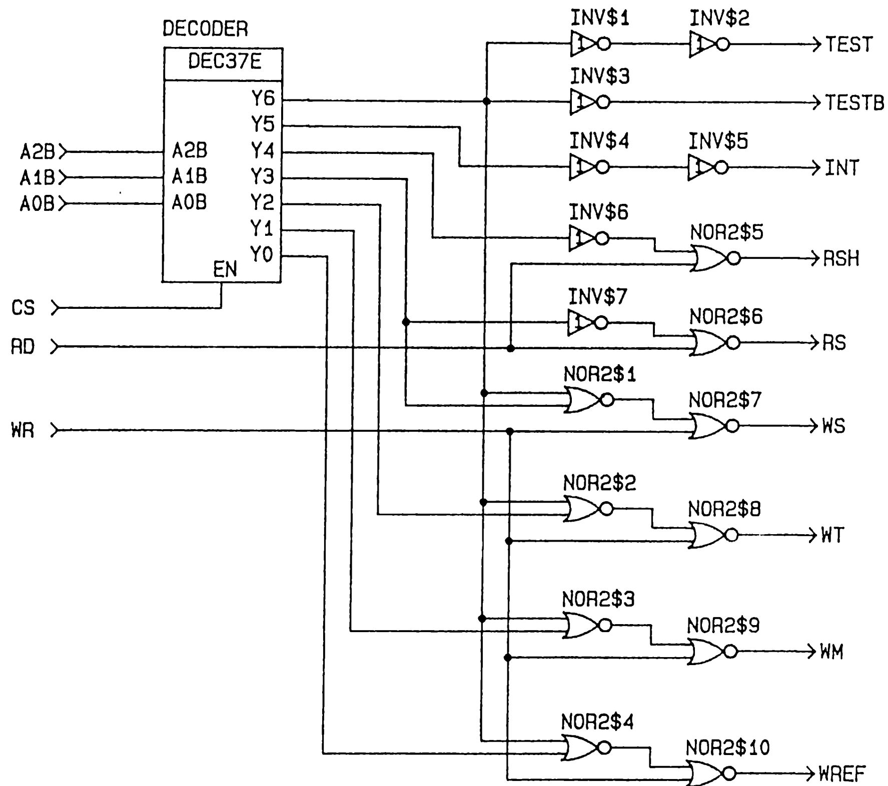


Figure 3.2: Logic diagram of the controller module.

The logic diagram of the decoder is shown in Fig. 3.3. It is a 3-to-7 decoder since the combination in which all address lines are high is not used. In the controller, CS signal is used to enable or disable the decoder outputs. The inverters used for TEST and INT outputs serve as buffers due to fan-out considerations. These outputs are independent of the RD and WR signals. TEST output is used only in testing of the registers and so when this output is high, register clocks behave as a single clock (i.e. they change their states simultaneously).

The status register has been designed as shown in Fig. 3.4. It holds some status bits necessary for changing operation of the chip. It contains 5 master-slave flip-flops and 3 tristate buffers whose enable signals (RS) are supplied by the controller. The flip-flops are loaded from the data bus in one write cycle of the μP. The purpose of using tristate buffers is to check the state of SYN and INV outputs of the decision maker by means of the data bus. To do this, μP must also check the state of CLK because SYN and INV outputs become stable while the CLK is high. Thus the first three bits, $D0 - D2$, of the data bus are used for both reading and writing. Here data collision never occurs since WS and RS are obtained from the μP's write and read signals and these signals do not become active together in any time.

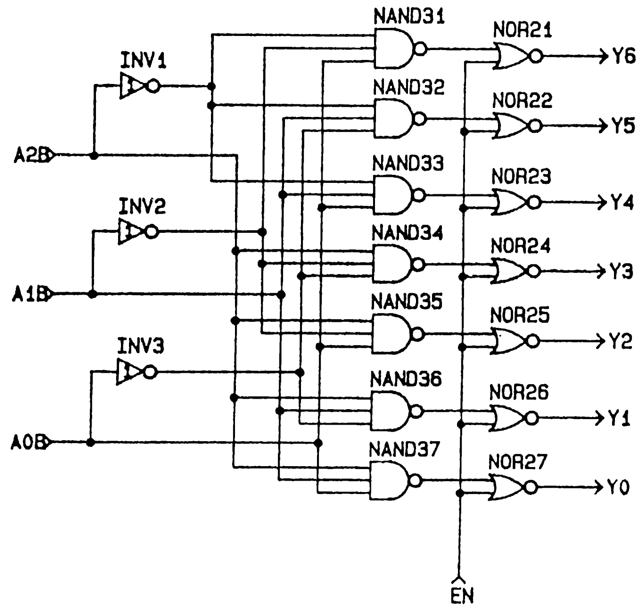


Figure 3.3: Decoder circuit used in the controller.

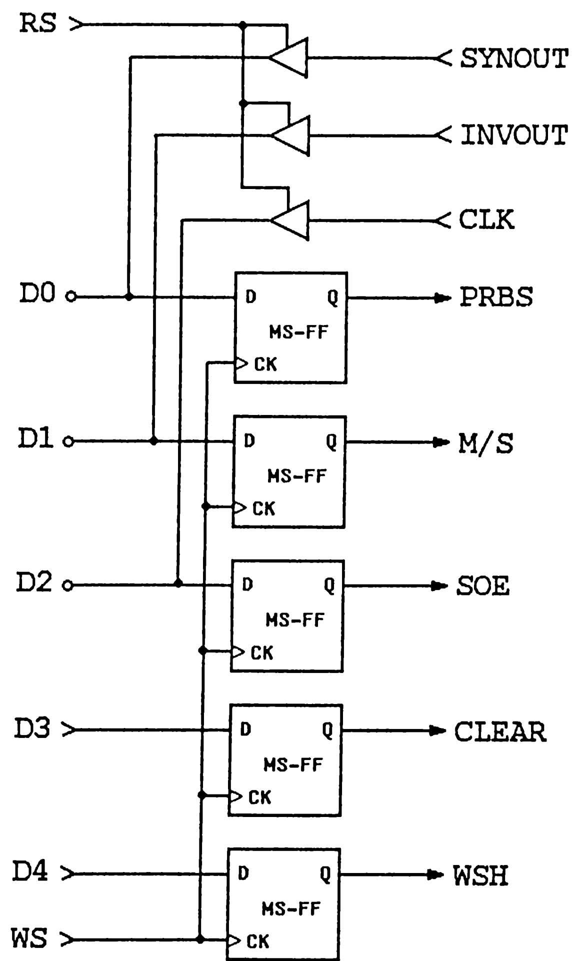


Figure 3.4: Logic diagram of the status register.

Here it is useful to mention about the master-slave flip-flop because a total of 405 master-slave flip-flops are used throughout the design. Its schematic is given in Fig. 3.5. It consists of four tristate inverters and two ordinary inverters. The tristate inverters can also be called as clocked inverters because when the clock becomes active their outputs are valid otherwise they drive their outputs into a high-impedance state. As it is seen, a clock called PHI and its inverse PHIB are used as clocks of the master and slave flip-flops, respectively. If the PHI is high, tristate inverters TRINV1 and TRINV4 are enabled and so the new input data is latched to the master while the old data is preserved in the slave since TRINV3 is in high-impedance state. However, if the PHI is low, TRINV2 and TRINV3 are enabled and thus the new data reaches the slave output, Q, and it is also preserved in the master.

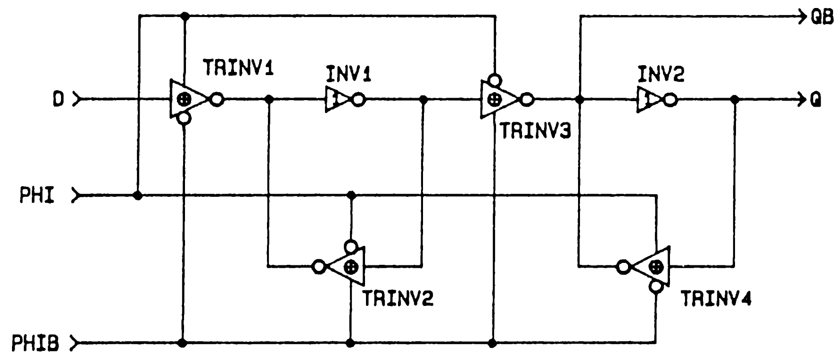


Figure 3.5: Master-slave flip-flop.

One of the five registers, the 128-bit shift register has been designed as shown in Fig. 3.6. It has a serial input (SIN) and a serial output (SOUT). In addition to 128 master-slave flip-flops, it contains 11 multiplexers, 8 inverters, and 8 tristate inverters. In the correlator mode of the chip, the shift register is serially loaded from SIN taking its clock from CLK input. In fact, all multiplexers and inverters are needed for PRBS generation because in PRBS mode, shift register is loaded from data bus instead of SIN to increase the loading speed. For writing into the shift register from the data bus, the data bus lines are multiplexed with the corresponding shift register outputs using eight multiplexers whose select signals are WSH bit held in the status register. Also the clock of the shift register is synchronized with the write signal of the μP instead of the CLK signal using a multiplexer which is controlled by the WSH bit. As well as writing from data bus, the first eight bits of the register can be read through the data bus using tristate inverters shown in Fig. 3.6. As mentioned in the subsection 2.4.2, the least significant bit (LSB) of the 1's counter output is replaced with SIN and SOUT of the shift register during PRBS generation.

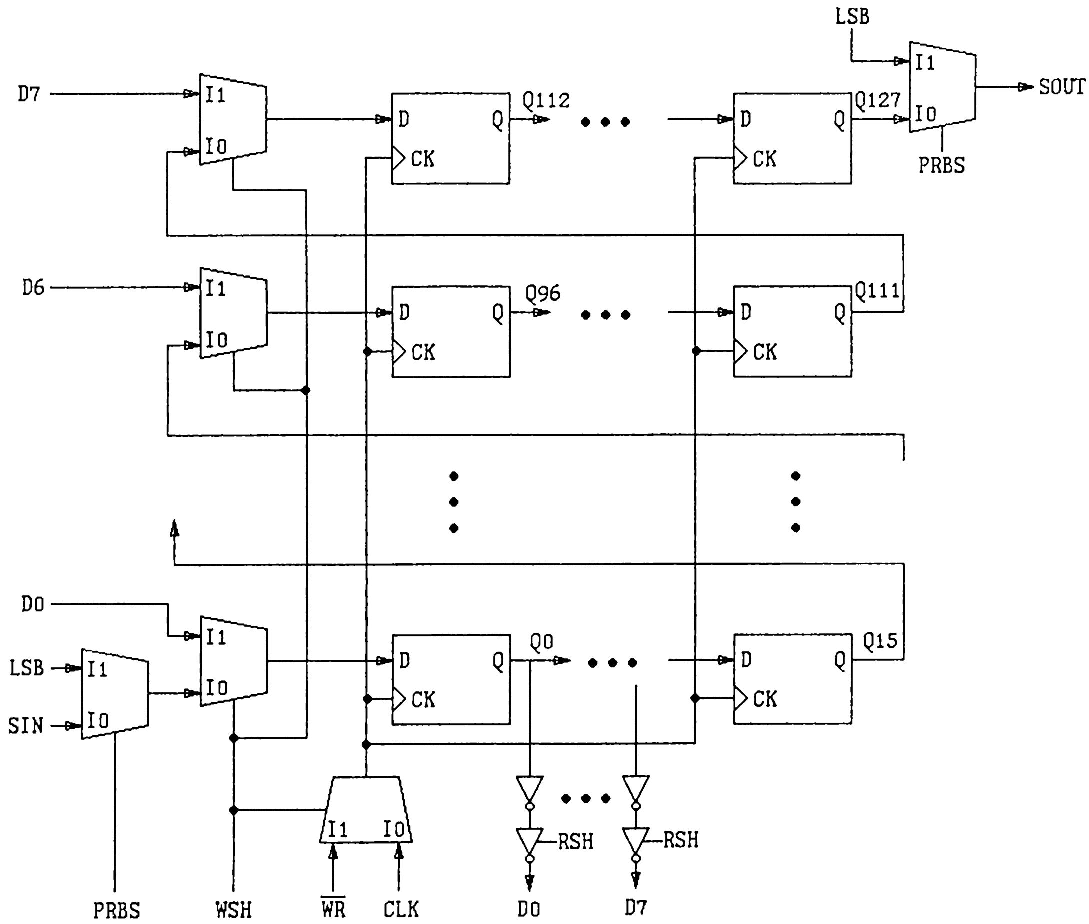


Figure 3.6: Schematic of the 128-bit shift register.

The reference and mask registers are exactly similar to each other. Their logic diagram is shown in Fig. 3.7. They contain 128 master-slave flip-flops grouped as 16-bit shift registers against data bus. All flip-flop outputs of these two registers go to the comparator block together with the 128-bit shift register outputs. Their clocks, WREF and WM, are supplied by the controller. The threshold register is very similar to reference and mask registers but it consists of 16 master-slave flip-flops grouped as 2-bit shift registers. Its logic diagram is given in Fig. 3.8.

The comparator block is a simple combinational logic and so its design is rather easy than the other blocks. It has three 128-bit input busses and a 128-bit output bus. The logic diagram of the comparator is shown in the Fig. 3.9. It simply consists of two-input EX-NOR and OR gates. Here EX-NOR gates check the match or mismatch between the shift and reference register bits. If these bits are same the EX-NOR output will be 1 otherwise it will be 0. The OR gates are used for masking the desired bits such that if the mask register bit is 1, the comparator output is 1 irrespective of the corresponding reference and shift register bits.

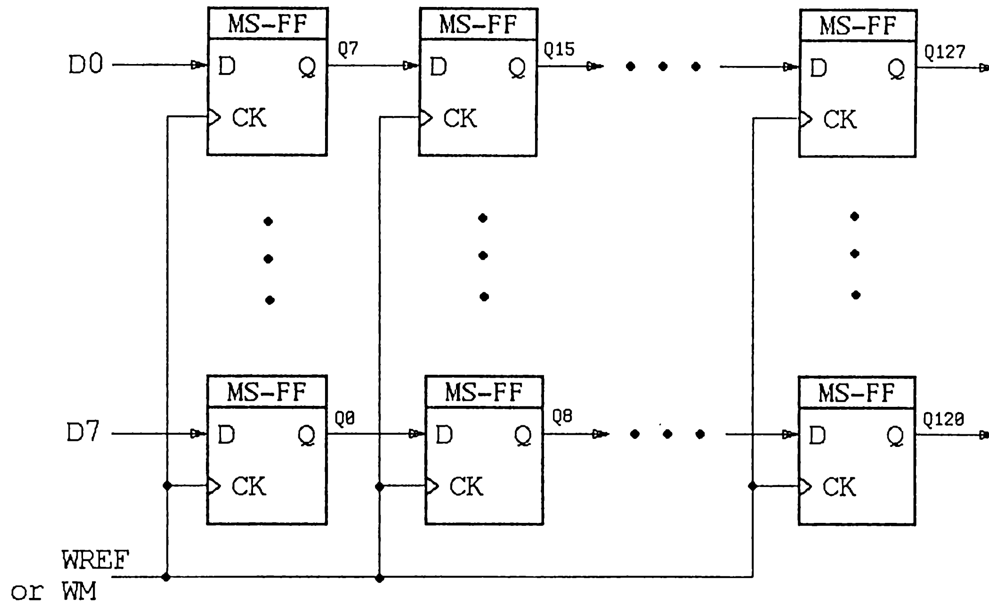


Figure 3.7: Logic diagram of the reference and mask registers.

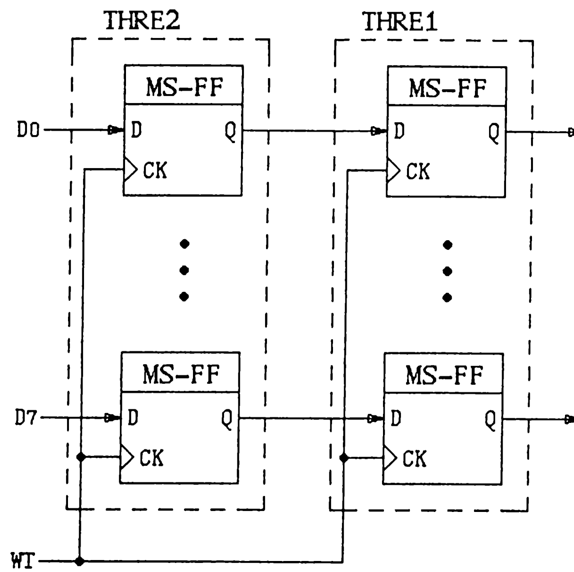


Figure 3.8: Logic diagram of the threshold register.

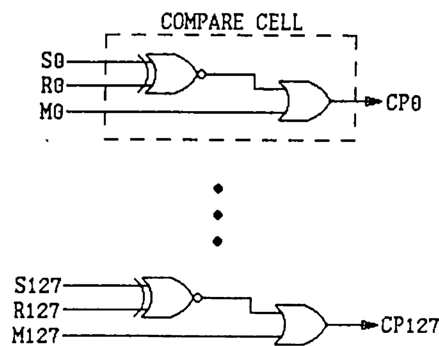


Figure 3.9: Logic diagram of the comparator.

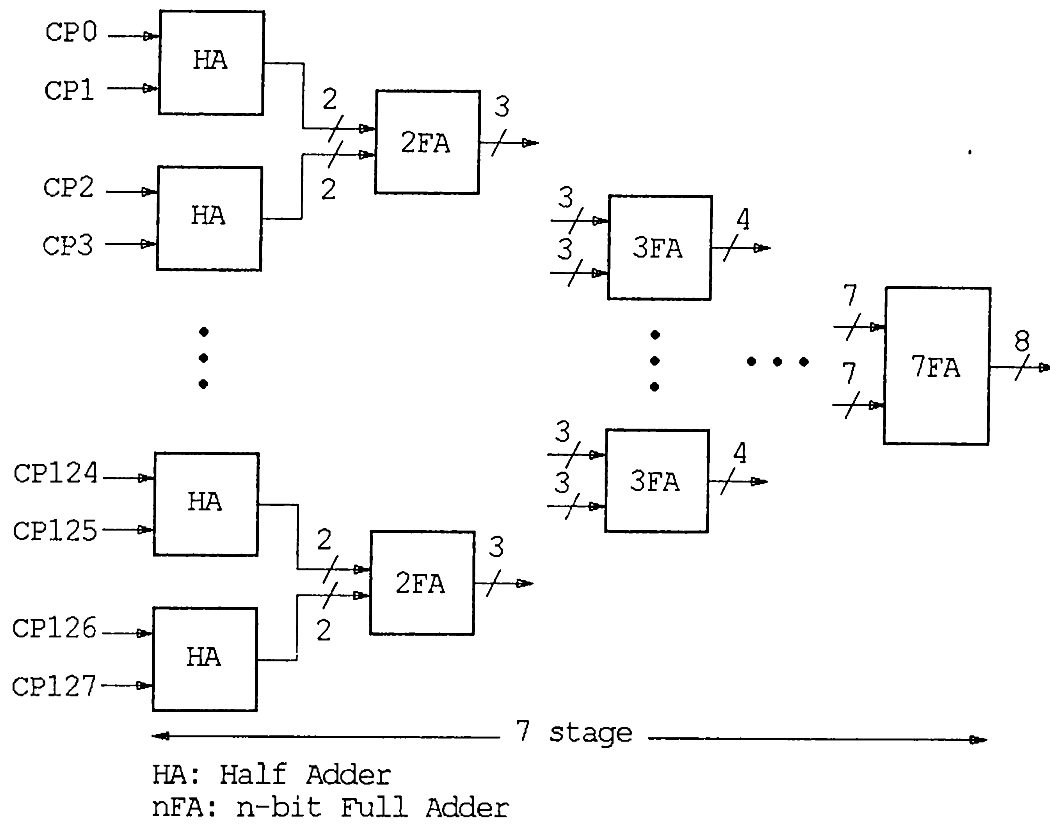


Figure 3.10: Logic diagram of the 1's counter.

The 1's counter in the integrator block is a combinational logic consisting of half-adders and full-adders as shown in Fig. 3.10. It is used to find the number of 1's at the 128-bit comparator output. First, 64 half-adders sum the one-bit numbers. Then 32 two-bit full-adders sum the half-adder outputs and two-bit full-adder outputs are summed by 16 three-bit full-adders. It continues like this up to seven-bit full-adder in an inverse binary tree structure. The 8-bit output of the 1's counter can have values between 0 and 128. Each of the n-bit full-adders used in the 1's counter has been designed by using simple half-adders and full-adders as shown in Fig. 3.11. Half-adders are used in the places where there is no carry input. As a result, a total of 127 half-adders and 120 full-adders are used in the 1's counter.

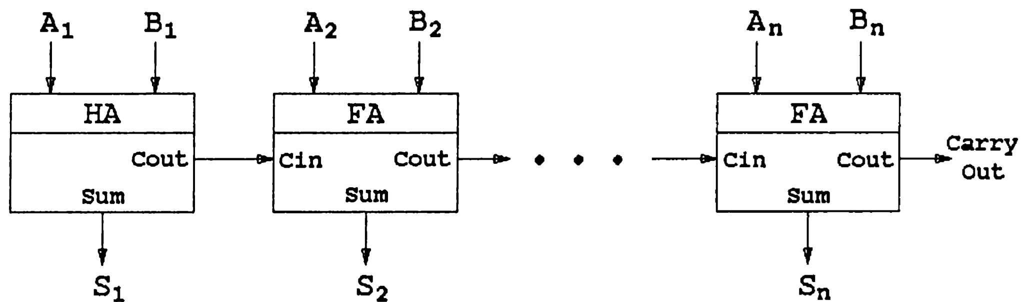


Figure 3.11: The circuit of an n-bit full-adder.

However, the 9-bit full-adders used in decision maker have been designed in a different way because some of the sum outputs of these adders are not used. The circuit of these 9-bit adders is shown in Fig. 3.12. In the places where sum output is not needed, carry generation and/or propagation is sufficient and thus a carry cell is used. Each carry cell generates and/or propagates the carry to the succeeding carry stages according to the truth table shown in Table 3.2. In the first stage of the 9-bit adders, since there is no carry input, the carry cell is replaced with a NAND gate. Since, only if both inputs, A_1 and B_1, are high, a carry is to be generated, this function can be performed by a NAND gate. While using these 9-bit adders in decision maker, we have a 9-bit number and an 8-bit number to be summed. Here the 8-bit number has a zero at the 9'th bit and therefore this input line of the adders is connected to ground.

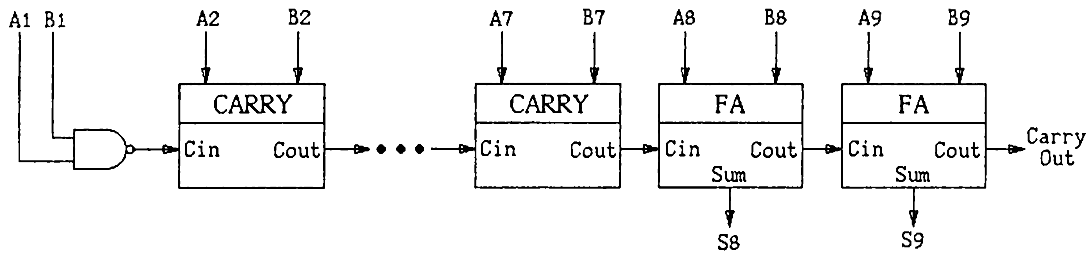


Figure 3.12: The 9-bit full adder circuit.

C_{in}	A	B	C_{out}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 3.2: Truth table of a carry cell.

The logic design of the correlator chip has been performed by using the schematic editor (CASS) of the Silvar-Lisco Standard Design System (SDS) on APOLLO workstations under the operating system AEGIS. Test pattern generation and fault simulation will be done on this circuit.

3.2 The Circuit Design

The last step before the layout design is the circuit design. In fact, it is difficult to separate the circuit and layout design processes because they are intimately meshed. So it is useful to mention about both design phases together. In the circuit design phase, the logic gates or blocks are designed using transistors which are the primitive elements. Here transistors can be thought as simple switches. Now the goal is to use minimum number of transistors in realizing the circuit design. Because the chip area and power consumption increase directly proportional with the transistor count.

There are various types of CMOS logic structures such as complementary CMOS logic, pseudo-nMOS logic, dynamic CMOS logic, clocked CMOS logic, CMOS domino logic, etc.. In designing the correlator chip, the complementary CMOS logic has been used since the others have an increased design and operational complexity and, possibly, decreased circuit stability [7].

In CMOS complementary logic, it is possible to merge several simple gates into a complex gate performing the same function. For example, the single compare cell has been designed in this way. The compare cell consists of an EX-NOR gate and an OR gate as shown in Fig. 3.9. It has been designed in the transistor level as shown in Fig. 3.13. As seen in the figure, it is realized by using 10 transistors. If it is realized by implementing the EX-NOR with simple gates, this number is approximately doubled. In the same way, full-adders, half-adders, and carry generators are designed with transistor counts less than that would be in standard methods. Thus a considerable number of transistors are saved in the whole circuit.

The layout of the chip has been implemented by using 3-micron double-metal double-poly n-well CMOS technology by İ. Enis Ungan [6]. For drawing the layout and extracting the netlist, Magic layout editor has been used on SUN workstations under the 4.3 BSD UNIX operating system. Magic is a hierarchical layout editor with the features of circuit extraction, CIF file generation, and on-line design rule checking [16]. The resultant chip [18] has 28 pins and contains approximately 15,000 transistors occupying a silicon area of $5.6mm \times 5.2mm$.

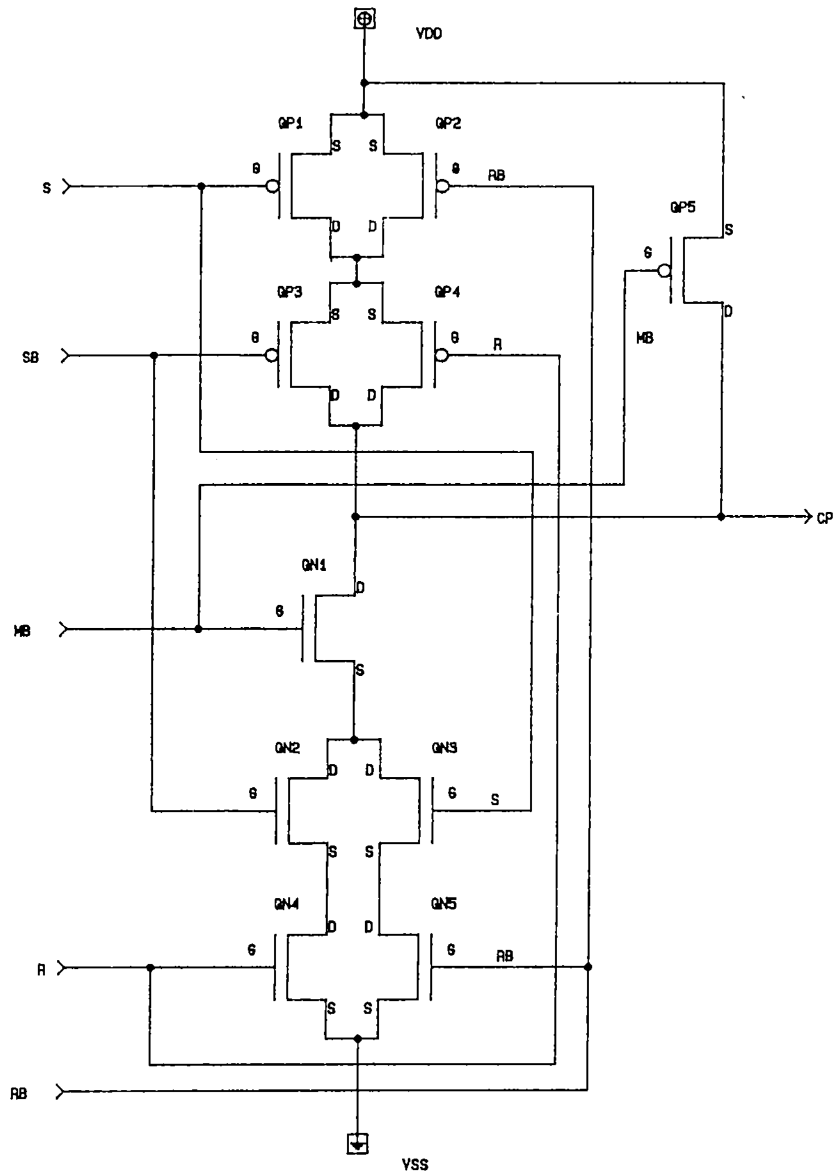


Figure 3.13: Transistor circuit of a compare cell.

4. DESIGN FOR TESTABILITY

VLSI has brought exciting increases in circuit density and performance capability along with the decreases in gate costs. However it has also aggravated the problem of testing. Testing is the problem of determining, in a cost-effective way, whether a chip, module, board, or system has been manufactured correctly. A standard among people familiar with the testing process is: if it costs $0.30 to detect a fault at the chip level, then it would cost $3 to detect that same fault at the board level; $30 when it is embedded at the system level; and $300 when it is embedded at the system level but it has to be found in the field. Thus if a fault can be detected at the chip level, then significantly larger costs per fault can be avoided at subsequent levels of packaging [19,20,21].

The circuit complexities, in terms of the number of gates, have increased the cost of testing; however, this increase in cost arises not only from the size of the circuit but also from the way in which the circuit has been designed, that is, from the fact that emphasis has been placed during design phase on the implementation of the function at the expense of consideration of the way in which the circuit would ultimately be tested. Regardless of the size of the circuit, various factors make the testing process difficult. The first is the inability to initialize memory elements in the circuit into some known state independent of the present state. The second factor is the inability to control or observe signal values on internal nodes in the circuit. Finally, the realization of logic functions using pass transistors which do not map readily onto equivalent gate models [23].

One general approach to addressing this problem is embodied in a collection of techniques known as "Design For Testability (DFT)" whose objective is to reduce the cost of testing in general, by

- making internal nodes in a circuit more accessible;

- transforming sequential circuits into combinational circuits or decomposing complex circuits into less complex sub-functions for the purposes of testing;
- reducing the amount of test data which is needed to test the circuit.

There are two key concepts in Design For Testability: *controllability* and *observability*. Control and observation of a circuit are central to implementing its test procedure.

The collection of techniques that comprise Design For Testability are, in some cases, general guidelines; in other cases, they are hard and fast design rules. Although these design methodologies which improve the testability of a circuit are highly desirable, ultimately they incur some penalty with respect to the effect upon the designer and the performance of the circuit. If a particular design style adopted to ease the testing problem is chosen, it must be easy to apply and not so constraining as to inhibit the flexibility of the designer. Furthermore, since most design for testability methods involve either additional hardware and/or routing, the physical size of circuit increases and subsequently has an effect on the yield. The additional hardware also introduces extra signal delays into the circuit which affect the performance of the circuit. There can be also a requirement for extra input/output pins, increasing the cost of packaging. In view of these penalties it must be demonstrated that a particular design style produces a marked reduction in test generation costs with respect to CPU time and personal effort [23,22].

Designing for testability implies some modifications to the circuit to enhance the process of test pattern generation and application. These techniques have been categorized into three main groups [20]:

1. *ad hoc* methods;
2. structured approaches;
3. built-in test methods.

4.1 Ad Hoc DFT Methods

Ad hoc methods evolved through the necessity to solve a particular testing problem, rather than trying to solve the problem of testing complex circuits in

general by using some design methodology. The methods evolved can be classified as test point insertion, pin amplification, blocking or degating logic for partitioning, bus architecture, control and observation switching, test state registers, and signature analysis [23,20]. *Ad hoc* methods for improving the testability of a circuit have the advantage of not imposing severe constrains on the designer but have the disadvantage that the methods cannot be automated, and consequently there is no software support for these techniques of designing for testability. In addition, these techniques solve the testing problem for a given design and are not generally applicable to all designs.

In designing the correlator chip, some *ad hoc* methods have been used such as partitioning, pin amplification, and control and observation switching. For partitioning, additional gates have been incorporated into the design to inhibit data flow along certain paths, thus dividing the circuit into smaller subcircuits which are much more manageable for the purpose of testing. These subcircuits have not been necessarily designed with any design for testability in mind. This approach is also called as "Divide and Conquer". In applying this method, address pins A0, A1, A2 and additional gates in the controller have been used to control data flow among the modules thus reducing the testing problem into dealing with the smaller modules.

Since the number of pins available for test purposes was minimal, the number of test pins was increased by multiplexing of the normal input/output pins to perform an additional function of acting as test inputs and outputs. For example, the address pin A2 and the SYN output pin were utilized in this direction. This technique is called as pin amplification. However, the delays introduced by the multiplexers/demultiplexers would degrade the circuit response under the normal operating conditions. For control and observation switching technique signal lines whose logic values were either easily controlled or observed have been identified in the circuit. And these have been used in conjunction with demultiplexers/multiplexers to improve the access to nodes, in close proximity to these lines, whose logic values were difficult to control or observe. Test mode control inputs to the multiplexers/demultiplexers determine whether the lines, whose signal values can be easily controlled or observed, are to be used for transmitting normal data or test data.

In fact, the chip already has similar features with some other DFT methods like bus architecture and test state registers. Because the chip is bus structured with a data bus going to different modules on the chip and with an address bus controlling the selection of these modules. Bus structured

architectures are successfully used in the partitioning problem. In addition, the 128-bit shift register can be thought of as a test state register since it is a serial input parallel output shift register which may be used to increase the number of test control signals applicable to the circuit at any given time.

4.2 Structured DFT Approaches

In contrast to the *ad hoc* methods, the structured approaches to design for testability are more formal and are incorporated into a design from the outset rather than introduced as an afterthought as with the *ad hoc* methods. The objective in developing the structured approach was to facilitate the testing of complex circuits; consequently these approaches are directed at increasing the controllability and observability of the internal states of the circuit, essentially transforming the testing of a sequential circuit into the simpler task of testing a combinational circuit. In other words most structured design approaches are built upon the concept that if the values in all the latches can be controlled to any specific value, and if they can be observed with a straightforward operation then the test generation, and possibly, the fault simulation task, can be reduced to that of doing test generation and fault simulation for a combinational logic circuit [23,20].

Over the past years several structured approaches have evolved, namely Level-Sensitive Scan Design (LSSD), Scan Path, Scan/Set, and Random Access Scan [23,20]. In design phase of the correlator chip level-sensitive scan design approach has also been applied. LSSD which is an IBM's discipline for structural design for testability incorporates two design concepts namely level sensitivity and scan path. "Scan" refers to the ability to shift into or out of any state of the circuit. "Level-sensitive" refers to constraints on circuit excitation, logic depth, and handling of clocked circuitry. The concept of a level sensitive design requires that the operation of the circuit is independent of the dynamic characteristics of the logic elements, that is, rise and fall times and propagation delays. Furthermore in a level sensitive design the next state of the circuit is independent of the order in which changes occur when a state change involves several input signals; this circuit characteristic implicitly places a constraint on what signal changes can occur in the circuit, these constraints however are usually applied to the clocking signals.

The major element in a level sensitive design is the "shift register latch" (SRL) generally used to implement all storage elements in the circuit. The

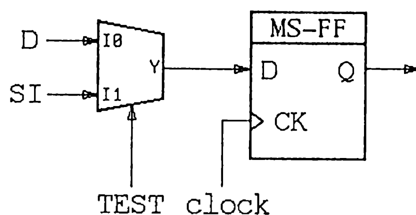


Figure 4.1: Symbolic representation of a shift register latch.

shift register latch used in the correlator chip has been designed as shown in Fig. 4.1. It consists of a simple master-slave flip-flop and an additional 2-to-1 multiplexer. Input D is the normal signal input to the SRL and SI is the scan path input to the element. In LSSD applications the shift register latches are driven by two nonoverlapping clocks which can be readily controlled from the primary inputs to the circuit. But here only one clock signal is used to control the normal operation of the element and movement of data through the SRL when it is used as part of the scan path. And this clock signal is controlled from a primary input pin of the chip.

In addition to providing a circuit response which is independent of delays, the SRL play an important function in the testing of the circuit. Because all the SRLs have the ability to be configured into a long serial shift register called a "scan path", which permits the internal states in the circuit to be easily controlled or observed, access to each storage element in the circuit is available via the scan path. Fortunately, the registers used in the correlator circuit are already composed of shift register blocks but not single flip-flops. Therefore the SRL shown in Fig. 4.1 is used only at the first stages of these shift register blocks. For example, the logic diagram of the reference and mask registers modified with respect to LSSD technique is shown in Fig. 4.2. As it is seen, only 8 multiplexers are sufficient for making these registers LSSD testable. The multiplexers are switched from the normal operation mode to the test mode by using a test signal produced in the controller block via decoding the address bus. In the test mode (i.e., TEST=1) the shift register latches take their inputs from other latches instead of data bus. Also both registers have a single scan input (SCI) and a single scan output (SCO).

Similarly, the threshold and status registers have been modified using LSSD technique as shown in Fig. 4.3 and Fig. 4.4. As it is seen additional 8 multiplexers for threshold register and 5 multiplexers for status register are needed. As in the reference and mask registers, these multiplexers switch from normal operation to test mode according to the state of the TEST signal. Again each register has a scan input and a scan output.

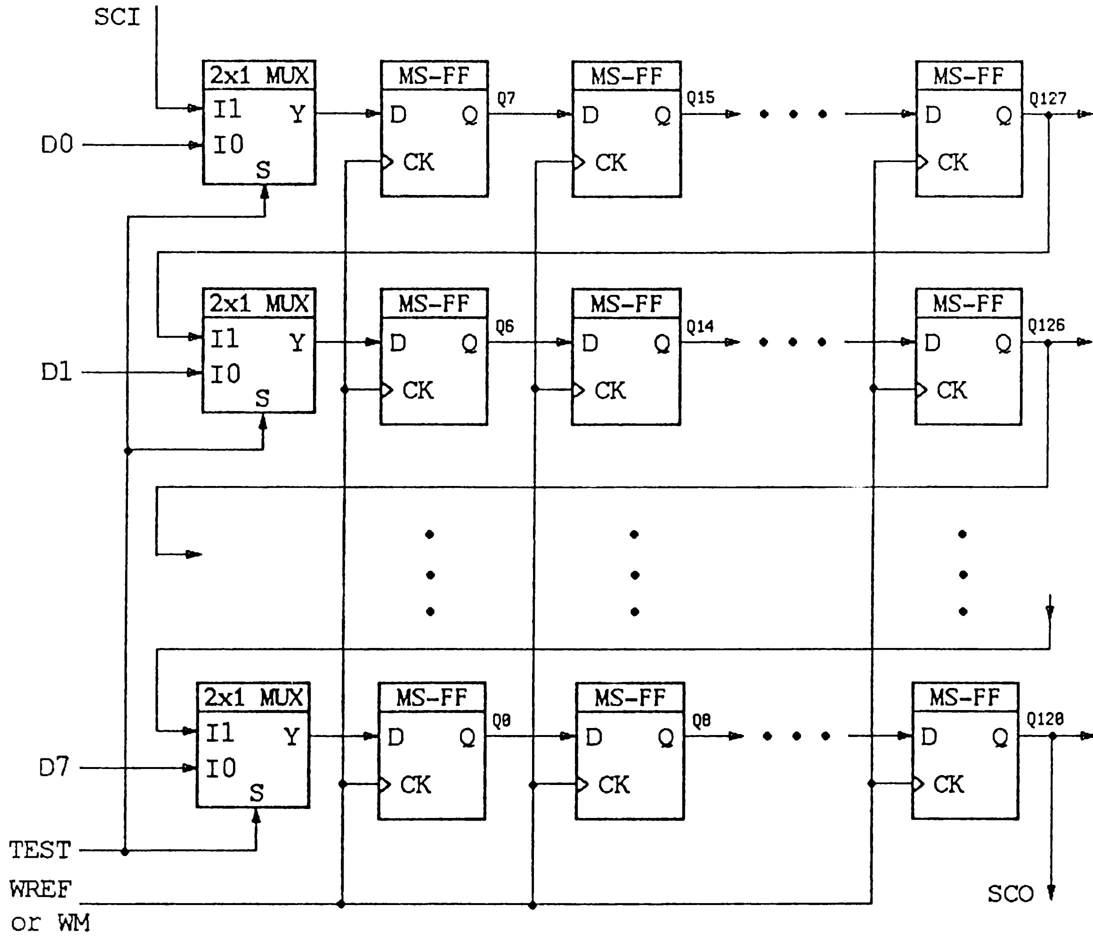


Figure 4.2: Level sensitive scan design of the reference/mask registers.

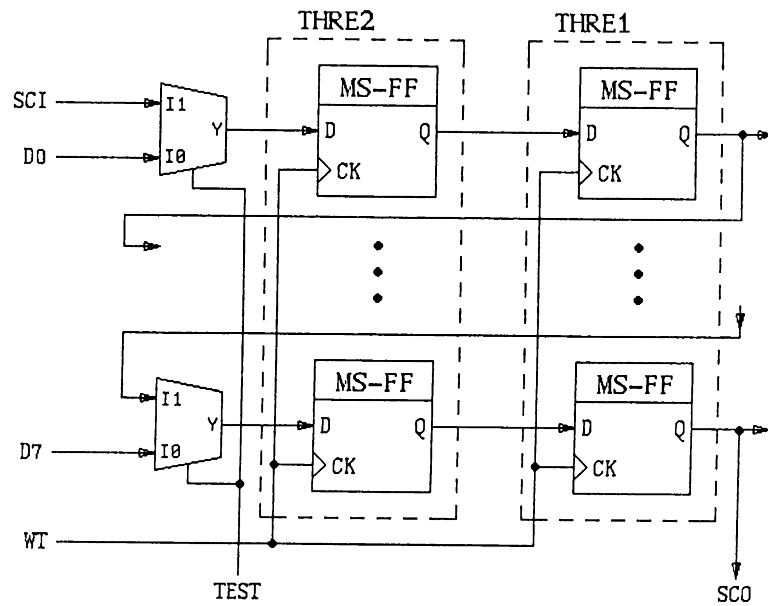


Figure 4.3: Level sensitive scan design of the threshold register.

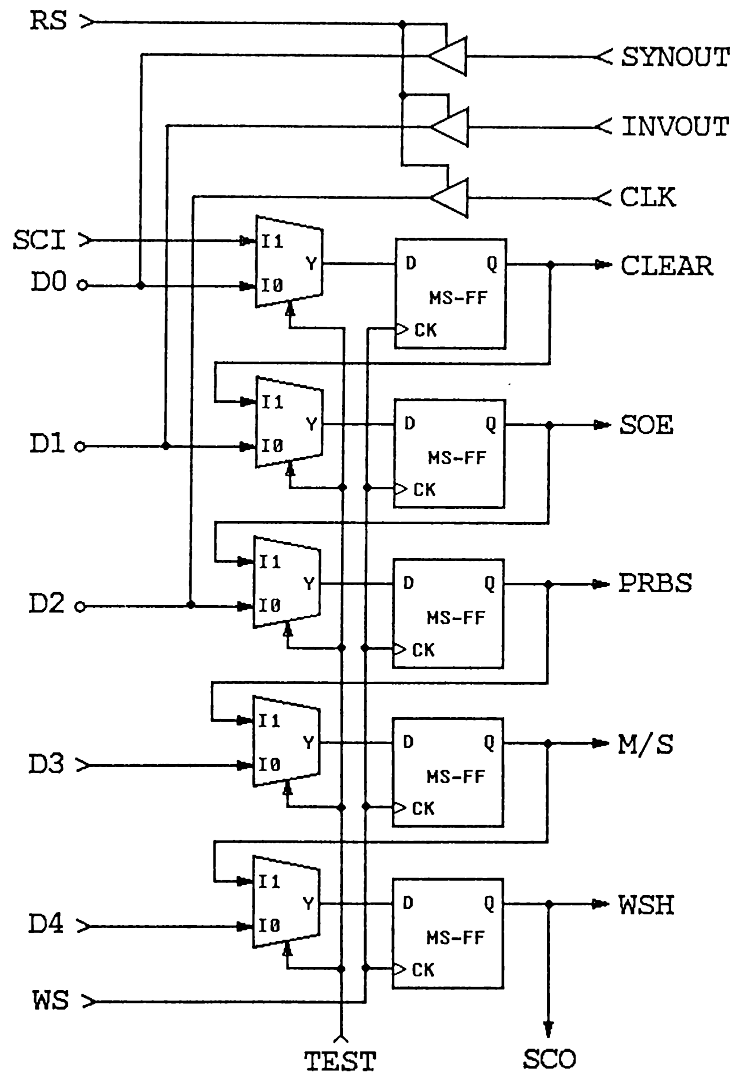


Figure 4.4: Level sensitive scan design of the status register.

In order to complete the scan path, scan output of a register block has been connected to the scan input of another register block as shown in the Fig. 4.5. In this LSSD configuration four registers commanded as shown act as a single serial shift register with a scan input (SCI of the reference) and a scan output (SCO of the status). Here the reference register gets its scan input from data bus line D0. This means that the scan path input can be accessed from a primary input pin. To be able to observe the scan path output, SCO of the status register has been multiplexed with the SYN output of the decision maker block. Thus, in the test mode, the scan path has a primary input and a primary output. In addition to these, the clocks of the four registers constituting the scan path behave as a single clock whose state depends on the μP's write signal which is a primary input. Note that the 128-bit shift register holding the incoming data for correlation is not included in the scan path because it has already a primary input and a primary output.

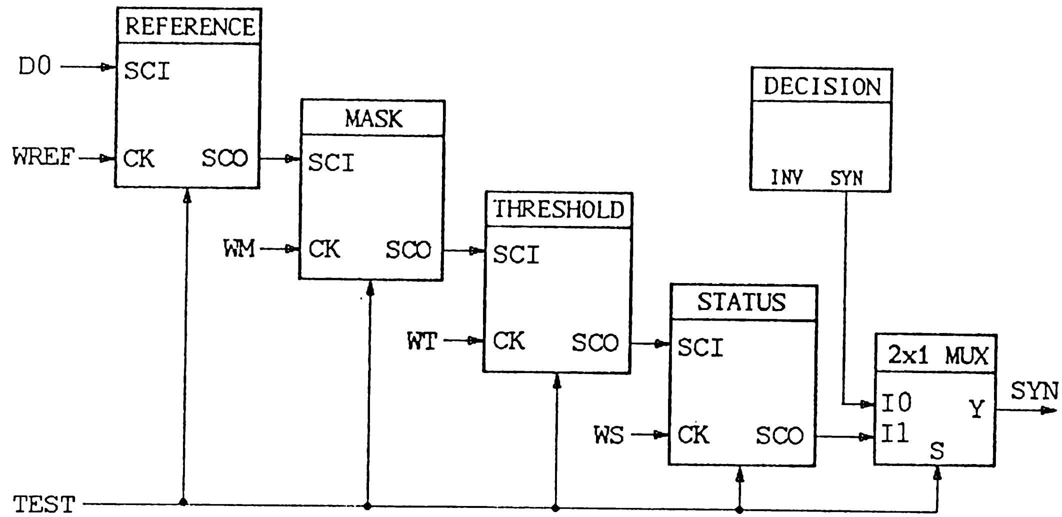


Figure 4.5: LSSD configuration of the registers.

An important feature of the LSSD technique is that for the purposes of testing, a large circuit is implicitly partitioned into smaller subcircuits, consisting entirely of combinational logic functions, by the shift register latches, which will be instrumental in applying the test patterns to the combinational blocks. And by using the scan path, future states can be set up independently of the present state of the system and internal states can be easily observed. It turns out that the circuit can now be thought of as purely combinational, where tests are applied via primary inputs and register outputs. Therefore the main advantages of the LSSD technique are that it removes the necessity of performing detailed timing analysis on the circuit since it is level sensitive; automatic test pattern generation is simplified since tests need only be generated for a combinational circuit; and finally this technique is a disciplined design methodology.

However, the LSSD technique has several disadvantages; first, the designer is constrained to implement his system as a synchronous sequential circuit. This condition is not important for the correlator circuit because it already satisfies this constraint. Second, the shift register latches in the registers are, logically, two or three times as complex as simple latches. But in the correlator chip, although the SRL element shown in Fig. 4.1 is not used for all register latches but only for 7 percent of them, it additionally contains only a multiplexer subsequently having 4 transistors more than the other latches which has 20 transistors. Third, additional input/output pins are required for the scan-in/out ports and clocks. This overhead has also been totally removed because the scan input and scan output are used also as functional input and output of the chip. And the clock of the scan path is controlled by

the write signal of the microprocessor, consequently an extra pin on the chip is not needed. Finally, test times are increased since the input and output data must be scanned serially. Test times for the correlator can be reduced significantly because the four registers in the scan path can also be loaded from data bus in parallel instead of serially and there remains only observing the states of the registers at the scan path output as serially.

As a result, all design for testability techniques applied to the correlator chip has increased the overall transistor count by 1 % while improving the testability of the chip considerably.

5. SIMULATIONS AND TESTING OF THE CHIP

After the layout design has been completed, simulations of the chip had to be done. Two types of simulations has been performed in the simulation part, namely: functional simulation and timing simulation. The important thing to mention is that both types of simulations has been carried out using the layout design of the chip on SUN workstations.

5.1 Functional Verification

The purpose of the functional simulation is to verify whether the chip performs its various functions correctly not regarding the timing characteristics such as rise and fall times or propagation delays of the elements. That is, by such a simulation only operation of the chip can be checked with the specified functions in the beginning of the design phase. This simulation has been done by using an event driven switch level simulator for nMOS or CMOS transistor circuits, **ESIM** [16]. The speed of ESIM is comparatively high: it takes approximately 15 minutes to simulate the overall circuit for 5000 input vectors.

In the functional simulation phase, first each block shown in the Fig. 2.3 has been simulated individually. To do this, separate input batch files containing the input vectors and ESIM commands to direct the simulation for each block have been prepared and then these files have been given to ESIM subsequently simulating the circuit and saving the results in the separate output files for each block. In fact, the electrical circuit descriptions of these blocks has been obtained by using the netlist extractor of Magic layout editor. For example, the input batch file to ESIM for the controller block is

given below. As it is seen, first the circuit is initialized with an **I** command. Here the **w** command describes the nodes to be watched in the circuit during the simulation and the **V** command defines the input vectors to the circuit in which the vectors consist of ones and zeros. The controller block has six input nodes and all combinations of the states of these inputs are given in the vectors. Now the circuit can be simulated and this is achieved by giving a **G** command to ESIM. This command runs the simulator as long as the longest vector and reports back the states of all the watched nodes as vectors succeedingly for every column of the input vector array. Finally, the **q** command is used to terminate the simulation run.

```
I
w A2 A1 A0 CS WR RD
w WRE WM WT WS RS RSH INT TEST
V A2  0000000000000000000000000000000011111111111111111111111111111111
V A1  0000000000000000111111111111111100000000000000001111111111111111
V A0  0000000011111111000000001111111100000000111111110000000011111111
V RD  0101010101010101010101010101010101010101010101010101010101010101
V WR  0011001100110011001100110011001100110011001100110011001100110011
V CS  0000111100001111000011110000111100001111000011110000111100001111
G
q
```

After this input batch file has been created, ESIM has been invoked in batch mode by such a command

esim cntl.sim -cntl.out cntl.in

where the file "cntl.sim" contains the circuit description, "cntl.in" file described above has the input vectors, and "cntl.out" is the output file to which the simulation results will be written instead of the computer screen. The results of this simulation saved in the output file "cntl.out" is given below. The input node values are also listed as well as output nodes and as it is seen the read and write signals (i.e., clocks) for registers and the test signal is properly generated. Here the sign > represents the prompt of ESIM and at the last row it gives some information such as the number of transistors and the number of nodes in the whole correlator circuit.

```
>0000000000000000000000000000000011111111111111111111111111111111:A2
>0000000000000000111111111111111100000000000000001111111111111111:A1
>0000000011111111000000001111111100000000111111110000000011111111:A0
>0000111100001111000011110000111100001111000011110000111100001111:CS
>0011001100110011001100110011001100110011001100110011001100110011:WR
>0101010101010101010101010101010101010101010101010101010101010101:RD
>0000001100000000000000000000000000000000000000000000001100000000:WRE
>0000000000000011000000000000000000000000000000000000001100000000:WM
>0000000000000000000000110000000000000000000000000000001100000000:WT
>0000000000000000000000000000001100000000000000000000001100000000:WS
>0000000000000000000000000000010000000000000000000000000000000000:RS
>0000000000000000000000000000000000000100000000000000000000000000:RSH
>0000000000000000000000000000000000000000000011110000000000000000:INT
>0000000000000000000000000000000000000000000000000000111100000000:TEST
14912 transistors, 8307 nodes (6 pulled up)
```

In a similar manner, all the other blocks have been simulated and then the overall functional simulations of the correlator chip have been carried out by using ESIM. The results of these simulations are given in Appendix A. In one of the overall simulations, first the reference, mask, threshold, and status registers have been loaded where the error tolerance in the threshold register was zero, and then the input data which was exactly same as the reference data has been loaded serially into the 128-bit shift register from SIN. In the 128'th cycle of the "CLK" input signal, the sync word has been detected at the SYN output of the chip. Similar simulations have been performed for inverted sync word detection and also for 256-bit correlation.

5.2 Timing Verification

The timing simulation has been divided into two steps in which SPICE and RNL [17] simulators have been used. SPICE is a general-purpose circuit simulation program for nonlinear dc, nonlinear transient, and linear ac analyses. It has been used for the detailed and precise timing simulations of the small sized layout cells having the number of transistors up to 300. The SPICE simulations has been done by İ. Enis Ungan [6]. RNL is a timing logic simulator for digital MOS circuits. It is an event driven simulator that uses a simple RC (resistance capacitance) model of the circuit to estimate node transition times and the effects of charge sharing. This simulation program

has been used in the timing simulations of the large and complicated layout blocks which could not be analyzed by using SPICE in a practical time. For example, loading process of the 128-bit reference register through the data bus is shown in Table 5.1.

time	db	WR	reflast
50	0b11111111	1	0b00000000
100	0b11111111	0	0b00000000
150	0b11111111	1	0b00000000
200	0b11111111	0	0b00000000
250	0b00000000	1	0b00000000
300	0b00000000	0	0b00000000
350	0b00000000	1	0b00000000
400	0b00000000	0	0b00000000
	.	.	.
.	.	.	.
.	.	.	.
1450	0b00000000	1	0b00000000
1500	0b00000000	0	0b00000000
1550	0b00000000	1	0b00000000
1600	0b00000000	0	0b00000000
1650	0b11111111	1	0b11111111
1700	0b11111111	0	0b11111111
1750	0b11111111	1	0b11111111
1800	0b11111111	0	0b11111111
1850	0b00000000	1	0b00000000
1900	0b00000000	0	0b00000000
1950	0b00000000	1	0b00000000
2000	0b00000000	0	0b00000000
2050	0b11111111	1	0b11111111
2100	0b11111111	0	0b11111111
2150	0b11111111	1	0b11111111
2200	0b11111111	0	0b11111111

Table 5.1: Timing simulation of the reference register.

Here some nodes are grouped as binary vectors. The vector "db" represents the data bus and it stands for "db=D0 D1 D2 D3 D4 D5 D6 D7". The vector "reflast" represents the reference register outputs. These vector definitions are done in the control file (*.l* file) of RNL with the command

(defvec '(bin db D0 D1 D2 D3 D4 D5 D6 D7))

Since base of the vectors are chosen as binary in "defvec" command, they are printed as binary numbers at the end of each simulation step. In the RNL simulation results the first two characters, 0b, of the vectors indicate the base (they would be 0 for octal, 0x for hexadecimal, and none for decimal).

In this RNL simulation of the reference register time is presented in terms of nanoseconds (ns). So, every simulation step takes $50ns$ and the period of the WR signal is chosen to be $100ns$. As it is seen, after 16 write cycles, the patterns given at the data bus appear at the reference register outputs.

In addition to the simulations of the blocks, also the timing characteristics of the whole layout of the correlator has been checked for consistency with the timings of the microprocessor. For instance, simulation results for the 128-bit sync detection is shown in Table 5.2. In order to perform such a simulation, the reference, mask, threshold, and status registers are loaded previously. Then a serial input data is applied to the serial input (SIN) of the 128-bit shift register. The aim is, first, to produce a sync pulse at SYNOUT and then to produce an inverted sync pulse at INVOUT. Here the time values are expressed in terms of nanoseconds. This simulation is carried out in steps of $500ns$ and the clock period is $1000ns$. This means that the correlation frequency is $1MHz$.

As shown in Table 5.2, after 128 clock cycles, the data in the 128-bit shift register correlates well with the content of the reference register and a sync pulse is generated. After 256 clock cycles, the shift register data changes wholly such that an inverted sync pulse is generated.

5.3 Test Pattern Generation (TPG) and Fault Simulation

In addition to the problems of verifying complex circuit designs prior to fabrication, a second major issue in the design of VLSI circuits is that of isolating faulty devices immediately after fabrication by means of testing. The VLSI testing problem is the sum of a number of problems which relate to the cost of doing business in the final analyses. There are two basic problem areas:

1. test generation,
2. test verification via fault simulation.

time	ab	CLK	SIN	SOUT	SYNOUT	INVOUT
40000	0b110	1	0	0	0	0
40500	0b011	1	1	0	0	0
41000	0b011	0	1	0	0	0
41500	0b011	1	1	0	0	0
42000	0b011	0	1	0	0	0
42500	0b011	1	1	0	0	0
43000	0b011	0	1	0	0	0
.		.		.		
	.	.		.		
.	.	.	.	.	.	.
165500	0b011	1	1	0	0	0
166000	0b011	0	1	0	0	0
166500	0b011	1	1	0	0	0
167000	0b011	0	1	0	0	0
167500	0b011	1	1	0	0	0
168000	0b011	0	1	1	0	0
168500	0b011	1	1	1	1	0
169000	0b011	0	1	1	1	0
169500	0b011	1	1	1	1	0
170000	0b011	0	1	1	1	0
170500	0b011	1	0	1	1	0
171000	0b011	0	0	1	1	0
171500	0b011	1	0	1	0	0
172000	0b011	0	0	1	0	0
172500	0b011	1	0	1	0	0
173000	0b011	0	0	1	0	0
173500	0b011	1	0	1	0	0
174000	0b011	0	0	1	0	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
295500	0b011	1	0	1	0	0
296000	0b011	0	0	1	0	0
296500	0b011	1	0	1	0	0
297000	0b011	0	0	1	0	0
297500	0b011	1	0	1	0	0
298000	0b011	0	0	0	0	0
298500	0b011	1	0	0	1	1
299000	0b011	0	0	0	1	1
299500	0b011	1	0	0	1	1
300000	0b011	0	0	0	1	1

Table 5.2: Timing simulation of the sync detection.

The problem with test generation is that as logic circuits get larger, the ability to generate test patterns will become more and more difficult. Fault simulation is that process by which the fault coverage is determined for a specific set of input test patterns. In particular, at the conclusion of the fault simulation, every fault that is detected by the given pattern set is listed. However, it is a very time-consuming, and hence, expensive task. It has been observed in the literature [20] that the computer run time to do test generation and fault simulation is approximately proportional to the number of logic gates to the power of 3; hence, small increases in gate count will yield quickly increasing run times.

Test pattern generation methods are divided into two classes, namely those which use the logic equations to generate test patterns and are referred to as algebraic or functional methods and those which use topological gate level descriptions for test pattern generation, referred to as the structural methods. Although adequate test pattern generation methods exist for detecting faults in combinational circuits, the same cannot be said for sequential circuits. The major difficulty is that the output response of a sequential circuit depends not only on the present input values but also on the stored states in the circuit. Therefore in order to detect a fault in a sequential circuit a series of test vectors must be applied, rather than a single test vector as in the combinational circuit [23,25].

Fault simulators are used extensively to predict the behavior of the circuit under fault conditions. As circuit complexities increased efficient fault simulation techniques were developed: (1) parallel fault simulation, (2) deductive fault simulation, and (3) concurrent fault simulation [23]. In fact, fault simulations are performed in conjunction with automatic test pattern generation programs: first to determine the overall fault coverage of a set of test vectors; second, to determine what other faults a given test vector will detect, since the generation of a test for a fault condition is a time-consuming process, so that when a test vector is generated a fault simulator is used to determine what other faults the test will detect, since this is considered more economical than test vector generation.

Test pattern generation methods have been directed at generating test patterns to detect faults with the use of fault models. That is, test pattern generation began with the concept of fault models [26,27]. A model of faults which does not take into account all possible defects, but is a more global type of model, is the stuck-at model. The stuck-at model being the most common one assumes that a logic gate input or output is fixed to either a

logic zero or a logic one. As stated, not all defects which occur in present-day integrated circuits can be modeled with a stuck-at-fault model. Some examples of the faults that cannot be modeled by stuck-at model are shorted or open interconnections which change the logical function of the circuit: for example, a short circuit between two signal lines may introduce an asynchronous feedback loop into a section of combinational circuitry, or in the case of CMOS circuits a simple NOR gate in the presence of a stuck-open fault is transformed into a logic function which has a 'memory'.

For the correlator circuit test patterns were generated via computer by using the existing fault models and known techniques. Mainly, bridging faults (shorts) and stuck-at faults were taken into consideration related with chip layout. Since there was a tight interaction with the layout, the location of possible shorts between the interconnections could easily be seen. As mentioned in the previous chapter, LSSD technique was applied in designing the correlator circuit. A feature of this design technique is that for the purposes of testing, a large circuit can be partitioned into smaller subcircuits, consisting entirely of combinational logic, by the blocks of shift register cells which are implicit in the LSSD style of design. These shift register blocks are instrumental in applying the test inputs to the subcircuits; consequently the registers must be tested first for possible faults before testing can begin on the combinational subcircuits. The tests performed on the registers comprise a 'flush' and a 'shift' test; that is, clocking blocks of 1's and 0's or a pattern 001100, which checks all combinations of initial and next states through the registers, which can be concatenated into a single serial shift register. Of course, this testing of the registers was performed in the test mode of the chip in which a scan path was formed. The test patterns were applied to the scan path input (SCI of the reference register), producing a one-stage shift along the registers for every write cycle of the microprocessor, and the resulting pattern was then scanned out of the registers from scan path output (SCO of the status register). As it is clear, testing of the registers is very easy, however, the input and output data must be handled serially.

The number of test vectors necessary for testing of the comparator block was also very small: 8. This is because all the 128-bit outputs of it were independent of each other. But in order to apply these test vectors to inputs of the comparator block, one must load first the reference, mask, and 128-bit shift registers because the comparator inputs can only be accessed through these registers and it takes some time. Then comes the testing of the 1's counter block which is a large combinational circuit having the 30 percent of overall transistor count in the chip. The logic diagram of the 1's counter is

shown in Fig. 3.10. This block has 128 inputs and it is impossible to either generate or apply all the combinations of input values (2^{128}). Therefore it was carefully analyzed and as a result it was observed that there are independent paths between the inputs and outputs. In other words, it has seven stages from half-adders to seven-bit adder where all the inputs entering to these stages can be controlled directly by accessing to the comparator outputs. To do this, 128-bit shift register and mask register must be loaded with all 1's and with all 0's, respectively, so that the contents of the reference register could reach to the comparator outputs thus having the ability to control the 1's counter inputs through the data bus. The outputs of 1's counter can be observed from the primary output pins, C0-C7, of the chip.

Here it is convenient to mention that the 1's counter is tested functionally. Now, we assume a single stuck-at fault or bridging fault in this block. This is called as the single stuck-at fault model. Then the whole process of testing the 1's counter block proceeds as follows: it is started by testing of the half-adders at the first stage. A half-adder has two inputs whose combinations are $2^2 = 4$. Since the inputs of each half-adder are independent of those of others, 64 half-adders can be tested by using only 4 test vectors which are 00, 01, 10, and 11. When these vectors are applied in this order to the input of every half-adder, the value of 8-bit output of 1's counter must be 0, 64, 64, and 128. If the 1's counter output deviates from these numbers then this means that there is a faulty half-adder at the first stage. At the second stage there are two-bit full-adders whose inputs are connected to outputs of the half-adders at the first stage. Now let's think about a single two-bit full-adder at the second stage. It performs the addition of 2 two-bit numbers whose values can be 0, 1, or 2. So the combinations of the input values of a two-bit adder is $3^2 = 9$. This means that all the two-bit adders can be tested functionally by using only 9 test vectors shown in Table 5.3 with the corresponding numbers which must be at the 1's counter output. As stated before, the inputs of these adders can be controlled from the 1's counter inputs. Note that every two-bit adder is manipulated independent of the others.

The test vectors for three-bit full-adders at the third stage of the 1's counter is found similarly. If a three-bit adder is considered alone, it adds 2 three-bit numbers whose values can range between 0 and 4, thus having $5^2 = 25$ combinations as shown in Table 5.4 with the corresponding numbers that must be seen at the 1's counter output. Therefore all the three-bit adders can be tested by using these 25 test vectors. In a similar manner other stages can be tested too. Consequently, the number of test vectors needed to test

A2	A1	B2	B1	counter output
0	0	0	0	0
0	0	0	1	32
0	0	1	0	64
0	1	0	0	32
0	1	0	1	64
0	1	1	0	96
1	0	0	0	64
1	0	0	1	96
1	0	1	0	128

Table 5.3: Test vectors for two-bit full-adders.

each stage in the 1's counter is shown in the Table 5.5. However there are some overlaps among the test vector sets for the stages. That is, certain vectors may be used for testing of more than one stage and so they are considered more than once but these vectors can be eliminated easily.

For testing process of the 1's counter two programs have been written in C programming language using the techniques mentioned above. One of the programs called **gen** generates all the test vectors for the 1's counter block in the format of the ESIM functional simulator. That is, the test vectors generated by this program can be simulated on the 1's counter using ESIM. This program also calculates the actual outputs of the 1's counter corresponding to each test vector. So we know what the 1's counter output must be while applying the test vectors using ESIM and subsequently we have the chance to check the simulation results. The program **gen** writes the generated test vectors and the calculated actual outputs for each stage of the 1's counter into different files. For example, 'Tcount.vec1', 'Tcount.out1' and 'Tcount.vec2', 'Tcount.out2' files are created by this program for the first and second stages, respectively.

The second program called **check** compares the actual outputs of the 1's counter produced by the first program with the ESIM simulation outputs for the 1's counter. If there is a difference between these two outputs, this program finds the difference and gives information about which stage is faulty with the test vector causing the problem. Thus, by using these two programs, **gen** and **check**, first all the test vectors for the 1's counter have been generated as well as the actual outputs and these vectors have been applied to the

A3	A2	A1	B3	B2	B1	counter output
0	0	0	0	0	0	0
0	0	0	0	0	1	16
0	0	0	0	1	0	32
0	0	0	0	1	1	48
0	0	0	1	0	0	64
0	0	1	0	0	0	16
0	0	1	0	0	1	32
0	0	1	0	1	0	48
0	0	1	0	1	1	64
0	0	1	1	0	0	80
0	1	0	0	0	0	32
0	1	0	0	0	1	48
0	1	0	0	1	0	64
0	1	0	0	1	1	80
0	1	0	1	0	0	96
0	1	1	0	0	0	48
0	1	1	0	0	1	64
0	1	1	0	1	0	80
0	1	1	0	1	1	96
0	1	1	1	0	0	112
1	0	0	0	0	0	64
1	0	0	0	0	1	80
1	0	0	0	1	0	96
1	0	0	0	1	1	112
1	0	0	1	0	0	128

Table 5.4: Test vectors for three-bit full-adders.

stg1	stg2	stg3	stg4	stg5	stg6	stg7	TOTAL
4	9	25	81	289	1089	4225	5722

Table 5.5: The number of test vectors for stages in the 1's counter.

1's counter by means of ESIM, finally a check between ESIM outputs and the actual outputs has been done. Consequently, the 1's counter has been tested exhaustively by using 5722 test vectors. The two programs and example files are given in Appendix B.

As a result all the registers and a significant part of the combinational circuitry in the correlator chip has been tested using the extracted netlist from the layout. The testing of the registers and the comparator block was easy with a small number of test vectors. However, for testing of the 1's counter and the 8-bit adder the number of test vectors needed was somewhat large, about 6000. But this number can be reduced considerably by doing a fault simulation using the generated test vectors. Finally, with respect to the ratio of the number of transistors or number of nodes in the part of the chip that can be tested to the whole chip, the expected fault coverage of the test patterns generated is 95 percent.

6. CONCLUSION

A microprocessor compatible 128-bit digital correlator has been designed to be implemented as a full-custom single VLSI chip. This work has been jointly carried on with İ. Enis Ungan. The details of the circuit and layout designs and SPICE simulations can be found in [6].

The chip is to be placed in a microprocessor (μP) based portable data terminal using HF radio communication. It marks the beginning of a synchronous data stream received from the very noisy channel by detecting a synchronization (sync) word. It can detect sync words of length up to 128-bits but two chips can be cascaded to make a 256-bit correlator. The sync word can be detected for either inverted or non-inverted incoming data streams. The correlator is fully programmable by a microprocessor to set the number of tolerable errors in detection and to select the bits of the 128-bit (or 256-bit) data stream to be included in the correlation. The latter feature makes the chip capable for use in detection of distributed sync words and PRBS generation.

The digital correlator compares, bit-by-bit, a given data sequence of binary digits against a standard (correct) sequence. If these two binary sequences are derived from different sources, the operation is cross-correlation. In contrast, auto-correlation is the comparison of a single binary sequence against a time-shifted copy of itself. The chip fits many of the correlator applications which include: detection of differences between two data sequences, determination of the time delay between two similar signals, correction of errors in expanded-code data streams, multiplexing of data among several users, recognition of specified patterns within an incoming data stream, synchronization of a decoding process or analyzer with an incoming data stream, identifying periodicities within a data stream, and extracting a periodic signal from its random noise back-ground. In addition to these, it could be used in the automated testing of digital circuits, in principle.

While designing the correlator chip some "Design For Testability (DFT)" methods, especially scan design and partitioning techniques, have been used. These methods have increased the testability of the chip significantly resulting in an easy way of test generation and fault simulation although they have involved very small additional logic such as an overhead in the transistor count only by 1 percent. The chip contains approximately 15,000 transistors occupying a silicon area of 5.6mm×5.2mm and it has 28 pins.

Two types of simulations, functional and timing simulations, has been performed on the chip layout by using **ESIM** and **RNL** simulators, respectively. Test patterns for the registers have been generated manually and for testing of the combinational part, two programs **gen** and **check** have been written in C programming language. The test patterns and the correct outputs for the combinational circuitry have been generated by using the first program and these test vectors have been simulated against the layout using ESIM. Then the second program has been used to check for any difference (i.e., fault) between the correct outputs and the ESIM outputs. In the result, approximately 6000 test vectors have been generated and applied to the correlator chip with an expected fault coverage of 95 %.

REFERENCES

[1] R. A. Williams, Communication Systems Analysis and Design - A Systems Approach, Prentice-Hall, New Jersey, 1987.

[2] J. Eldon, *Correlation: A Powerful Technique for Digital Signal Processing*, TRW LSI Publication, TP17B-4/81.

[3] G. R. Cooper and C. D. McGillem, *Modern Communications and Spread Spectrum*, McGraw-Hill, 1986.

[4] K. Ramachandran and R. R. Cordell, "A 30 MHz Programmable CMOS Video FIR Filter and Correlator", in *Proc. of ISCAS'88*, pp. 705-708, 1988.

[5] İ. E. Ungan, S. Topcu, and A. Atalar, "VLSI Implementation of a Microprocessor Compatible 128-bit Programmable Correlator", in *Proc. of Third Inter. Symp. on Comp. Infor. Sci.*, Çesme, İzmir, Turkey, pp. 791-797, 1988.

[6] İ. Enis Ungan, *VLSI Implementation of a Microprocessor Compatible 128-bit Programmable Correlator*, M.S. Thesis, Bilkent University, May 1989.

[7] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Reading MA: Addison-Wesley, 1985.

[8] L. A. Glasser and D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits* Addison-Wesley, 1985.

[9] C. Mead and L. Conway, *Introduction to VLSI Systems.* Reading, MA: Addison-Wesley, 1980.

[10] M. J. Foster and H. T. Kung, "The Design of Special-Purpose VLSI Chips", *IEEE Computer*, pp. 26-40, Jan. 1980.

[11] R. D. Davies, "The Case for CMOS", *IEEE Spectrum*, pp. 26-32, Oct. 1983.

[12] E. Hörbst, C. Müller-Schloer, and H. Schwärtzel, Design of VLSI Circuits - Based on VENUS, Springer-Verlag, Heidelberg, 1987.

[13] M. Ş. Toygar, *Design and Computer Simulation of a Microprocessor Compatible Correlator*, B.S. Project, Middle East Technical University, 1987.

[14] S. Topcu, İ. E. Ungan, Ş. Toygar, and A. Atalar, "Design and testing of a microprocessor compatible 128-bit correlator," in *Proc. of Third Inter. Symp. on Comp. Infor. Sci.*, Çesme, İzmir, Turkey, pp. 798-804, 1988.

[15] M. M. Mano, *Digital Design*, Prentice-Hall, 1979.

[16] *Berkeley CAD Tools User's Manual*, EECS Dep., University of California at Berkeley, 1986.

[17] *VLSI Tools Reference Manual*, TR#87-02-01, Release 3.1, NW Lab. Int. Sys., Dep. Computer Sci., University of Washington, Feb.1987.

[18] İ. E. Ungan, S. Topcu, and A. Atalar, "A 128-bit Microprocessor Compatible Programmable Correlator Chip for Use in Synchronous Communication", in *Proc. of Third Annual European Computer Conf. on VLSI and Computer Peripherals*, Hamburg, West Germany, pp. 5.146-5.147, May 1989.

[19] T. W. Williams, "VLSI Testing", *IEEE Computer*, pp. 126-136, Oct. 1984.

[20] T. W. Williams and K. P. Parker, "Design for Testability-A Survey", *Proc. IEEE*, vol. 71, no. 1, pp. 98-112, Jan. 1983.

[21] T. W. Williams and K. P. Parker, "Testing Logic Networks and Designing for Testability", *IEEE Computer*, pp. 9-21, Oct. 1979.

[22] F. F. Tsui, *LSI/VLSI Testability Design*, McGraw-Hill, 1987.

[23] G. Russell, D. J. Kinniment, E. G. Chester, M. R. McLauglan, *CAD for VLSI*, Van Nostrand Reinhold, England, 1985.

[24] R. J. Feugate and S. M. McIntyre, *Introduction to VLSI Testing*, Prentice-Hall, New Jersey, 1988.

[25] S. K. Jain and V. D. Agrawal, "Modeling and Test Generation Algorithms for MOS Circuits", *IEEE Trans. Comput.*, vol. C-34, no. 5, pp. 426-433, May 1985.

[26] J. A. Abraham and W. K. Fuchs, "Fault and Error Models for VLSI", *Proc. of IEEE*, vol. 74, no. 5, pp. 639-654, May 1986.

[27] S. A. Al-Arian and D. P. Agrawal, "Physical Failures and Fault Models of CMOS Circuits", *IEEE Trans. Circuits and Systems*, vol. CAS-34, no. 3, pp. 269-279, March 1987.

APPENDIX A

FUNCTIONAL SIMULATION RESULTS OF THE CORRELATOR CHIP

In this appendix two example simulation results are presented; one for simulation of the registers and one for simulation of the sync detection. These results given are the output files created by ESIM. The input files to ESIM, not shown here, are prepared as described in the section 5.1. There is one thing to note that the input vectors as well as output vectors are given in the following results. The symbol **X** in the output vectors represents an undefined state. Also notice that the lines beginning with vertical bar (|) are treated as comments and ignored by ESIM.

Simulations of the Registers

Here the shift, reference, mask, threshold, and status registers are simulated with the input vectors described in the section 5.3. Since ESIM runs two simulation steps for a clock cycle, these input vectors are chosen as the pattern 000011110000. In addition the watched nodes are described by the corresponding label in the layout after a ":" sign. For the reference, mask, and threshold registers it is sufficient to watch only eight nodes because they consist of shift register blocks.

```
initialization took 19654 steps
initialization took 64 steps
initialization took 7 steps
initialization took 0 steps
|***
|*** LOAD THE STATUS REGISTER ***
|***
CS=0 A2=0 A1=1 A0=1
>000011110000:D0
>000011110000:D1
>000011110000:D2
>000011110000:D3
>000011110000:D4
>010101010101:WR
```

```
>000001111000:CLEAR
>100001111000:SOE
>100001111000:PRBS
>X00001111000:M|S
>100001111000:WSH
|***
|*** READ THE STATUS REGISTER ***
|***
CS=0 A2=0 A1=1 A0=1
>000011110000:SYNOUT
>000011110000:INVOUT
>000011110000:CLK
>010101010101:RD
>000011110000:D0
>000011110000:D1
>000011110000:D2
|***
|*** LOAD THE THRESHOLD REGISTER ***
|***
CS=0 A2=0 A1=1 A0=0
>000011110000111100001111:D0
>000011110000111100001111:D1
>000011110000111100001111:D2
>000011110000111100001111:D3
>000011110000111100001111:D4
>000011110000111100001111:D5
>000011110000111100001111:D6
>000011110000111100001111:D7
>010101010101010101010101:WR
>100000011110000111100001:thdm_0/B0
>111111100001111000011110:thdm_0/B1B
>000000011110000111100001:thdm_0/B2
>111111100001111000011110:thdm_0/B3B
>100000011110000111100001:thdm_0/B4
>111111100001111000011110:thdm_0/B5B
>100000011110000111100001:thdm_0/B6
>111111100001111000011110:thdm_0/B7B
```

|***
|*** LOAD THE MASK REGISTER ***
|***
CS=0 A2=0 A1=0 A0=1
>000011110000111100001111000011110000111100001111:D0
>000011110000111100001111000011110000111100001111:D1
>000011110000111100001111000011110000111100001111:D2
>000011110000111100001111000011110000111100001111:D3
>000011110000111100001111000011110000111100001111:D4
>000011110000111100001111000011110000111100001111:D5
>000011110000111100001111000011110000111100001111:D6
>000011110000111100001111000011110000111100001111:D7
>010101010101010101010101010101010101010101010101:WR
>011000000000000000000000000110000001111000011110:mix21_14/B
>100000000000000000110011000011110000111100011110:mix21_8/B
>000000011110000110000000011000000001111000011110:mix21_10/B
>000110011110000000000000110000000000111100011110:mix21_0/B
>000111111001111000000000111100000000111100011110:mix21_2/B
>111000000000110000000000000000000000111100011110:mix21_7/B
>000001100000000000000000111100110000111100011110:mix21_5/B
>100000000000000000111100000001100000011110001110:MOUT
|***
|*** LOAD THE REFERENCE REGISTER ***
|***
CS=0 A2=0 A1=0 A0=0
>000011110000111100001111000011110000111100001111:D0
>000011110000111100001111000011110000111100001111:D1
>000011110000111100001111000011110000111100001111:D2
>000011110000111100001111000011110000111100001111:D3
>000011110000111100001111000011110000111100001111:D4
>000011110000111100001111000011110000111100001111:D5
>000011110000111100001111000011110000111100001111:D6
>000011110000111100001111000011110000111100001111:D7
>010101010101010101010101010101010101010101010101:WR
>000111100000000000000000011110000111100001111 0:mix21_15/B
>100000011110000000111100000011110000111100011110:mix21_9/B
>000111100000000110011000011111100001111000011110:mix21_11/B
>111000011000000011110000000000011000011110001110:mix21_1/B

```
>00011110011111110000000000001111XX00001111000011110:mx21_3/B
>11100000011111110000001100000XX00001111000011110:mx21_6/B
>000001111111000000000111111000011000011110000011110:mx21_4/B
>1001111000011000011111100001111000011110000011110:ROUT
|***
|*** LOAD THE SHIFT REGISTER ***
|***
CS=0 A2=1 A1=0 A0=0
>01010101010101010101010101010101010101010101010101010101010101010
1010101010101010101010101010101010101010101010101010101010101010101
0101010101010101010101010101010101010101010101010101010101010101010
1010101010101010101010101010101010101010101010101010101010101010101
010101010101010101010101010101010101010101010101:CLK
>0000111100001111000011110000111100001111000011110000111100001111000111
100001111000011110000111100001111000011110000111100001111000011110001111
0000111100001111000011110000111100001111000011110000111100001111000011110
000111100001111000011110000111100001111000011110000111100001111000111100
001111000011110000111100001111000011110000111100001111:SIN
>1111111111111111111111111111111111111111111111111111XX111111111111
11111111111111111111111111111111111111111111111XX111111111111111111
1111111111XX1111111111111111111111111111XX11111111111111111111111111
1111111111111111111111111111111111XX1111110011110000111100001111000111
10000111100001111000011110000111100001111000111100:SOUT
14912 transistors, 8307 nodes (6 pulled up)
```

Simulation of the Sync Detection

In this part the reference, mask, and status registers are loaded with all zeros and the threshold register is loaded by a pattern such that the error tolerances are zero in both THRE1 and THRE2. Then the 128-bit shift register is loaded, first, with all zeros and then with all ones. Thus a noninverted sync and an inverted sync pulses are detected at the SYNOUT and INVOUT nodes.

```
initialization took 19654 steps
initialization took 64 steps
initialization took 7 steps
```

```
initialization took 0 steps
|***
|*** LOAD THE STATUS REGISTER ***
|***
CS=0 A2=0 A1=1 A0=1
>01:WR
>00:CLEAR
>10:SOE
>10:PRBS
>X0:M|S
>10:WSH
|***
|*** LOAD THE THRESHOLD REGISTER ***
|***
CS=0 A2=0 A1=1 A0=0
>0101:WR
>1000:thdm_0/B0
>1111:thdm_0/B1B
>0000:thdm_0/B2
>1111:thdm_0/B3B
>1000:thdm_0/B4
>1111:thdm_0/B5B
>1000:thdm_0/B6
>1111:thdm_0/B7B
|***
|*** LOAD THE MASK REGISTER ***
|***
CS=0 A2=0 A1=0 A0=1
>01010101010101010101010101010101:WR
>01100000000000000000000000001100000:mux21_14/B
>1000000000000000011001100001111000:mux21_8/B
>00000001111000011000000000110000000:mux21_10/B
>000110011110000000000000011000000000:mux21_0/B
>0001111110011110000000001111000000000:mux21_2/B
>111000000000110000000000000000000000:mux21_7/B
>00000110000000000000000011110011000:mux21_5/B
>1000000000000000011110000001100000:MOUT
```

|***
|*** LOAD THE REFERENCE REGISTER ***
|***
CS=0 A2=0 A1=0 A0=0
>01010101010101010101010101010101:WR
>0001111000000000000000000001111000:mux21_15/B
>10000001111000000011110000001111000:mux21_9/B
>00011110000000011001100001111111000:mux21_11/B
>11100001100000001111000000000011000:mux21_1/B
>000111100111111000000000001111XX000:mux21_3/B
>111000000111111000000011000000XX000:mux21_6/B
>00000111111000000000111111000011000:mux21_4/B
>1001111000011000011111100001111000:ROUT
|***
|*** LOAD THE SHIFT REGISTER AND OBSERVE THE SYNC PULSES ***
|***
CS=0 A2=1 A1=0 A0=0
>1010101010101010101010101010101010101010101010101010101010101010
101010101010101010101010101010101010101010101010101010101010101010
101010101010101010101010101010101010101010101010101010101010101010
101010101010101010101010101010101010101010101010101010101010101010
101010101010101010101010101010101010101010101010101010101010101010
101010101010101010101010101010101010101010101010101010101010101010
101010101010101010101010101010101010101010101010101010101010101010
10101010101010101010:CLK
>0000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000011111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111
11111111110000000000:SIN
>1111111111111111111111111111111111111111111111111111111XX11111111
11111111111111111111111111111111111111111111111XX1111111111111111
1111111111111XX111111111111111111111111111111XX111111111111111111111
11111111111111111111111111111111111111XX1100000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000

000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000011111111111:SOUT
>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX0000000000000000XXXXXXXXXXXXXXXX000000000
000000000000000000000000000000000000000XXXXXXXXXXXXXXXX00110000
00000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000001100000000:SYNOUT
>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXX0000000000000000XXXXXXXXXXXXXXXX000000000
000000000000000000000000000000000000000XXXXXXXXXXXXXXXX00000000
00000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000001100000000:INVOUT
14912 transistors, 8307 nodes (6 pulled up)

APPENDIX B

TEST PATTERN GENERATOR AND OUTPUT CHECKER PROGRAMS FOR THE 1'S COUNTER BLOCK

In this appendix two programs, **gen** and **check**, written in C language are presented. One of these programs generates the test vectors and the correct outputs corresponding to these vectors for the 1's counter combinational block. After an ESIM simulation is performed on the 1's counter using the vectors generated, the other program checks for any difference between the correct outputs and the ESIM outputs. If there is a difference between these two outputs (e.g. fault in the 1's counter block), it also gives some information useful for making diagnostic.

Test Pattern Generator Program

```
#include <stdio.h>
#define MAXSTG 7
#define MAXROW 128
#define MAXCOL 4225

int VARRAY[MAXROW][MAXCOL];

main()
{
   char *FILE1,*FILE2;
   int  n,x,y;

   printf(``***************************************\n");
   printf(``***************************************\n");
   printf(``***                                 ***\n");
   printf(``***  Test vector generator program  ***\n");
   printf(``***                                 ***\n");
   printf(``***             < gen >             ***\n");
   printf(``***                                 ***\n");
   printf(``***************************************\n");
   printf(``***************************************\n");
   printf(``!!! wait...\n");
   FILE1=``Tcount.vec1";
   FILE2=``Tcount.out1";

   for(n=1;n<=MAXSTG;n++)
   {
      x=power(2,n);
      y=power((x/2+1),2);
      printf(``<< Executing stage %1d >>\n",n);
      genvec(n,x,y,FILE1,FILE2);
      *(FILE1+10) +=1;
      *(FILE2+10) +=1;
   }
}

genvec(n,x,y,FILE1,FILE2)
char *FILE1,*FILE2;
int  n,x,y;
{
   int  i,j,k=0,m=0,CNT;
   FILE *fopen(),*fp;
   /*** INITIALIZE VECTOR ARRAY ***/
   for(i=0;i<x;i++)
      for(j=0;j<y;j++)
         VARRAY[i][j]=0;
```

```
/*** WRITE TEST VECTORS INTO VECTOR ARRAY ***/
for(j=0;j<y;j++)
{
   for(i=0;i<m;i++)
      VARRAY[i][j]=1;
   k=k+1;
   if(k==(x/2+1))
   {
      k=0;
      m=m+1;
   }
}
for(k=0;k<y;k=k+(x/2+1))
{
   m=(x/2);
   for(j=k;j<k+(x/2+1);j++)
   {
      for(i=(x/2);i<m;i++)
         VARRAY[i][j]=1;
      m=m+1;
   }
}
/*** WRITE VECTOR ARRAY INTO A FILE ***/
if(n==7)
{
   printf(``Generating vectors for %1d-bit adder...\n",n);
   FILE1=``Tcount.vec71";
   for(m=0;m<y;m=m+(y/5))
   {
      fp=fopen(FILE1,``w");
      fprintf(fp,``N");
      for(i=0;i<128;i++)
      {
         fprintf(fp,``\nV srmc_0/CP%1d ",i);
         for(j=m;j<m+(y/5);j++)
            fprintf(fp,``%1d",VARRAY[i][j]);
      }
      fprintf(fp,``\nG\n");
      fclose(fp);
      if(*(FILE1+11)==9)
      {
         *(FILE1+11)=1;
         *(FILE1+10)+=1;
      }
      else
         *(FILE1+11) +=1;
   }
}
```

```
else
{
   printf(``Generating vectors for %1d-bit adders...\n",n);
   fp=fopen(FILE1,``w");
   if(n==1)
   {
      fprintf(fp,``I\nI\nI\nI\n");
      fprintf(fp,``w 7.0 7.1B 7.2 7.3B 7.4 7.5B 7.6 7.7B");
   }
   else
      fprintf(fp,``N");
   k=0;
   for(i=0;i<128;i++)
   {
      fprintf(fp,``\nV srmc_0/CP%1d ",i);
      for(j=0;j<y;j++)
         fprintf(fp,``%1d",VARRAY[k][j]);
      k=k+1;
      if(k==x)
         k=0;
   }
   fprintf(fp,``\nG\n");
   fclose(fp);
}

/*** WRITE ACTUAL OUTPUTS INTO A FILE ***/
if(n==7)
{
   printf(``Calculating outputs for %1d-bit adder...\n",n);
   FILE2=``Tcount.out71";
   for(m=0;m<y;m=m+(y/5))
   {
      fp=fopen(FILE2,``w");
      for(j=m;j<m+(y/5);j++)
      {
         CNT=0;
         for(i=0;i<x;i++)
            CNT=CNT+VARRAY[i][j];
         CNT=CNT*(128/x);
         fprintf(fp,``%3d ",CNT);
      }
      fprintf(fp,``\n");
      fclose(fp);
      if(*(FILE1+11)==9)
      {
         *(FILE1+11)=1;
         *(FILE1+10)+=1;
      }
```

```
         else
            *(FILE2+11) +=1;
      }
   }
   else
   {
      printf(``Calculating outputs for %1d-bit adders...\n",n);
      fp=fopen(FILE2,``w");
      for(j=0;j<y;j++)
      {
         CNT=0;
         for(i=0;i<x;i++)
            CNT=CNT+VARRAY[i][j];
         CNT=CNT*(128/x);
         fprintf(fp,``%3d ",CNT);
      }
      fprintf(fp,``\n");
      fclose(fp);
   }
}

power(x,n)   /* raise x to n-th power; n>0 */
int x,n;
{
   int p;
   for(p=1;n>0;--n)
      p=p*x;
   return(p);
}
```

Output Checker Program

```
#include <stdio.h>
#define MAXSTG 7
#define MAXCOL 4225

main()
{
   char *FILE1,*FILE2,*FILE3;
   int  y,n,m;

   printf("********************************\n");
   printf("********************************\n");
   printf("***                          ***\n");
   printf("***  Output checker program  ***\n");
   printf("***                          ***\n");
   printf("***         < check >        ***\n");
   printf("***                          ***\n");
   printf("********************************\n");
   printf("********************************\n");
   printf("!!! wait...\n");
   FILE1="Tcount.esim1";
   FILE2="Tcount.out1";
   FILE3="Tcount.err1";
   for(n=1;n<=MAXSTG;n++)
   {
      y=power((power(2,n)/2+1),2);
      if(n==7)
      {
         printf("<< Executing for %1d-bit adder >>\n",n);
         y=y/5;
         FILE1="Tcount.esim71";
         FILE2="Tcount.out71";
         FILE3="Tcount.err71";
         for(m=1;m<=5;m++)
         {
            printf("-- Tcount.esim7%1d --",m);
            checker(y,FILE1,FILE2,FILE3);
            *(FILE1+12) +=1;
            *(FILE2+11) +=1;
            *(FILE3+11) +=1;
         }
      }
      else
      {
         printf("<< Executing for %1d-bit adders >>\n",n);
         checker(y,FILE1,FILE2,FILE3);
         *(FILE1+11) +=1;
```

```
\setlength{\baselineskip}{5mm}
         *(FILE2+10) +=1;
         *(FILE3+10) +=1;
      }
   }
}

checker(y,FILE1,FILE2,FILE3)
char *FILE1,*FILE2,*FILE3;
int  y;
{
   char c;
   int i,j,k,n,CNT,SUM,ERR[MAXCOL],ARRAY[8][MAXCOL];
   FILE *fopen(),*fp;
   /*** INITIALIZE ARRAY ***/
   for(i=0;i<8;i++)
      for(j=0;j<MAXCOL;j++)
         ARRAY[i][j]=0;

   /*** WRITE ESIM OUTPUTS INTO ARRAY ***/
   fp=fopen(FILE1,"r");
   if(fp!=NULL)
   {
      printf("Reading esim outputs...\n");
      fscanf(fp,"%c",&c);
      while(c !='>')
         fscanf(fp,"%c",&c);
      for(i=0;i<=8;i++)
      {
         for(j=0;j<y;j++)
         {
            fscanf(fp,"%1d",&n);
            ARRAY[i][j]=n;
         }
         if(i !=8)
         {
            fscanf(fp,"%c",&c);
            while(c !='>')
               fscanf(fp,"%c",&c);
         }
      }
      fclose(fp);

      /*** COMPARE ESIM OUTPUTS WITH ACTUAL OUTPUTS ***/
      fp=fopen(FILE2,"r");
      if(fp!=NULL)
      {
         printf("Checking esim outputs ...\n");
```

```
        k=0;
        for(j=0;j<y;j++)
        {
           SUM=0;
           for(i=1;i<=7;i=i+2)
           {
              if(ARRAY[i][j] == 1)
                 SUM = SUM+power(2,i);
           }
           for(i=0;i<=8;i=i+2)
           {
              if(ARRAY[i][j]==0)
                 SUM = SUM +power(2,i);
           }
           fscanf(fp,"%3d",&CNT);
           if(SUM!=CNT)
           {
              ERR[k]=j;
              k+=1;
           }
        }
        fclose(fp);
        if(k>0)
        {
           printf("!!! ERROR :: It is written on %s\n",FILE3);
           fp=fopen(FILE3,"w");
           for(i=0;i<k;i++)
              fprintf(fp,"error found in column %d\n",ERR[i]);
           fclose(fp);
        }
        else
           printf("!!! No errors found\n");
     }
     else
        printf("\n:: File %s not found\n\n",FILE2);
  }
  else
  printf("\n:: File %s not found\n\n",FILE1);
}

power(x,n)   /* raise x to n-th power; n>0 */
int x,n;
{
   int p;
   for(p=1;n>0;--n)
      p=p*x;
   return(p);
}
```