Shih-Chuan Hung for the degree of Master of Science in Electrical and Computer Engineering presented on August 5, 1988

Title: THE DESIGN OF A SINGLE-CHIP LOGIC STATE ANALYZER

Redacted for Privacy

Abstract approved: 

John M. Murray

This thesis describes the design of a single-chip logic analyzer. The utility of such a chip in measuring and recording digital system parameters is illustrated. Possibilities for future applications of this chip in performance monitoring of computer systems are discussed.
THE DESIGN OF A SINGLE CHIP LOGIC STATE ANALYZER

by

Shih-Chuan Hung

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed August 5, 1988
Commencement June, 1989
APPROVED:
Redacted for Privacy

Redacted for Privacy

Redacted for Privacy

Head of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented  August 5, 1988
ACKNOWLEDGEMENTS

I would like to give special thanks to my major professor, Prof. John M. Murray, for his support throughout my thesis at OSU. I wish to thank Prof. Rudolf S. Engelbrecht, my minor professor, for his guidance. Special thanks to Prof. S. John T. Owen and Prof. Robert J. Schultz for acting as my committee members and for giving me useful advice.

I would like to express my deepest thanks to Mr. Scott A. Starr and Mr. Russell Ruby for their help in correcting and editing my thesis. Without their help, I would not have been able to finish my thesis in a timely manner. I am also grateful to Mr. Craig Overhage from Tektronix for his help and time in obtaining materials dealing with the theory of logic analyzers, and his tutorial in fundamental logic analyzer concepts.

I would like to take this opportunity to thank my parents and family members for their financial support and encouragement, without which I would not have been able to come to the USA for my studies.
# TABLE OF CONTENTS

1. Introduction .............................................. 1  
   1.1 Definition of a Logic Analyzers ...................... 1  
   1.2 An Overview of Logic Analysis ........................ 1  
   1.3 Present And Future Markets For Logic Analyzers .... 3  
   1.4 Logic Analyzer Microarchitecture .................... 4  
      1.4.1 Sampling Circuit ................................ 6  
      1.4.2 Triggering Circuit ................................ 7  
      1.4.3 Storage System ................................... 14  
      1.4.4 Computing Engine ................................ 14  
      1.4.5 Display .......................................... 14  
   1.5 The State of The Art in Logic Analyzers ............ 16  
   1.6 An Example of a Single-Chip Logic Analyzer .......... 19  
      1.6.1 Brief Description of Logic Analyzer Operation .... 19  
      1.6.2 An Example ....................................... 22  
2. Logic Analyzer Chip Architecture ....................... 29  
   2.1 System Specification ................................. 29  
      2.1.1 Handshaking Approach ............................. 30  
      2.1.2 The Main Controller Algorithm .................... 33  
   2.2 Microarchitecture .............................. 36  
   2.3 An Implementation of The Microarchitecture ........... 39  
      2.3.1 The Hardware Module .............................. 39  
      2.3.2 The Memory Module ................................. 42  
      2.3.3 The Controller Module ............................. 42  
      2.3.4 Chip Netlisting ................................ 45  
      2.3.5 Chip Floorplanning ............................... 45  
3. Silicon Compilation ................................. 47  
   3.1 An Overview of the GENESIL Silicon Compiler ........ 47  
   3.2 Design Methodology .................................. 47  
4. Functional Simulation ................................. 53  
   4.1 Simulation Environment .............................. 53  
   4.2 MASM Source Code .................................. 54  
5. Timing Analysis .................................. 57  
   5.1 Timing Analysis Environment ........................ 57  
   5.2 Timing Reports ..................................... 58  
6. Conclusions ........................................... 62  
   6.1 Extensions And Critiques ............................ 62  
   6.2 Conclusions ......................................... 62  
Bibliography ................................................. 64
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Block Diagram of a Logic Analyzer</td>
<td>5</td>
</tr>
<tr>
<td>1-2</td>
<td>Block Diagram of the Triggering Circuit</td>
<td>8</td>
</tr>
<tr>
<td>1-3(a)</td>
<td>Synchronous Word Recognition</td>
<td>9</td>
</tr>
<tr>
<td>1-3(b)</td>
<td>Asynchronous Word Recognition</td>
<td>10</td>
</tr>
<tr>
<td>1-3(c)</td>
<td>Word Recognition With a Qualifier Input</td>
<td>12</td>
</tr>
<tr>
<td>1-4</td>
<td>An Example of Sequential Triggering</td>
<td>13</td>
</tr>
<tr>
<td>1-5</td>
<td>Block Diagram of the Storage System</td>
<td>15</td>
</tr>
<tr>
<td>1-6(a)</td>
<td>A Timing Diagram</td>
<td>17</td>
</tr>
<tr>
<td>1-6(b)</td>
<td>A State Diagram for a Hexidecimal Display</td>
<td>17</td>
</tr>
<tr>
<td>1-7</td>
<td>Logic Analyzer Architectural Block Diagram</td>
<td>20</td>
</tr>
<tr>
<td>1-8</td>
<td>A Program Path</td>
<td>23</td>
</tr>
<tr>
<td>2-1(a)</td>
<td>The Input Format Specification</td>
<td>31</td>
</tr>
<tr>
<td>2-1(b)</td>
<td>Another Possible Input Format Specification</td>
<td>31</td>
</tr>
<tr>
<td>2-2</td>
<td>A Brief Flowchart of the Main Algorithm</td>
<td>34</td>
</tr>
<tr>
<td>2-3</td>
<td>The Chip Hierarchy</td>
<td>37</td>
</tr>
<tr>
<td>2-4</td>
<td>The Main Module Microarchitecture</td>
<td>38</td>
</tr>
<tr>
<td>2-5</td>
<td>Hardware Module Interconnections</td>
<td>40</td>
</tr>
<tr>
<td>2-6</td>
<td>Block Diagram of the Memory Module</td>
<td>43</td>
</tr>
<tr>
<td>2-7</td>
<td>Block Diagram of the Controller Module</td>
<td>44</td>
</tr>
<tr>
<td>3-1</td>
<td>Genesil Silicon Development System</td>
<td>48</td>
</tr>
<tr>
<td>3-2</td>
<td>The Silicon Compilation Process</td>
<td>52</td>
</tr>
<tr>
<td>4-1</td>
<td>Simulation Environment</td>
<td>55</td>
</tr>
<tr>
<td>5-1</td>
<td>Timing Analysis Environment</td>
<td>59</td>
</tr>
</tbody>
</table>
1. Introduction

This thesis presents the design and simulation of a single-chip logic state analyzer. The GENESIL Silicon Compiler System was utilized in its implementation.

1.1 Definition of a Logic Analyzer

A logic analyzer is a device used to observe and record the operations occurring within a digital system. The logic analyzer presents to the user timing diagram, numeric displays, and mnemonic opcodes indicative of these operations.

1.2 An Overview of Logic Analysis

The logic analyzer is a measurement device that has its origins in the oscilloscope. Like the oscilloscope, a logic analyzer is used to measure the voltage levels of waveforms over time. Unlike oscilloscopes, which are used to measure continuous waveforms, logic analyzers are used to collect and analyze digital or discrete waveforms. Digital waveforms are generated by logic circuits and microprocessor systems.
The oscilloscope, an electronic instrument is used to acquire and present analog data in the form of an x-y coordinate graph. It displays the amplitudes of the input signals on the vertical axis, and time on the horizontal axis. Logic analyzers perform a similar function, but are limited by the digital amplitude requirements imposed on the input signals. Pseudo-waveforms, based on the crossing of a specified logic threshold by the input voltage, are typically displayed by logic analyzers.

Oscilloscopes and logic analyzers can accept more than data input stream at a time, each stream being called a channel. Today, the largest number of channels that can be displayed on an oscilloscope screen is 8. An oscilloscope must shift or multiplex rapidly from channel to channel to acquire many channels of data. For some digital applications, the number of inputs available from a conventional oscilloscope is not enough; for instance, a microprocessor-based product may require between 24 and 48 channels to be observed simultaneously. A logic analyzer can collect data on all of its channels simultaneously, and is not; in general, limited to eight channels.

A logic analyzer does not provide the voltage resolution or time interval accuracy as the oscilloscope. The key feature of a logic analyzer is that it can be triggered by user specified patterns of waveforms occurring on the input channels, and store such input data as might be of interest accordingly. Logic analyzers are
particularly useful when observing the timing relationships between signals occurring on a microprocessor data, address, or control bus. Logic analyzers are capable of decoding the information on these buses and presenting it in a form meaningful to the user.

1.3 Present And Future Markets For Logic Analyzers

Modern digital systems contain an increasing number of internal and external signal nodes. Microprocessor address, data, and control buses constitute examples of such nodes. In general, these nodes need to be viewed simultaneously. Very few of today's test instruments have this capability. Therefore, instruments such as logic analyzers, which can handle this problem, are becoming more important. The rapid expansion of digital systems and VLSI technology will increase the need for powerful analysis tools, and will also help to make the same analysis tools more powerful and more affordable. VLSI circuitry is becoming more reliable, but systems using this circuitry are becoming more complex, and need to be evaluated carefully, via logic analysis techniques, before they can be said to be correct.

This thesis explains the detailed design of a unique single-chip logic state analyzer. As a single integrated circuit, this logic analyzer could be included easily as part of a larger digital system. Current logic analyzers are large and costly devices that are often attached to a
digital system after a fault has made itself evident. The ability to continuously monitor digital system performance should, by itself, make this an attractive device. In the future, such analyzers may constitute on-chip diagnostic tools for ULSI (Ultra-Large Scale Integration) systems containing the equivalent of several hundred of today’s computer systems.

The Hewlett-Packard corporation has developed a similar circuit in response to market demand for lower-cost, higher-performance logic analyzers. It is not, however; intended to be included as an isolated component used for general logic analysis or system performance analysis purpose in an independent digital system. The device under consideration in this thesis is well-suited for microprocessor applications, although it is not as powerful as a complete, stand-alone system because of the cost-performance and power limits imposed by current VLSI technology. As the technology improves, so will the single-chip logic analyzer.

1.4 Logic Analyzer Microarchitecture

Fig.1-1 shows a functional block diagram of a general purpose logic analyzer. This section discusses the microarchitecture of this system. We begin this section by examining the blocks shown in Fig.1-1.
Fig. 1-1

Block Diagram of a Logic Analyzer

user's system → sampling circuit → storage system

qual trigger

→ trigger system

→ compute engine

→ display screen
1.4.1 Sampling Circuit

There are two kinds of digital data sampling or acquisition: Synchronous data acquisition and asynchronous data acquisition.

A. Synchronous Data Acquisition

The logic analyzer user might choose to use a clock that is external to the logic analyzer and synchronous to the system being monitored. The process of using such a clock in acquiring data is referred to as "synchronous data acquisition." The clock being used to sample the data is part of the system under test and tells that system when data is valid.

B. Asynchronous Data Acquisition

Many logic analyzers use a clock generated internal to the logic analyzer to acquire input data. This is called "asynchronous data acquisition." Whether all or part of the system under test is synchronous, the clock internal to the logic analyzer serves to tell the logic analyzer when to acquire data. There is no guarantee that the right data will be acquired without error unless the internal clock is much faster than the external clock.
1.4.2 Triggering Circuit

Fig.1-2 shows the block diagram of the logic analyzer triggering circuit. In a storage oscilloscope, a trigger event starts the sweep and, hence, the acquisition and storage of data. In a logic analyzer, a trigger event stops the data acquisition process. A trace event, conversely, starts the data acquisition process. If the user wants to acquire data after the trigger, the logic analyzer can be programmed to stop acquiring data after a certain number of clock cycles have occurred, rather than immediately, via controller timers.

Several methods of triggering a logic analyzer exist. Those will be discussed in the following paragraphs:

A. Word Recognition

A word recognizer is a circuit which produces a trigger when the combination of high and low signals at its inputs matches a specified set of conditions. A word recognizer is functionally equivalent to a large AND gate. When all inputs are true, a trigger signal is generated.

Fig.1-3(a) shows a synchronous word recognizer which performs recognition only on a system clock edge.

Fig.1-3(b) shows an asynchronous recognizer which trigger on events that are not synchronized to a system clock. In this case, no explicit clock is used.
Fig. 1-3(a) Synchronous Word Recognition
Fig. 1-3 (b) Asynchronous Word Recognition
B. Word Qualifier

Fig.1-3(c) shows a word qualifier circuit which functions as a word recognizer with additional word-qualifier inputs. It is common to word recognize and acquire data from the same lines on the system under test. However, there are applications where one or two additional lines are needed for word recognition. These signals need not be acquired and displayed. These lines are referred to as word-qualifier inputs. This implementation functions to keep irrelevant data from being displayed and saves memory space in the logic analyzer.

C. Sequential Triggering

One necessary requirement for a logic analyzer is the ability to trigger the system following the occurrence of a particular sequence of words on a bus in a digital system. This process is called tracing. Trace tells the logic analyzer when it should start storing the user data. Sequential triggering allows the user to qualify data storage more accurately than with a single trigger.

Sequential triggering is especially useful for observing the information flow occurring in a program subroutine starting from a specific point in the execution of the program. This makes it possible to store only a limited portion of the complete program in the logic analyzer memory. Fig.1-4 shows an example of this
Fig. 1-3 (c) Word Recognition With a Qualifier Input
Fig. 1-4  An Example of Sequential Triggering
situation. The numbers in this figure indicates program addresses.

To execute a program trace along the highlighted path as shown in Fig.1-4, the analyzer needs to detect 2849, 284A, 284C and 284E in sequence at its address inputs before it starts the trace.

1.4.3 Storage System

The logic analyzer storage system shown in Fig.1-5 is used to save appropriate incoming and retrieve data for display. It stores system control parameters as well as the data collected by the analyzer. Fig.1-5 shows one approach to this circuit. The "pipeline" block shown in this figure functions as a buffer for the internal memory. In some logic analyzers, the user can choose to stop storing data before or after trigger event. The "postfill control" block is used as an index counter to aid this approach. These two blocks are not included in this design.

1.4.4 Computing Engine

This part of circuit controls the writing and reading of data to and from memory respectively. It also handles the task of sending data to the display screen.

1.4.5 Display

The display section presents data from the logic analyzer to the user. Often a CRT display is used to
Fig.1-5  Block Diagram of the Storage System
physically present the data, but any display technology may be used.

The information displayed always includes at least a timing diagram or a state diagram. Timing diagram are graphical displays that show a clock signal for a timing reference, and the state of each input channel as a function of time. Waveforms are displayed with the channels spread vertically and time displayed horizontally. A state diagram or state table shows the binary value of each individual channel. Information can be displayed in octal, binary, decimal, or hexadecimal format. Fig.1-6(a) shows a timing diagram and Fig.1-6(b) shows a state diagram for a hexadecimal display.

Data may also be displayed in the form of microprocessor instruction mnemonics. Specialized hardware, firmware, or software customizes the logic analyzer for the particular instruction set and hardware characteristics of the specific microprocessor under analysis.

1.5 The State of the Art in Logic Analyzers

Specialized logic analyzers are available for many different applications. Among these are general purpose, serial, signature, and microprocessor analyzers. More than one of these specialized analysis tools may be found in the same physical instrument.
Fig.1-6(a) A Timing Diagram

Fig.1-6(b) A State Diagram for a Hexidecimal Display
All general purpose logic analyzers can be connected to various points in a digital system and can acquire data with or without an external clock. The type of displayed included will vary, but will always include a timing diagram and/or state table.

For some applications, a serial analyzer is necessary to perform high speed serial-to-parallel conversion on a serial input data stream. Appropriate data manipulation is then performed on the parallel data to yield a meaningful display and flexible triggering.

A signature analyzer displays a 4-character signature that is formed by the compression of the data acquired at a single point on a board or chip. The signature is not interpreted, but rather compared with a known good signature to determine whether the system is operating properly at that particular point on the board.

Another type of logic analyzer often encountered is a microprocessor analyzer. This type of analyzer is ideal for use in a microprocessor-based system. It typically has 24 or more input channels which are used to acquire data on the data bus, address bus, and control lines of a microprocessor. Displays will vary, but often include instruction mnemonics which are disassembled machine code. To display these mnemonics, a specialized personality card or module is required for each microprocessor to which the analyzer is connected.
1.6 An Example of a Single-Chip Logic Analyzer

1.6.1 Brief Description of Logic Analyzer Operation

As described above, there exist several types of logic analyzers. Presented in this thesis is a single-chip logic state analyzer that aids program debugging. It is a part of a complete logic analyzer system. This design uses sequential triggering. The design is required to examine the input signals for comparison purposes. To use the device, the user sends a trace address to the device, which indicates the point in the program at which to start the logic analyzer, and a trigger address, telling the logic analyzer when to stop storing data. Conditional addresses may also be provided if more than one possible path of code execution will lead to a suspected trouble spot. This allows the analyzer to determine if the program is following the desired path of execution by examining the given conditional addresses sequentially. The user can specify a debugging path by supplying the analyzer with information about the total number of conditional addresses, and the specific conditional, tracing, and triggering addresses prior to execution of a program and the beginning of the analysis process. Conditional, tracing, and triggering addresses are applied via the con_address nodes shown in Fig.1-7.

The architectural block diagram of the single-chip logic analyzer is shown in Fig.1-7. In this design,
fig. 2-7

Logics Analyser Architectural Block Diagram

- UCLK
- VCC
- VSS
- ready
- active
- data
- address
- con-address
- number
- data-out
- address-out
- clock
- run$
- iv
- controller
- module
- memory
- module

8)

3)

-11

(16)

(8)

(16)

3)

(8)

(8)

(16)
handshaking is used for data input and output operations. Handshaking is a means for insuring that the sender and the receiver of data are both ready to exchange data. It avoids communication problems between the logic analyzer and the external system. The operational flow of the logic analysis process is as follows:

1. The "clear" signal shown in Fig.1-7 resets the chip.

2. Handshaking is initialized between the chip and the external system to allow information transfer.
   a. The chip waits for the external "request" signal, indicating a readiness on the part of the external device to provide data to the chip.
   b. The chip acknowledges the "request" signal with the "ready" signal.
   c. The external system sends data and the "up_active" signal to indicate that data is ready.
   d. The chip acknowledges receipt of the data with the "active" signal.
   e. Step 2 is repeated until there is no more data to be transferred.

3. The external "run" signal initiates the data acquisition process.

4. The chip begins comparing and storing data.

5. Upon external demand, the chip will begin
handshaking operations for retrieving data from storage.

a. The chip sends out the "ready" signal to tell the external system to begin the retrieval of data stored in the chip.

b. The chip waits for the "request" signal, indicating a readiness on the part of the external system to receive data from the chip.

c. The chip sends data to the external system from storage, plus the "active" signal is asserted to indicate that data is ready.

d. The chip waits for the "up_active" signal, indicating that data transmission is complete.

e. Step 5 is repeated until there is no more data to be transmitted.

6. Once the data has been sent, a new "clear" signal will restart the analysis process.

1.6.2 An Example

A hypothetical problem will serve to illustrate the use of this logic analyzer. Suppose that a program, such as the one diagrammed in Fig.1-8, exists. It exhibits a fatal flaw; the location of this flaw has been confined to the flow of execution which is highlighted in Fig.1-8.

Referring to Fig.1-8, the tracing and the triggering addresses are 0027(trace=start) and 0179(trigger=stop)
Fig. 1-8  A Program Path
respectively, and the conditional address is 0257. These addresses are chosen for the critical path in this example.

**Operations:**

1. The chip is reset with the "clear" signal.
2. The chip waits for the "request" signal.
3. The chip acknowledges with the "ready" signal.
4. The external system sends a number indicating the number of conditional address in this process. In this case the number is 1. The external system also provides the con_address of 0027(tracing address) and the "up_active" signal.
5. The chip sends the "active" signal.
6. The chip waits for another "request" signal.
7. The chip acknowledges with the "ready" signal.
8. The external system sends the con_address of 0179(triggering address) and the "up_active" signal.
9. The chip sends the "active" signal.
10. The chip waits for the "request" signal.
11. The chip acknowledges with the "ready" signal.
12. The external system sends the con_address of 0257(conditional address) and the "up_active" signal.
13. The chip sends the "active" signal.
14. The chip waits for the "run" signal.
15. The chip activates trace features. See TABLE 1.
16. The chip sends the "ready" signal.
17. The chip waits for the "request" signal.
18. The chip sends the following data: address_out = 0027, data_out = 39, and the "active" signal.
19. The chip waits for the "up_active" signal.
20. The chip repeats steps 15 through 20 until there is no more data to be sent out.
21. The chip turns off the "ready" signal to stop the whole chip’s operation.
22. The chip waits for a new "clear" signal to restart the process.
### TABLE 1

<table>
<thead>
<tr>
<th>User's Data</th>
<th>Address</th>
<th>Address</th>
<th>User's Address</th>
<th>Memory Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>0023</td>
<td>0257</td>
<td>0</td>
<td>Data 0</td>
</tr>
<tr>
<td>2e</td>
<td>0057</td>
<td>0257</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2f</td>
<td>*0257(conditional)</td>
<td>0257</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>0300 address</td>
<td>*0027</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>e5</td>
<td>0001</td>
<td>0027</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>*0027(tracing)</td>
<td>0027</td>
<td>0</td>
<td>39 0027</td>
</tr>
<tr>
<td>35</td>
<td>0048 address</td>
<td>*0179</td>
<td>1</td>
<td>35 0048</td>
</tr>
<tr>
<td>ee</td>
<td>0265</td>
<td>0179</td>
<td>2</td>
<td>ee 0265</td>
</tr>
<tr>
<td>9f</td>
<td>*0179(triggering address)</td>
<td>0179</td>
<td>3</td>
<td>9f 0179</td>
</tr>
</tbody>
</table>

The internal process terminates.
Table 1 shows a step-by-step illustration of the internal and external states of the logic analyzer at each step in the sample program trace. The first two columns show the state of the system's memory external to the analyzer; the remaining columns show the state of the memory internal to the analyzer.

Column 1, User's Data, shows the contents of memory at the address specified in column two. This data has no effect in initiating the trace or performing any triggering function. Three critical addresses are marked with asterisks in column 2. These are, in order from top to bottom, the conditional address, the tracing address, and the triggering address. The third column shows the state of the internal address comparison register. This register holds the next address for which the analyzer is waiting; the analyzer also stores the type of this address: conditional, trace, or trigger. When one of these addresses is reached, it is replaced with its successor, and the state of the analyzer changes. The address types are replaced in the following order: conditional is replaced by tracing, and tracing is replaced by triggering. If no conditional address is given, the tracing address is loaded first, and the analyzer behaves accordingly.

The fourth column, Memory Address, shows the internal memory pointer, which dictates where the next set of data will be stored in internal memory once the analyzer has reached the trace state. The fifth and sixth columns show
the contents of internal memories at the place pointed to by Memory Address. It may be observed from Table 1 that the data acquisition (in memory) starts from the tracing address, which in this example is 0027.
2. Logic Analyzer Chip Architecture

2.1 System Specification

As a result of the logic analyzer architectural strategy outlined in the previous chapter, a chip was designed to meet the following (extended) specifications:

1. The chip is contained in an 84-pin square package.
2. On-chip memory size is 8 words.
3. Eleven distinct groups of input signals exist for the chip:
   a. UCLK, the external system clock.
   b. Clear, a signal used to reset the chip.
   c. Rise, a signal which indicates when an input to the analyzer is valid.
   d. Request, a signal which initiates the data transfer sequence.
   e. Run, a signal which initiates data collection.
   f. Up_Active, a signal which indicates that external parameter data is valid.
   g. Data, an eight-bit data transfer bus that is connected to the external system’s data bus.
   h. Number, a three-bit integer that indicates the number of conditional addresses to be considered.
   i. Address, a sixteen-bit address bus connected to
the external system’s address bus.

j. Con_Address, a sixteen-bit address bus used to input tracing, triggering and conditional addresses sequentially.

4. Maximum clock rate = 16.69 MHz.

5. Maximum external system clock rate = 4.17 MHz.

6. Ts = Setup Time = 16.7 ns from UCLK. Th = Hold Time = 0 -> 0.1 ns. The setup time is time during which data must be stable prior to the falling edge of the clock. The hold time is the amount of time that data should be valid after the falling edge of the clock.

7. The input format of the user’s address and data is shown in Fig.2-1(a) and Fig.2-1(b).

8. The design assumes synchronous data acquisition.

9. Handshaking is used for input and output operations.

10. Vss is equal to 0 volt and Vdd (power) is equal to 5 volt.

2.1.1 Handshaking Approach

In this chip, handshaking is used for data input and data output operations. The handshaking used in this design is the four-cycle or four-phase approach which requires four steps to complete a data transfer operation between the chip and the external system. The following is a description of the handshaking strategy:
**Fig. 2-1(a) The Input Format Specification**

**Fig. 2-1(b) Another Possible Input Format Specification**
Handshaking for Data Input:
1. Initially, the external system generates a "request" signal to request permission to send data to the chip.
2. The chip will generate a "ready" signal in response to this request.
3. After receiving the "ready" signal from the chip, the external system sends data to the chip while holding the "up_active" signal high.
4. After acquiring data, the chip responds with an "active" signal.

Handshaking for Data output:
1. The chip first generates a "ready" signal to tell the external system that data is available to be transferred to the system.
2. If the external system is capable of receiving this data, it then emits a "request" signal.
3. The chip will not transfer data to the external system until it receives a "request" signal. After receiving this signal, the chip transfers data while holding the "active" signal high.
4. The outside system will respond with the "up_active" signal after receiving data.
2.1.2 The Main Controller Algorithm

The algorithm for the control system responsible for controlling the operation of this chip follows below. One controller is implemented on the chip using a Finite State Machine (FSM). Fig.2-2 shows the flowchart of the main algorithm.

1. If the "clear" signal is on, then go to state 2; else, go to state 1.
2. Initialization:
   a. The internal data memory address pointer is initialized to 0.
   b. The trace and triggering addresses are fetched sequentially.
   c. The "number" which specifies the number of conditional addresses is fetched. If "number" is equal to 0, then the comparison address is also the trace address. Go to state 6. Else, go to state 3.
3. One conditional address is fetched and "number" is decremented by one. If "number" is not equal to 0, then go to state 3, else, go to state 4 and the first comparison address (the first conditional address) is loaded into the comparison register. At the same time, "number" is returned to its original value.
reset

Initialization

run

NO

number=0

YES

comp. addr.
= cond. addr.

match?

YES

reduce " number "

NO

comp. addr.
= tracing addr.

match?

NO

start storing data

copm. addr.
= triggering address

YES

match ?

NO

stop storing data

YES

output data

stop

Fig. 2-2 A Brief Flowchart of the Main Algorithm
4. If the "run" signal is asserted then go to state 5, else go to state 4.

5. If the address on the host system's address bus is different from the comparison address, go to state 5, else if "number" is equal to zero, put the trace address into the comparison register and go to state 7; else if "number" does not equal 0, put the next conditional address into the comparison register, decrement "number" by one and go to state 5.

6. If the "run" signal is active then go to state 7, else go to state 6.

7. If the host system's current address is different from the trace address, then go to state 7, else put the trigger address into the comparison register, enable the memory-write signal, and go to state 8.

8. If the host system's current address is different from the trigger address then enable the memory-write signal and go to state 8, else disable the memory-write signal and go to state 9.

9. If the internal memory pointer has exceeded memory limits and reset itself, it points to the first data word to output, else the pointer is reset to 0. At the same time, the "ready" signal is set to notify the external system that data is ready to be retrieved. Go to state 10.
10. The chip transmits all data to the external system. After each dataset has been transmitted, the internal address pointer is incremented by one. After transmitting all of the data, the "ready" signal is turned off. Go to state 11.

11. The "ready" signal is set to a logical low value. If the "clear" signal is on, go to state 1, else go to state 11.

2.2 Microarchitecture

The GENESIL Silicon Compiler provides a hierarchical design environment. Using GENESIL, a particular system can be transformed into a microarchitecture consisting of GENESIL modular building blocks, each of which will perform a specific function. At first, the chip is specified by defining the physical package characteristics and a general module which contains all the internal processing functions. The next element defined is the chip's main module. The main module contains three sub-modules: the hardware sub-module, memory sub-module, and the controller sub-module. These modules are implemented as specific instances of parameterized blocks contained in the compiler's block library. Fig.2-3 shows the ship hierarchy and Fig.2-4 shows the main module microarchitecture.
The Chip Hierarchy

Fig. 2-3

- chip
- pads
- mod (module)
- hardware
- controller
- memory

14 blocks
3 blocks
2 blocks
Fig. 2-4  The Main Module Microarchitecture
2.3 An Implementation of the Chip Microarchitecture

In this section, we describe the hardware, memory and controller modules.

2.3.1 The Hardware Module

This is the main operational module. It contains 14 blocks to implement all required logic analyzer functions. The connections between blocks in the main module are shown in Fig.2-5. Below is an explanation of each block's function.

1. Num is a parallel datapath. It tells the FSM in the Controller Module how many conditional addresses this process needs. It also functions as a decrementing counter to establish which conditional address are available.

2. Regwr (register-read&write) is a parallel datapath. It includes a shifter to control the read/write operation of the register file (for the conditional addresses only, not the triggering and the tracing addresses).

3. Pointer is a parallel datapath. It sends the memory address to the memory module for memory read/write operations.

4. Rand1 and rand2 are random logic blocks (D-type FFs) and are connected sequentially. Rand1 fetches the input address and also sends its output to
Fig. 2-5 Hardware Module Interconnections
5. Rand3 is a random logic (latch-SA), and its input is from rand2. Its output is connected to datawr and mem-addr (in the Memory Module).

6. Condata is a parallel datapath also known as a register file. It saves the conditional, trace and trigger addresses. Its output, comp, connected to datawr is the address available for comparison purpose.

7. Datawr is a parallel datapath used to signal the equality of the two comparands. Its output, eq2, will tell FSM when to change the comparison address.

8. Datal and data2 are random logic blocks (D-type FFs). These two blocks are connected sequentially. Datal fetches the input data and the output of data2 is connected to mem-data.

9. Pr is a parallel datapath (also an equality flag). Its inputs are from datal and data2. Its output, eq, helps FSM to solve the timing problems of memory writing.

10. Overwr is a parallel datapath. The content of the ROM or constant element contained in this datapath sets the size of mem-addr and mem-data in the Memory Module. Its output is connected to overwrcon (in the Controller Module) to indicate whether overwriting has occurred or not.
11. Memwrt is a random logic block. It controls memory writing operations.

12. Conaddr is a random logic block (latch-SA). This block is responsible for fetching and saving con-address, and saving this data to condata.

2.3.2 The Memory Module

The memory module is used to store data acquired from the external system for the user. This module includes two RAM (Random Access Memory) blocks: mem-addr and mem-data. mem-addr is 16-bits wide and mem-data is 8-bits wide. These two RAMs have the same address range and their address lines are connected together. Fig.2-6 shows the block diagram of this module.

2.3.3 The Controller Module

There are three Programmable Logic Arrays (PLAs) under this module. They are FSM, clkchoose and overwrcon. Fig.2-7 is the block diagram of the controller module.

Module FSM is the main controller, supervising the operation of this chip. It starts working when the clear signal is applied and stops working when the process is complete.

As mentioned before, if the input format matches the format shown in Fig.2-1(b) the rise signal must be on. clkchoose was created to deal with this problem. It makes
Fig. 2-6  Block Diagram of the Memory Module
Fig. 2-7  Block Diagram of the Controller Module
sure that the inputs are always available at the right phases.

Overwrcon deals with memory overwriting. With its help, the FSM can decide the correct address for retrieving and storing data.

2.3.4 Chip Netlisting

After the hardware, the controller and the memory modules were implemented in GENESIL, they were simulated individually (We will discuss simulation in Chapter 4).

Following the design procedure, all three modules were connected together via the netlisting process. Netlisting is the process that is done at the module and chip level to specify the interconnections of blocks or modules. Following netlisting, the main module was simulated to verify the correctness of this module. Once the functionality of the main module was verified and found to be correct, it was time to define connections for the inputs, outputs, clock and power supply. Again the connections and the main module were netlisted together, then simulated to verify correctness at the chip level.

2.3.5 Chip Floorplanning

When the functionality of the chip was verified and found to be correct, floorplanning was performed. Floorplanning is the process of orienting or placing design objects such as FSM, datapaths, etc. on the chip, the
specification of their fusion order, and the connections of the chip pins to the package wiring points. Floorplanning is done bottom up from module level to chip level.

Once the floorplanning of the chip is complete, the GENESIL description of the chip is converted into a complete model of the chip. This important process is called chip compilation. Following a successful chip compilation, the chip is presumed to be complete and functional. This is referred to in the literature as "correctness by construction".
3. Silicon Compilation

A silicon compiler is a design automation system which allows the designer to specify the design of an integrated circuit at a high-level of abstraction. The compiler automatically generates a description of the objects to be fabricated on silicon based on the user’s input specifications. These specifications may be behavioral or structural in nature, depending on the specific compiler.

3.1 An Overview of the GENESIL Silicon Compiler

The GENESIL Silicon Design System is a silicon compiler capable of transforming a structural specification of an integrated circuit into a database from which an IC can be produced. GENESIL offers the system designer the ability to create custom VLSI circuits from specification to tapeout. Fig.3-1 shows a block diagram of GENESIL Silicon Development System.

3.2 Design Methodology

In order to manage the complexity associated with the design of a single-chip logic state analyzer, the entire VLSI chip was synthesized using a top-down design approach, coupled with a bottom-up implementation. The chip was
Fig. 3-1  Genesil Silicon Development System
decomposed hierarchically, with each element of the hierarchy being synthesized, simulated, and incorporated into an appropriate higher level structure. This process was continued until the top-most level of the design, the chip level, was completed.

The design methodology includes the following activities:

A. Exploratory Design

A strength of the GENESIL chip design methodology is the ability to do "exploratory design", that is, to quickly perform a design and evaluate how well a "first guess" design meets the design specifications. It provides rapid feedback about the size, power, performance and functionality of the chip. This process allows the designer to identify strengths and weakness in the architecture, and allows rapid modification of the architecture to achieve an optimal chip design.

B. Detailed Logic Design

The next step in the design process involves completing the detailed specification of each block and module in the chip. This is accomplished by specifying these parameters using forms provided in the GENESIL system. These forms are interpreted by the compiler and translated into functional blocks in silicon.
C. Verification Phases

As the design progresses, the entire design must be checked for proper functionality, performance, power, and area.

The GENESIL system includes functional or logic simulation and timing analysis capabilities to assure that the design functional and performance requirements are met. Functional simulation verifies the logic; timing analysis verifies performance. Other tools contained in the compiler provide information about power and chip area.

D. Netlisting

Netlisting is the process in which a designer specifies how the blocks are interconnected. This is done by defining how the signals or nodes on a particular block are connected to nodes on other blocks.

After netlisting is completed at a particular level in the design hierarchy, it is possible to simulate and perform timing analysis on the interconnected entity as a whole rather than on the individual blocks in isolation.

E. Physical Design

Assuming that the results of the simulation and timing analysis phases of the design are successful, it is necessary to establish where each block will be located on
the chip and how it is to be oriented; this is known as floorplanning.

F. Manufacturing Phases

The final design step, tapeout, produces a text file containing fabrication information which may be loaded on a magnetic tape that is sent to a silicon foundry. At the foundry the chip will be fabricated. The tooling tape may be formatted in industry standard CIF or GDSII tape formats for photomask or customized for in-house fabrication processes. GENESIL also provides a text specification language which allows a compact representation of the entire design to be transferred quickly to a storage or transmission device.

Fig.3-2 illustrates the flow of operations when using GENESIL.
Fig. 3-2  The Silicon Compilation Process
4. Functional Simulation

The goal of functional or logic simulation is to verify that both the design as implemented and the layout generated by that design work as intended. The GENESIL simulator simulates a design with functional models in order to verify the design.

Functional simulation or modeling applies patterns of bits created by the designer to the inputs of each functional block at a particular level of the design hierarchy. The outputs of the block is observed to see if proper behavior is obtained given these input conditions. Functional simulation is useful in verifying both sequential and combinational logic.

4.1 Simulation Environment

Once a block, module, or chip is defined via the GENESIL specification forms, it can be simulated. If more than one block is to be simulated, then the netlist or interconnection scheme must also have been defined. Simulation is first performed with functional models. These models are independent of technology and layout because they do not depend on the actual delay time in a circuit, but only on the circuit’s behavior and changes in input signals. Only the definition of the blocks and the netlist are required for functional simulation. After the user
has verified the design’s functionality, then the user completes the floorplan and compiles the layout. From the layout, switch-level simulation models are compiled by GENESIL. A final simulation is performed with the switch-level model to verify that the behavior associated with the chip elements matches that of the actual layout extracted from the functional model. The block diagram shown in Fig.4-1 illustrates the simulator environment.

The ability to efficiently create, edit, and run a simulation file is important. Typically, thousands of test vectors or patterns of 1’s and 0’s are used to verify the functionality of a chip design against expected results. Given the efficient way in which designs are created using a silicon compiler, a designer typically will spend more than 80% of the overall design cycle performing simulation or simulation related activities.

As a result, specialized tools for creating and modifying test vectors are necessary. Test vectors in GENESIL are written in the microassembler (MASM) language or in a specialized high-level language. The simulator processes the test vectors and verifies the simulated results against the expected results provided by the user.

4.2 MASM Source Code

The results of running a MASM source code file during a simulation can be viewed on the CRT display screen or
Fig. 4-1  Simulation Environment
captured in a text file. Detailed source code files and trace output files of the chip level simulation for the logic analyzer chip are available for reference upon request.
5. Timing Analysis

Timing analysis is used to establish the timing performance of the chip under consideration. The timing analyzer contained in the GENESIL system provides timing information relative to an internal 2-phase non-overlapping clock generated on-chip. Timing information regarding asynchronous inputs may also be obtained. The timing analyzer extracts timing information directly from the physical layout, creates a non-linear RC timing model, and solves the resulting timing equations to produce a timing report. This timing information is highly technology dependent. As a result, the timing analyzer is calibrated to each fabrication process utilized.

5.1 Timing Analysis Environment

The timing analyzer reports the following key timing relationships:

1. The maximum speed at which the object under analysis will run. The timing analyzer may be run on any level of the design hierarchy.
2. Critical paths in the chip that limit the clock frequency.
3. Duty-cycle (phase high time) constraints.
4. Input setup and hold times.
5. Output delays.
6. Setup and hold times and signal delays for any internal nodes.
7. Path delays between internal nodes.

All data and control signals are characterized in terms of timing as valid, stable, or propagate with respect to a particular clock phase pair, eliminating other explicit timing dependencies on other data and control signals. Valid signals are available for one phase of the clock cycle, stable signals are available for both phases of the clock cycle, and propagate signals are independent of the clock.

Because of the two-phase methodology and the static timing analysis algorithms, timing analysis and simulation are run as independent processes. These algorithms run several orders of magnitude faster than SPICE, allowing the designer to rapidly perform "what-if" design iterations.

Timing models for each block are created by GENESIL and combined with parasitic timing parameters extracted from the chip layout to produce timing reports. Fig.5-1 illustrates the timing analysis environment.

5.2 Timing Reports

The clock report provides information about the maximum frequency and duty cycle limitations of the circuit
Fig. 5-1  Timing Analysis Environment
under analysis. In this design, the clock report presents the following information:

1. Minimum Phase High Time:

   The phase-high time is less than or equal to the accumulated delay of the critical path, which is the path on the chip which has the longest delay time. If the phase-high time is less than the accumulated delay, the difference is due to the clock delay of the latch at the end of the critical path.

   A. Phase 1 High: This parameter constitutes the minimum acceptable Phase_A high time, and is 25.5 ns for this chip.

   B. Phase 2 High: This parameter constitutes the minimum acceptable Phase_B high time, and is 24.1 ns for this chip.

2. Cycle

   A. Cycle (from Phase 1): Reports the longest delay through a sequence of Phase_A and Phase_B latches and is 59.9 ns for this chip.

   B. Cycle (from Phase 2): Reports the longest delay through a sequence of Phase_A and Phase_B latches and is 41.0 ns for this chip.
3. Minimum Cycle Time:

This parameter is the minimum total clock cycle time, which is derived from the two phase_high time and the cycle time for each phase, independent of the duty cycle. The minimum cycle time for this chip is 59.9 ns.

4. Symmetric Cycle Time:

This parameter is either the minimum cycle time, or twice the longest phase time minus the clock delay calculation, whichever is longer. The symmetric cycle time for this chip is 59.9 ns.

The symmetric cycle time gives an indication of the speed of this chip, assuming a symmetrical clock. A symmetric cycle time of 59.9 ns for this chip indicates a maximum speed of 16.69 MHz.
6. Conclusions

6.1 Extensions and Critiques

There are some functions not implemented in this design, such as glitch detection, and alternative triggering possibilities.

Since glitches in a system create problems, it is often necessary to include glitch detection. However, it costs more in terms of chip area to include the circuitry required for this purpose. A more comprehensive design should include this feature.

In chapter 1, many triggering methods were discussed. In this design, only one form was included: sequential triggering. Future designs having more than one triggering method should be considered.

This logic analyzer stops storing data immediately after triggering. It doesn’t allow the user to save data at some time after or before triggering. In the future, the design can be expanded to have the capability to stop storing data after or before triggering.

6.2 Conclusions

In this paper, a single-chip logic state analyzer was designed and discussed in detail. An 84-pin package was selected for this design. This chip can be either a part of
a complete logic analyzer system, or it can act as an Application-Specific Integrated Circuit (ASIC) for program debugging, performance analysis, or in general purpose digital system monitoring and diagnostic applications. This chip provides sufficient capabilities to fulfill the original design goals.
BIBLIOGRAPHY


