AN INTEGRATED LOW-COST FUNCTIONAL TESTER FOR CMOS LOGIC

By

William Low

B. A. Sc., University of British Columbia, Canada, 1991

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

in
THE FACULTY OF GRADUATE STUDIES
ELECTRICAL ENGINEERING

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

1993
© William Low, 1993
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

(Signature)

Department of Electrical Engineering
The University of British Columbia
Vancouver, Canada

Date Dec 29/1993
Abstract

This thesis focuses on improving the quality of tests performed by low-cost testers for Very Large Scale Integration (VLSI) chips. The testing of timing parameters become increasingly important with higher performance technology. Circuits that operate correctly at low speeds may fail at higher speeds because of timing problems. Most low-end test systems lack the performance required to conduct the timing diagnostics needed to perform more advanced testing of modern VLSI chips.

Areas of improvement in low-cost test systems include increasing vector memory, test speed, or I/O timing and wave formatting capability. Increasing I/O timing and wave formatting capability provides a good compromise of improved tester functionality at a reasonable cost. These features provide considerably more improvements in test quality than simply increasing the raw test speed (or test pattern rate).

A test strategy was developed that exploits this improved timing and I/O waveform formatting capability to generate short high-speed clock bursts which can be used to perform testing at speeds equivalent to a much higher rate than the tester’s pattern rate. This strategy can be used to test many digital designs. However, using this strategy for “high-speed” testing requires more effort in test pattern development and more test vectors than an equivalent test performed on a high-speed tester. This approach can also be used in existing high-end testers that already have the required timing and waveform formatting capabilities.

A functional tester system for CMOS logic was developed that provides these improved timing and I/O formatting features. This system integrates most of tester circuits into modular functional tester chips (FTCs) which are used in parallel to form a tester. Vector encoding combined with an internal lookup table was used to reduce the memory bandwidth required to support the increased functionality.

A single-channel FTC was designed and implemented on a 3.8mm × 2.9mm die and requires
only 40-pins which gives considerable allowance for future increases in memory width and channels. Measurements from earlier prototypes of the FTC waveform formatter give worst case timing resolutions of 1ns and maximum rise-fall skews of 1-2ns. Worst case skews between devices were on the order of 1ns. These results are very reasonable considering the relatively conservative standard cell design using 1.2μm CMOS.
# Table of Contents

Abstract ii  
List of Tables vii  
List of Figures ix  
Acknowledgements xi  

1 Introduction 1  
1.1 Background on VLSI Testing ................................................. 1  
1.1.1 Design for Testability .......................................................... 2  
1.1.2 IC Testers .............................................................................. 4  
1.2 Thesis Objective and Organization .............................................. 5  

2 Functional Testing and System Specifications 7  
2.1 Functional Testing ...................................................................... 7  
2.1.1 Waveform Driving and Sensing ................................................ 7  
2.1.2 An Example ............................................................................. 10  
2.2 Functional Tester System Specifications ....................................... 13  
2.2.1 Limited High-Speed Testing ...................................................... 15  

3 Architectures and System Design 19  
3.1 Tester Architectures .................................................................... 19  
3.1.1 Shared Resource Architecture .................................................. 21  
3.1.2 Tester Per-Pin Architecture ...................................................... 22
3.2 Exploiting VLSI Technology .................................................. 23
  3.2.1 Systems Without Wave Formatting .................................. 24
  3.2.2 Systems With Wave Formatting ...................................... 26

3.3 System Design ................................................................. 27
  3.3.1 Functional Tester System Overview ................................. 27
  3.3.2 System Operation ....................................................... 28
  3.3.3 Functional Tester Chip ................................................ 29
  3.3.4 Board Layout ............................................................ 29
  3.3.5 Pin Count and Vector Depth ........................................ 31
  3.3.6 Speed and Bandwidth .................................................. 32

4 Functional Tester Chip ...................................................... 34
  4.1 Overview ......................................................................... 34
  4.2 Control .......................................................................... 34
  4.3 Registers ......................................................................... 36
  4.4 Memory Interface ............................................................ 36
  4.5 Clock Module ................................................................. 38
  4.6 Test Data Generation ....................................................... 38
    4.6.1 Format Memory .......................................................... 38
    4.6.2 Instruction Encoding and Decoding ............................... 39
    4.6.3 Pipelining .................................................................. 41
  4.7 Waveform Formatting and Timing ....................................... 42
    4.7.1 Timing Generation ...................................................... 42
    4.7.2 Propagation Delay Sensitivities .................................. 44
    4.7.3 Formatting ............................................................... 46
    4.7.4 Dead-Zones ................................................................ 46
    4.7.5 Skew ........................................................................ 48
## 4.8 Input Comparison .......................................................... 49
## 4.9 Pin Electronics .............................................................. 51

## 5 Implementation and Results ............................................. 52
### 5.1 Implementation .......................................................... 52
#### 5.1.1 FTC Area ............................................................. 52
### 5.2 Performance .............................................................. 55
#### 5.2.1 Timing Resolution .................................................. 56
#### 5.2.2 Timing Skew ........................................................ 59

## 6 Conclusions and Future Work .......................................... 67
### 6.1 Conclusions .............................................................. 67
### 6.2 Future Work ............................................................. 68

### Bibliography .............................................................. 70

### A FTC Pin Description .................................................... 74

### B WFC Measurements ..................................................... 76

### C FTC Implementation Details ......................................... 78
#### C.1 FTC Circuit Documentation ........................................ 78
#### C.2 FTC Timing ............................................................ 96
#### C.3 FTC Circuit Schematics ............................................. 98
List of Tables

2.1 Example test vectors for testing flip-flop. ........................................... 11
2.2 Example timing parameters for flip-flop. ........................................... 11
3.1 Characteristics of VLSI technologies. .................................................. 23
4.1 FTC Modes. ......................................................................................... 35
4.2 Address space of FTC registers. ............................................................ 37
4.3 Format memory word fields. .................................................................. 39
4.4 Test data stored per byte for scheme A instruction distributions. .......... 41
4.5 Test data stored per byte for scheme B instruction distributions. .......... 41
4.6 Pulse generator 0 triggering conditions. .............................................. 47
4.7 Pulse generator 1 triggering conditions. .............................................. 47
4.8 Leading edge skew. ............................................................................. 49
4.9 Trailing edge skew. ............................................................................ 49
5.1 Bidirectional pads available from CMC’s CMOS4S cell library. ............ 54
5.2 Simulated FTC edge placement times in ns (normalized). ..................... 57
5.3 Average WFC edge placement times for a rising edge at t₁ in ns (normalized) . 58
5.4 Rise-fall skew measurements ............................................................... 63
5.5 Rise-fall skew calculations ................................................................. 63
A.1 FTC pinout. ....................................................................................... 75
B.1 Rising edge at t₁. ............................................................................... 76
B.2 Falling edge at t₁. .............................................................................. 76
B.3 Rising edge at $t_2$. ......................................................... 77
B.4 Falling edge at $t_2$. ......................................................... 77

C.1 Memory map of address space registers in the caregmod module. ...................... 80
C.2 Input signals for chmod. ....................................................... 81
C.3 Output signals for chmod. ..................................................... 81
C.4 Control signals generated by instruction decoder. ............................................ 86
C.5 Control signals for executing instructions. ......................................................... 86
C.6 Address decoding in modebits module. ......................................................... 88
C.7 Truth table for generating SP0.D in pulsegen0 module. ................................. 89
C.8 Karnaugh map for generating SPO.D in pulsegen0 module. ............................ 89
C.9 Truth table for generating WP0.D in pulsegen0 module. .................................... 90
C.10 Karnaugh map for generating WPO.D in pulsegen0 module. .......................... 90
C.11 Truth table for generating SP1.D in pulsegen1 module. ................................... 91
C.12 Karnaugh map for generating SP1.D in pulsegen1 module. ............................ 91
C.13 Truth table for generating WP1.D in pulsegen1 module. .................................. 92
C.14 Karnaugh map for generating WP1.D in pulsegen1 module. ............................ 92
List of Figures

2.1 Waveform formats ................................................... 8
2.2 Comparison methods................................................. 9
2.3 Flip-flop test example ............................................. 10
2.4 Flip-flop test waveforms ......................................... 12
2.5 Flip-flop formatted test waveforms .............................. 13
2.6 Generating clock waveforms ....................................... 16
2.7 State machine model of static synchronous circuit .......... 17

3.1 Functional block diagram of a generic tester .................. 21
3.2 Generic channel electronics in systems without waveform formatting ........................................ 25
3.3 MacTester channel multiplexing .................................. 25
3.4 Block diagram of DGR Architecture ............................. 26
3.5 Test system architecture .......................................... 28
3.6 Pin driver/receiver placement .................................... 30

4.1 Block diagram of FTC .............................................. 35
4.2 Address space example ............................................ 38
4.3 Vector encoding .................................................... 40
4.4 FTC pipeline ....................................................... 42
4.5 Functional block diagram of timing generator ................ 42
4.6 Fine vernier implementations .................................... 43
4.7 Functional block diagram of waveform formatting circuit .. 46
4.8 Dead-zones ......................................................... 48
4.9 Functional block diagram of input comparison circuit. 

4.10 Comparison result output encoding. 

5.1 FTC chip layout. 

5.2 Delay line macrocell layout. 

5.3 Timing measurement locations. 

5.4 Simulated and measured delays. 

5.5 Interchannel skew measured for rising and falling edges at $t_1$. 

5.6 Deskewed placement times for rising edges at $t_1$. 

5.7 Rise-fall skew at $t_1$. 

5.8 Rise-fall skew at $t_2$. 

5.9 Rise-fall deskew circuit. 

C.1 Timing diagram showing the fetching of instructions to the decoding stage.
Acknowledgements

I would like to take this opportunity to sincerely thank my supervisor, Dr. André Ivanov, for his guidance, support, patience, and encouragement throughout the course of this work. I must also thank Yuejian Wu, one of my supervisor’s Ph. D. students, for originally suggesting this research project to my supervisor and for his many enlightening discussions on VLSI testing. I would like to thank Dr. Rick Hobson for taking the time to show me the operation of the IMS tester and for his thoughts on IC testers. I would also like to thank Jeffery Chow, Barry Tsuji, Peter Bonek, Robert Lam, Oswaldo Antezana, and Andrew Bishop for their contributions of various resources, and helpful discussions and suggestions. Special thanks goes to Dave Gagne for his help and patience in suggesting and providing solutions to my VLSI problems. The work reported here would not be possible without the facilities and resources for integrated circuit fabrication generously provided by the Canadian Microelectronics Corporation and Northern Telecom Limited respectively. I would like to thank the many faculty, staff, and students who made my experience at UBC interesting and pleasant. Finally, I thank my family for their constant support and encouragement over the past two years.
Chapter 1

Introduction

Improvements in semiconductor technology, transistor densities, and manufacturing yields are providing more powerful Very Large Scale Integration (VLSI) technology at less cost. Coupled with advances in computer-aided design (CAD) tools, the technology to produce fairly complex integrated circuits (ICs) has become more accessible. As a result, chipsets and Application Specific Integrated Circuits (ASICs) are being used more frequently to implement circuits that once required discrete components. Despite these improvements, VLSI design and fabrication remains an imperfect process. Consequently, testing continues to be an important step in the design and manufacture of VLSI chips that is necessary to ensure a working final product.

1.1 Background on VLSI Testing

The purpose of VLSI testing is to detect faulty chips. Faults in a VLSI chip may arise from many sources. For example, fabrication defects and contamination may cause faults such as shorts or opens, excessive leakage currents, etc. [Director90]. Packaging and subsequent handling may introduce additional faults.

Unfortunately, an IC containing faults may not reveal its defective nature very easily. In the past ICs were much simpler and it was feasible to verify that a chip was good simply by exhaustively testing that it performed its specified function. This strategy, known as behavioral [Levitt92] (or functional) testing, does not scale well with the higher pin counts and integration densities of modern ICs [Levitt92]. Instead of exhaustively testing a chip as a single functional unit, exploiting the structure of the circuit elements within the chip can lead to significantly shorter pseudoexhaustive tests that are equivalent to the exhaustive tests [Abramovici90, page 305].

1
Chapter 1. Introduction

The detection of faults in a VLSI chip is accomplished by stimulating the chip inputs so that potentially faulty circuits inside the chip produce a result at the chip's outputs that indicates whether the circuit is operating correctly. Therefore, controllability and observability of a chip's internal circuit nodes is important for making a chip testable [Abramovici90, pages 343-358]. One of the main difficulties with VLSI testing is that nodes deep within a chip are not so easily controlled and observed from the available I/O pins. Much computer time is needed to generate test patterns (or test vectors) that could stimulate and observe these difficult to reach nodes to achieve a high coverage of potential faults. These problems continue to grow as researchers strive to achieve higher speeds and integration densities. The cost of allowing defective parts to pass can be very expensive. It is often cited that the cost of finding a defective component increases by a factor of 10 for each level of assembly, i.e., from device to board level, system level, and field installation [Levitt92]. The recognition of these difficulties have caused IC designers to more seriously consider test issues and design for testability techniques [Williams83].

1.1.1 Design for Testability

Design for testability (DFT) techniques make an IC more testable by improving its controllability and/or observability. Today, many VLSI chips are employing design for testability techniques to facilitate testing. (E.g., Intel 80386 [Gelsinger87], Motorola 68340 [Bishop90], etc.) There are many DFT schemes available to a designer [Williams83, Levitt92]. Two of the more established techniques are scan-path design [Abramovici90, pages 358-382][Eichelberger78] and built-in self-test (BIST) [Abramovici90, pages 457-539][McCluskey81].

Scan design is a technique used to simplify the testing of sequential circuits. This method uses special latches or flip-flops that provides an additional set of data and clock (or control) inputs. Essentially, these special memory elements provide "dual input ports." One set of inputs is used for normal circuit design. The other set of inputs is used to connect the memory elements into one long shift register (or scan chain). During normal operation, the regular latch inputs are used and the circuit performs its normally specified function. During test mode, the alternate latch inputs
are used and the memory elements act as a shift register. Any internal state can be initialized by shifting the appropriate state bits into the memory elements. Similarly, any internal state can be observed by shifting the state bits out of the memory elements. Thus, complete controllability and observability is achieved. The problem of testing a sequential circuit is reduced to one of testing combinational logic and a shift register. The costs associated with this scheme are in the area required for the additional circuits, routing, and pins. In addition, the performance of the flip-flop may be degraded. The serial nature of this test strategy results in a relatively slow test.

The idea behind BIST is to provide additional circuits in a design that are capable of testing the design from within the chip. The term BIST generally includes a wide variety of test strategies that satisfies this criterion. The BIST circuits generate the test stimulus and evaluate the response of the circuit under test. Being inside the chip, these test circuits have the advantage of being better able to control and observe nodes that are difficult to reach from outside. Typically, pseudorandom techniques are used to efficiently generate test stimulus and to compress the test results [Bardell87]. Generally, a BIST test is performed by providing a specified number of clock cycles followed by go/no-go comparison of the test result. Testing performed under BIST is generally very fast. The main cost associated with BIST is the increased area.

CrossCheck [Chandra93] is a recently introduced ASIC DFT strategy based on a proprietary technology. This scheme employs an embedded test matrix to provide massive controllability and observability. Besides the limited availability of this technology due to licensing, a drawback associated with CrossCheck is the area overhead [Levitt92]. For gate bound designs, the test structures could occupy 25% of the core area while routing bound designs may experience an overhead of 12% to 20% [Chandra93].

There are many DFT strategies available. In practice, a design may employ a combination of formal and application specific DFT methods, e.g. Intel 80386 [Gelsinger87].
1.1.2 IC Testers

Testing an integrated circuit involves three major steps: i) the generation of suitable test patterns, ii) the application of these patterns to the device under test (DUT), and iii) the analysis of the test response from the DUT. The first step, test pattern generation, is usually CPU intensive and is dependent on the DFT strategy. These test patterns are usually prepared ahead of time, although some test systems provide algorithmic pattern generators for special applications such as memory testing [Kikuchi89].

The last two steps, test pattern application and test response evaluation, involve the use of an IC tester. Electrical tests applied to a DUT could be classified into parametric (or DC) tests and functional (or AC) tests [Feugate88]. Parametric tests measure the analog quantities (voltages and currents) at the pins of the device under test (DUT). Typically, these tests measure leakage currents, output voltage levels, and check for shorts and opens at the pins. Parametric tests usually take place at fairly low speeds because of the relatively long settling times of the precision instruments involved and the slow switches used to connect these instruments to the DUT pins.

In the context of IC testers, the term functional test refers to the high-speed operating mode of a tester. It does not imply anything about the test methodology mentioned earlier. To avoid confusion, we will refer to the functional test methodology as behavioral testing. Functional tests verify the correct operation of a DUT. This involves driving the DUT inputs with the appropriate logic stimulus and comparing its outputs with the expected response. Preferably test patterns are applied at the DUTs normal operating speed which allows testing of timing critical parameters such as propagation delay, setup/hold times, and maximum frequency.

The set of tests to be performed on the DUT is usually described using a test language (TL). Support for generating TL instructions range from highly automated, that is, mostly generated by software (e.g., [Organ91], [Walter88]) to manual writing of fairly low-level code. ATLAS, an IEEE standard test language[ATLAS84], has been published, but it has not been widely adopted for IC testers.
VLSI testers are available with various capabilities and performance. The specific requirements for a test system depends much on its intended use. For example, in a manufacturing environment, test throughput is an important criterion in addition to other requirements such as timing and input/output (I/O) capability. In [Bassett90], IBM (International Business Machines Inc.) reports producing a tester with a parametric measurement unit (PMU) dedicated to each pin for the purpose of increasing test throughput. The requirements for a design-prototyping environment are less demanding, however, test quality is an important issue in both environments.

With the increasing speeds and densities of VLSI chips, the testers that are capable of testing these high performance devices are becoming increasingly expensive and complex. High-end IC testers are large multi-million dollar machines with per-pin costs of $6000-$8000 [LaBuda90] [Bassett90]. Unfortunately, these high-priced testers that are needed to thoroughly test most VLSI chips are economical only for high volume users such as semiconductor manufacturers. Users without access to these testers could either rely on the chip manufacturer to perform the required tests, or use a more affordable, but inferior lower-end test system. Contracting the manufacturer to perform production testing is fine, but it is less practical for design debugging and diagnosis where a high degree of interaction is needed with a tester.

Low-end IC testers typically cut costs by providing smaller pattern memories, lower test speeds, and minimal timing and input/output (I/O) capability. A long test can usually be partitioned into several smaller tests that would fit into a small pattern memory. The reduced throughput because of pattern memory reloading is usually not a problem for prototype diagnostics. However, the remaining deficiencies degrade the test quality.

1.2 Thesis Objective and Organization

The this thesis focuses on improving the quality of functional tests provided by low-cost VLSI test systems. We show that increasing timing and I/O capability provides greater improvements in test quality than simply increasing the test speed. To exploit these capabilities, a strategy to
perform limited high-speed testing was developed. A simple tester architecture that supports these enhancements was designed. A key component of this test system, the integrated functional tester chip, was designed and implemented.

The remainder of this thesis is organized as follows. Chapter 2 provides more detailed background on functional testing using IC testers. The requirements for improved test quality in a low-cost test environment are developed. We examine some of the capabilities and limitations of this functional tester system. In Chapter 3, we review conventional tester architectures and related work in this area. An architecture for the functional tester system is developed. The requirements of the Functional Tester Chip (FTC) are given. Chapter 4 gives an overview of the design and implementation of the FTC. Simulated and experimental results for part of the FTC implementation are presented in Chapter 5. Conclusions and directions for future work are given in Chapter 6.
2.1 Functional Testing

The characteristics of the patterns required to test a DUT depends on the test strategy and the DFT features used. For example, a chip that uses scan design typically has most test activity confined to a few pins on the device [Tsui87] whereas a device using BIST may only require a number of clock cycles to operate the test followed by a “go/no-go” comparison to check the results. In order to test a device, a functional tester provides an interface consisting of a number of channels (or pins) that connect to the DUT. Through these channels, test waveforms are applied and the DUT’s responses are sensed.

2.1.1 Waveform Driving and Sensing

Waveforms to be generated by a tester are usually specified in segments of a given test period (or test cycle time). The length of the test period is programmable and usually remains fixed for the duration of a test. The speed of a tester is the frequency corresponding to the shortest test period that could be generated by the tester. Since transitions in the applied waveform may not necessarily fall on test period boundaries, the waveforms generated for each test period are usually formatted [Feugate88] to allow more flexibility in edge placement. The logic levels of formatted waveforms can be programmed to change at programmable time intervals following the start of the test period. Typical output formats provided by functional testers are shown in Figure 2.1a-d and described in the following [Feugate88]:
a) **Return low**: During the interval $t_1$ to $t_2$, the channel is driven with the logic level specified for the current test period. Outside this interval, the channel is driven with a low logic level.

b) **Return high**: During the interval $t_1$ to $t_2$, the channel is driven with the logic level specified for the current test period. Outside this interval, the channel is driven with a high logic level.

c) **Return complement**: During the interval $t_1$ to $t_2$, the channel is driven with the logic level specified for the current test period. Outside this interval, the channel is driven with the opposite logic level.

d) **No return**: The channel is driven with the logic level at the end of the previous test period up to time $t_1$. From $t_1$ to the end of the current test period, the channel is driven with the logic level specified for the current test period. The $t_2$ timing specification is not used.

---

**Figure 2.1: Waveform formats.**
Figure 2.1e shows how formatted waveform segments are used together to form the desired waveform. The test period must be chosen so that it partitions the desired waveform into segments containing one or two transitions that can be reproduced by the waveform formatter. Low-end testers usually provide only the NR format. A tester without waveform formatting essentially provides NR with \( t_1 \) set at the beginning of the test period.

Input comparison is performed using edge or window strobing (or comparison) [Feugate88]. Figure 2.2 illustrates the difference between the two comparison methods. Edge strobing samples the DUT output at one time instance (\( t_1 \)) within a test period while window strobing tests the DUT output over a time interval (\( t_1 \) to \( t_2 \)) or window. As shown with the third comparison in Figure 2.2, edge strobing is more vulnerable to producing erroneous comparisons with unstable output signals. Window comparison is usually provided in higher-end testers because of the additional resources required to specify and generate the window. Generally, only one logic level could be checked per test period because only one edge or window strobe is available. Moreover, only one vector value is associated with each test period for output driving or input comparison.

Figure 2.2: Comparison methods.
Chapter 2. Functional Testing and System Specifications

There are generally a number of constraints governing the relationship between the test period and the placement of $t_1$ and $t_2$. These constraints arise from hardware limitations and implementation issues that are tester dependent. For example, one fundamental limitation of input window comparison is that any changing of the expected value while the window is active will cause a comparison error.

2.1.2 An Example

As an example of functional testing, consider testing the flip-flop shown in Figure 2.3a. (The set and reset signals were omitted to simplify the example.) A test to determine whether the flip-flop could latch a logic 1 might consist of the test vectors given in Table 2.1. Each row under the vector column is a test vector. The test period column gives the period during which a test vector is active. The output vectors drive the flip-flop inputs while the compare vectors check the flip-flop outputs.

Before these test vectors could be used, more timing information about the $Q$ output is needed. The difficulty lies in determining when to test $Q$ for the latched $D$ input. $Q$ could not be checked at the rising edge of $clk$ because of propagation delay. Checking $Q$ just before the end of the $clk$ period may pass most flip-flops, but the resulting test criterion is not stringent enough to guarantee the operation of common circuit configurations of the type shown in Figure 2.3b. Ideally, the test waveforms should adequately stress the timing requirements of the DUT.

Table 2.2 gives the timing parameters needed for a more comprehensive test of the flip-flop.
Chapter 2. Functional Testing and System Specifications

<table>
<thead>
<tr>
<th>Test Period</th>
<th>Vectors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Output</td>
<td>Compare</td>
</tr>
<tr>
<td></td>
<td>clk</td>
<td>D</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.1: Example test vectors for testing flip-flop.

Figure 2.4 gives the more detailed set of test waveforms derived using these timing parameters. The actual flip-flop output is given by $Q_o$. The required flip-flop output is given by $Q_c$. For clarity, the minimum clk pulse width was not used. In using a slower clk speed, the waveforms also illustrate the situation that arises when a tester is too slow to test a device at its maximum speed.

<table>
<thead>
<tr>
<th>Description</th>
<th>Parameter</th>
<th>Typical (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propagation delay</td>
<td>$t_{pd}$</td>
<td>31</td>
</tr>
<tr>
<td>Minimum Clock Pulse Width</td>
<td>$t_w$</td>
<td>21</td>
</tr>
<tr>
<td>Setup Time</td>
<td>$t_{su}$</td>
<td>14</td>
</tr>
<tr>
<td>Hold Time</td>
<td>$t_h$</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.2: Example timing parameters for flip-flop.

The timing used for the $D$ input is designed to test the specified setup and hold times of the flip-flop. To test the setup time, the value to be latched is present at the $D$ input only for a time of $i_{su}$. Although the hold time is zero, the $D$ input could not change reliably at the same time instance as the rising edge of clk. To ensure that erroneous operation does not arise from the slight skew between $D$ and clk, the $D$ transition used to test the hold time takes place shortly after the rising clk edge. Testing the $Q$ output takes place after the specified propagation delay, $t_{pd}$.

---

1Based on SN74HCT74 Dual D-Type Positive-Edge-Triggered Flip-Flops with Clear and Preset by Texas Instruments.
Figure 2.4: Flip-flop test waveforms.

The actual propagation delay is probably less than this value. A guard band is needed between comparison regions of differing logic levels to allow the comparison hardware time to settle on the new value.

The test waveforms in Figure 2.4 could be specified using the appropriate output formats and input comparisons defined in Figures 2.1 and 2.2 respectively. The first step involves choosing a test period. Setting the test period equal to the clk period would be ideal for generating the clk and D signals as shown in Figure 2.5a. However, performing the input comparison for Q in test period 1 presents a problem because both the high and low logic levels could not be checked in one test period. A solution to this problem is given in Figure 2.5b. By choosing a test period equal to the clk phase, the applied waveform is still divided into segments containing only one or two transitions, but the output comparison now checks only one logic level per test cycle. Once the test period is established, the appropriate formats, timing, and vector values to be used in each test period could be chosen.

A complete test of a device may involve several functional tests because the tester may not have the resources (e.g., vector memory, timing generators, etc.) to complete the test in one operation. Each additional functional test requires reinitialization of some part of the tester. In
2.2 Functional Tester System Specifications

Clock rate is a frequently quoted performance figure for IC testers. However, as shown in the example in the previous section, this figure alone is insufficient as a measure of test capability. Other factors, such as waveform formatting and comparison methods are also important. In developing the specifications for an more effective low-cost functional test system, we want to concentrate on the features that provide the most test capability while maintaining reasonable cost. There are three main parameters to consider:

1. **Vector Memory** - The main advantage offered by increasing the vector memory size is the accommodation of longer tests without memory reloading. However, long tests can usually be partitioned into several shorter tests. As long as a workable amount of memory is available,
increasing this amount provides little gain in test capability. Large memories also increase the size of the test system and require more support hardware to implement.

2. **Tester Speed** - Increasing the speed of a tester provides some improvement in test capability mainly in the form of higher edge resolution and increased test speed. The difficulty with this approach is that the entire test system must operate at a higher speed. The improvement in edge resolution does not seem to justify this additional cost. For example, a tester with a 40MHz test frequency can provide a 25ns test period which is still fairly coarse for detailed timing tests. The memory system supporting this tester probably requires an access time of less than 25ns which requires either expensive static RAMs or memory interleaving schemes.

3. **I/O Timing and Formatting** - Efforts to improve I/O timing and formatting yields the most improvement in test capability at a modest cost. As shown in the example in Section 2.1.2, I/O timing and formatting flexibility is essential for performing detailed timing tests. Section 2.2.1 elaborates more on some of the possibilities of this test system. The main costs of this feature are in the increased complexity and the additional memory width needed to encode the desired formatted waveforms. Increased complexity can be accommodated in a VLSI chip. However, the increase in memory width reduces the potential channel density.

Providing a high degree of I/O timing and formatting flexibility seems to give the best enhancement in test capability at a reasonable cost. The specific requirements of the *functional tester system* (FTS) established here are:

- output waveform formatting (RC, RL, RH, and NR)
- input window comparison
- format and timing changes on-the-fly
- timing resolution should allow a wide selection of edge placements (perhaps a resolution of 1-3ns)
Chapter 2. Functional Testing and System Specifications

Other specifications include a reasonable memory size of approximately 16k vectors/channel, support for 48-64 channels, and test speed of 5MHz. We now examine the capabilities and limitations of this test system based on these specifications.

2.2.1 Limited High-Speed Testing

Clock Signals

One of the most basic signals generated by a tester is a clock waveform. As shown in Figure 2.6, there are several ways in which a tester could generate clock signals. In Figure 2.6a, the clock signal is generated using a NR format. This example is representative of the capabilities of a low-cost tester with limited output formats and timing flexibility. Even if the tester allows delaying the edges by a fixed time, as shown in Figure 2.6b, the frequency of this clock is still only half the tester frequency because of the format used.

Once waveform formatting is provided, considerably more edge placement flexibility is available and clock speeds from 1 to 1.5 times the test frequency could be generated as shown in Figures 2.6c and d. Providing waveform formatting alone allows 50% higher speed tests to be performed. From a diagnostics point of view, the purpose of using a higher frequency clock is to reveal faults arising from clock skew, propagation delay, and other timing critical factors. These problems may not appear during low-speed testing when there is adequate time between clock edges to allow correct circuit operation.

Strategy for Limited High-Speed Testing

So far, 50% duty cycles have been assumed for the clock signals. If symmetric clocks are not required, it is possible to generate short bursts of high frequency clock cycles by exploiting on-the-fly format and timing changes as shown in Figure 2.6e. Each pair of high frequency clock pulses is followed by one slow clock cycle. The frequency of the fast clock cycles are determined by the edge placement resolution and the minimum pulse width that could be generated by the tester. The
slow clock cycle has a frequency of approximately half the tester speed.

Generating these high-speed "bursty" clock and data signals enables a tester to perform potentially more stringent circuit tests. For example, consider testing a static synchronous ASIC design. Static synchronous designs are based on edge-sensitive, single-phase clocking schemes. All storage elements in the design are sensitive to either the rising or falling edge (not both) of a common clock signal. The clock period is longer than the settling time of the circuit. If the clock were to stop, the circuit would remain in its current state as long as power were maintained. An excellent description of this common design methodology is given in [Naish88].

The operation of a static synchronous circuit could be abstractly represented using a state machine as shown in Figure 2.7. A state transition could occur during each active clock edge. A functional test for this circuit would require testing that each transition takes place correctly at the circuit’s operating speed. Since the circuit is static (i.e., its state is retained when the clock is
stopped), the timing of each state transition is independent of other state transitions. The question to be answered through testing is: "From a given starting state, could a correct state transition occur using a particular clock period?"

![Block diagram of circuit shown as state machine.](image1)

![State diagram of circuit showing fast(F) and slow(S) state transitions due to applied test vectors.](image2)

Figure 2.7: State machine model of static synchronous circuit.

Figure 2.7b shows the state transition that occurs when the circuit is stimulated with bursty high-speed test patterns and clocking. Although one third of the transitions take place under a slow clock, additional test passes could be used to cover the slowly clocked transitions from the initial test. The key requirement of this scheme is that the circuit remain static. However, it may be possible to use this technique on dynamic circuits provided that the slow clock meets the minimal speed required for the circuit to retain its state.

**Limitations**

A limitation of many test systems, including the FTS, is the disparity between the capability of the output formatting and input comparison. In the FTS, up to three output transitions (1.5 clock cycles) could be generated per test period while only one input comparison could be performed. This problem arises because more timing resources are required to perform input comparison than are required to generate output waveforms. Using window comparison, six timing edges are required...
to test the three output transitions (edge strobing requires three timing edges).

Fortunately, most tests do not require three comparisons per test period. If the three transitions are used to generate 1.5 clock cycles and one result is generated per clock cycle, then only two comparisons are required per test period as shown in Figure 2.6f. This requirement still exceeds the capability of the input comparison by one comparison. Discarding the comparison of the slow clock cycle results still leaves the problem of having two comparisons on every second test period.

Figure 2.6g gives a reasonable compromise to this difficulty. We allow full flexibility in the comparison of the first result from the fast clock. The result from the second fast clock period is compared after one phase of the slow clock. The result from this second compare is valid because the timing stress comes from the application of short clock periods. The correct circuit state should be retained as long as the correct results are latched by the circuit.

This test strategy can be used in many digital designs, but more effort is needed to develop suitable test patterns to exploit the high-speed clock bursts. A larger set of test vectors, compared to conventional testing, is also needed because multiple test passes are required to produce one "equivalent" high-speed test pass. The DUT operating under this irregular clocking will have different levels of power dissipation and current consumption from a DUT operating under continuous high-speed. Therefore, problems arising from these sources may not be detected using this test approach. Despite these difficulties, this test strategy at least allows a low-cost test system to perform some limited high-speed testing that would otherwise require a much more expensive test system.
Chapter 3

Architectures and System Design

The previous chapter reviewed functional testing and defined some common I/O formats provided by testers. The desired specifications of the FTS, a low-cost test system, were given and the capabilities and limitations of this system were examined. In this chapter, we examine the architecture of an IC tester in more detail and we review some related work. We then present the system design of the FTS.

3.1 Tester Architectures

An IC functional tester is essentially a specialized parallel processing system with channels ranging from a few dozen to hundreds or even over a thousand (e.g., Hitachi [Kikuchi89], NTT LSI [Sudo87]). During each test period, a tester must apply data to the DUT pins or the DUT pin outputs must be sensed. It is this basic activity that a tester architecture must support with the appropriate organization and allocation of resources. Even with our modest tester requirements of 64 channels operating at 5MHz, the data rate required to support these operations is fairly high:

\[
4\text{bits/channel}^1 \times 5\text{MHz} \times 64\text{channels} = 320\text{Mbits/s}
\]

This figure does not include the additional bandwidth that may be required to store test results. Testers with sophisticated formatting, 250MHz operating speeds, and 512 pins clearly require data rates well into the high gigabit/second ranges. These high data rates are achieved through parallelism. The test vectors are stored in memory associated with the tester channels. During testing these vectors are delivered to the channels in parallel at high speeds.

\[^1\text{Each vector value is assumed to require 4 bits to encode. The actual system uses 8 bits to encode one vector value in the worse case (see Section 4.6.2).}\]
Figure 3.1 shows a functional block diagram of a generic IC tester. Generally, an IC tester consists of the following key systems:

1. **System Controller**: This system controls and coordinates the overall tester operation. A general purpose microprocessor is usually responsible for low-speed operations such as the loading of test vectors, system configuration, and interface functions. This microprocessor also controls a sequencer which is a high-speed controller used during functional testing to provide the control signals at the speeds needed by the tester channels. Essentially the sequencer provides the addresses of the test vectors to be used in the current test cycle. In more sophisticated systems, the sequencer may be programmed to perform looping, decision branching, or other higher level functions. The microprocessor has little responsibility over the real-time test operations other than initiating and stopping the test.

2. **Pin Electronics**: The term pin electronics refer to the formatting, driver, and receiver circuits common to each tester channel. The interface electronics block shown in Figure 3.1 contains the driver/receiver circuits that interface with the DUT pins. More advanced pin electronics have programmable high and low voltage levels for output driving and input threshold comparison [Parker87]. In more expensive test systems, active loading may be used during input comparison to source or sink programmable load currents. Since these programmable voltages and currents cannot be rapidly changed amidst testing, they are fixed for the duration of a test. Depending on the tester architecture, other additional circuits common to each tester channel may be designated as part of the pin electronics.

3. **Timing Generators**: The timing generators provide the timing edges needed by the formatting electronics to define waveforms for I/O purposes. The sequencer may also use a timing generator to adjust the test period. More advanced timing generators could change its timing output on-the-fly [Feugate88].

4. **Memory System**: This system provides storage for test patterns, timing specifications (used
by the timing generators), format specifications (used by the formatting electronics), and comparison results. Large memory systems may require multiple banks of memory devices. Interleaving or other schemes may be required to provide higher access speeds.

The actual organization of these systems (especially 2-4) depend much on the architecture and implementation of the tester. Tester architectures can generally be classified into shared resource [Feugate88] architectures and tester per-pin [Bisset83] architectures.

![Functional block diagram of a generic tester.](image)

**3.1.1 Shared Resource Architecture**

The shared resource architecture attempts to optimize the cost of a tester system through resource sharing. Usually timing generator signals are shared either by fixed groups of channels or dynamically distributed with a crossbar switching network. I/O channels cannot be shared, but they could
be designed to be splittable into separate input and output channels. This feature provides the opportunity to more efficiently allocate I/O resources as a single bi-directional channel, or as separate input and output channels. Since bi-directional signals are often buses, further savings could be obtained by specifying the I/O direction for a group of channels, instead of on an individual basis. For example, I/O direction control in the HP E1451/1452 pattern module are given for groups of eight channels [HP E1451/E1452 91]. Some manual wiring is usually needed to prepare an fixture board that interfaces the appropriate tester channels to the DUT because the functionality of the channels could be quite inhomogeneous. To minimize cost, a shared resource tester system is assembled with the minimal resources that would meet all its anticipated test requirements. As test needs grow, hardware is added to the test system to meet these demands. This architecture is popular for low-end testers because of its lower cost.

3.1.2 Tester Per-Pin Architecture

The tester per-pin architecture uses the opposite approach of the shared resource architecture. This architecture attempts to provide uniform functionality between all channels by allocating dedicated resources to each channel. Early studies of this architecture suggested that this approach is justified for its improved flexibility and productivity [Bisset83]. For example, test program development on such a system is simpler because resource (e.g., timing generators) allocation is no longer an issue. Hardware is also somewhat simpler because the complex crossbar switches used for resource sharing are not required which reduces cost and simplifies the calibration of internal timing paths. However, the net cost of tester per-pin systems are still more than shared resource systems because of the large number of replicated channel components. This architecture provides a great deal of flexibility that makes it attractive or even required in a manufacturing environment [Chang87]. The high cost associated with this architecture means that its use is mainly restricted to high-end test systems (e.g., IBM [Gruodis88], NTT [Sudo87]). Efforts have also been made towards lower cost hybrid systems (e.g., Texas Instruments [Fehr92]) that attempt to combine the best of the two approaches.
3.2 Exploiting VLSI Technology

To meet the specifications required by leading edge IC testers, manufacturers rely on high performance technologies such as emitter-coupled logic (ECL) to implement the high speed subsystems that are mainly comprised of the pin electronics. However, these technologies have low integration densities (see Table 3.1 from [Huber91]). Compensation for low integration densities is provided with aggressive packaging technologies (e.g. mult-chip modules, hybrid) which reduce size and improve signal distribution.

<table>
<thead>
<tr>
<th>Technology</th>
<th>Speed</th>
<th>Density</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMOS</td>
<td>50 MHz</td>
<td>Highest</td>
<td>Lowest</td>
</tr>
<tr>
<td>TTL</td>
<td>100 MHz</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>BiCMOS</td>
<td>50 MHz</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>ECL</td>
<td>&gt;1 GHz</td>
<td>Low</td>
<td>Highest</td>
</tr>
<tr>
<td>GaAs</td>
<td>&gt;3 GHz</td>
<td>Lowest</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Table 3.1: Characteristics of VLSI technologies.

In the past, pin electronics were custom designed by the tester manufacturer because commercial parts were not available for these special purpose functions. For similar reasons, this approach is still used in today's high-end systems. However, for less demanding mid-range and lower end systems, VLSI technology could be exploited to produce compact test systems. Research at Tektronix has produced a pair of CMOS chips that provide output formatting and input comparison with active loading [Branson89]. On chip timing generators and DACs make these ICs fairly self-contained. Recently, Credence has reported using CMOS technology to integrate both input and output functions into a single device [Lesmeister91]. While this implementation does not provide dynamic loading or high/low voltage thresholds for I/O, the technology to integrate these functions is clearly available. A number of commercial pin electronics chipsets have also become available that provide active loading, pin driving, and comparison functions [Goodenough90]. These chipsets are lower density implementations that are more general purpose than the custom
devices mentioned above.

CMOS technology is emerging as an attractive alternative where extremely high performance is not required. CMOS is attractive for its high density and low cost, but it is more sensitive to temperature and voltage variations than other technologies. Once compensation is provided for these problems [Chapman92], CMOS technology appears to be a serious contender for use in tester implementation [Branson89][Lesmeister91].

Recently efforts have also been made at universities to exploit CMOS VLSI technology in the design of IC testers. Industrial efforts mentioned earlier mainly emphasize highly functional, but lower channel density pin-electronics chipsets. Typically, one or more ICs are needed to support one tester channel. On the other hand, directions taken by university research emphasize a higher degree of system integration with less functionality. Typically, these systems integrate many functional tester channels into a single chip. Several chips are used in parallel to form a test system. The result of this research are several compact functional tester systems. The functionality provided by these integrated testers are quite different, but they could be classified into systems without waveform formatting [Miyamoto87][Butner87][McKenzie92] and systems with waveform formatting [Gasbarro89]. The architecture and features of these systems are briefly discussed in the next two sections.

3.2.1 Systems Without Wave Formatting

Test systems without waveform formatting are the most simple and can be built most compactly. In these systems, each test vector bit typically needs to be specified with a minimum of two bits: i) the output value or the expected DUT response, and ii) the I/O direction. Using external vector memory with this approach requires 3 device pins per channel (2 pins for i) and ii), and a single I/O pin for the channel itself). Figure 3.2a shows a block diagram of this channel. As shown in Figure 3.2b, several channels could be integrated into each chip. (Control signals were not shown to simplify the diagram.) Several chips are used in parallel to form a tester with the desired number of channels. Outputs are usually formatted using NR with $t_1$ fixed. Edge strobing is usually provided
for input comparison.

Figure 3.2: Generic channel electronics in systems without waveform formatting.

Figure 3.2a shows two data pins supporting one I/O channel. In multichannel chips, the number of pins required to support each channel could be reduced by multiplexing the data inputs. Multiplexing is used in the MacTester [McKenzie89] [McKenzie92] to reduce the pin requirements to a manageable level (3 pins per 2 channels) for implementation using Xilinx XC3020 PLDs. (See Figure 3.3.) Each XC3020 provides 22 I/O channels. The main disadvantage of pin multiplexing is that several bus cycles are needed to transfer the data required for one test cycle. For the MacTester, the speed penalty incurred for multiplexing was acceptable because it was primarily designed to be sequenced by the host computer in real-time (on-line testing) which is already very slow. When operating independent from the host computer (off-line mode), 13 state machine cycles are required to process each test vector. This gives a vector rate on the order of 1MHz. The key advantage offered by the MacTester system is the high degree of programmability. The main disadvantages of this system is its slow operating speed.

Figure 3.3: MacTester channel multiplexing.
Miyamoto [Miyamoto87] used a different approach to reduce pin-count without compromising speed in their Data Generator/Receiver (DGR) tester chip. They integrate the vector memory directly in the DGR chip which eliminates the large number of data pins normally required for I/O interfacing. Figure 3.4 shows the basic DGR architecture. A narrow external data bus (e.g., 8-bits) is used to initialize the relatively wide vector memory inside the DGR chip. During testing, test vectors could be internally transferred between memory and pin electronics without width constraints. Similar to the MacTester, two bits are used to encode each test vector bit. The advantages of this architecture include: i) a higher operating speed because of the built-in RAM, and ii) a reduced pin count which yields compact packaging. The main disadvantage of this approach is the limited size of the on-chip vector memory. A prototype sixteen channel DGR was implemented with 256 vectors/channels in an 84-pin device.

Figure 3.4: Block diagram of DGR Architecture.

3.2.2 Systems With Wave Formatting

Gasbarro expands on Miyamoto's approach by adding waveform formatting (NR, RH, RL, RC, and RT\(^2\)), on-chip timing generators, input window comparison and hardware decompression to allow more vectors to be stored in an embedded memory [Gasbarro89]. Formats and timing are per-pin programmable, but they remain fixed for the duration of a test. This restriction reduces the number of bits needed to encode each test vector bit to four. The four data bits specify the test vector bit, the expected input bit, the I/O direction, and the relevance of the comparison result.

\(^2\)The Return Tri-state (RT) format is similar to RH or RL (see Figure 2.1) except the logic level outside the interval \(t_1\) to \(t_2\) is tri-state (high-impedance).
Compression ratios on the order 2.5:1 were achieved. (This figure accounts for the area occupied by the decompression hardware.) Compressed test vectors are stored using internal DRAM (dynamic random access memory). Sixteen channels were integrated into one 68-pin chip.

3.3 System Design

In some ways, our tester requirements are similar to the systems discussed in the previous section. It is desirable to integrate many channels into one VLSI device which facilitates the construction of a compact tester and helps to reduce cost. Having a compact tester also makes it easier to maintain signal integrity across the tester-DUT interface because I/O driver pins could be located very near the DUT.

Our test system requires the I/O formatting functionality of the commercial pin-electronics chip-sets with the additional system integration and density of the university developed testers. Besides implementation issues, one of the problems that need to be addressed is the high-bandwidth required to support on-the-fly formatting, timing and I/O changes. The remainder of this chapter presents the design of our functional tester system.

3.3.1 Functional Tester System Overview

The proposed architecture of the tester system is shown in Figure 3.5. The tester provides channels implemented using functional tester chips (FTCs). Four identical channels are integrated into each FTC. The test vectors required by a FTC is stored in its own vector memory in an encoded format. Higher test speeds may be achieved using an embedded memory to store the test vectors, but at this stage external memory is used to simplify the design. The FTCs are located as close to the DUT as possible to reduce transmission line effects.

The overall activity of the tester is controlled and monitored by the host computer through an interface provided by the tester system controller (TSC). The address space of the FTCs and the associated vector memories are accessed through the TSC. The TSC is also responsible for
generating the control signals which specify the operating mode for the FTCs. For an IBM AT ISA (industry standard architecture) bus interface, a programmable logic device (PLD) should suffice for implementing the TSC.

![Test system architecture diagram](image)

**Figure 3.5: Test system architecture.**

### 3.3.2 System Operation

Timing specifications for a functional test are first written using a high-level test language which is compiled into a low-level instruction encoding for execution by the individual channels. For initial prototyping, the encoded instructions could be manually derived (analogous to machine language programming). Timing measurements from the channels may need to be collected for calibration purposes. The FTCs are initialized using the calibration data to correctly setup the required formats and timing. Testing could then begin. Depending on the configuration (or implementation) of the TSC, the host computer could either wait for an interrupt or poll to determine the tester status. Testing could also be halted at any time by the host computer. The time required to initialize the test system depends on the speed of the interface between the host computer and the functional tester system.
3.3.3 Functional Tester Chip

The FTC is the interface between the test system and the DUT. Each FTC contains the waveform formatters, input comparators, and timing generators needed for the four channels. In addition, the FTC includes a sequencer which controls and manages access to the external vector memory. Except for the sequencer, resources are dedicated to each channel. In this situation, using a per-pin approach is simpler because fewer external signals and chips are needed.

The FTCs mainly operate independently, however, some rudimentary communications between the TSC and possibly with other FTCs is needed. For example, a FTC may need to signal that testing has stopped because an error was detected or all the test vectors have been applied. Some control pins are also needed to initiate testing and to select other FTC operating modes. Access to internal registers can be provided by the memory interface. Chapter 4 discusses the design of the FTC in greater detail.

3.3.4 Board Layout

A standard method of placing pin drivers/receivers for an IC tester is shown in Figure 3.6. This layout allows the maximum number of drivers to connect with the DUT while maintaining relatively constant distances. Maintaining constant driver to DUT distances is important for consistent signal propagation across the drivers.

In large test systems, the channel electronics (e.g., waveform formatting, pattern generators, etc.) are located further from the pin drivers because of their size and numbers. Special efforts are made to propagate signals without distortion between the channel electronics and pin drivers because of the relatively large distances that separate them.

In our integrated tester, the functions of the pin electronics and the waveform formatter are combined into one chip. This allows all the pertinent circuitry to be located very near the DUT. The signal distribution problem is simplified and the resulting test system is very compact.

Our 64 channel tester system requires 16 ICs (4 channels per IC). Each IC has a footprint
of 26mm × 26mm. If each IC requires a spacing of 5mm for routing purposes, the circle of FTCs will have a radius of:

$$\text{radius} = \frac{(\# \text{ of chips}) \times (\text{footprint length} + 2 \times \text{spacing})}{2\pi} = \frac{16 \times (26\text{mm} + 2 \times 5\text{mm})}{2\pi} = 9.17\text{cm}$$

Therefore, a DUT to pin driver distance of 9.17 cm could be achieved. Guidelines given in [Cascade91] suggest that the transition time of a signal should be greater than twice the time it takes to propagate down its length of stripline to maintain good signal integrity without severe ringing and other undesirable effects. This relationship is given in equation (3.1) below.

$$L = \frac{T_r \times V_p}{2}$$  \hspace{1cm} (3.1)
where \( L \) = physical stripline length
\( T_{tr} \) = transition time
\( V_p \) = wave velocity
\[
\approx \frac{c}{\sqrt{\varepsilon_r}} \quad (\varepsilon_r \approx 4.9 \text{ for FR4 printed circuit board (PCB) material})
\]
\[
= \frac{3 \times 10^8 \text{m/s}}{\sqrt{4.9}}
\]
\[
= 1.355 \times 10^{10} \text{cm/s}
\]

Using equation (3.1) with \( L = 9.17 \text{cm} \) shows that the proposed layout can be used for transition times of 1.35ns or more. This result should be more than adequate for our design.

### 3.3.5 Pin Count and Vector Depth

The pin count (or the number of channels) and vector depth are two parameters that generally scale well in terms of size and communication requirements. The main issues in increasing pin count deal with density and the maintenance of minimum driver-DUT distances. Some of the ways to increase the pin count for our test system are:

1. Increasing the size of the circle allows more pin drivers to fit while also increasing the driver-DUT distances. This approach may also require a larger printed circuit board (PCB).

2. Using smaller IC packaging will allow more pin drivers to fit around the same radius circle.

3. The footprint of the FTC could be significantly reduced by mounting it on a daughter card which is in turn mounted on the motherboard. This approach essentially reduces the footprint of the FTC to the thickness of a PCB plus the area required for routing.

There are limits to this scalability as high pin-count testers require more exotic packaging and circuit board technology to achieve the high density requirements. For our requirements of 48-64 pins, ordinary PCB and packaging technology should suffice.

The depth of the vector memory is another parameter that scales well. Currently, the tester memory system calls for one static random-access memory (RAM) chip. Relatively inexpensive
static RAMs with densities of 64-256k bits range should be adequate for our system. Using a single memory chip keeps the memory system small and simple. Multiple devices could be used to increase the vector depth, but a small memory system is sufficient to test the functionality of initial prototypes.

3.3.6 Speed and Bandwidth

Test systems that support on-the-fly format and timing changes require significantly more bits to encode test vectors than other simpler systems. The extra bits specify the formatting and timing information required during each test period. Since each channel is probably going to use only a small subset of the possible formats and timings of each channel, it is possible to reduce the width of a vector bit encoding by storing only an index to an on-chip table which in turn specifies the actual format and timing. This approach substantially reduces the bits needed to encode each test vector value, but at a slight reduction in flexibility.

While the width of the encoding is reduced in this scheme, it is still significantly more than the two to four bits used in the other integrated test systems. We restrict the width of the encoded vector values to a maximum of 8 bits. The relatively wide memory bus and high memory bandwidth required per channel makes it impractical to integrate many channels into one FTC while maintaining a low pin count. Using a shared bus, it is possible to integrate four channels into one FTC at the cost of reducing the potential test speed by a factor of four. Alternatively, internal vector memory could be used to eliminate FTC I/O bandwidth and pin requirements, but the chip area would increase significantly and the size of the vector memory would be limited. Even systems with a two or four bit encoding uses either multiplexing or embedded memory to achieve their high densities.

A tester must be fast enough to generate the desired waveforms and perform the required comparisons. For chips containing dynamic circuits, the tester must also meet a minimum "keep-alive" speed. The main constraint on test speed is imposed by the memory system. A 20MHz operating speed requires a memory access time of less than 50ns. There are well-known techniques
for improving the access speed of a memory system such as interleaving [Rau79]. These schemes should work well because test vector memory access is highly sequential. However, these techniques increase the size, complexity, and cost of the memory system.
Chapter 4

Functional Tester Chip

This chapter discusses the design of the main components of the functional tester chip. Implementation details and schematics are given in Appendix C.

4.1 Overview

Figure 4.1 shows a block diagram of the Functional Tester Chip (FTC). The FTC consists of the following modules: control, registers, memory interface, clock, and channels. Each channel contains an instruction decoder, format table, a waveform generator, and an input comparator. Although the original design supported four channels per FTC, the number of channels used in the actual prototype chip was reduced to one because of limited fabrication resources. It would still be possible to evaluate the functionality of the FTC with this one channel. The remaining sections of this chapter discuss the functional blocks of the FTC in more detail.

4.2 Control

The operation of the FTC is essentially controlled by three pins: halt and mode<1:0>. The mode and halt signals are intended to be connected in parallel to all the FTC chips. The halt signal behaves as an open drain active-low signal and allows external devices to stop the FTC. While in the halt state, the tester channels are tristated. The values present at the mode pins determine the operating mode of the FTC. Changes between operating modes are synchronized with the clk input (falling edge). Table 4.1 gives the decoding of these mode bits. Future use of the unused mode (01) include a “keep-alive” mode for maintaining the state of a dynamic circuit during pattern reload.
The operation of the FTC under these modes is described in the following:

1. **Test Mode**: In this mode, the FTC is actively fetching and executing instructions from the vector memory. The FTC could leave test mode and enter a *halt* state for one of three reasons: i) end of instructions, ii) comparison error, or iii) external *halt* signal. Once the FTC enters the *halt* state it could only recover by leaving test mode.

2. **Program Mode**: While in this mode, the tester channels are inactive and tristated. The FTC operates as a memory allowing read/write operations on internal registers through its memory interface.

3. **Calibration Mode**: This mode facilitates calibration operations on the FTC. Tester channels

---

**Table 4.1: FTC Modes.**

<table>
<thead>
<tr>
<th>Mode&lt;1:0&gt;</th>
<th>Mode Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Calibration Mode</td>
</tr>
<tr>
<td>01</td>
<td>Unused</td>
</tr>
<tr>
<td>10</td>
<td>Program Mode</td>
</tr>
<tr>
<td>11</td>
<td>Test Mode</td>
</tr>
</tbody>
</table>
are active in this mode, but test pattern data are not fetched from the pattern memory.
Instead, the host computer directly writes data into the channel buffers. The contents of the
channel buffers are repeatedly executed.

The behavior of the FTC under various error conditions depend on the state of the system bits
which are discussed in the next section.

4.3 Registers

A memory map of the FTC registers and status bits are given in Table 4.2. The system bits specify
the behavior of the FTC during test mode. The tclock period is programmable from 4 to 259 times
the clk period. The address wrap bits enable the vectors to continue at the wrap address. Otherwise
testing stops because a halt condition is reached. The stop on compare error bits enable a compare
error to trigger a halt condition. The 3-bit synch select register selects the signals that would be
multiplexed out the synch pin for synchronization purposes. The address end and compare error
status bits are set to indicate the cause of the halt condition. The functions of the remaining
registers are described in the related sections.

4.4 Memory Interface

Access to the external vector memory is multiplexed between the four channels. The address space
of each channel is defined by the start, end, and wrap addresses. Figure 4.2 illustrates the use of
these registers. Vector decoding begins at the start address and continues sequentially until the
end address is reached (arrow a). From the end address, decoding continues at the wrap address
(arrow b) if wrap execution mode is enabled for the channel (i.e., the wrap bit is set). Otherwise,
the FTC enters a halt state after the instruction at the end address has been executed. Under the
wrap operating mode, execution continues until termination by a halt condition.

This organization allows initialization waveforms to be placed between the start and end
addresses. Since the address space of each channel is completely configurable, the available pattern
memory can be more efficiently distributed between the four channels. Two channels on the same FTC that are producing the same waveforms could even share the same address space.

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;10:6&gt;</td>
<td>&lt;5:0&gt;</td>
</tr>
<tr>
<td>00000</td>
<td>System Space</td>
</tr>
<tr>
<td>000000</td>
<td>tclk period</td>
</tr>
<tr>
<td>000001</td>
<td>address wrap (0-4) / stop on compare error (0-4)</td>
</tr>
<tr>
<td>000010</td>
<td>synch output select (5 bits)</td>
</tr>
<tr>
<td>0001cc</td>
<td>buffer of channel cc during calibration mode</td>
</tr>
<tr>
<td>001000</td>
<td>LSB of channel 0 error count</td>
</tr>
<tr>
<td>001001</td>
<td>MSB of channel 0 error count</td>
</tr>
<tr>
<td>001010</td>
<td>LSB of channel 1 error count</td>
</tr>
<tr>
<td>001011</td>
<td>MSB of channel 1 error count</td>
</tr>
<tr>
<td>001100</td>
<td>LSB of channel 2 error count</td>
</tr>
<tr>
<td>001101</td>
<td>MSB of channel 2 error count</td>
</tr>
<tr>
<td>001110</td>
<td>LSB of channel 3 error count</td>
</tr>
<tr>
<td>001111</td>
<td>MSB of channel 3 error count</td>
</tr>
<tr>
<td>010000</td>
<td>address end status (0-4) / compare error status (0-4)</td>
</tr>
<tr>
<td>00001</td>
<td>Channel Address Space Bounds</td>
</tr>
<tr>
<td>0cc000</td>
<td>LSB of channel cc wrap address</td>
</tr>
<tr>
<td>0cc001</td>
<td>MSB of channel cc wrap address</td>
</tr>
<tr>
<td>0cc010</td>
<td>LSB of channel cc end address</td>
</tr>
<tr>
<td>0cc011</td>
<td>MSB of channel cc end address</td>
</tr>
<tr>
<td>0cc100</td>
<td>LSB of channel cc start address</td>
</tr>
<tr>
<td>0cc101</td>
<td>MSB of channel cc start address</td>
</tr>
<tr>
<td>00010</td>
<td>Format Memory</td>
</tr>
<tr>
<td>cccfbb</td>
<td>Addresses byte bb of format ff of channel cc. Currently only &lt;10:6&gt;=00010 is used. Byte ordering: 0 is LSB; 3 is MSB.</td>
</tr>
</tbody>
</table>

Table 4.2: Address space of FTC registers. (LSB = least significant byte; MSB = most significant byte)

Encoded vectors are read sequentially in round-robin fashion into a separate buffer for each channel. External memory is accessed only when a buffer is empty. Buffering allows idle periods in the bus cycle to be potentially used for other purposes. However, it is difficult to reliably exploit these idle periods because they are highly dependent on the encoding stream. Depending on the
waveforms to be generated, these idle periods may not exist at all.

![Diagram of Address Space Example](image)

Figure 4.2: Address space example.

4.5 Clock Module

The clock module derives the test clock ($tclk$) and other clock related timing signals from the master clock ($clk$) input. The $tclk$ period is programmable from 4 to 259 times the $clk$ period. This provides coarse adjustment of the test period. Fine adjustment of the test period could be provided by changing the $clk$ frequency externally. The test period is fixed for the duration of a test.

4.6 Test Data Generation

Test data must be provided during each test period to the formatting, timing, and comparison circuits. The generation of this test data is the responsibility of the format memory and instruction decoder.

4.6.1 Format Memory

The format memory specifies the set of formats and timings to be used for the current test. An index is associated with each format memory that identifies the active memory word. The data in the memory word is used in combination with the incoming test data bits to generate the formatted
test data or comparison data. The contents and function of the fields in the format memory word are given in Table 4.3.

<table>
<thead>
<tr>
<th>Position</th>
<th>Field Width</th>
<th>Field Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..0</td>
<td>2</td>
<td>Format (NR=11, RL=10, RH=01, RC=00).</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>I/O direction (Input=0 / Output=1).</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Vector value for waveform formatting during input comparison.</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>Select alternate data source (Data generator=0 / Other=1).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Width Timing Generator</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>Delay line tap enable.</td>
</tr>
<tr>
<td>11..6</td>
<td>5</td>
<td>Delay line tap select (0 to 31).</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>Phase delay (phase delay=1 / no phase delay=0).</td>
</tr>
<tr>
<td>18..13</td>
<td>6</td>
<td>Synchronous delay counter (0 to 63 clk cycles).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Edge Timing Generator</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>Delay line tap enable.</td>
</tr>
<tr>
<td>24..29</td>
<td>5</td>
<td>Delay line tap select (0 to 31).</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>Phase delay (phase delay=1 / no phase delay=0).</td>
</tr>
<tr>
<td>31..36</td>
<td>6</td>
<td>Synchronous delay counter (0 to 63 clk cycles).</td>
</tr>
</tbody>
</table>

Table 4.3: Format memory word fields.

4.6.2 Instruction Encoding and Decoding

Since the timing, formatting and I/O data is stored in the on-chip format memory, the simplest encoding scheme that allows on-the-fly format changes is given in Figure 4.3a. This encoding simply specifies the test datum and the table entry used to format the datum. Therefore, each test vector bit requires one byte to encode.

Using the slightly more sophisticated encoding (scheme A) shown in Figure 4.3b, the available pattern memory could be more efficiently used. The FTC recognizes three 8-bit instructions: vector sequence, format, and repeat. The desired waveform is described using these instructions. The vector sequence instruction specifies 7-bits of test data. The format instruction specifies the table entry to use for formatting the test data. Starting with the data bit included in the format

---

1To simplify the text, the term format will henceforth imply format, timing, and I/O direction.
Chapter 4. Functional Tester Chip

instruction, this new format is used on all subsequent test data until it is changed by another format instruction. The 5-bit format index allows up to 32 formats per channel. The repeat instruction repeats the last data bit from 1 to 64 times.

![Diagram of instruction encoding](image)

- **a)** Minimal encoding.

- **b)** Instruction encoding scheme A.

- **c)** Instruction encoding scheme B.

Note: \( d = \) test data bit; \( i = \) I/O direction

Figure 4.3: Vector encoding.

The number of test bits stored per byte depends on the distribution of instructions used to encode a particular test waveform. Table 4.4 gives the test data per byte for some instruction distributions using encoding scheme A. Case I is a test containing mainly vector sequences. This case is representative of a test with few changes in I/O direction and/or fairly simple timing requirements. Case II is a test containing primarily format instructions. Having a high number of format changes indicates that the channel is either I/O intensive or requires complex timing. Case III is unlikely to represent a real instruction distribution except when used in conjunction with an algorithmic pattern generator in the FTC (not implemented). This encoding scheme is fairly optimal for channels that operate strictly as inputs or outputs.

Encoding scheme B shown in Figure 4.3c handles I/O channels better than scheme A and yields fairly good encoding for strictly input or output channels. This scheme is similar to the first one in providing the same three instruction types. However, the I/O direction associated with a test data bit is also given. Therefore, the I/O direction is no longer part of the internal format table. The vector sequence instruction now encodes only three test data and the format instruction
can only choose from a selection of 16 formats.

Table 4.5 summarizes test data per byte of instruction distributions using this scheme. Again, Case I is a test containing mainly vector sequences. However, this case covers I/O intensive tests as well as test with simple timing requirements. Case II contains primarily format instructions which indicates that complex timing is required.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction Distribution (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector Seq.(7)</td>
<td>90-100 10-0 0</td>
</tr>
<tr>
<td>Format(1)</td>
<td>10-0 90-100 0</td>
</tr>
<tr>
<td>Repeat(64)</td>
<td>0 0 100</td>
</tr>
</tbody>
</table>

Table 4.4: Test data stored per byte for scheme A instruction distributions.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction Distribution (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector Seq.(3)</td>
<td>90-100 10-0 0</td>
</tr>
<tr>
<td>Format(1)</td>
<td>10-0 90-100 0</td>
</tr>
<tr>
<td>Repeat(64)</td>
<td>0 0 100</td>
</tr>
</tbody>
</table>

Table 4.5: Test data stored per byte for scheme B instruction distributions.

4.6.3 Pipelining

During each test period, one instruction for each channel is read. Fetch, decoding, and execution is done using a 3-stage pipeline as shown in Figure 4.4. The execution step, which involves generating the specified waveforms, occurs simultaneously for the four channels and lasts for four memory cycles. The format instruction requires the highest memory bandwidth because it encodes only one test data bit per byte. Therefore, the memory interface must satisfy this worst case situation and operate at 4xMHz to support a xMHz test frequency.
4.7 Waveform Formatting and Timing

4.7.1 Timing Generation

The timing generators need to produce timing edges at discrete intervals over the given test period. A functional block diagram of the timing generator is shown in Figure 4.5. Each test period ($T_{clk}$) consists of 4 to 259 system clock periods ($T_{clk}$). Timing edges are placed relative to the rising or falling edge of $clk$. A counter and a $clk$ phase delay circuit is used to determine from which edge of $clk$ to trigger the fine vernier. Therefore, the fine vernier must provide enough delay to span one $clk$ phase.

One traditional approach to implementing the fine delay vernier involves using a ramp generator, a digital to analog converter (DAC), and a comparator as shown Figure 4.6a. The voltage level of the linear ramp is compared with the output of the precision DAC to produce a delayed edge. This circuit is difficult to design using CMOS VLSI because high-resistance current sources and high precision DACs are required [Chapman92]. Furthermore, the relatively long settling times of DACs (100ns to 100$\mu$s) make this approach unsuitable for on-the-fly timing changes at test speeds.
Figure 4.6b shows the implementation of the fine vernier used in the timing generator. The edge generated by the coarse vernier passes through the variable length delay line configured to place the timing edge at the desired offset from the rising or falling edge of the master clock. The basic delay element, $\tau_D$, is a CMOS buffer which is essentially a pair of inverters. The number of buffers used in each successive delay segment is binary ratioed, i.e., twice the number used in the previous segment. This implementation is compact and gives a resolution of one buffer delay. The buffer delay element determines to a large extent the resolution of the waveform formatter.

The range of delays generated by the fine vernier is given by:

$$\tau = \tau_{\text{mux}} + n\tau_D \quad \text{where} \quad 0 \leq n \leq 31$$

For the buffer cell used, $\tau_D \approx 0.63\text{ns}$ which yields a range of 19.5ns for $31\tau_D$. This delay range supports a 25MHz master clock. The propagation delay through the six multiplexors, $\tau_{\text{mux}}$, amount to approximately 4ns. Bypassing the first five multiplexors allow an edge to be propagated within 1ns.
4.7.2 Propagation Delay Sensitivities

Since we are relying on the propagation delays of CMOS gates to generate timing edges, it is important to examine the sensitivity of these delays to process, supply voltage, and temperature variations. Process variations lead to fixed differences in propagation delay which could be accounted for through calibration. Temperature and power supply variations are dynamic in nature and therefore require active compensation.

CMOS circuits primarily drive capacitive loads because of their high impedance inputs. Therefore, the propagation delay of these circuits is inversely proportional to the drive current supplied by the MOSFETs that comprise the CMOS circuit. The drain-source current, $I_{DS}$, of a MOSFET operating in saturation (or pinch-off) is given by (4.1). This equation contains both temperature and voltage rail sensitive parameters.

$$I_{DS} = \frac{1}{2}\mu C_{ox}(\frac{W}{L})(V_{GS} - V_t)^2$$  \hspace{1cm} (4.1)

Mobility (both electrons and holes), $\mu$, was experimentally measured to have a temperature dependence of approximately $T^{-3/2}$ due to lattice scattering (in the temperature range of interest) [Pulfrey89]. The threshold voltage, $V_t$, is also temperature sensitive and decreases approximately 2mV per °C increase [Sedra87]. However, the dependency of $I_{DS}$ on temperature is dominated by $\mu$. The gate-source voltage, $V_{GS}$, depends primarily on the supply rail. These characteristics of CMOS circuits must be considered in the design of the timing and formatting circuits.

To get an idea of the severity of these problems, we need to examine data showing the propagation delay sensitivities of CMC's CMOS4S library to supply voltage and temperature variations. Unfortunately, this information is not available. However, propagation delay curves specified by VLSI technology for their 1.0μm CMOS library are available and should give an indication of the expected variations\(^2\) [VLSI89]:

- Temperature: +0.003x per °C

---

\(^2\)These factors were linearly interpolated from curves published by VLSI Technology.
Chapter 4. Functional Tester Chip

- Voltage: $-0.019 \text{x per 100 mV}$

For example, an increase in 1 °C increases the propagation delay by 0.3% while an increase in 100 mV decreases the propagation delay by 1.9%.

These sensitivities result in a fairly small, but cumulative skew in the delay line. Early taps in the delay line experience only a small skew with respect to the expected delay, but the skew experienced by the later taps may accumulate beyond the resolution of the delay line. For our 20ns delay line, the 0.63ns steps are 3.15% of the total delay length. Therefore, variations of as little as 200mV could shift a timing edge at the end of the delay line by one step.

On-chip temperature regulation and phase-locked loop methods that compensate for these drifts in propagation delay are given in [Chapman92]. In [Lesmeister91], a phase-locked loop was used for stabilizing timing edges against temperature variations. These techniques are suitable for compensating slow drifts in temperature and voltage.

Compensating for temperature and voltage variations require considerable efforts in simulations, chip characterization, custom layout and analog design. These efforts are beyond the scope of this project due to limited time and resources. Furthermore, it is not certain that compensation is needed without actual measurements. For example, the timing generator implemented by Gasbarro using a 2μm CMOS process did not use temperature or voltage compensation circuits, but still had fairly good timing stability [Gasbarro90]:

- Temperature: 20ps per °C per delay step.
- Voltage: 70ps per Volt per delay step.

In [Branson89], maximum thermal drifts of 1.6ns were reported for pin-electronics operating at 100MHz. It may be possible to ignore thermal effects on the FTC because of its relatively low operating frequency.
4.7.3 Formatting

The waveform formatter provides NR, RH, RL and RC formats. Except for the NR format, two timing edges are required to specify the formats. Figure 4.7 shows a block diagram of the formatting circuit. The waveform formatter uses a negative edge sensitive flip-flop to synchronize transitions in the formatted output.

![Figure 4.7: Functional block diagram of waveform formatting circuit.](image)

The edge timing generator (TG) causes the flip-flop to latch the test data at $t_1$ which places the leading pulse edge. Depending on the format, the width TG triggers either the P0 or P1 pulse generators to clear or set the flip-flop at $t_2$ which places the trailing edge. The width TG is disabled for the NR format since only the leading edge is required. P0 or P1 may also initialize the flip-flop to a set or clear state at the start of the test period. The format and test datum to be used in the next test period is made available to the pulse generators to allow this initialization. The triggering conditions for P0 and P1 are given in Tables 4.6 and 4.7 respectively. For example, the RC format requires the flip-flop cleared if the test datum is 1. Flip-flop initialization is not required for the NR format.

4.7.4 Dead-Zones

Dead-zones (or dead-bands) are regions in the test period where edges cannot be placed even though the region is larger than the edge resolution of the tester. These dead-zones arise from design and hardware limitations. For example, the timing generators and formatting circuits takes time to
initialize and reset. Dead-zones result if this time exceeds the resolution of the timing generators. A common technique to reduce or eliminate the dead-zone involves interleaving (or multiplexing) between two timing generators and waveform formatting circuits. Each circuit now has one test period for recovery and initialization. At the cost of redundancy and increased complexity, dead-zones caused by initialization time is essentially eliminated. Interleaving was not used in the formatting and timing circuits for simplicity, and reduced area.

The dead-zones that occur in the FTC arise primarily from insufficient recovery time between edges. The set (S) and reset (Rb) signals of the flip-flop are asynchronous level sensitive inputs. Since these inputs have priority over the clock and data inputs, it is desirable to minimize the duration of the reset and set pulses. The pulse durations are reduced using a feedback connection from the flip-flop output (not shown) which switches off the pulse and resets the pulse generator when the flip-flop reaches the desired low or high state. By keeping the duration of the pulse short, waveform edges could be placed closer to the test cycle boundaries without interfering.

<table>
<thead>
<tr>
<th>Next Test Data Bit</th>
<th>Next Format</th>
<th>Clear Flip-flop</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>RC</td>
<td>Yes if v = 1 otherwise No</td>
</tr>
<tr>
<td>v</td>
<td>RH</td>
<td>No</td>
</tr>
<tr>
<td>v</td>
<td>RL</td>
<td>Yes</td>
</tr>
<tr>
<td>v</td>
<td>NR</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 4.6: Pulse generator 0 triggering conditions.

<table>
<thead>
<tr>
<th>Next Test Data Bit</th>
<th>Next Format</th>
<th>Set Flip-flop</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>RC</td>
<td>Yes if v = 0 otherwise No</td>
</tr>
<tr>
<td>v</td>
<td>RH</td>
<td>Yes</td>
</tr>
<tr>
<td>v</td>
<td>RL</td>
<td>No</td>
</tr>
<tr>
<td>v</td>
<td>NR</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 4.7: Pulse generator 1 triggering conditions.
with the next test cycle. Since the pulses cannot be infinitesimally short, some interference is expected.

The relationship between these timing edges is shown in Figure 4.8. Simulations show that the pulses are approximately 6.1 ns in duration and take approximately 2.3 ns from the triggering edge to activate. The dead-zone at the beginning of the test period is actually less than 8.4 ns because the start of the test period is determined by the time of the earliest possible edge. This earliest possible edge is also triggered by \textit{tclk}, but it is delayed by the propagation delay through the coarse and fine vernier in \textit{edge} TG. This delay amounts to 5.3 ns (4.3 ns + 1 ns) which leaves a 3.1 ns dead-zone at the start of the test period. The dead-zone after the trailing edge is primarily caused by the pulse width of the pulse generators and amounts to approximately 6.1 ns.

![Diagram showing dead-zones at the beginning and after the trailing edge.](image)

\textbf{Figure 4.8: Dead-zones (all times in ns).}

### 4.7.5 Skew

The waveforms generated by the FTC will exhibit some skew relative to the input clock. This skew is not a serious problem as long it is consistent amongst the channels. A more important concern is the skew, caused by the inconsistent propagation delay of rising and falling edges through logic gates, i.e., rise-fall skew. From Figure 4.7 we see that sources of rise-fall skew are the timing
Chapter 4. Functional Tester Chip

generators, the pulse generators, the flip-flop, and the I/O pad. Since the same \textit{tclk} and \textit{clk} edges are used, these signals do not contribute to rise-fall skew. Similarly, the timing generators do not add to the rise-fall skew since the flip-flop and pulse generators are both active on the falling edges. The calculated skew contribution of the remaining modules to the leading and trailing edges of a formatted waveform are summarized in Tables 4.8 and 4.9 respectively. These calculations show that the rise-fall skew is fairly small. In Chapter 5, we examine the rise-fall skew measured from a fabricated waveform formatter and discuss ways to reduce this skew.

<table>
<thead>
<tr>
<th>Module</th>
<th>LH Delay (ns)</th>
<th>HL Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flip-flop (clk)</td>
<td>1.96</td>
<td>2.26</td>
</tr>
<tr>
<td>I/O Pad</td>
<td>4.33</td>
<td>3.76</td>
</tr>
<tr>
<td><strong>Total Delays</strong></td>
<td><strong>6.29</strong></td>
<td><strong>6.02</strong></td>
</tr>
<tr>
<td><strong>Total Skew</strong></td>
<td><strong>0.27</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.8: Leading edge skew.

<table>
<thead>
<tr>
<th>Module</th>
<th>LH Delay (ns)</th>
<th>HL Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flip-flop (Set/Reset)</td>
<td>1.00</td>
<td>0.96</td>
</tr>
<tr>
<td>Pulse generators (Set/Reset)</td>
<td>2.56</td>
<td>2.34</td>
</tr>
<tr>
<td>I/O Pad</td>
<td>4.33</td>
<td>3.76</td>
</tr>
<tr>
<td><strong>Total Delays</strong></td>
<td><strong>7.89</strong></td>
<td><strong>7.06</strong></td>
</tr>
<tr>
<td><strong>Total Skew</strong></td>
<td><strong>0.83</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.9: Trailing edge skew.

4.8 Input Comparison

Figure 4.9 gives a functional block diagram of the input comparison circuit. During input comparison, the waveform formatter is used to generate formatted waveforms that define the compare window. When this waveform is high, the compare window opens which sensitizes an error latch to the result of the comparison between the DUT output (pad input) and the expected value. Any
detected discrepancy produces an error. Although strobe sampling is not provided, it could be approximated using a very narrow window.

Skew in the formatted waveform causes part of the output of the previous test period to be present for a brief duration after the I/O transition. If this output happens to be high, the comparison window may be inadvertently opened and cause an erroneous comparison. To avoid this problem, the window define circuit is edge sensitive and opens the compare window on a rising edge and closes the compare window a falling edge.

Similarly, skew in the formatted waveform may leave the comparison window open while the next comparison datum is loaded at the beginning of the test period. This transition in the comparison data is almost certain to cause an error to be recorded. To avoid this problem, the vector synchronization circuit allows comparison data to change only when the compare window is closed. Hspice simulations suggest that pulses on the order of 1.5ns could be detected.

Comparison results are not stored in the current design because of I/O and chip area limitations. However, this function could be easily provided with using a simple encoding scheme such as the one shown in Figure 4.10. Comparison results are returned as binary data (e.g., 1=correct; 0=error). The result sequence encoding describes 7 bits of comparison results while the repeated result encoding describes a string of 1 to 32 ones or zeros. Although this feature requires
relatively little I/O (a maximum of 1/8th the bandwidth of instruction memory), an additional memory bus is needed to provide the required bandwidth.

\[
\begin{array}{c|c}
7 & 6 \\
\hline
0 & 1 \\
\end{array}
\]

Result Sequence

\[
\begin{array}{c|c}
7 & 6 \\
\hline
0 & 1 \\
\end{array}
\]

Repeated Result

Note: \( r = \) comparison result

Figure 4.10: Comparison result output encoding.

Instead of storing the comparison results, a counter with an overflow bit is used to log the number of successful comparisons before a failure. This counter could be examined after a test to determine in which test cycle the first error occurred.

4.9 Pin Electronics

Pin electronics in high-end testers usually have programmable voltage supplies for output driving and input high-low threshold sensing. Dynamic loading is provided to facilitate testing under various load conditions. For initial prototyping, we can use standard CMOS I/O pads to drive and sense the DUT pins. The short distances between the FTC and the DUT should allow even CMOS pads to directly drive the DUT inputs without secondary buffers. However, we are probably limited to testing only CMOS devices because of the limited drive current available from the pad. The chosen pad can provide 4.2mA of drive current and has a rise-fall time on the order of 1ns.
Chapter 5

Implementation and Results

5.1 Implementation

The Canadian Microelectronics Corporation (CMC) provides two CMOS technologies for implementing the FTC – a 3.0μm process [CMOS3DLM] and a 1.2μm process [CMOS4SV2]. The 1.2μm technology was chosen for several reasons. Its smaller feature size allows the design to be implemented with a substantially smaller die. A 3.0μm implementation would require a die size of over twice that required by a 1.2μm implementation. More importantly, critical properties of the timing generators and waveform formatter are speed dependent. Using the higher speed 1.2μm technology increases the timing resolution and generally reduces the skew. Better performance could be obtained using a CMOS4S standard cell design without resorting to a full custom design effort.

5.1.1 FTC Area

Even when using the higher density technology, a four channel FTC required an area of 5733μm × 5122μm (core area: 4632μm × 4072μm). The control circuits were implemented with approximately 2200 gates while each channel required approximately 4200 gates. A four channel FTC therefore requires approximately 19k gates. This implementation was scaled down for initial prototyping, due to the limited fabrication resources of the CMC. An outline of the prototype FTC layout is shown in Figure 5.1. The number of channels integrated into the FTC was reduced to one which decreased the size of the chip substantially without changing the functions of the FTC that needs to be tested.
Figure 5.1: FTC chip layout.
Reducing the FTC to one channel left the chip is I/O bound, that is, its area is determined by the number of pads in the perimeter of the layout. Further area reduction could be accomplished by using either narrower or fewer pads. Since the FTC directly interfaces with the DUT, the pads that give the best performance should be chosen. Table 5.1 summarizes the performance of the available bidirectional pads [CMOS4SV2]. Compared to the other available pads, the 4mA wide pads gave the best combination of rise/fall times, propagation delay and skew. Therefore, the wide family of pads were retained in the layout.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Available Bidirectional Pads</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Wide</td>
<td>4mA</td>
<td>8mA</td>
<td>4mA</td>
<td>8mA</td>
</tr>
<tr>
<td>Rise time ($t_r$)</td>
<td>1.0 + 0.3C_L</td>
<td>1.0 + 0.15C_L</td>
<td>5.0 + 0.34C_L</td>
<td>6.0 + 0.036C_L</td>
<td></td>
</tr>
<tr>
<td>Fall time ($t_f$)</td>
<td>1.0 + 0.3C_L</td>
<td>0.5 + 0.16C_L</td>
<td>7.2 + 0.21C_L</td>
<td>6.0 + 0.036C_L</td>
<td></td>
</tr>
<tr>
<td>Low-high prop. delay ($t_{PLH}$)</td>
<td>2.9 + 0.11C_L</td>
<td>4.8 + 0.067C_L</td>
<td>4.3 + 0.15C_L</td>
<td>6.0 + 0.073C_L</td>
<td></td>
</tr>
<tr>
<td>High-low prop. delay ($t_{PHL}$)</td>
<td>2.2 + 0.12C_L</td>
<td>2.7 + 0.080C_L</td>
<td>6.2 + 0.10C_L</td>
<td>5.2 + 0.060C_L</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Bidirectional pads available from CMC’s CMOS4S cell library.

The number of pads were decreased by reducing the width of the address bus to 10-bits and by using the corner locations for supply and ground pads. This reduced layout is still slightly I/O bound and occupies an area of 3873.8μm \times 2931.4μm (39% of the area required by the four channel version). Most of the design was implemented with standard cells and automatically placed and routed. The single-channel FTC requires only 40 pins which gives considerable allowance for expanding the memory width and the number of channels. A description of the FTC pins is given in Appendix A.

The delay lines used to implement the fine vernier in the timing generators were placed in the lower right corner of the FTC layout (see Figure 5.1). These circuits were laid out by hand to provide more consistent delays. Figure 5.2 shows the layout the delay line macrocell. The multiplexors were placed adjacent to each other to minimize the zero delay skew. The smaller delay elements were placed closest to the selection multiplexors to reduce the affect of wire delays. This
circuit occupies an area of 292\(\mu\text{m} \times 174.8\mu\text{m}\).

5.2 Performance

Since the waveform formatting circuit and the associated timing generators are such a critical part of the FTC, these circuits were fabricated to provide an indication of the expected performance. Four out of the five manufactured chips were functional. The measured results given in this chapter were obtained from these four chips. Unfortunately, results from the actual FTC are unavailable because it has not yet returned from fabrication.

The resolution and skew performance of these waveform formatting chips (WFCs) were characterized by measuring the four edge placement parameters shown in Figure 5.3: i) the rising edge placement time at \(t_1\), ii) the falling edge placement time at \(t_1\), iii) the rising edge placement time at \(t_2\), and iv) the falling edge placement time at \(t_2\). We are mainly interested in characterizing

![Figure 5.2: Delay line macrocell layout.](image-url)
the asynchronous delay generated by the fine vernier. The fine vernier allows the \( t_1 \) and \( t_2 \) edges to be placed over the range of one clock phase. The actual range is slightly delayed after the clock edge due to skew. From these measurements the resolution and skew were determined.

![Figure 5.3: Timing measurement locations.](image)

### 5.2.1 Timing Resolution

Timing resolution is measured as the size of the interval between which successive edges can be placed. Table 5.2 shows the simulated edge placement delays of the FTC. The change in (\( \Delta \)) delay column gives the difference between the current delay and the previous delay. The absolute delays marked with an asterisk (*) are the results obtained from Silos simulation. These simulated results were used to compute the other values in the table. The delay values were normalized to eliminate the skew offset from the reference edge. Timing skew is an important issue that is discussed in the next section.

From this table, the resolution is determined to be 1.2ns, the largest \( \Delta \) delay. These results show that most of the \( \Delta \) delays fall in the 0.5ns to 0.8ns range. The outliers with delays of 0.2ns and 1.2ns are caused by skew mainly in the delay elements. For example, consider the transition from seven to eight delay elements. The delay line implementation shown in Figure 4.6b shows that a seven element delay involves \( (1+4)=5 \) high-low (HL) transitions and 2 low-high (LH) transitions while the eight element delay involves eight LH transitions. The buffer delay elements propagate a
Table 5.2: Simulated FTC edge placement times in ns (normalized).

<table>
<thead>
<tr>
<th>delay</th>
<th>Δ(ns)</th>
<th>delay</th>
<th>Δ(ns)</th>
<th>delay</th>
<th>Δ(ns)</th>
<th>delay</th>
<th>Δ(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tap</td>
<td>abs.</td>
<td></td>
<td></td>
<td>tap</td>
<td>abs.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0.0*</td>
<td></td>
<td></td>
<td>8</td>
<td>4.54*</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.63*</td>
<td>0.6</td>
<td></td>
<td>16</td>
<td>10.10*</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.17*</td>
<td>0.5</td>
<td></td>
<td>17</td>
<td>10.72</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.80</td>
<td>0.5</td>
<td></td>
<td>18</td>
<td>11.27</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2.56*</td>
<td>0.8</td>
<td></td>
<td>19</td>
<td>11.90</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3.19</td>
<td>0.6</td>
<td></td>
<td>20</td>
<td>12.66</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3.73</td>
<td>0.5</td>
<td></td>
<td>21</td>
<td>13.28</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4.36</td>
<td>0.6</td>
<td></td>
<td>22</td>
<td>13.83</td>
<td>0.5</td>
<td></td>
</tr>
</tbody>
</table>

HL transition with 0.58ns delay while a LH transition is propagated with 0.51ns delay. Therefore, seven delay elements provide a 3.92ns delay and an eight delay elements provide only a 4.08ns delay (i.e., an additional 0.16ns). The cause of the 1.2ns Δ delay is similar.

Table 5.3 lists the average edge placement times obtained from the WFCs for a rising edge at \( t_1 \). Again, the absolute delays marked with an asterisk(*) were measured from the WFCs while the unmarked delays were derived from the measurements. Test stimulus for the WFC were generated with an HP 8180A Data Generator. The results were measured using a HP 54600A 100MHz oscilloscope (with 13pF load from probe). The output edges were very stable with no observable jitter. Measurement errors are estimated to be ±0.1ns.

An initial observation from comparing Tables 5.2 and 5.3 is that the measured 15ns delay range is substantially less than the simulated 19ns delay range. This result is not totally surprising considering that timing characteristics can be quite different due to process variations. However, it is quite possible that the propagation delays used in the simulation models are overly pessimistic. The resolution given by the largest Δ delay is 1ns.

A second observation is that the WFCs appear to exhibit a greater variation in Δ delay when compared to the simulated results. Closer examination of the measurements show that
Table 5.3: Average WFC edge placement times for a rising edge at \( t_1 \) in ns (normalized).

<table>
<thead>
<tr>
<th>delay</th>
<th>abs.(ns)</th>
<th>( \Delta )(ns)</th>
<th>delay</th>
<th>abs.</th>
<th>( \Delta )</th>
<th>delay</th>
<th>abs.</th>
<th>( \Delta )</th>
<th>delay</th>
<th>abs.</th>
<th>( \Delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0*</td>
<td>—</td>
<td>8</td>
<td>3.78*</td>
<td>0.3</td>
<td>16</td>
<td>7.60*</td>
<td>0.3</td>
<td>24</td>
<td>11.38</td>
<td>0.3</td>
</tr>
<tr>
<td>1</td>
<td>0.23*</td>
<td>0.2</td>
<td>9</td>
<td>4.01</td>
<td>0.2</td>
<td>17</td>
<td>7.83</td>
<td>0.2</td>
<td>25</td>
<td>11.61</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>1.25*</td>
<td>1.0</td>
<td>10</td>
<td>5.02</td>
<td>1.0</td>
<td>18</td>
<td>8.85</td>
<td>1.0</td>
<td>26</td>
<td>12.62</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>1.48</td>
<td>0.2</td>
<td>11</td>
<td>5.26</td>
<td>0.2</td>
<td>19</td>
<td>9.08</td>
<td>0.2</td>
<td>27</td>
<td>12.85</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>2.00*</td>
<td>0.5</td>
<td>12</td>
<td>5.78</td>
<td>0.5</td>
<td>20</td>
<td>9.60</td>
<td>0.5</td>
<td>28</td>
<td>13.38</td>
<td>0.5</td>
</tr>
<tr>
<td>5</td>
<td>2.23</td>
<td>0.2</td>
<td>13</td>
<td>6.01</td>
<td>0.2</td>
<td>21</td>
<td>9.83</td>
<td>0.2</td>
<td>29</td>
<td>13.61</td>
<td>0.2</td>
</tr>
<tr>
<td>6</td>
<td>3.25</td>
<td>1.0</td>
<td>14</td>
<td>7.03</td>
<td>1.0</td>
<td>22</td>
<td>10.85</td>
<td>1.0</td>
<td>30</td>
<td>14.63</td>
<td>1.0</td>
</tr>
<tr>
<td>7</td>
<td>3.48</td>
<td>0.2</td>
<td>15</td>
<td>7.26</td>
<td>0.2</td>
<td>23</td>
<td>11.08</td>
<td>0.2</td>
<td>31</td>
<td>14.86</td>
<td>0.2</td>
</tr>
</tbody>
</table>

The variations originate mainly from non-linearities in the two shorter delay elements. These non-linearities are more easily seen when the data is plotted as shown in Figure 5.4. The individual chip measurements were plotted separately so that variations between chips are more apparent. Process variations are the main cause of the deviations in delay between devices. Most of the measurements fall on a fairly straight line except for those from the one element delay. Examining the layout of the WFC showed that the single buffer delay element was ideally placed between its adjacent multiplexors while the other larger delay elements were placed further apart. (Figure 4.6b gives a block diagram of the delay lines.) Therefore, the non-linearity may be caused by the disparity between the additional delay introduced by the longer routing of larger delay elements and the shorter delays caused by minimal routing of the single delay element. In some cases, the delay of the single buffer is barely measurable.

A third observation is that the WFC measurements are significantly more linear than the simulated data. This result indicates that the skew in the buffer delay elements are less severe than modeled in the CMOS4S library. Comparison with timing measurements of the other three edge placement parameters (see Appendix B) give similar results.

Resolution provides a measure of the interval at which a tester can place distinct waveform edges. The amount of resolution required is highly dependent on the application. Usually, a test
Simulated and Measured Delays

Figure 5.4: Simulated and measured delays. (Measured times are for rising edge at t₁.)

is scaled to adapt to the available resolution and other tester parameters. While a fine resolution is desirable, it may not be justified if a less costly system with coarser resolution is adequate. Measurements from the WFCs give a resolution of 1 ns. Resolutions of 0.5 ns are achievable if the non-linearities in the shorter delay elements could be eliminated.

5.2.2 Timing Skew

So far, in the discussion of timing resolution, the channel outputs have been considered independently. In practice, however, ICs are tested with groups of waveforms. Once more than one signal is used, the issue of skew becomes important. Skew refers to the deviation (sometimes called *phase difference*) between the actual and desired edge placement times. There are two types of skew that
we are concerned with: i) *interchannel skew*, and ii) *rise-fall skew*. Generally, the presence of skew degrades the accuracy with which edges could be placed. Skew cannot be completely eliminated because the environment in which the electrical signals operate cannot be perfectly duplicated between channels. However, the goal is to minimize skew.

**Interchannel Skew**

Interchannel skew arises from a phase difference between edges generated by different channels from either the same device or from different devices. Figure 5.5 shows a plot of the typical interchannel skew measured from the WFCs at edge $t_1$. The delay times are measured relative to a fixed reference edge (an edge of the input clock). The rising edges are graphed with the heavier lines while the falling edges are graphed with the lighter lines.

From this plot, the maximum interchannel skew obtained is 1.2ns for both rising and falling edges. Several calibration techniques are available that could be used to correct for this skew. One method of skew compensation is to introduce an additional delay line at either the timing generator output or the waveform formatter output [Healy85, Dahl87]. Faster edges are delayed so that they arrive at the same time as the slower edges. The net effect is an upward shifting of the graphs to minimize skew. Figure 5.6 shows how this deskewing technique can be used to reduce most of the skew to 0.3ns or less. (The offset from 0ns for absolute delay was uniformly reduced to 1ns so that the maximum skew could be plotted on the same graph in larger scale.) This deskewing scheme is commonly used in shared resource tester architectures [Healy85].

An alternative deskewing scheme uses the timing generators already present in each channel to compensate for the skew [Catalano83] [Healy85]. The idea is to choose the appropriate delay that minimizes skew for a given edge placement time. This scheme is only effective in the regions where the delay ranges for the different channels overlap. The edges of the delay ranges are not usable, i.e., they are dead-zones. By using sufficiently wide delay ranges, these dead-zones could be located outside the active region of the current test cycle. Multiplexing is probably required to eliminate these dead-zones.
Interchannel Skew
for rising and falling edges at t1

Figure 5.5: Interchannel skew measured for rising and falling edges at t1.

Rise-fall Skew

CMOS logic circuits usually do not propagate both rising and falling edges with the same delay. This difference in propagation delay is the cause of rise-fall skew. Rise-fall skew primarily degrades the accuracy with which edges could be placed.

The measured rise-fall skew at t1 and t2 are summarized in Table 5.4. The calculated skews are shown in Table 5.5. The magnitude of the measured skews are fairly close to the calculated skews and the relative positions of the edges are correct. The waveform formatter implemented in the WFC could not produce all four formats because of a design error. The revised waveform
formatter in the FTC may have a slightly longer propagation delay for $t_1$ edges, but the overall rise-fall skew is reduced. The overall propagation delay is reduced because of the 4mA wide pads. The effect of these skews is more clearly seen when the actual delay ranges for $t_1$ and $t_2$ are plotted as shown in Figures 5.7 and 5.8 respectively.

Rise-fall skew can be reduced using custom transistor sizing and/or subsequent deskewing. Designing the waveform formatter circuit with minimum rise-fall skew involves appropriately sizing its transistors for a given capacitive load. This problem requires a considerable amount of effort. Even sizing a pair of inverter chains (consisting of two and three gates) for the purposes of generating complementary clocks with minimal skew and pulse distortion requires as many a 100 brute force spice simulations [Argade89] depending on the desired precision. In comparison, the waveform

Figure 5.6: Deskewed placement times for rising edges at $t_1$. 

![Diagram of Deskewed Rising Edge at t1](image)
Chapter 5. Implementation and Results

<table>
<thead>
<tr>
<th>Tap</th>
<th>Rise-fall skew at $t_1$ (ns)</th>
<th>Rise-fall skew at $t_2$ (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>2.0</td>
<td>1.6</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>4</td>
<td>1.7</td>
<td>1.6</td>
</tr>
<tr>
<td>8</td>
<td>1.9</td>
<td>1.9</td>
</tr>
<tr>
<td>16</td>
<td>2.1</td>
<td>2.1</td>
</tr>
</tbody>
</table>

Mean: 1.61
Standard Deviation: 0.31
Variance: 0.09

<table>
<thead>
<tr>
<th>Tap</th>
<th>Rise-fall skew at $t_1$ (ns)</th>
<th>Rise-fall skew at $t_2$ (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>2.0</td>
<td>1.6</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>4</td>
<td>1.7</td>
<td>1.6</td>
</tr>
<tr>
<td>8</td>
<td>1.9</td>
<td>1.9</td>
</tr>
<tr>
<td>16</td>
<td>2.1</td>
<td>2.1</td>
</tr>
</tbody>
</table>

Mean: 0.63
Standard Deviation: 0.24
Variance: 0.06

Table 5.4: Rise-fall skew (rising edge – falling edge) measurements.

<table>
<thead>
<tr>
<th>Component</th>
<th>WFC</th>
<th>FTC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t_1$</td>
<td>$t_2$</td>
</tr>
<tr>
<td></td>
<td>rise</td>
<td>fall</td>
</tr>
<tr>
<td>Wave formatter</td>
<td>3.06</td>
<td>2.89</td>
</tr>
<tr>
<td>I/O pad</td>
<td>6.95</td>
<td>5.98</td>
</tr>
<tr>
<td>Total Delay</td>
<td>10.01</td>
<td>8.87</td>
</tr>
<tr>
<td>Total Skew</td>
<td>1.14</td>
<td>1.14</td>
</tr>
</tbody>
</table>

Table 5.5: Rise-fall skew calculations (all times in ns).

The wave formatter circuit is significantly larger with 40 gates (160 transistors) and the presence of feedback further complicates optimization. Application specific optimization methods could significantly reduce the effort (e.g., [Argade89]), but these methods need to be developed.

Deskewing rise-fall skew using calibration circuits requires more sophisticated methods than those used for interchannel skew because we need to selectively adjust rising and falling edges at $t_1$ and $t_2$. Figure 5.9 shows a block diagram of the waveform formatting module including the deskewing circuits. The waveform formatter is essentially a memory element with synchronous load, set, and reset. The latch signal generates either a rising or falling edge at $t_1$ depending on the vector data. The set signal generates a rising edge at $t_2$ while the reset signal generates a falling
edge at $t_2$. Additional control signals (not shown) determine whether the set or reset inputs are active. These control signals are set up ahead of time and do not affect the skew. The timing of the latch signal is controlled by the edge timing generator ($\tau_e$) while the timing of the set and reset signals are controlled by the width timing generator ($\tau_w$).

The placement of the four edges A, B, C, and D are given by the following equations:

\[
\begin{align*}
A &= \tau_e + \tau_3 + \tau_1 \\
B &= \tau_e + \tau_4 + \tau_1 \\
C &= \tau_w + \tau_2 + \tau_4 + \tau_2 \\
D &= \tau_w + \tau_1 + \tau_3 + \tau_2 
\end{align*}
\]
where $\tau_{r_i}$ and $\tau_{f_i}$ is the respective rising and falling edge propagation delay at $t_i$ (for $i = 1$ or 2). Differences between the $\tau_{r_i}$ and $\tau_{f_i}$ delay values causes rise-fall skew at $t_i$. To deskew the edges at $t_1$ and $t_2$, we want:

$$A = B$$

$$\tau_e + \tau_3 + \tau_{r_1} = \tau_e + \tau_4 + \tau_{f_1}$$

$$\tau_3 + \tau_{r_1} = \tau_4 + \tau_{f_1}$$  \hspace{1cm} (5.1)$$

and,

$$C = D$$
Figure 5.9: Rise-fall deskew circuit.

\[
\tau_w + \tau_2 + \tau_4 + \tau_{f_2} = \tau_w + \tau_1 + \tau_3 + \tau_{f_2} \\
\tau_2 + \tau_4 + \tau_{f_2} = \tau_1 + \tau_3 + \tau_{f_2}
\] (5.2)

These equations show that any differences between \(\tau_{r_i}\) and \(\tau_{f_i}\) can be deskewed by an appropriate choice of \(\tau_1, \tau_2, \tau_3,\) and \(\tau_4\) delay values.

If we can assume that \(\tau_{r_1} > \tau_{f_1}\) and \(\tau_{r_2} > \tau_{f_2}\), (as in our case) \(\tau_2\) and \(\tau_3\) are no longer needed. Therefore, equations (5.1) and (5.2) can be reduced to equations (5.3) and (5.4) as follows:

\[
\begin{align*}
\tau_{r_1} - \tau_{f_1} &= \tau_4 \\
\tau_{r_2} - \tau_{f_2} &= \tau_4 - \tau_1
\end{align*}
\] (5.3) (5.4)

In practice, the degree of deskewing is limited by the resolution of the delay lines \(\tau_1\) to \(\tau_4\). The resolution of the existing delay lines could be enhanced by using inverters which have a propagation delay of 0.26ns. Further increases in resolution could be achieved by varying the capacitive loading on the inverters [Branson89].
Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis focuses on improving the test quality of low-end VLSI functional tester systems. Three potential areas of improvement that were considered are: memory size, tester speed, and I/O capability (timing and formatting). Increasing the memory size has little effect on test quality. Simply increasing the test speed (test pattern rate), does not provide sufficient gain in test quality compared to cost. We found that the best improvement in test quality can be obtained by improving the timing and I/O wave formatting capability of a tester.

A test strategy was developed which exploits this improved timing and I/O waveform formatting capability to generate short high-speed clock bursts. These high-speed clock bursts can be used to perform testing at speeds equivalent to a much higher rate than the tester's pattern rate. This strategy can be used to test many digital designs. However, using this strategy for “high-speed” testing requires more effort in test pattern development and more test vectors than an equivalent test performed on a high-speed tester. This approach can also be used in existing high-end testers that already have the required timing and waveform formatting capabilities.

A functional tester system for CMOS logic was developed that provides these timing and waveform formatting enhancements. The simplicity of this system was achieved by integrating most of the required circuits into modular functional tester chips (FTCs) that are used in parallel to form a tester. Vector encoding combined with an internal lookup table was used to reduce the memory bandwidth required to support the increased functionality.

A single-channel FTC, was designed and implemented. The chip has a die size of 3.8mm ×
2.9mm and requires only 40-pins which gives considerable allowance for future increases in memory width and channels. Measurements from earlier prototypes of the FTC waveform formatter give worst case timing resolutions of 1ns and maximum rise-fall skews of 1-2ns. Worst case skews between devices were on the order of 1ns. These results are very reasonable considering the relatively conservative standard cell design using 1.2μm CMOS. Improved performance is expected from a more customized and aggressive design.

6.2 Future Work

The work presented in this thesis represents an initial step towards implementing the functional tester system (FTS). More work is needed to produce a final version of the FTC. Software will also be needed to configure and control the FTS. Once completed, FTS could be used to provide a platform from which to explore VLSI testing. Some directions for future work are given below.

The effectiveness of the high speed “burst” testing strategy depends on the implementation of the device to be tested. More work is needed to formally quantify the effectiveness of this strategy for different circuit types in terms of the effort required for test vector development and the increase in the number of test vectors. It would be useful to examine the capabilities and limitations of this approach with some more concrete example circuits. The industrial benchmark circuits from MCNC (Microelectronics Corporation of North Carolina) and ISCAS (International Symposium on Circuits and Systems) may be a useful source.

The test vectors in the FTS were stored using an encoding scheme to reduce the bandwidth required between the FTC and the external memory. It will be useful to develop more optimal encoding by examining instruction distributions for test vectors. It may be possible to use data from other test systems, but there is some machine dependency in the actual test patterns.

Many aspects of the existing FTC implementation could be enhanced with more customized design. Accuracy could be improved by including skew calibration, and compensation for thermal
and/or voltage drifts which provides more stable timing. Improved timing resolution and performance may be achieved using the BiCMOS technology and the analog cells that are now available from CMC. Custom pads could be designed that supports programmable high/low voltage thresholds and drive currents for the purposes of I/O and dynamic loading. This would allow the FTC to function in a variety of environments.
Bibliography


Bibliography


Appendix A

FTC Pin Description

A description of the FTC signals is given in Table A.1.
### Appendix A. FTC Pin Description

<table>
<thead>
<tr>
<th>Name</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>I</td>
<td>This is an asynchronous active-high reset signal for the FTC chip. The reset signal must be asserted for two clock cycles for proper operation.</td>
</tr>
<tr>
<td>clk</td>
<td>I</td>
<td>This is a single-phase clock input for the FTC chip.</td>
</tr>
<tr>
<td>Data&lt;7:0&gt;</td>
<td>I/O</td>
<td>This is an 8-bit bidirectional data bus. During test or calibration mode this bus operates as an input. During program mode this bus is bidirectional.</td>
</tr>
<tr>
<td>Addr&lt;9:0&gt;</td>
<td>I/O</td>
<td>This is a 10-bit bidirectional address bus. During test mode this bus generates the addresses from which to fetch instructions. During program or calibration mode this bus accepts addresses to access internal FTC memory elements.</td>
</tr>
<tr>
<td>cs</td>
<td>I</td>
<td>This is an active-high chip select signal. This signal must be active before read/write operations on internal FTC memory elements can be initiated by the memiordb and memiowrb signals.</td>
</tr>
<tr>
<td>memiordb</td>
<td>I</td>
<td>This is an active-low read signal for FTC internal registers.</td>
</tr>
<tr>
<td>memiowrb</td>
<td>I</td>
<td>This is an active-low write signal for FTC internal registers.</td>
</tr>
<tr>
<td>mode&lt;1:0&gt;</td>
<td>I</td>
<td>These pins are used to control the current operating mode of the FTC. Mode changes are latched on the falling edge of clk.</td>
</tr>
<tr>
<td>memcs</td>
<td>I</td>
<td>This input gives external chip select signals for the memory.</td>
</tr>
<tr>
<td>MEMCS</td>
<td>O</td>
<td>This output provides the chip select control signal for the vector memory. During test mode this output is always asserted.</td>
</tr>
<tr>
<td>rwb</td>
<td>I</td>
<td>This input gives external read/write memory operations.</td>
</tr>
<tr>
<td>RWB</td>
<td>O</td>
<td>This output provides the read/write control signal for the vector memory. During test mode this signal always specify the read operation.</td>
</tr>
<tr>
<td>SYNC</td>
<td>O</td>
<td>This output provides programmable access to certain internal control signals which could be used for synchronization of other test equipment.</td>
</tr>
<tr>
<td>Fv</td>
<td>I/O</td>
<td>This pin provides the channel output which connects to the DUT.</td>
</tr>
<tr>
<td>HALTB</td>
<td>I/O</td>
<td>This signal behaves as an open drain output and should be resistively pulled high. It is active when the signal is low.</td>
</tr>
</tbody>
</table>

Table A.1: FTC pinout.
Appendix B

WFC Measurements

Tables B.1 to B.4 summarizes the WFC measurements. The average rising and falling edge times for the input clock is 3.4ns and 3.0ns respectively. The average rising and falling edge times for the WFC measurements is 3.7ns and 2.4ns respectively.

<table>
<thead>
<tr>
<th>Delay Tap</th>
<th>Delay in ns for Chip #</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Sample Var.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bypass</td>
<td>13.9 14.2 13.7 13.6</td>
<td>13.85</td>
<td>0.23</td>
<td>0.07</td>
</tr>
<tr>
<td>0</td>
<td>17.8 17.8 16.8 17.0</td>
<td>17.35</td>
<td>0.46</td>
<td>0.28</td>
</tr>
<tr>
<td>1</td>
<td>17.6 17.9 17.5 17.3</td>
<td>17.58</td>
<td>0.22</td>
<td>0.06</td>
</tr>
<tr>
<td>2</td>
<td>18.7 19.1 18.4 18.2</td>
<td>18.60</td>
<td>0.34</td>
<td>0.15</td>
</tr>
<tr>
<td>4</td>
<td>19.4 19.9 19.1 19.0</td>
<td>19.35</td>
<td>0.35</td>
<td>0.16</td>
</tr>
<tr>
<td>8</td>
<td>21.2 21.8 20.9 20.6</td>
<td>21.13</td>
<td>0.44</td>
<td>0.26</td>
</tr>
<tr>
<td>16</td>
<td>25.0 25.6 24.8 24.4</td>
<td>24.95</td>
<td>0.43</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Table B.1: Rising edge at $t_1$.

<table>
<thead>
<tr>
<th>Delay Tap</th>
<th>Delay in ns for Chip #</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Sample Var.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bypass</td>
<td>11.3 11.8 11.4 11.3</td>
<td>11.45</td>
<td>0.21</td>
<td>0.06</td>
</tr>
<tr>
<td>0</td>
<td>15.8 16.2 15.4 15.4</td>
<td>15.70</td>
<td>0.33</td>
<td>0.15</td>
</tr>
<tr>
<td>1</td>
<td>16.6 17.0 16.4 15.8</td>
<td>16.45</td>
<td>0.43</td>
<td>0.25</td>
</tr>
<tr>
<td>2</td>
<td>17.0 17.3 16.8 16.6</td>
<td>16.93</td>
<td>0.26</td>
<td>0.09</td>
</tr>
<tr>
<td>4</td>
<td>17.7 18.3 17.6 17.5</td>
<td>17.78</td>
<td>0.31</td>
<td>0.13</td>
</tr>
<tr>
<td>8</td>
<td>19.3 19.9 19.4 19.1</td>
<td>19.43</td>
<td>0.29</td>
<td>0.12</td>
</tr>
<tr>
<td>16</td>
<td>22.9 23.5 22.8 22.8</td>
<td>23.00</td>
<td>0.29</td>
<td>0.11</td>
</tr>
</tbody>
</table>

Table B.2: Falling edge at $t_1$.
### Appendix B. WFC Measurements

#### Table B.3: Rising edge at $t_2$.

<table>
<thead>
<tr>
<th>Delay Tap</th>
<th>Delay (ns) for Chip #</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Sample Var.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>13.7</td>
<td>14.2</td>
<td>13.8</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>17.6</td>
<td>17.8</td>
<td>17.4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>17.7</td>
<td>18.2</td>
<td>17.4</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>18.6</td>
<td>19.2</td>
<td>18.4</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>19.8</td>
<td>20.4</td>
<td>19.5</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>21.8</td>
<td>22.4</td>
<td>21.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25.8</td>
<td>26.5</td>
<td>25.5</td>
</tr>
</tbody>
</table>

#### Table B.4: Falling edge at $t_2$.

<table>
<thead>
<tr>
<th>Delay Tap</th>
<th>Delay in ns for Chip #</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Sample Var.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>13.1</td>
<td>13.3</td>
<td>12.8</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>16.8</td>
<td>17.2</td>
<td>16.6</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>17.6</td>
<td>18.2</td>
<td>16.8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>17.9</td>
<td>18.5</td>
<td>17.8</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>19.3</td>
<td>19.9</td>
<td>19.0</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>21.0</td>
<td>21.5</td>
<td>21.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25.0</td>
<td>25.8</td>
<td>24.6</td>
</tr>
</tbody>
</table>
Appendix C

FTC Implementation Details

C.1 FTC Circuit Documentation

acquire

The acquire module provides the input window comparison function. This module activates when a channel switches to input mode (fmt<2>=0). The compare value and the DUT value is compared by the XOR gate 163. The result of this comparison is used to set the error bit (EB) when the comparison window is open. The comparison window opens when fv is high. The opening and closing of the comparison window is synchronized to the edges of fv. As long as the comparison remains error free (EB.Q=0) and the channel is in input mode, the compare counter (cmpcnt) increments on every tclk cycle.

The comparison vector changes on the falling edge of tclk. This change does not occur on the test cycle boundary because of skew between tclk and the test cycle. The “cv synchronizer” synchronizes comparison vector changes to the actual test period boundary. Comparison windows that extend beyond the current test cycle retains the same comparison vector in the next cycle as in the current cycle. That is, the comparison vector (cv) will not be changed. Since cv changes occur on the test cycle, comparisons should not occur near the test period boundaries without a guard-band. Otherwise errors are guaranteed to be reported.

addregc

The addregc (address register counter) is a 13-bit loadable ripple counter. The lower 8 bits or the upper 5 bits could be read through the DOUT<7:0> bus using rdb<0>=0 and rdb<1>=0


respectively. $DOUT$ is tristated when $rdb<1:0> = 11$.

When $(loaden \text{ or } enb)=1$ data is loaded from the $ain<12:0>$ bus. The $enb$ signal also switches the $ckb$ signal of the counters. Transitions on the $enb$ signal should occur only when the destination clock signal is high.

Data is loaded from the $ain<12:0>$ bus when $(loaden \text{ or } enb) = 1$. The $enb$ signal also selects one of two signals for clocking the counters. Consequently, the $enb$ signal should select the next clock signal only when the next clock is in the high state. This avoids inadvertently generating a falling edge which would trigger the counter. The counter increments when $counten$ is high. The $counten$ signal should not change on the active edge of the counter.

adecmod

The $adecmod$ (address decoder module) decodes the address space of the chip into blocks of 32 addresses each. As shown in the schematic, the full address space is not fully decoded – only bits 6 and 7 are used to identify a block.

aregmod

In a multichannel implementation, this schematic would contain one channel address register module ($caregmod$) for each channel. The decoders (118 and 119) select from which $caregmod$ to read or write.

caregmod

The $caregmod$ (channel address register module) schematic has the registers that contain the current address to be fetched, the restart address (lower bound) and the ending address (upper bound). The restart and ending addresses are included inside the address range.

The address is incremented on the falling edge of the $incb$ signal. Disabling $counten$ prevents the address from incrementing. Disabling $wrapenb$ prevents the current address from wrapping to the restart address after it reaches the ending address. Asserting the $aoutenb$ signal
Table C.1: Memory map of address space registers in the caregmod module.

will tristate the AOUT<12:0> address bus. The tm_enb (test mode enable) signal must be asserted before the current address is incremented.

Memory Map

The decoders DR and DW selects the registers to read and write according to the memory map given in Table C.1.

END Signal

END signals the controller to terminate test mode at the end of the current test cycle. This signal is used to stop testing at the end of a channels instruction sequence (instead of wrapping). Since instructions are pipelined and execute in variable number of cycles, the last instruction will not be executed until a variable number of cycles after it has been fetched.

The END GENERATOR is the circuit responsible for generating the END signal at the correct time to stop testing after the last instruction in the channels address space has been executed. This circuit is enabled when wrapenb is disabled.

A low on I78.Q enables I69 to generate END when creadyb is high (i.e., the channel finished executing the last instruction and is ready for the next one.) The flip-flops I62, I66 and I78 are used to delay the low signal generated by CMP.EQB, which indicates that the end address is reached, until the last instruction actually completed execution in the pipeline.
Appendix C. FTC Implementation Details

chmod

The chmod channel module contains the vector decoder and format memory (vectmod), the waveform formatter (wfmt2) and the input comparison (acquire) circuits.

Signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rst&lt;1:0&gt;, sreset</td>
<td>These are the reset (active high/low) and the start reset signals.</td>
</tr>
<tr>
<td>enb</td>
<td>This activates the channel module.</td>
</tr>
<tr>
<td>rdb, wrb, addr&lt;3:0&gt;, din&lt;7:0&gt;</td>
<td>These pins provide the interface to the format memory.</td>
</tr>
<tr>
<td>ac_rdb&lt;1:0&gt;</td>
<td>These signals are used to read the acquire counter.</td>
</tr>
<tr>
<td>ivectin&lt;7:0&gt;, c_lat</td>
<td>These signals are for the instruction bus.</td>
</tr>
<tr>
<td>ck&lt;1:0&gt;, tck&lt;1:0&gt;, loaden&lt;1:0&gt;</td>
<td>These are clocking signals.</td>
</tr>
<tr>
<td>padin_v</td>
<td>This pin provides the pad input used for input comparison.</td>
</tr>
<tr>
<td>fu_en</td>
<td>This pin provides the pad enable signal used to activate the input comparison.</td>
</tr>
</tbody>
</table>

Table C.2: Input signals for chmod.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>unformatted vector.</td>
</tr>
<tr>
<td>FV_ENABLE</td>
<td>enable/disable output.</td>
</tr>
<tr>
<td>FV</td>
<td>formatted output.</td>
</tr>
<tr>
<td>CREADYB</td>
<td>channel ready, signal to fetch next instruction.</td>
</tr>
<tr>
<td>DOUT</td>
<td>data out for read access to format memory.</td>
</tr>
</tbody>
</table>

Table C.3: Output signals for chmod.
chselmod

This module generates the following timing signals for pipelining the memory fetch:

1. \textit{CHSELB<3:0>} (channel select): This signal is an active low pulse.

2. \textit{CECHSEL.PDEL<3:0>} (channel enabled channel select with phase delay i.e., delayed 1 phase from \textit{CHSELB}): This signal is basically a phase delayed version of \textit{CHSELB<3:0>} that is enabled by \textit{CHENB<3:0>}.

3. \textit{CHENB<3:0>} (channel enable): This signal is generated by latching the \textit{creadyb<3:0>} (channel ready) signal using the rising edge of \textit{CHSELB<3:0>}.

Four versions of these three signals are generated - one for each channel. The \textit{CHSELB<3:0>} and \textit{CECHSEL.PDEL<3:0>} signals are asserted once per \textit{tclkb} (test clock) period. Consequently, the \textit{CHENB<3:0>} signal can change at most once per \textit{tclkb} period.

\textbf{Counter Reset Circuit}

This circuit ensures that \textit{CHSELB<3:0>} and \textit{CECHSEL.PDEL<3:0>} are asserted once per \textit{tclkb} period. (The \textit{tclk} signal may range from 4-255 \textit{clk} cycles.) Once the 2-bit counter reaches 11, further counting is disabled using I36 until \textit{tclkb} is 1. I37 is needed to disable I36 for the first cycle.

\textbf{Channel Status Register}

This circuit stores the status (\textit{CREADY}) of each channel.

\textbf{2-bit counter}

This two bit counter is decoded to generate \textit{CHSEL<3:0>} and \textit{CECHSEL.PDEL<3:0>}.

\textbf{Counter Decoder and Output Sync}

This circuit decodes the 2-bit counter and provides the appropriate output synchronization.
Appendix C. FTC Implementation Details

chsmod

This schematic contains one chmod for each channel in the chip. The decoders select which channel to perform the read/write operation on. The 142 FF is used to override the $FV_{EN}$ signal when the channel is inactive.

clkmod

The clock module contains all the circuits that generate timing related signals.

cmp13

This is a 13-bit comparator with equality indicated by an active low output.

cmpcnt

This 12-bit counter has an overflow bit to indicate that a counter overflow has occurred. This counter is reset by $sreset$ which is asserted at the start of each test.

cntrl

This module generates internal control signals for the various operating modes and monitors the internal and external signals for termination conditions. Mode changes (program, calibration, and test) are synchronized to the falling edge of $clk$.

Test Mode

Once test mode is entered, it could be terminated by several causes: i) an external halt, ii) a mode change, iii) an end of instruction sequence, or iv) a comparison error. If the termination signal (I62) is received more than one $clk$ cycle prior to the end of the $tclk$ cycle, test mode will be terminated at the end of the current $tclk$ cycle. Otherwise, testing will stop at the end of the next $tclk$ cycle.
Appendix C. FTC Implementation Details

Test mode stop causes the clock to stop which unasserts tmclk_enb on the next falling edge of clk. This action causes tm_enb to be unasserted after stopped is asserted. The aendb termination signal should be timed to cause testing to stop when the ending instruction has completed execution.

Termination caused by compare errors (signaled by cmp_okb) always cause testing to stop at the end of the next test cycle, because the termination signal is always received at the beginning of the test cycle following the erroneous test cycle. Termination due to internal causes will generate an external halt signal.

delaycnt

This 6-bit ripple counter provides synchronous delay for placing waveform edges.

dlyln

This variable-length delay line allows lengths from 0 to 31 gate delays. Bypass of the delay line is also provided by setting del<0>=0.

fmem

The format memory is 31 bits wide. The current implementation has 4 words. Data is written into this memory 8 bits at a time, but is read 32 bits at once. For external (off-chip) reading, the multiplexor tree selects which byte to output. During test mode (i.e. tm_enb==0) reading is always enabled.

fmemmod

The “format memory index” points to the format last specified by the format instruction. At the start of a test, this register is reset to point at the format zero, the default format. This five bit register could index up to 32 formats although only 4 formats are provided in this prototype.

The outputs of the format memory are partially buffered by I12. The unbuffered outputs are latched elsewhere (directly into counters located in the waveform formatting circuits).
The $FMT_{30:0}$ bus provides the current format data while the $NFMT_{18,1:0}$ bus provides format data that will be used during the next test period. Certain wave formatting circuits need this information ahead of time.

The $fmt_{2}$ bit is reset to disable output before the start of test.

**fmword**

This is a 31-bit word for the format memory.

**gatedel**

This provides the minimal delay element for the delay line. (The delay line is actually laid out so this schematic is only used for simulation purposes. Changing this schematic will not change the actual implementation.) The edgedel (edge delay) module generates a falling edge that latches the current test vector for driving the output. This module comprises of three major components:

1. $delaycnt$ (delay counter): A synchronous counter which subdivides the test period ($tclk$ periods) into a number of $clk$ periods.

2. **one-shot pulse generator**: The dffrbs and the combinational logic forms an active low pulse generator that is triggerable once per $tclk$ period. This pulse generator is triggered synchronously by WFSTART and $clk$. The pulse duration is either the low or high phase of $clk$ depending on $del_{5}$. The $tclkb$ asynchronously resets the pulse generator on its falling edge.

3. $dlyln$ (delay line): A delay line formed using buffer (buf) cells as gate delays.

More timing details about this module is given in the hand drawn timing diagrams.

**idecdel**

The idecdel (instruction decode) module decodes the instruction specified by the 2-bit opcode (opcode<1:0>). The operations performed by these instructions are executed by combinations of the four control signals given in Table C.4.
### Table C.4: Control signals generated by instruction decoder.

<table>
<thead>
<tr>
<th>Control Signal</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>WXFD.ENB</td>
<td>- used to load format index&lt;br&gt;- indicates a format instruction</td>
</tr>
<tr>
<td>SLD.ENB</td>
<td>- used to load the shift register</td>
</tr>
<tr>
<td>SCLD.ENB</td>
<td>- enables the shifting of the shift register&lt;br&gt;- indicates a vector instruction</td>
</tr>
<tr>
<td>RLD.ENB</td>
<td>- used to load the counter</td>
</tr>
</tbody>
</table>

Table C.5 shows how these four control signals are used to implement the functionality of the four instructions.

<table>
<thead>
<tr>
<th></th>
<th>WXFD.ENB</th>
<th>SLD.ENB</th>
<th>SCLD.ENB</th>
<th>RLD.ENB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Format</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Repeat</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Note: 0=active 1=inactive (i.e. active-low)

Table C.5: Control signals for executing instructions.

When a channel is not ready to decode a new instruction, cready becomes low which causes the four control signals to synchronously become unasserted (high). The inactive control signals allow the various controlled modules to complete their operations. Once an instruction completes execution, its channel is ready for the next instruction and cready becomes high.

The latching edges for the four control signals are:

- \( tck \) falling edge: \( WXFD.ENB \)
- \( loadenb \) falling edge: \( SLD.ENB, SCLD.ENB, \) and \( RLD.ENB \)

\textbf{iobufmod}

The I/O buffer module provides buffering for input instructions. Actually, this level of buffering is only needed in designs that exploit the idle periods in the instruction stream. This module is used
for input buffering in the current design.

**latreg13**

This 13-bit resister with output enable is used for static address bound registers. The memory I/O module contains the address registers and the I/O buffers. The I30 and MUX allows the iobufmod to be selectively written during calibration mode. During test mode, the write signals are internally generated.

**modebits**

This module contains the bits for the following three modes: i) stop on compare error, ii) stop on address end, and iii) sync signal output. The address decoding that is also done in this module is listed in Table C.6.

**pulsegen0**

The pulsegen0 (pulse generator active low pulse) circuit generates an active low pulse used for clearing the vector flip-flop. This pulse is generated at the beginning of the test period (e.g. RL) and/or at the t1 edge (the width time). In the following description, these pulses will be referred to as the START PULSE and the WIDTH PULSE respectively.

**START PULSE**

The start pulse is generated only for the RL format and the RC format with \( V=1 \). Since the start pulse occurs right at the beginning of the test period (it is triggered by \( tclk\rightarrow0 \)), the D input data must be setup before \( tclk\rightarrow0 \). These inputs (\( nfmt<1:0> \) and \( nv \)) are determined by looking ahead to the next test period. The start pulse is activated when SP0.Q is 1. Table C.7 gives the truth table for generating SP0.D. This table was reduced using the karnaugh map shown in Table C.8. Equation C.1 gives the actual logic function that was implemented.
### Appendix C. FTC Implementation Details

<table>
<thead>
<tr>
<th>10:6</th>
<th>5:0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td></td>
<td>System Space</td>
</tr>
<tr>
<td>00000</td>
<td></td>
<td>tclk period [W/O]</td>
</tr>
<tr>
<td>00001</td>
<td></td>
<td>address wrap (4) / stop on compare error (4) [W/O]</td>
</tr>
<tr>
<td>00010</td>
<td></td>
<td>sync. output select (3-bit) [W/O]</td>
</tr>
<tr>
<td>0001X</td>
<td></td>
<td>Channel buffers during calibrate mode [W/O]</td>
</tr>
<tr>
<td>01000</td>
<td></td>
<td>Addr. end status (4) / comp. error status (4) [R/O]</td>
</tr>
<tr>
<td>00100</td>
<td></td>
<td>LSB ch#0 error count [R/O]</td>
</tr>
<tr>
<td>00101</td>
<td></td>
<td>MSB ch#0 error count [R/O]</td>
</tr>
<tr>
<td>00110</td>
<td></td>
<td>LSB ch#1 error count [R/O]</td>
</tr>
<tr>
<td>00111</td>
<td></td>
<td>MSB ch#1 error count [R/O]</td>
</tr>
<tr>
<td>0011X</td>
<td></td>
<td>LSB ch#2 error count [R/O]</td>
</tr>
<tr>
<td>00111</td>
<td></td>
<td>MSB ch#2 error count [R/O]</td>
</tr>
<tr>
<td>00111</td>
<td></td>
<td>LSB ch#3 error count [R/O]</td>
</tr>
<tr>
<td>00111</td>
<td></td>
<td>MSB ch#3 error count [R/O]</td>
</tr>
<tr>
<td>00001</td>
<td></td>
<td>Input Address Space Bounds</td>
</tr>
<tr>
<td>0XX00</td>
<td></td>
<td>LSB restart address for ch#XX</td>
</tr>
<tr>
<td>0XX01</td>
<td></td>
<td>MSB restart address for ch#XX</td>
</tr>
<tr>
<td>0XX01</td>
<td></td>
<td>LSB end address for ch#XX</td>
</tr>
<tr>
<td>0XX11</td>
<td></td>
<td>MSB end address for ch#XX</td>
</tr>
<tr>
<td>0XX10</td>
<td></td>
<td>LSB starting address for ch#XX</td>
</tr>
<tr>
<td>0XX11</td>
<td></td>
<td>MSB starting address for ch#XX</td>
</tr>
<tr>
<td>00010</td>
<td></td>
<td>Format Memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Currently using only block 00010</td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ccffbb</td>
<td></td>
<td>cc = channel number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ff = format number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bb = byte number (byte ordering: 3 2 1 0)</td>
</tr>
</tbody>
</table>

Table C.6: Address decoding in *modebits* module.
Appendix C. FTC Implementation Details

Table C.7: Truth table for generating SP0.D in pulsegen0 module.

<table>
<thead>
<tr>
<th>$n_v$</th>
<th>$nfmt&lt;1:0&gt;$</th>
<th>SP0.D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1 0 (RL)</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1 1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 (RC)</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 0 (RL)</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1 1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table C.8: Karnaugh map for generating SP0.D in pulsegen0 module.

\[

nfmt<1:0>
\begin{array}{cccc}
00 & 01 & 11 & 10 \\
0 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 \\
\end{array}

\]

\[
SP0.D = nfmt<0> + nfmt<1> \cdot n_v
\]

\[
= nfmt<0> + nfmt<1> + n_v
\]  \hspace{1cm} (C.1)

The SP0 flip-flop is active in the 1 state (since this causes $PULSE_RB \rightarrow 0$). The SP0 FF is reset when $fub == 1$ (i.e. the vector FF is 0).

**WIDTH PULSE**

The width pulse is generated only for the RL format and the RC format with $V=1$. If these conditions are met, the width pulse is generated when wpin goes low. The width pulse is activated when WP0.Q is 1. This depends on the combinational logic at the D input. Table C.9 gives the truth table for generating WP0.D. This table was reduced using the karnaugh map shown in Table C.10. Equation C.2 gives the actual logic function that was implemented.
Table C.9: Truth table for generating WP0.D in pulsegen0 module.

<table>
<thead>
<tr>
<th>$v$</th>
<th>fmt&lt;1:0&gt;</th>
<th>WP0.D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1 0 (RL)</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1 1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 (RC)</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 0 (RL)</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1 1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table C.10: Karnaugh map for generating WPO.D in pulsegen0 module.

\[
\begin{array}{cccc}
fmt<1:0> & 00 & 01 & 11 & 10 \\
v & 0 & 0 & 0 & 0\quad 1 \\
1 & 1 & 0 & 0 & 1
\end{array}
\]

The WPO flip-flop is active in the 1 state (since this causes PULSE_RB→0). The WP0 FF is reset when $\text{fub} == 1$ (i.e. the vector FF is 0).

**pulsegenn1**

The *pulsegenn1* (pulse generator active high pulse) circuit generates an active high pulse used for setting the vector flip-flop. This pulse is generated at the beginning of the test period (e.g. RH) and/or at the t1 edge (the width time). In the following description, these pulses will be referred to as the START PULSE and the WIDTH PULSE respectively.

**START PULSE**
The start pulse is generated only for the RH format and the RC format with \( V=0 \). Since the start pulse occurs right at the beginning of the test period (it is triggered by \( tclkb\rightarrow0 \)), the D input data must be setup before \( tclkb\rightarrow0 \). These inputs \( (nfmt<1:0> \) and \( nv \) are determined by looking ahead to the next test period. The start pulse is activated when \( SP1.Q=0 \). Table C.11 gives the truth table for generating WP1.D. This table was reduced using the karnaugh map shown in Table C.12. Equation C.3 gives the actual logic function that was implemented.

<table>
<thead>
<tr>
<th>( nv )</th>
<th>( nfmt&lt;1:0&gt; )</th>
<th>( SP1.D )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 (RC)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0 0 1 (RH)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1 0 1 (RH)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Table C.11: Truth table for generating \( SP1.D \) in pulsegen1 module.

\[
\begin{array}{cccc}
   \text{nfmt}<1:0> & 00 & 01 & 11 & 10 \\
   \text{nv} & 0 & 0 & 0 & 1 & 1 \\
   & 1 & 1 & 0 & 1 & 1 \\
\end{array}
\]

Table C.12: Karnaugh map for generating \( SP1.D \) in pulsegen1 module.

\[
SP1.D = nfmt < 1 > + nv * fmt < 0 > \\
= nfmt < 1 > * nv * nfmt < 0 > \tag{C.3}
\]

The \( SP1 \) flip-flop is active in the 0 state (since this causes \( PULSE.RB\rightarrow1 \)). The \( SP1 \) FF is reset when \( fvb == 0 \) (i.e., the vector FF is 1).
WIDTH PULSE

The width pulse is generated only for the RH format and the RC format with \( V=0 \). If these conditions are met, the width pulse is generated when \( wpin \) goes low. The width pulse is activated when \( WP1.Q \) is 0. This depends on the combinational logic at the D input. Table C.13 gives the truth table for generating \( WP1.D \). This table was reduced using the karnaugh map shown in Table C.14. Equation C.4 gives the actual logic function that was implemented.

\[
\begin{align*}
\text{Table C.13: Truth table for generating } WP1.D \text{ in } pulsegen1 \text{ module.} \\
\begin{array}{c|c|c}
\text{v} & \text{fmt<1:0>} & WP1.D \\
\hline
0 & 0 0 (RC) & 0 \\
0 & 0 1 (RH) & 0 \\
0 & 1 0 & 1 \\
0 & 1 1 & 1 \\
1 & 0 0 & 1 \\
1 & 0 1 (RH) & 0 \\
1 & 1 0 & 1 \\
1 & 1 1 & 1 \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\text{fmt<1:0>} \\
\begin{array}{c|c|c|c|c}
\text{v} & 00 & 01 & 11 & 10 \\
\hline
0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
WP1.D &= fmt < 1 > + v \cdot fmt < 0 > \\
&= fmt < 1 > \cdot v + fmt < 0 > \\
\end{align*}
\]

(C.4)

The WP1 flip-flop is active in the 0 state (since this causes \( PULSE.RB \rightarrow 1 \)). The WP1 FF is reset when \( fub == 0 \) (i.e. the vector FF is 1). When \( resetb \) or \( sresetb \) is low, \( PULSE.S \) also goes low to reset the VFF.
rscnt

The repeat/shift counter keeps track of the number of cycles required by an instruction. Multicycle instructions must load the appropriate count to postpone decoding of the next instruction via creadyb. Single cycle instructions do not need to initialize this counter. The vector instruction generates scld.enb==0 which causes d<5:0>=111001 regardless of din<5:0>. The repeat instruction simply loads the desired number of repetitions into the counter.

sdisrbit

This shift register bit has the following truth table:

<table>
<thead>
<tr>
<th>s</th>
<th>rb</th>
<th>din</th>
<th>load1enb</th>
<th>d2in</th>
<th>load2enb</th>
<th>clk</th>
<th>D</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>↓</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>a</td>
<td>1</td>
<td>↓</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>b</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>↓</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

* D has the same decoding as the second part of the table.

shifter

Test data bits must go through this 7-bit shift register to be output to the wave formatter. This 7-bit shift register is primarily used to perform parallel to serial conversion on the vector sequence instruction. The multiplexor, I27, allows copying the test data bit from bit 5 of the format instruction into the MSB of the SR for immediate output. The NV output gives the vector that would be used in the next test period.

sync

This module multiplexes several signals to the SYNC output.
Appendix C. FTC Implementation Details

<table>
<thead>
<tr>
<th>ss_sel&lt;2:0&gt;</th>
<th>SYNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>creadyb</td>
</tr>
<tr>
<td>001</td>
<td>cmp_okb</td>
</tr>
<tr>
<td>010</td>
<td>v (unformatted vector)</td>
</tr>
<tr>
<td>011</td>
<td>fv_en (formatted vector I/O direction)</td>
</tr>
<tr>
<td>100</td>
<td>tclkb</td>
</tr>
<tr>
<td>101</td>
<td>loadenb</td>
</tr>
<tr>
<td>110</td>
<td>sresetb</td>
</tr>
<tr>
<td>111</td>
<td>tm_enb</td>
</tr>
</tbody>
</table>

tcbl

The tcbl (tester chip block level) schematic contains the core blocks of the tester chip.

tcil

The tcil (tester chip interface level) schematic adds some of the interface related blocks to the tcbl.

tclkgen2

The tclkgen2 (test clock generator 2 *) module generates the tclk, tclkb, loadenb and stopped signals. Since tclk is synchronously generated from clk, its period is an integral number of clk periods. The low phase of tclk is always two clk periods; while the high phase of tclk can vary from 2-259 clk periods (specified by value loaded into the tclk high-phase register). When tclkgen2 is inactive, tclk and loadenb are high while tclkb is low. The tm_enb signal is used to active tclkgen2 and should be asserted on the rising edge of clk. The first tclk period begins one clk phase after tm_enb is asserted. When tm_enb is unasserted, tclkgen2 remains active until the current tclk period is over. The stopped signal gives the current status of tclkgen2.

Clean Stop Circuit

Upon receiving a disable signal, this circuit causes the clock generator to stop at the end of the current test period. This feature is intended to support cleanly stopping and restarting of
testing. The reset signal stops the clock generator immediately.

tcpl

The tcpl (tester chip pad level) schematic adds the pads to the chip.

tmdel

This circuit provides the appropriate synchronous delay in the test mode signal to properly start-up the pipeline. This module also generates the start reset signal (sresetb).

ts

The ts (tester system) module provides a system level simulation of the tester chip. A ROM module, I1, is used to store the encoded vectors used for simulations. This ROM module was produced using a ROM generator.

vectmod

This module contains the instruction latch (I4), instruction decoder (I36), the format memory module (I25), a 7-bit shift register (I63) and a counter (I3). These components generate the vector and formatting data required by the wave formatting module.

The CREADYB signal indicates whether the channel is ready for more instructions. This signal is used for instructions that could provide vectors for more than one test cycle. In these cases, the counter (I3) is used to determine when new data is required. The instruction decoder (I36, idecdel) generates the control signals required to execute the various instructions.

vectsel

This circuit selects the appropriate test data source depending on the format chosen.
Appendix C. FTC Implementation Details

wfmt2

This waveform formatter module generates a formatted waveform given the desired format and test data value.

widthdel

The `widthdel` (width delay) module generates an active low pulse used to reset the flip flop after the test vector has been applied for an appropriate duration ("width"). The operation of this module is similar to that of the `edgedel` except that the D input of I82 provides a means for disabling the pulse (in `edgedel` the D input is tied high).

C.2 FTC Timing

Figure C.1 gives a timing diagram showing the fetching of instructions to the decoding stage. The signals shown in gray are used in the four channel version.
Appendix C. FTC Implementation Details

Figure C.1: Timing diagram showing the fetching of instructions to the decoding stage.
C.3 FTC Circuit Schematics

This section contains the schematics corresponding to the circuit documentation provided in Section C.1.
Appendix C. FTC Implementation Details
Appendix C. FTC Implementation Details
Appendix C. FTC Implementation Details
Appendix C. FTC Implementation Details