To the Graduate Council:

I am submitting herewith a thesis written by Christopher L. Spearman entitled "Design and Development of a Digital Time-Based Error Analyzer Circuit for Implementation in a Mixed-Signal Integrated Circuit." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Donald W. Bouldin

Don Bouldin, Major Professor

We have read this thesis
and recommend its acceptance:

Robert E. Bodenheimer

T. Vaughn Blalock

Accepted for the Council:

Carolyn Minkel

Associate Vice Chancellor
and Dean of The Graduate School

Design and Development of a Digital Time-Based Error Analyzer Circuit for Implementation in a Mixed-Signal Integrated Circuit

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Christopher L. Spearman

December 1995

Abstract

The phenomenon of "ghosting" in terrestrial broadcast television is one of the most significant sources of picture distortion in television receivers. As part of a cost reduced consumer-based Ghost Cancellation product developed at Philips Consumer Electronics Company, a mixed-mode integrated circuit (IC) was co-developed with Mitsubishi Electronics. The IC includes a digital circuit to detect channel changes and VCR playback sources on a NTSC Composite Video Baseband Signal using Time-Based Error Analysis techniques. Two separate designs for the Time-Based Error Analyzer were prototyped in Xilinx Field Programmable Gate Arrays (FPGA's) using two different design methodologies. The first methodology used a combination of low-level schematics and VHDL Hardware Description Language, and the second methodology used the Synopsys VSS Simulator, FPGA Compiler, Design Compiler, and Test Compiler tools. The results of both designs and design methodologies, as well as the final implementation in silicon, are discussed in detail.

Table of Contents

List of Figures

Chapter 1

Introduction

Engineering is a problem solving profession, and one of the most valuable tools for any engineer to possess is the ability to approach problems systematically. Most problems faced by engineers do not have definitive answers, but most problems do have solutions. A systematic approach to solving problems consistently yields good solutions to even the most difficult engineering problems. A systematic approach to problem solving consists of:

1. Defining the problem
2. Gathering information
3. Development of a solution to the problem
4. Testing the solution
5. Formatting and documenting formally the final solution

The problem definition is the most important step. A valid problem definition is to break the problem into several smaller, more manageable pieces. Solutions to each piece are developed, using the same systematic approach, to arrive at an overall solution to the problem. Gathering information consists of researching previous work, collecting and analyzing data, discussing the problem with peers, etc. Development of a solution to the problem is usually a direct result of the information gathering step. Testing the solution is the most difficult step because the designer, in many cases, is reluctant to accept that his/her solution is anything less than the optimal solution. Since the designer is most often the tester, a self-critical eye is a requirement for non-biased evaluation of the solution. Difficulty during the testing of a solution is avoided by eliminating ambiguity in the problem definition. Steps 2, 3, and 4 form an iterative process, which provide the mechanism for arriving at good solutions by eliminating bad and improving sufficient solutions. Invariably, the solution that results from steps 2 through 4 is in rough form. Many times a solution must be formatted to fit another environment especially if the solution was developed outside the framework of the intended environment. Reformatting of a solution is a dangerous process, and precaution must be taken to insure that the solution is not corrupted during reformatting. Also, documentation that was maintained during the solution development phase should contain as much

information on failures and dead ends as the final solution. Formal documentation of the final solution consist of writing a paper, submitting a patent disclosure, giving a presentation to peers, etc. By formally documenting the problem and the solution, other engineers can re-use, adapt, or build on the documented problem and solution.

The Design and Development of a Digital Time-Based Error Analyzer Circuit for Implementation in a Mixed-Signal Integrated Circuit is the formal documentation for a circuit that was co-developed by Philips Consumer Electronics Company and Mitsubishi Electronics, Inc. The "problem" is defined in Chapter 2 by describing the circuit requirements within the overall system framework. In Chapter 3, general and background information about the problem is presented. The solution development phase is presented in Chapters 4, 5, and 6. Chapter 4 discusses the environment and the method of implementing the solution. Chapter 5 demonstrates an entire iteration of steps 2, 3, and 4 of the systematic problem solving approach discussed above. The failure of the solution presented in Chapter 5 required another iteration of steps 2 through 4, and Chapter 6 documents a second solution to the problem, which was the final solution. Chapter 7 details the reformatting process applied to the final solution, and the method of insuring that solution integrity was maintained. Chapter 8 describes the results of the final implementation of the circuit. The strengths and weaknesses of the final solution/implementation, possible improvements, and the effectiveness of the systematic problem solving approach are discussed in Chapter 9.

The manner in which the information is presented reflects the systematic problem solving approach that was used during the design process. The information presented is not necessarily in chronological order. The level of detail presented is intended so that the reader can understand the circuit operation well enough to reproduce the circuit from the documentation alone. Hopefully, the information presented is general enough for the reader to build on in future designs.

Chapter 2

Overview of Ghost Cancellation

History

Investigation into Ghost Cancellation by Philips began as early as 1988 by the research staff at Philips Research Laboratories, Briarcliff Manor, NY. The Ghost Cancellation project was initially a three phase project, with the final objectives being development of a low cost filter chip and transfer of Ghost Cancellation technology for product development at Philips Consumer Electronics Company, Knoxville, TN. One of the most significant accomplishments by the Philips Research Laboratories effort was the adoption of the Philips Ghost Cancellation Reference (GCR) signal as the U.S. standard by the Advanced Television Systems Committee (ATSC). The second major accomplishment was the development of a professional Ghost Cancellation system called the "Vector", which was intended primarily for broadcasters and cable head-ends[6].

Development of a low-cost, consumer Ghost Cancellation system began at Philips Consumer Electronics Company, Knoxville, TN in 1993. The primary ingredient for a successful consumer Ghost Cancellation system was cost reduction. Ghost Cancellation devices were introduced into the Japanese market that retailed over $500, and the cost of these devices was considered to be the principal factor for the low market penetration. Therefore, the consumer target price for the Philips consumer Ghost Cancellation device was approximately $100.

Theory

The term "ghosts" is derived from the visualization of scaled, time-delayed, and phase-shifted echoes of the main signal on a video display. The most common cause of "ghosting" is multi-path distortion in terrestrial broadcast; however, the same phenomenon occurs in cable systems. The theory behind ghost cancellation is to insert a known reference signal into the video signal prior to transmission.

The receiving station uses the received reference signal and the ideal reference signal, which is stored in memory at the receiver, in an adaptive filter algorithm to produce coefficients for a programmable digital filter array. The digital filter is then used to eliminate "ghosts" from the incoming video. The U.S. Ghost Cancellation Reference signal is shown in Figure 2-1[4].

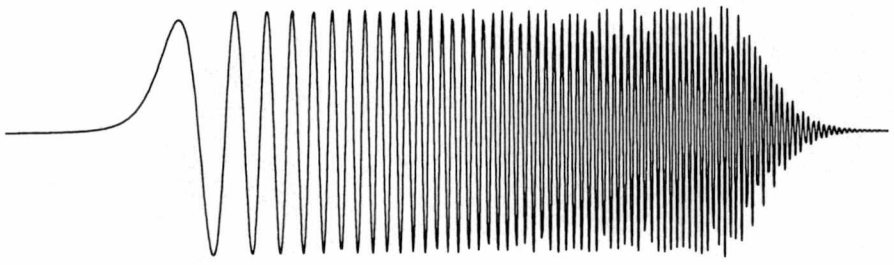

Figure 2-1: U.S. Ghost Cancellation Reference Signal

The characteristics of the U.S. GCR are[3]:

- High energy in short time period
- Immunity to data on adjacent Vertical Blanking Lines
- Flat spectrum-by design
- Linear group delay-by design
- Unlimited ghost delay range
- Multiple processing algorithms are possible

The GCR is transmitted in an eight field sequence of alternating polarity in order to cancel the video synchronization information, while enabling averaging of the GCR to increase the signal to noise ratio of the received GCR. An adaptation algorithm is then applied to the averaged GCR to calculate filter coefficients. The filter structure is a combination of a Finite Impulse Response (FIR) and an Infinite Impulse Response (IIR) filter. A FIR filter is sufficient to cancel short pre-ghosts and post-ghosts, and an IIR filter is required to cancel long post-ghosts and FIR residues. A great deal of research has been directed at adaptation algorithms for the filter structure. Most algorithms to date have been derivatives of the Least Mean Square (LMS) algorithm, which is a classical algorithm for channel equalization using adaptive FIR filters. Adaptation of the IIR filter coefficients is particularly difficult to achieve primarily

because of instability. Once new filter coefficients are calculated and put into the filter array, the entire process is repeated beginning with the eight field averaging step.

Ghost Cancellation System Architecture

The basic architecture of the consumer ghost canceler is shown in Figure 2-2. The Analog Pre-Processing functions shown are classical video processing functions such as anti-aliasing low-pass filtering, back porch clamping, horizontal and vertical sync separation, burst-locked clock generation, and analog-to-digital conversion. The quantized CVBS, horizontal and vertical synchronization pulses, and burst-locked clock signals are used in the Digital Signal Processing section to implement the ghost cancellation algorithm. The deghosted digital data is then passed to the Analog Post-Processing section for digital-to-analog conversion and output smoothing.

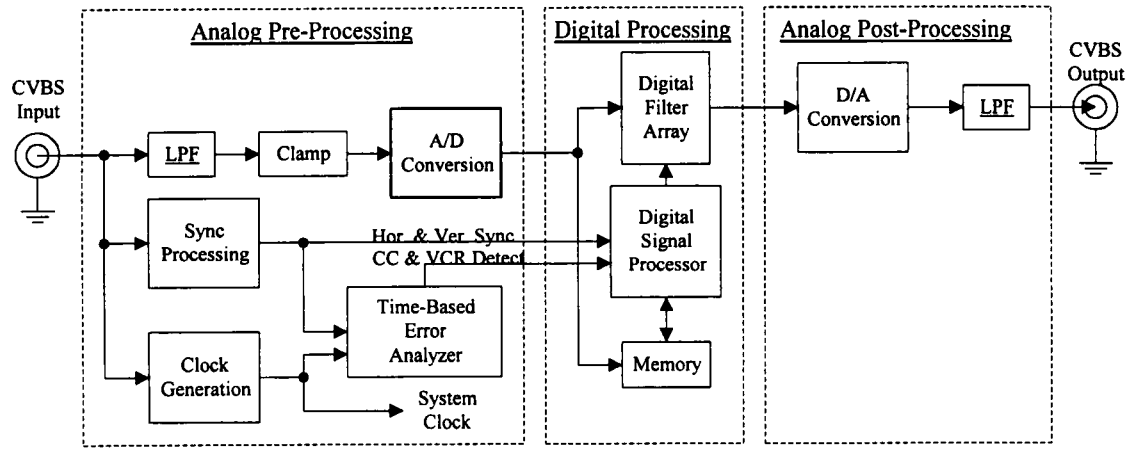

Figure 2-2: Ghost Cancellation System

The system architecture assumes that a Composite Video Baseband Signal (CVBS) is available. For professional applications, RF modulators/demodulators are readily available; however, for consumer applications, the only widely available CVBS source other than the television set is a Video Cassette Recorder (VCR). Therefore, the consumer Ghost Cancellation system was designed to operate on the

CVBS between a VCR and a television set. This poses a unique set of problems to the consumer based ghost canceler not faced by other professional or consumer video processing products.

The filter adaptation algorithm proposed has both a fast and a slow adaptation speed. Fast adaptation occurs immediately after a change of channel by the user, and converges on a solution in less than a second. Slow adaptation is a continuous process that converges to a solution over many seconds, and is intended to deal with slow changes in the ghosting conditions or a missed channel change. The problem is detecting channel changes because no other information other than the CVBS input is available. In television sets or VCR's, a microprocessor receives input from either the remote control or a push-button and signals the tuner to change channels. Since the ghost cancellation device operates independently of the VCR and television, access to these signals is not possible, and a means of detecting channel changes based on only the CVBS input is required.

The second problem faced by the consumer ghost cancellation device is VCR playback. With the ghost cancellation device operating between the VCR and the television, there is no way to deghost the video before it is recorded by the VCR. Since the ghost canceler is always operating between the VCR and television, the video can be deghosted during playback. However, VCR playback has a significantly different channel characteristic than standard video and must be treated differently in the adaptation algorithm. As with channel change detection, the ghost canceler must detect VCR playback conditions from only the CVBS input.

Since few products have been introduced into the consumer market that operate only on the CVBS, there has been only a limited need to detect a VCR playback source and virtually no need to detect channel changes from the CVBS. The few applications that have attempted to detect VCR playback used time-based error analysis techniques, which requires only separated horizontal and vertical sync and a burst reference signal. Since a burst-locked clock was being used for the digital signal processing section of the ghost canceler, the same clock can be used as the burst reference in a digital circuit that measures time-based errors on the horizontal and vertical drive signals. If a burst-locked reference is not available, time-based errors in the horizontal and vertical syncs are still measurable.

One item of the ghost canceler development schedule was to develop a circuit to detect channel changes and VCR playback sources using only the separated horizontal and vertical sync signals and either a freerunning or a burst-locked clock. Since the nature of horizontal sync, vertical sync, and the clock are time-based, and errors in these signals were the basis for detecting channel changes and VCR playback, the circuit is referred to as a Time-Based Error Analyzer. The circuit was to be a digital circuit that consisted of less than 3000 gates, and was to be implemented as a digital block inside the IC that provided all of the analog processing functions shown in Figure 2-2 except for the input and output low-pass filters and the analog-to-digital converter. The author was responsible for the design and development of the Time-Based Error Analyzer block, which is the primary topic of study for this thesis.

System Implementation

The majority of the ghost cancellation system of Figure 2-2 was implemented with three integrated circuits. Each of the IC's and their associated circuit functionality are shown in Table 2-1. The input and output low-pass filters were unfeasible to implement in any of the IC's; therefore, external anti-aliasing and smoothing filters were used. Agreements were reached for the Mixed-Signal IC to be developed by Mitsubishi Electronics of Japan, the Digital IC to be developed by Zoran of Israel, and the A/D Converter to be provided by Philips Semiconductors, Eindhoven, Netherlands.

Table 2-1: IC Functionality

Mixed-Signal IC	Digital IC	A/D Converter
Input Clamp Circuit	Digital Filter Array	A/D Conversion
A/D Voltage Reference Generator	Digital Signal Processor	
Analog Bypass Switch	Memory	
Horizontal and Vertical Sync Separation		
Channel Change & VCR Detection		
D/A Converter		
$4f_{SC}$ Burst-Locked Clock Generator		

Chapter 3

Background

NTSC Composite Video Waveform

The drawing shown in Figure 3-1 depicts the key features of the NTSC Composite Video waveform[7]. The horizontal synchronization pulse is defined in Detail YY of Figure 3-1. The horizontal sync pulse is processed in televisions to provide a control signal to the horizontal deflection circuits that scan an electron beam horizontally across a Cathode Ray Tube (CRT). When processing digital video, the extracted horizontal sync pulses are useful for providing time-based information such as pixel position or the number of pixels present in a particular line of video.

The color burst shown in Detail ZZ of Figure 3-1 is 8 cycles of a pure 3.579645 MHz sinusoid. Color information in a NTSC signal is phase modulated, and the color burst is used as a phase reference in color demodulating circuits. Burst locked clocks are commonly used when processing digital video because coherent relationships exist between the color burst, the horizontal sync pulse, and the vertical sync pulse. In quantized video, if the sample clock is phase locked to an even multiple of the color burst frequency, then all the samples in a field are aligned vertically. Figure 3-1 shows the four color field sequence for the Vertical Blanking Interval (VBI) of the NTSC waveform, which consists of lines 1 through 21 of the video field. The VBI lines are used for a variety of purposes, such as closed captioning (line 21), station identification (SID), Teletex, and also is the location for the GCR signal (line 19). Lines 1 through 9 comprise the Vertical Interval (VI), from which the vertical synchronization pulse interval is from lines 4 to 6. The vertical sync pulse is used much like the horizontal sync pulse in televisions because the electron beam is pulled from the bottom of the CRT to the top during the Vertical Interval. The Vertical Synchronization Pulse is also used for determining time-based information in the video signal. Table 3-1 summarizes the frequencies and frequency relationships for color burst, horizontal sync, and vertical sync.

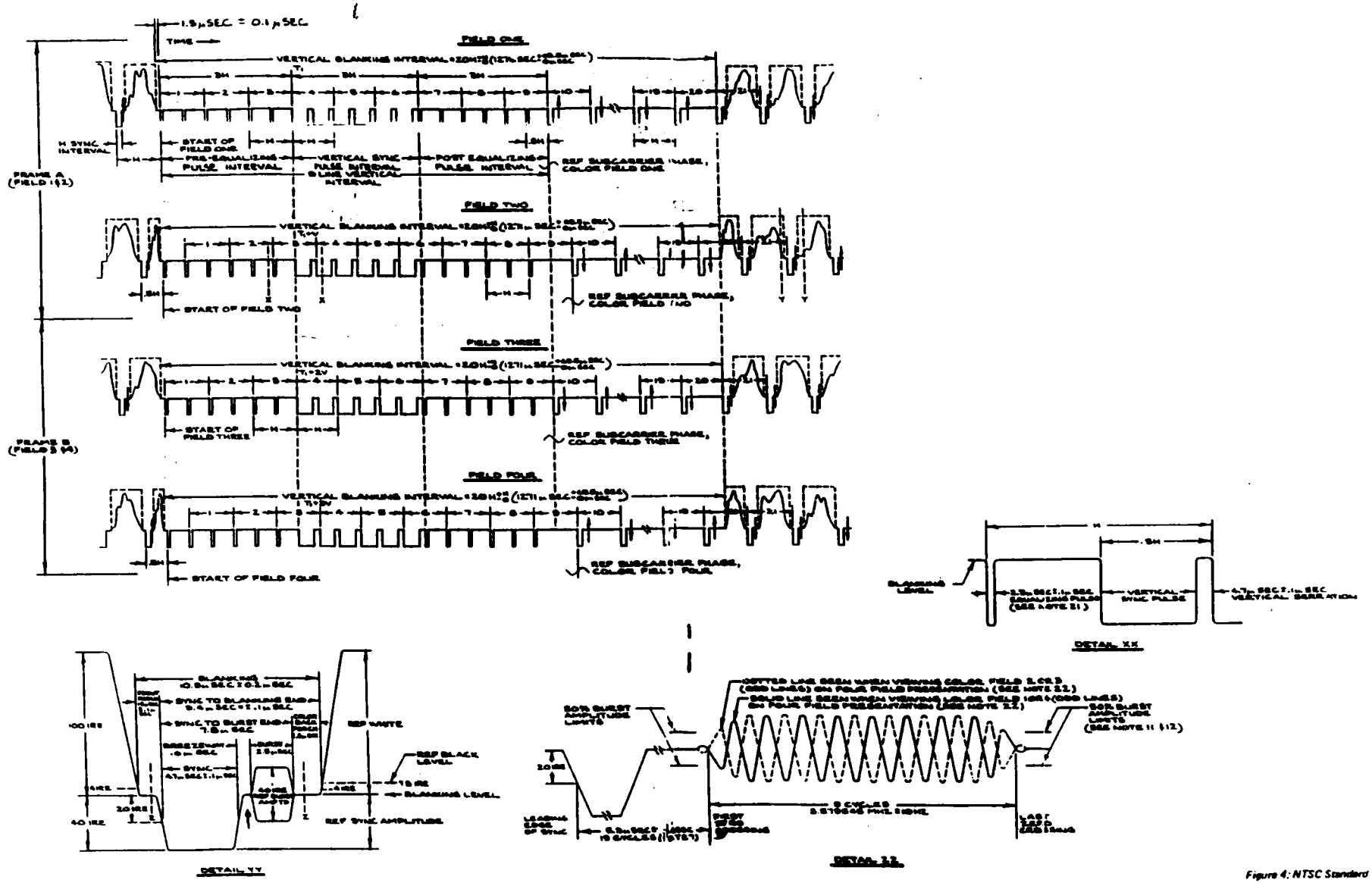

Figure 3-1: NTSC Composite Waveform

Figure 4: NTSC Standard

Table 3-1: Frequency Relationships between Synchronization Signals

Video Feature	Frequency	Relationship
Color Burst, f_{sc}	3.579545 MHz	$f_{sc} = \frac{455}{2} f_h$
Horizontal Sync, f_h	15.734264 kHz	$f_{sc} = \frac{455}{2} f_h$
Vertical Sync, f_v	59.94 Hz	$f_v = \frac{f_h}{262.5}$

Circuits for Synchronization Signal Extraction

Figure 3-2 is a block diagram of a typical horizontal and vertical sync separator circuit. The output of the lowpass filter is commonly referred to as composite sync, which is simply the composite video signal stripped of chrominance and luminance information. The slicer shown is usually implemented by a comparator. The output of the slicer is taken as the reference input to a Phase-Locked Loop (PLL). The purpose of the PLL is to maintain precise phase alignment of the generated horizontal drive pulse to the horizontal synchronization pulse of the input video. The PLL shown is comprised of a phase detector, a loop filter, a voltage-controlled oscillator (VCO), and a divide-by-32 counter. The vertical drive pulse is generated primarily by slicing the integrated composite sync; however, noise immunity is increased when a countdown circuit is used to "window" the predicted position of the vertical drive pulse. If the slice level is not triggered by the end of the "window", then a vertical drive pulse is blindly generated by the terminal count of the countdown circuit.

The system architecture described in Figure 2-2 requires the generation of a single synchronous system clock for the entire ghost cancellation system. In NTSC CVBS, the highest frequency component is at 4.2 MHz, and in order to satisfy the Nyquist Criterion, the sampling rate must be at least 8.4 MHz to prevent aliasing. Common sampling schemes for video use some multiple of the subcarrier frequency as the sampling rate since the horizontal and vertical rates are also derived from this frequency. Therefore, a $4f_{sc}$ sampling rate is sufficient to satisfy Nyquist criterion, while at the same time relaxes the constraints

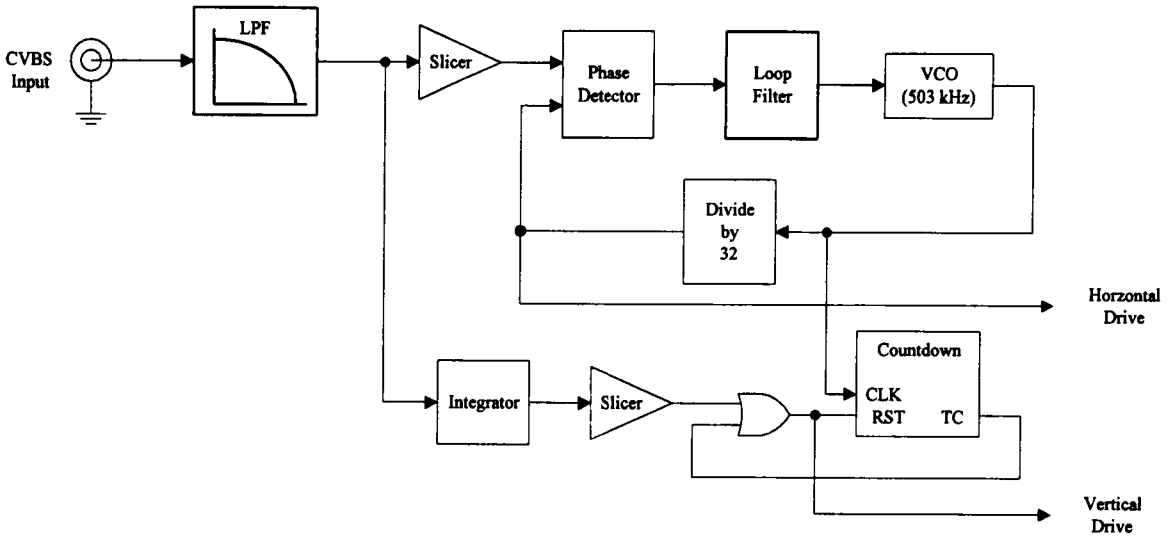

Figure 3-2: Horizontal and Vertical Synchronization Separation Circuit

on the anti-aliasing filter. Figure 3-3 is a block diagram of a $4f_{SC}$ clock generation circuit, which is also referred to as a burst-locked clock.

The first component of a $4f_{SC}$ clock generation circuit is a bandpass filter, which is centered about the color burst frequency. The bandpass filtered signal is then gated through a Voltage Controlled Amplifier (VCA) via a burst-gate pulse signal. The burst-gate pulse is generated by a fixed time delay from the horizontal sync edge, which creates a window on the color burst. The gated color burst is then used as an reference input to another PLL; however, the $4f_{SC}$ PLL is required to be much more accurate than the Horizontal PLL (typically around +/- 500 Hz), which requires the use of a Voltage Controlled Crystal Oscillator.

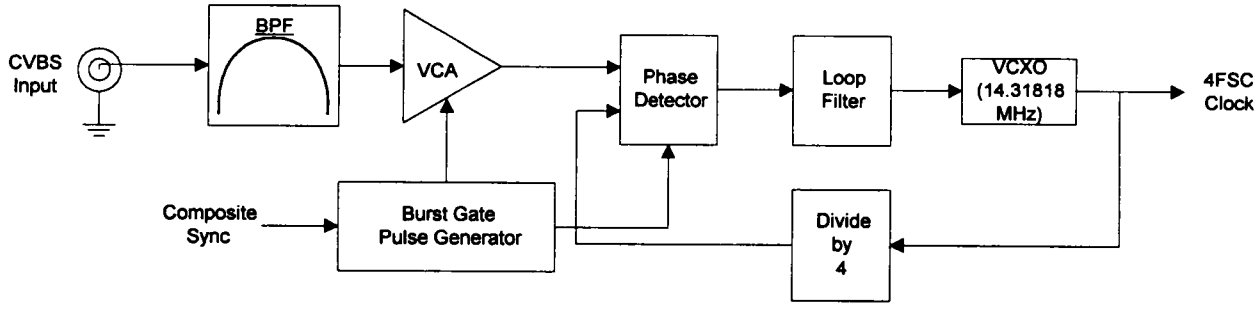

Figure 3-3: Burst-Locked Clock Generation

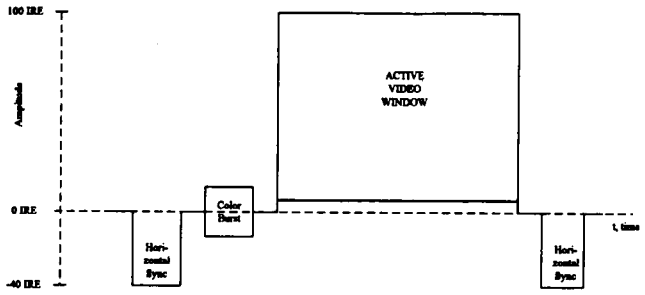

Figure 3-4: Abbreviated NTSC Waveform

The discussion of the NTSC waveform so far has been about the synchronization signals and the purpose they serve. However, none of the information presented so far describes the picture information carried in the NTSC waveform. The picture information in NTSC video signals is commonly referred to as luminance and chrominance and refers to the signal levels present during the "active video window", which is depicted in Figure 3-4. Luminance information is essentially the DC level present in the video at any particular point in the active video window. Monochrome television waveforms consist only of luminance video components. Chrominance information in the NTSC waveform is phase modulated, using the color burst as a phase reference, and amplitude modulated. Color hue is represented as phase deviations from the color burst reference phase, and color saturation depends on signal amplitude. The NTSC waveform consist of both luminance and chrominance components, which have overlapping frequency bands in order to compress all of the video information into a 4.2 MHz channel. Luminance occupies the frequency band from 0 to 4.2 MHz, and chrominance occupies the frequency band from 3.08 to 4.08 MHz[1].

Video Tape Recorders

The most common VCR format found in the world market today is the 1/2 inch, type H (VHS) helical scan video recorder, which was invented by the JVC company in 1959. Since VCR playback mode is of significant importance in the operation of the ghost canceler system, some of the basic concepts of VCR operation will be discussed. Figure 3-5 shows the configuration of a helical scan head and the positioning of the magnetic tape to the scan head, which is the method used to record and playback video

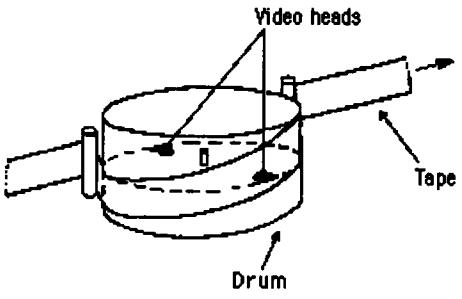

Figure 3-5: Two Head Helical VCR[5]

in the VHS video recorder. The drum rotates at a relatively high rate (1800 rpm's for a typical two head scanner) compared to the velocity of the magnetic tape across the scan head (approximately 1.3 in/sec). Video information is recorded on diagonal tracks of the tape as shown in Figure 3-6. Each track contains only the non-VBI lines of one field. The NTSC waveform is reconstructed in a two head system, as shown in Figure 3-6, by the VCR synthesizing the VBI, scanning a complete field of video information (track A) from a single 180° rotation of the scan head. A second VBI is synthesized before the track B is scanned by the second head during the next 180° rotation, and this process continues throughout the duration of VCR playback. The process of synchronizing the synthesized VBI to the recorded video is controlled mechanically by servos, which operate as a feedback mechanism to adjust the rotation speed of the scan head. The servos are referenced to the horizontal frequency, which enables the VCR to compensate for the effects of tape stretch. Tape stretch causes the head to be off track during the read of a track, and by adjusting the speed of the scan head, track alignment can be maintained.

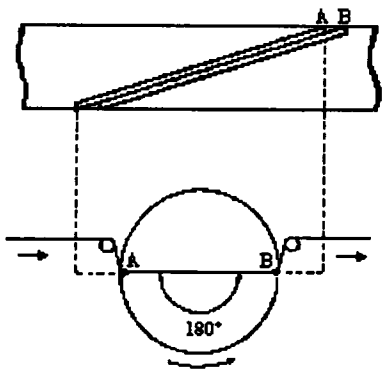

Figure 3-6: Two Head VCR Operation[5]

In the evolution of the VCR, one of the most persistent problems was time-based errors, which caused poor or faulty recording of chrominance signals. These time-based errors were the resultant of modulation of the video signal before being recorded to the magnetic tape. Modulation of the video signal is required because of the useable spectrum of magnetic tape, which has a notch at DC and sharp roll off at high frequencies. The modulation scheme used in the VHS format VCR is called the "color under" system, which is depicted in Figure 3-7. The chrominance component signals, which for the NTSC waveform reside in the frequency band centered at 3.58 MHz +/- 0.5 MHz, are heterodyned or down converted to a lower frequency band centered at 629 kHz +/- 0.5 MHz. The luminance component signals

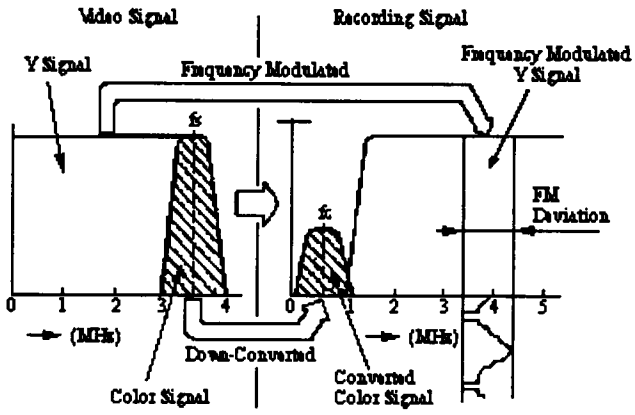

Figure 3-7: Color Under System[5]

are frequency modulated to the 3.4 to 4.4 MHz band, where 4.4 MHz corresponds to peak white, and 3.4 MHz to sync tip. The frequency modulated luminance and the down converted chrominance are mixed before being recorded to the magnetic tape. During playback, the same local oscillator that was used to down convert the chrominance signal is used to up convert the chrominance back to 3.58 MHz, which provides a color burst signal that is stable enough to lock a television PLL. The demodulated luminance is mixed with the up converted chrominance to reproduce an NTSC waveform. The advantage of the "color under" system is that a high quality color image is attainable without the need for an expensive time-based error corrector. The disadvantage is that the reproduced waveform does not comply with the NTSC standard because the horizontal sync frequency lies within the luminance band, and the horizontal sync is demodulated with luminance and not the local oscillator used to up/down convert the chrominance. Therefore, there is no phase relationship between the color burst and the horizontal sync pulse, which is required by the NTSC standard[2,5].

The VCR playback characteristics discussed have a significant effect on the ghost cancellation system performance. The most important characteristic being the effective channel bandwidth of the VCR playback signal. Since luminance and chrominance information have overlapping frequency space, a VCR playback signal should have luminance components from DC to 4.2 MHz. Typically this is not the case. A study was performed by Philips Consumer Electronics Company using VCR's from a variety of

manufacturers and a variety of video cassettes. The conclusion of this study was that the luminance bandwidth in VCR playback was flat from DC to approximately 2.0 MHz, with a -3dB point at approximately 2.5 MHz. The chrominance bandwidth was centered about 3.58 MHz, with about a 1 MHz passband. Since the ghost cancellation system equalizes the channel using a stored copy of the GCR, filter instability is a problem if a VCR playback signal is equalized using the full bandwidth GCR shown in Figure 2-1. The solution is to store a VCR-GCR model in the ghost cancellation system for use when VCR playback is detected. The Time-Based Error Analyzer circuit is used to provide an indication of when to use the VCR-GCR.

Effects of Ghosting on Standard NTSC Signals

Ghosting is caused by multi-path distortion in the RF domain. Sources of the multi-path distortion are reflections from tall buildings and mountains, mis-aligned receiving antennas, impedance mismatching, etc. The visible effect of ghosting is one or more time-delayed, scaled, and phase-shifted copies of the main signal in the video image. Figure 3-8 through Figure 3-11 are included to demonstrate the effects of ghosting on the NTSC composite waveform. The NTSC signals shown were generated with an NTSC waveform generator. The output of the waveform generator was fed into a ShibaSoku TG98AX ghost generator, which synthesizes the effects of ghosting on the baseband input. The ShibaSoku ghost generator has the ability to create eight separate ghosts, each of which the user can configure the strength (in dB), delay (in μs), and phase (in degrees). The Shiba-Soku ghost generator can create a multitude of ghosting conditions that closely resemble many real-world conditions, while affording the control over the ghosting conditions that is necessary in a laboratory environment. The output of the Shiba-Soku ghost generator was monitored with a Tektronics VM-700A, which was used to generate the plots of Figure 3-8 through Figure 3-11.

Figure 3-8 shows line 127 of a white circle on a black background NTSC pattern. The white circle on black background pattern is particularly useful in demonstrating the effects of ghosting on a NTSC signal. Since the white circle on black background pattern is a monochrome image, a single line of the waveform has two impulses where the white circle is present in the active video window. Elsewhere

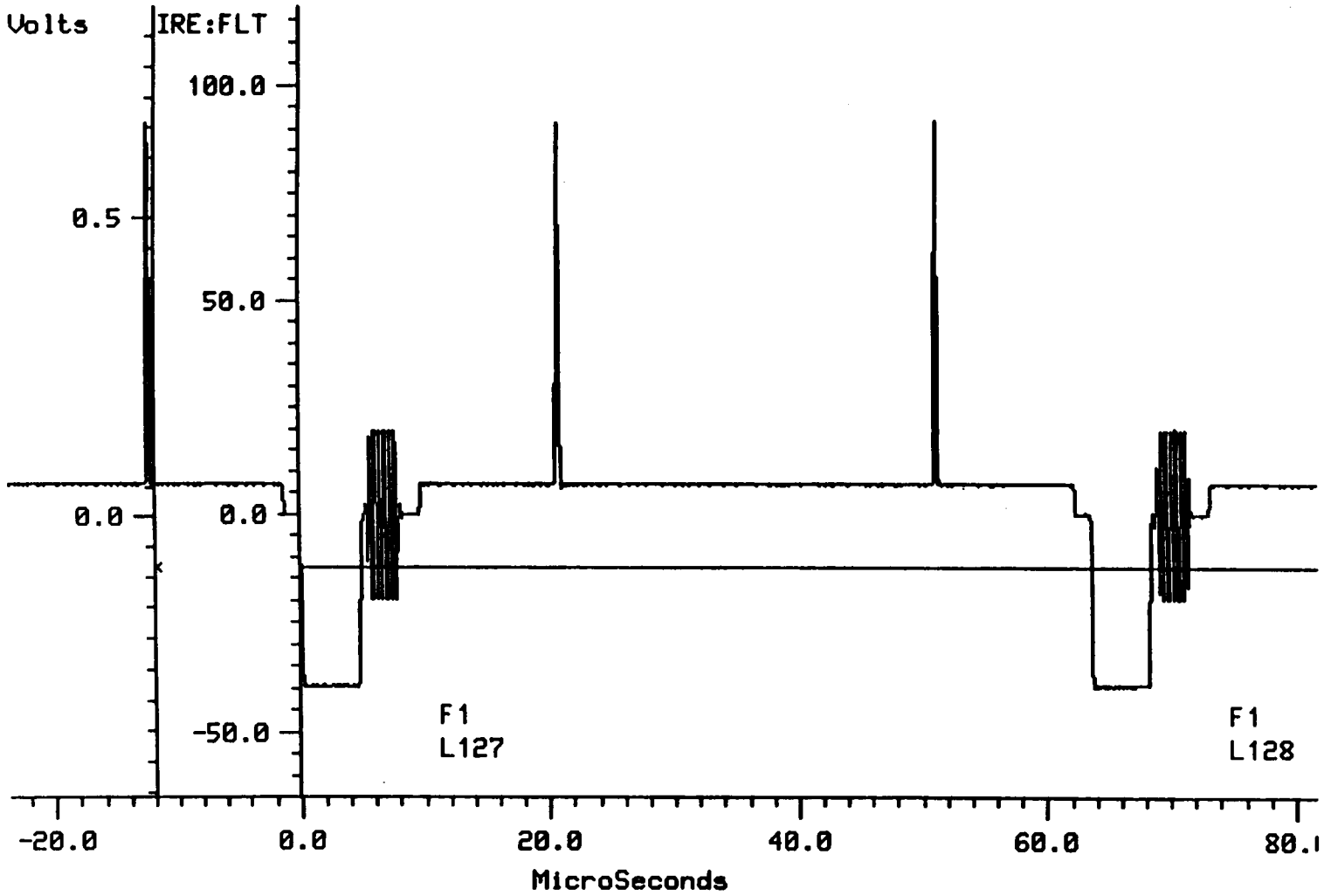

Figure 3-8: White Circle on Black Background Test Pattern

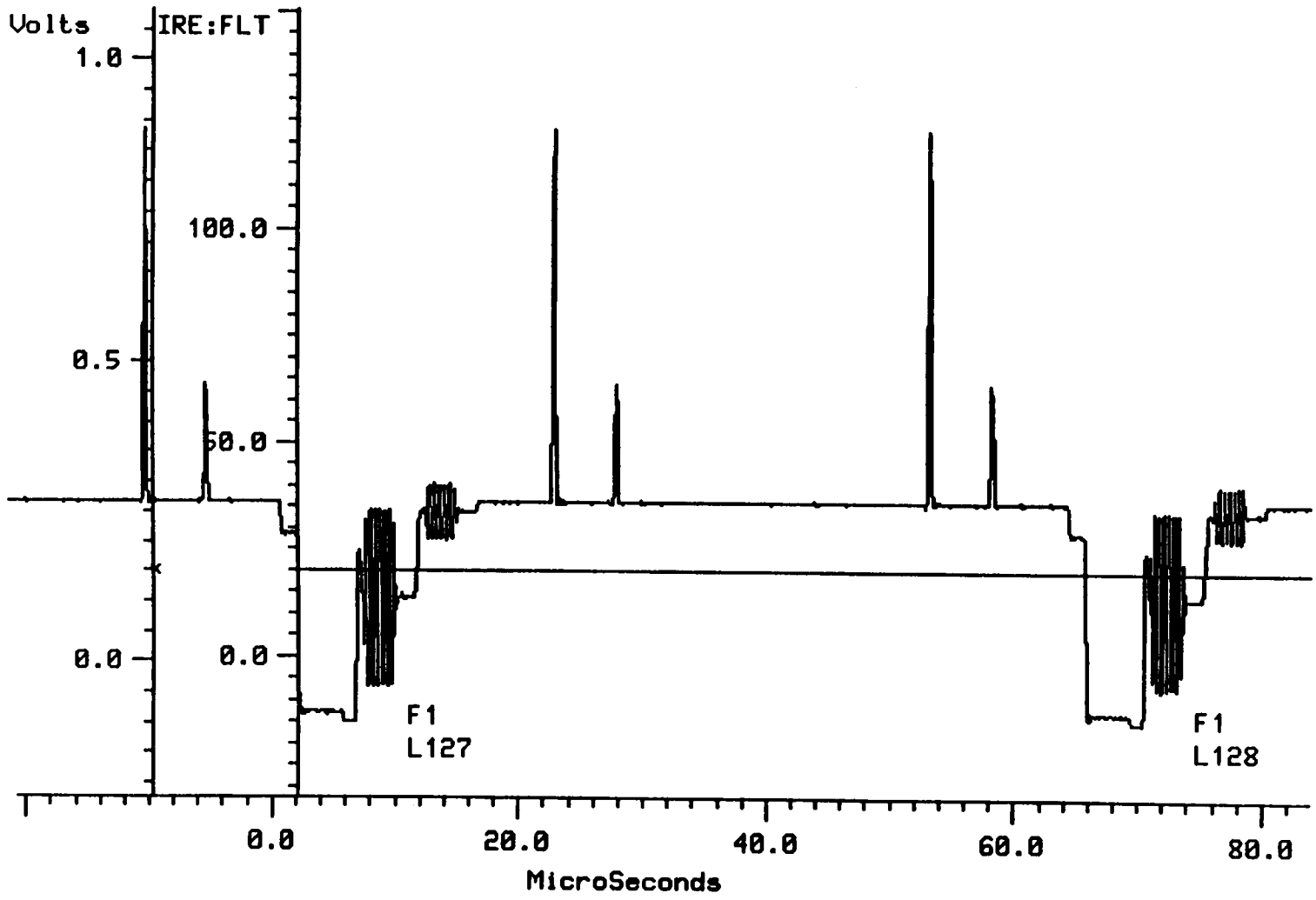

Figure 3-9: Ghosted White Circle on Black Background Test Pattern

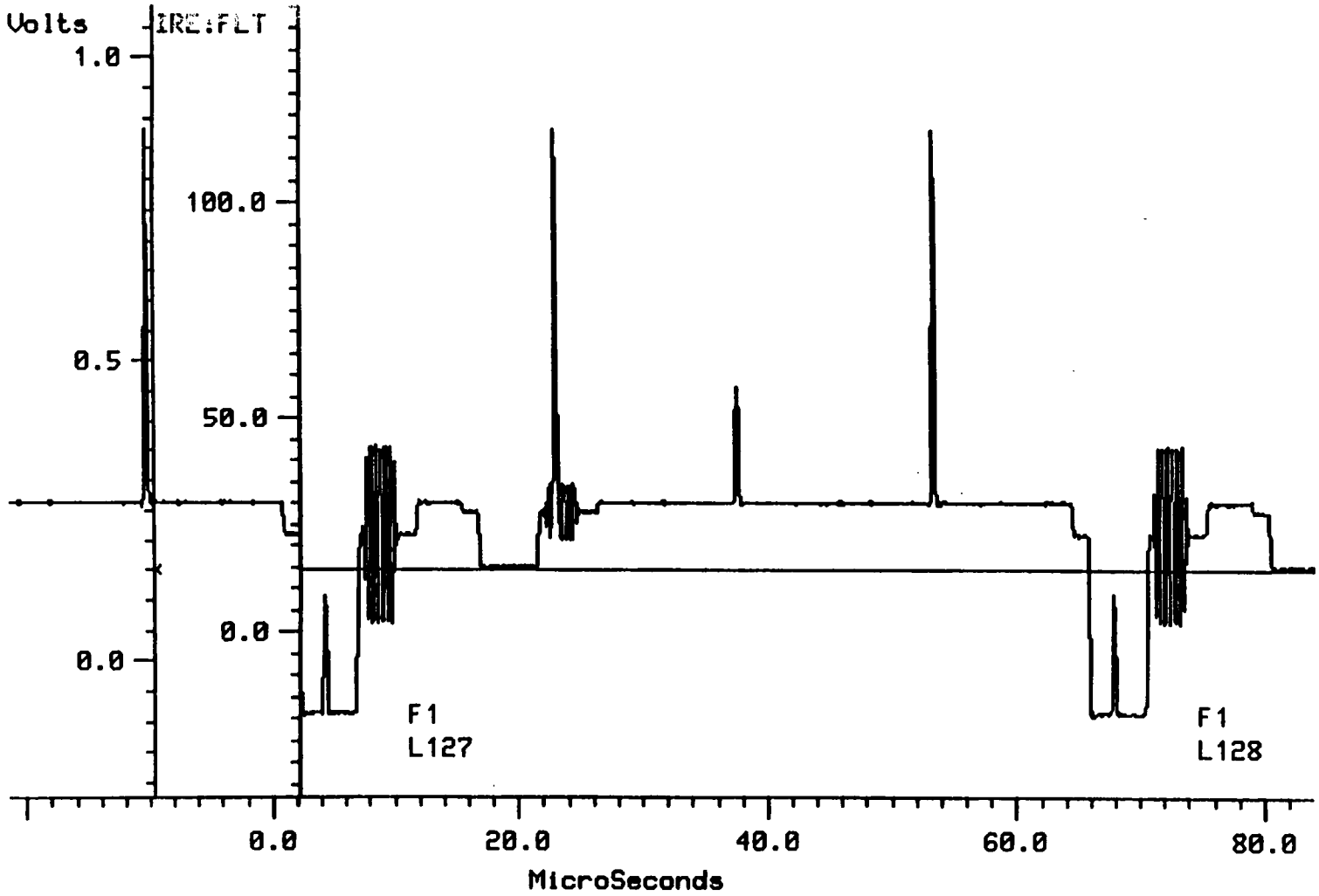

Figure 3-10: Effects of Ghosting on Horizontal Sync

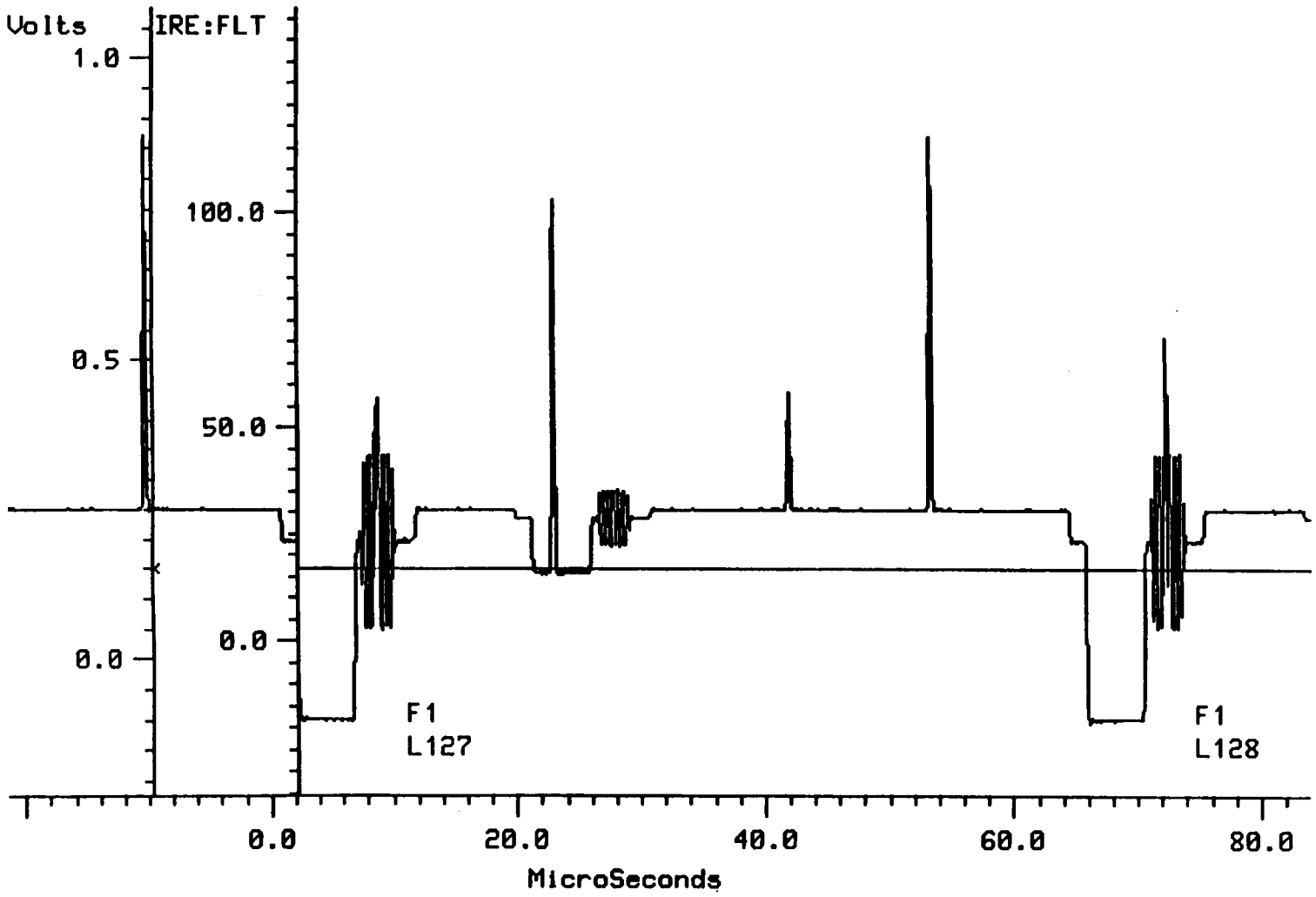

Figure 3-11: Effects of Ghosting on Color Burst

in the active video is the black level. Figure 3-9 shows the effect of a -10 dB, 5 *us*, 0 degree ghost on a white circle on black background NTSC pattern. The ghost is clearly seen in the active video as the time-delayed, scaled copy of the main signal. The problem of actually eliminating the ghost from the picture is the function of the Digital Processor block shown in Figure 2-2. The Mixed-Signal IC must perform the functions defined in Table 2-1 *before* deghosting of the signal is performed, which poses some unique problems for classical sync extraction circuits and burst locked clock generators.

The major problem that occurs is that the active video can be ghosted into the horizontal blanking interval. Figure 3-10 demonstrates the effects of the active video ghosting into the horizontal blanking interval, which causes horizontal sync pulse suppression. Since the slicer (Figure 3-2) is set at a fixed point, the ghosted video in the horizontal sync can cause improper slicing of the horizontal sync edge. To complicate this even more, the nature of the active video window is dynamic under normal viewing conditions; therefore, incorrect slicing of the horizontal sync combined with changing video content will cause the output frequency of the horizontal sync separation circuit of Figure 3-2 to vary. The generation of the vertical drive pulse is also effected by sync suppression caused by ghosting. If the horizontal sync pulses are suppressed by ghosting, then the slope of the integrated composite sync during the VBI is changed, which causes the signal input to the vertical sync slicer to reach the fixed slice level at the wrong time. The effect is that the vertical drive pulse, which is supposed to occur predictably within a window on a particular line, is delayed. The delayed vertical drive pulse makes determining the exact location of a particular line of video impossible.

Active video may also ghost into the color burst, which affects the operation of the burst locked clock extraction circuit of Figure 3-3. In Figure 3-11, the ghosted impulse is seen in the middle of the color burst. Since active video contains chrominance information, the chrominance of the active video will pass through the bandpass filter and mix with the color burst, which may cause phase shift in the color burst. Once again, the changing video content in the active video window will cause constant change in the phase of the color burst on a line-to-line basis, which will cause the output of the burst locked clock circuit of Figure 3-3 to jitter.

Figure 3-8 through Figure 3-11 show that ghosting of composite signals can cause severe time-based errors, which have to be dealt with in two ways. First, the sync separation circuit of Figure 3-2 and the burst-locked clock generation circuit of Figure 3-3 must be designed to withstand ghosting conditions. Second, circuits that use horizontal and vertical drive and burst-locked clocks must include some means of preventing ghosting conditions from effecting circuit operation. The Time-Based Error Analyzer circuit was developed with the understanding that time-based errors caused by ghosting had to be eliminated from normal VCR playback and channel change detection.

Chapter 4

Design and Development Environment

As previously stated, the Time-Based Error Analyzer circuit was to be implemented as a digital block inside the Mixed-Signal IC. In the agreement between Mitsubishi Electronics and Philips Consumer Electronics, the design and development of the Time-Based Error Analyzer circuit was the responsibility of Philips Consumer Electronics. The transfer of the design from Philips to Mitsubishi was to be in the form of a gate level netlist, and Philips was also responsible for providing both functional and manufacturing test vectors to Mitsubishi for testing of the IC.

Operations performed on video signals are extremely subjective by nature. Many times a straight forward function implemented on a video signal will appear to be correct from an algorithmic standpoint, but when viewed on a television monitor, faults or errors are immediately apparent. For this reason, prototypes of video processing systems are common. Also, simulation of video signals can take an extremely long time, and possibly reveal less than a prototyped system. The Time-Based Error Analyzer circuit was designed and developed on a prototype of the entire ghost cancellation system of Figure 2-2. The prototype system and the design and development environment will be described.

Prototyping Environment

The digital systems group at Philips Consumer Electronics Company, Knoxville, TN developed a Rapid Prototyping System in conjunction with Philips Research Labs, Briarcliff, NY. The purpose of this board was to provide a large number of gates for prototyping digital circuits. The Rapid Prototyping System consists of 23 Xilinx XC4005 Field Programmable Gate Arrays (FPGA) for reprogrammable prototyping, a single Xilinx XC4010 (FPGA) motherboard controller, a battery-backed SRAM for program storage, and a PC parallel cable interface for program download. Interconnect between XC4005 parts is accomplished by either shorting sockets or wirewrap. The shorting sockets also act as connection points

between the XC4005 parts and daughter boards that can be mounted directly to the prototyping system. The Rapid Prototyping System has been applied to a variety of projects at PCEC including Picture-in-Picture (PIP), QuAM modem, as well as Ghost Cancellation.

Each Xilinx XC4005 part consists of roughly 5000 programmable gates. To implement Figure 2-2 on the Rapid Prototyping System, a significant amount of partitioning was required. The Analog Pre-Processing and Post-Processing blocks were implemented on a single daughter board. The digital filter array was implemented as a separate daughter board. The memory was implemented with four separate daughter boards, each consisting of a 32kx16 SRAM. A Motorola ADS56002 DSP and DSP development environment was used as a microprocessor interface for implementing the top level functionality. The horizontal drive, the vertical drive, and the $4f_{sc}$ burst-locked clock inputs to the Time-Based Error Analyzer circuit were provided by the analog daughter board, which used a Mitsubishi sync processor that had been co-developed with Philips for a Picture-in-Picture product. Since the Time-Based Error Analyzer circuit was to be under 3000 gates, a single Xilinx XC4005 part was sufficient to prototype the entire circuit.

Design Entry Tools

When the development of the Time-Based Error Analyzer circuit began, most of the rest of the ghost cancellation system had been prototyped by a combination of Mentor Graphics 7.0 schematic entry and ABEL HDL. The ABEL code was used to implement state machine control over instantiated blocks in the Mentor 7.0 schematic. The ABEL compiler that was being used also compiled and synthesized VHDL. Since VHDL was an acceptable design form for Mitsubishi, the design and development environment for the Time-Based Error Analyzer circuit was initially a combination of VHDL and Mentor Graphics 7.0 schematics. The limitations of these tools soon became apparent, and the Synopsys VHDL simulation and synthesis tools were evaluated as design flow possibility. The Synopsys design flow proved to be far superior to the Mentor/ABEL design flow, but both design strategies will be discussed since a great deal of knowledge was gained from trying to make both methodologies successful.

The Mentor schematic capture environment at Philips was somewhat unique because a considerable development effort went into a "Universal Library". The Universal Library is simply a collection of digital functions ranging in complexity from two input AND gates to FIR filter blocks. What makes the Universal Library unique is that different target technologies could be mapped from a single Universal Library schematic. The primary target technology for the Universal Library was Xilinx, but assuming that a vendor library was setup for use with the Universal Library, the design could be prototyped in Xilinx, then retargeted to the vendor library. The ability to retarget technologies is the primary benefit in a HDL synthesis design flow, but for schematic capture based design flows, retargeting technologies is usually done by redrawing schematics by hand. However, the Universal Library has a major problem because new releases of Mentor Graphics require that all the Universal Library components be converted to a new library database format, which is a mammoth task.

Two designs were completed for the Time-Based Error Analyzer circuit. The first design was performed using a combination of Mentor Graphics 7.0 schematics and synthesized VHDL code. The first design did not use the $4f_{sc}$ burst-locked clock as the system clock, but a free running clock was used instead. Problems with the operation of the first Time-Based Error Analyzer design forced a complete redesign of the circuit. Since schedule became a predominant issue, the design cycle time for the second design had to be reduced, which was the reason for switching to the Synopsys toolset. Each of the designs are discussed in Chapters 5 and 6.

Design Flow

The design flows for Design I and Design II are shown in Figure 4-1 and Figure 4-2 respectively. In Design I, low-level functionality, such as n-bit registers, adders, multiplexers, etc., were implemented using Universal Library components. By using different expansion scripts, either a Xilinx or a Mitsubishi netlist was generated from the schematic. ABEL 5.0 was used to compile the VHDL code, and a LCA fitter was used to create the Xilinx netlist from the compiled VHDL code. The Xilinx XACT 5.1 place and route tools were used to combine the Mentor Graphics generated netlist with the ABEL generated netlist to

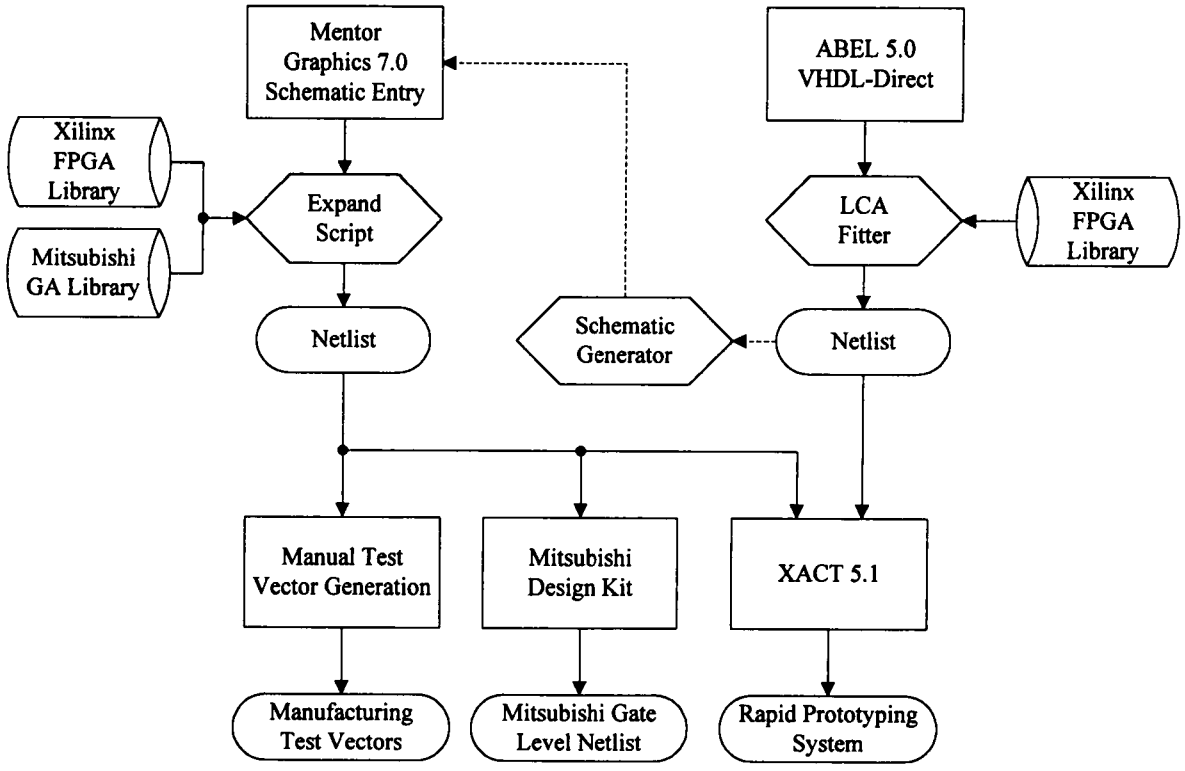

Figure 4-1: Design I Flow

produce a program file for download to the Rapid Prototyping System. In order to produce a gate level netlist in Mitsubishi gates, and also to simulate the design for test vector generation, the netlist generated by the ABEL 5.0 compiler and the LCA Fitter was converted to a Mentor Graphics schematic by using a schematic generator utility. However, the netlist was in Xilinx primitives, and the schematic had to be hand edited for conversion to Universal Library components. The hand editing process was extremely tedious and error-prone; therefore, the conversion was performed only after the Design I circuit operation was determined to be satisfactory. In order to verify the circuit operation after hand conversion, a program file for the Rapid Prototyping System was generated from the Mentor Graphics schematic alone. The Mitsubishi gate level netlist was generated from the Mentor Graphics schematic, after the hand conversion process, by using an expansion script and several utilities provided in the Mitsubishi Design Kit for Mentor Graphics 7.0. The manufacturing test vectors were generated by breaking the design into functional pieces

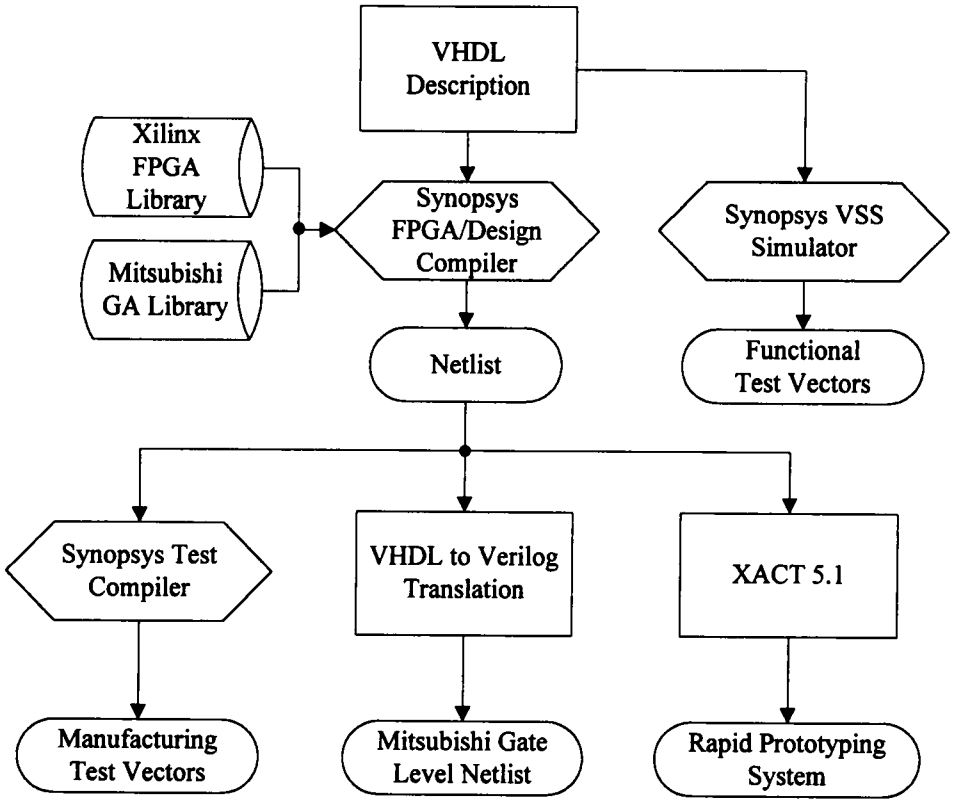

Figure 4-2: Design II Flow

and using Mentor Graphics' QuickSim to exercise each piece.

The Design II Flow, shown in Figure 4-2, began with a VHDL description of the circuit. The program file for the Rapid Prototyping System was generated using the Synopsys FPGA Compiler and Xilinx XACT 5.1 place and route tools. Once the circuit operation was determined to be satisfactory, the VHDL description was synthesized to a Mitsubishi gate level netlist in structural VHDL form. The structural VHDL was converted to a Verilog gate level netlist by Mitsubishi engineers. Manufacturing test vectors were generated from the gate level netlist using Synopsys Test Compiler. Functional test vectors were generated using the VHDL description of the circuit and a VHDL testbench stimulus file. The functional simulation was performed using the Synopsys VSS simulator.

Chapter 5

Design I: Time-Based Error Analyzer Using a Free-running Clock

The circuit shown in Figure 5-1 was built in order to examine the time-based error of the horizontal and vertical drive signals. The circuit consists of a pixel counter and a line counter, and the system clock was generated by a free running 14.31818 MHz oscillator. The pixel counter counts clock cycles per line and is reset on the rising edge of horizontal drive. The line counter counts rising edges of the horizontal signal and is reset on the falling edge of vertical drive. For standard NTSC composite signals, the expected pixels per line and lines per field is calculated from Table 3-1:

$$pixels\ per\ line \approx \frac{455}{2} \times 4 f_{sc} = 910$$
$$lines\ per\ field \approx 262.5$$

The output of the pixel and line counters were observed on Philips PM3585 Logic Analyzer. By triggering the logic analyzer on the horizontal sync edge, the variation in the horizontal frequency can be observed as variations from a 910 terminal count. Figure 5-2 demonstrates the variance in horizontal frequency for standard NTSC signals. In order to determine what information about channel changes and VCR playback is available from the horizontal frequency variations, a logic analyzer was used to capture data in the same manner data was taken for Figure 5-2. Figure 5-3 through Figure 5-5 demonstrate the horizontal frequency variation with the input signal a VCR playback, undergoing a change of channel, and having low signal-to-noise ratio, respectively. The horizontal frequency variations under the VCR playback conditions is attributable primarily to tape stretch and the VCR servos adjusting after the synthesized VBI. The characterization of the horizontal frequency variation for VCR playback is extremely complicated because at least two feedback loops are controlling the rate of variation. However, the exhibited phenomenon appeared to be consistent enough to detect VCR playback conditions. Channel changes were seen to produce a large perturbation in the horizontal frequency for many lines, which made measurement of the horizontal frequency variation a likely means of detecting channel changes. The effects of a low signal-to-noise ratio input signal was performed to see what real world conditions did to the horizontal frequency.

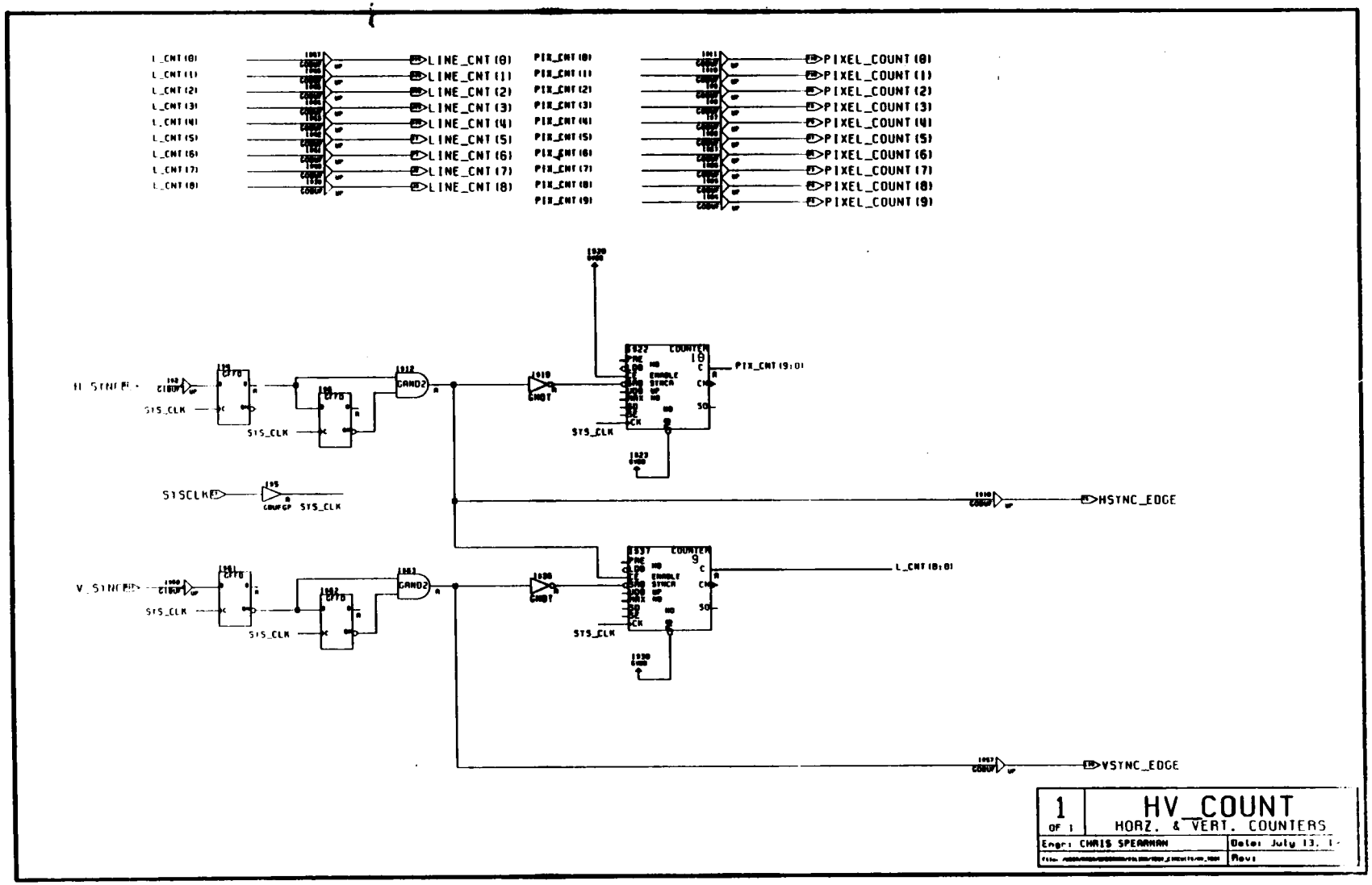

Figure 5-1: Test Circuit

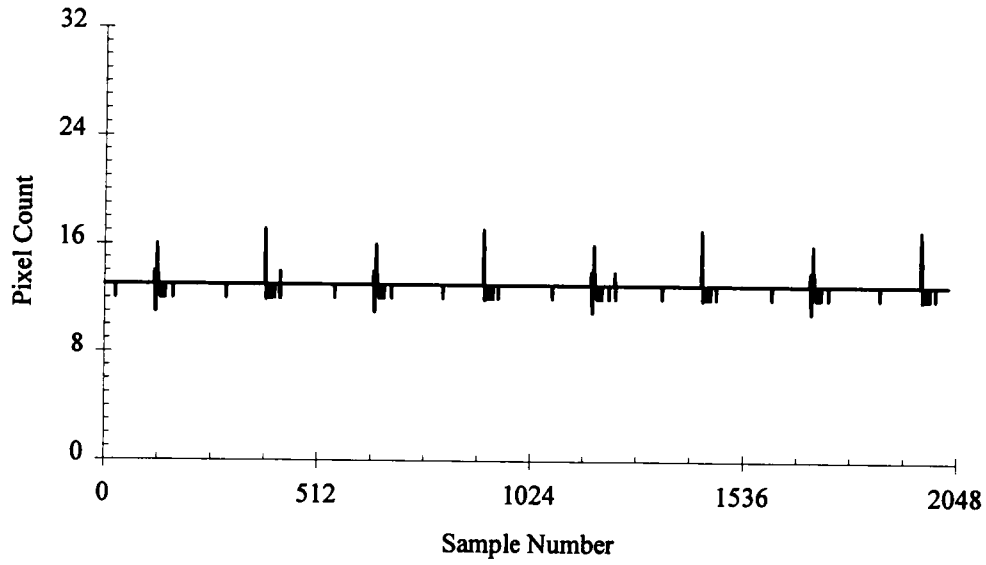

Figure 5-2: Standard Signal Horizontal Frequency Variation

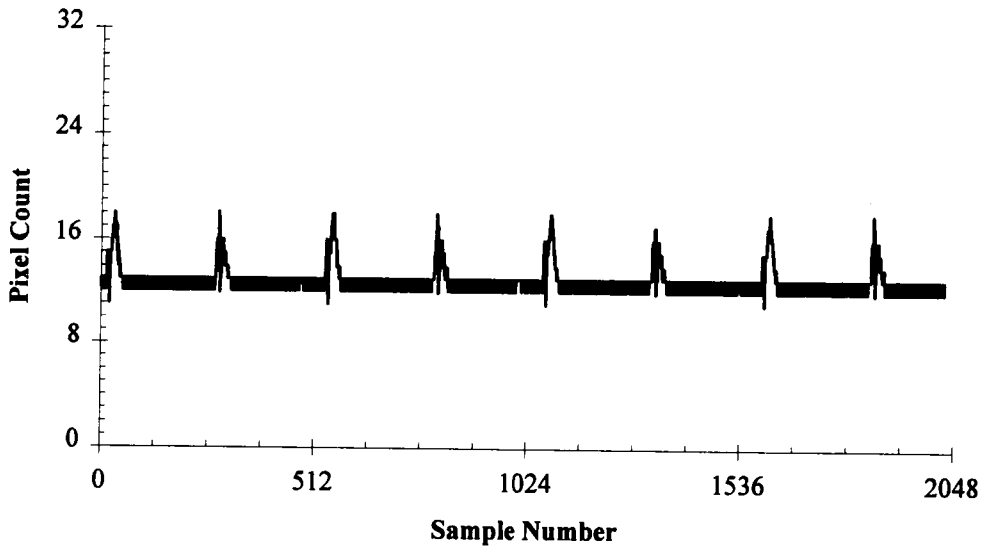

Figure 5-3: VCR Playback Horizontal Frequency Variation

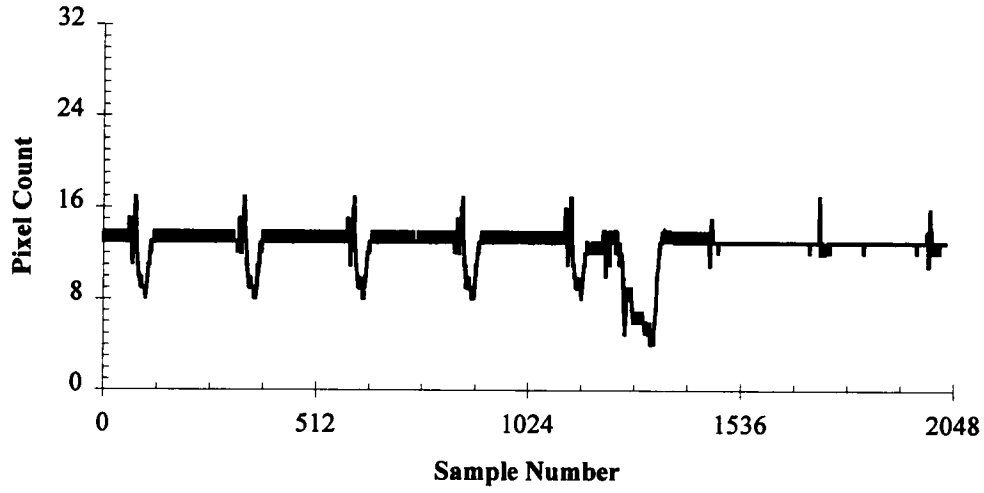

Figure 5-4: Channel Change Horizontal Frequency Variation

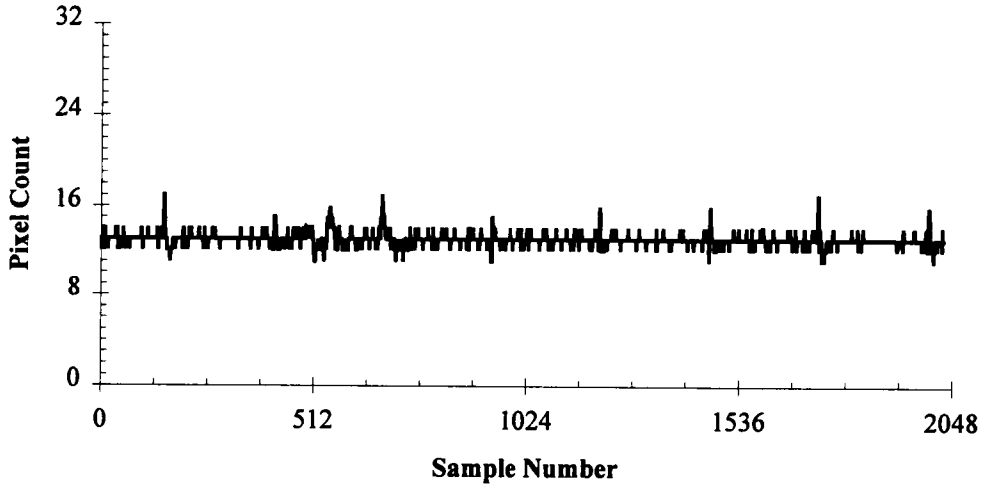

Figure 5-5: Low Signal-to-Noise Ratio Horizontal Frequency Variation

From Figure 5-5, low signal to noise effects were seen to be similar to VCR playback operation. Since low signal-to noise ratio signals are considered to be standard signals, the Time-Based Error Analyzer circuit could not mistake a VCR playback for a low signal-to-noise ratio input source. For ghosted input signals, the horizontal frequency variation was similar to the low signal-to-noise ratio signal shown in Figure 5-5.

Figure 5-6 shows the output of line counter for standard signals. The line count at vertical sync is seen to alternate between 262 and 263 decimal. Essentially, there were no vertical time-based errors seen for VCR playback, low signal-to-noise, or ghosted input signals. For channel changes, there were very large deviations in the vertical line count as witnessed in Figure 5-7. The vertical frequency variation during channel change is primarily due to the operation of the tuner in the television set. When the tuner changes channels, a finite amount of time is required to lock to the new channel. During this time, there is no sync to the input of the sync detector. From Figure 3-2, if no vertical sync is detected when the vertical countdown reaches terminal count, then a vertical sync pulse is automatically generated. From the Mitsubishi data on the vertical sync processor, the vertical countdown is approximately 35 lines, which

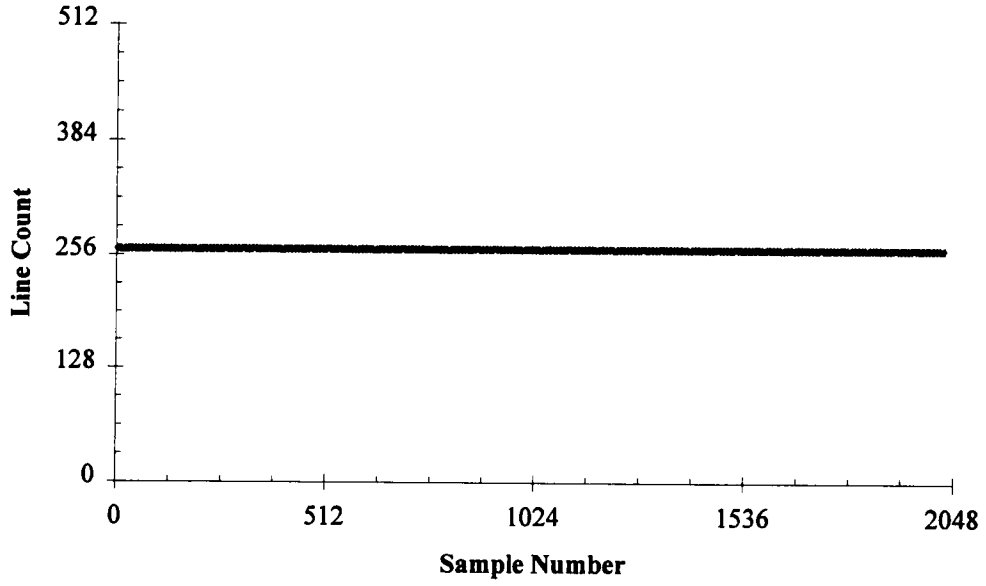

Figure 5-6: Standard Signal Vertical Line Count

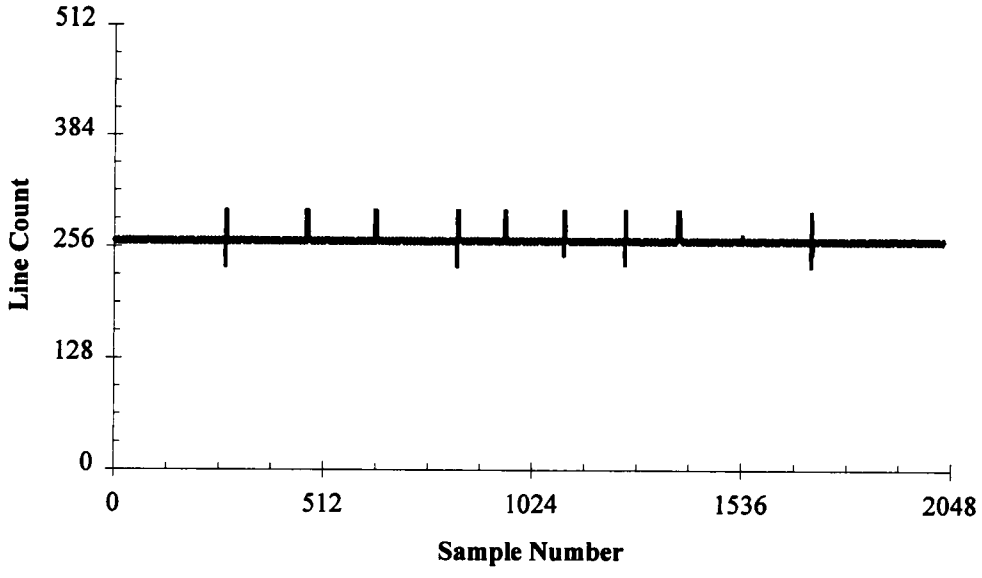

Figure 5-7: Vertical Line Count Variation Under Channel Change Conditions

corresponds to the maximum line count of 297 measured in Figure 5-7.

From the relatively simple circuit shown in Figure 5-1, enough information was obtained to begin designing a circuit to detect channel changes and VCR playback conditions. Channel changes were to be detected by measuring variations in the vertical line frequency, and VCR playback was to be detected by measuring variations in the horizontal frequency. Since VCR playback, low signal-to-noise ratio, and ghosted signals had little effect on the vertical line frequency, channel change detection was straight forward. Using horizontal frequency variations to detect VCR playback required that the effects of low signal-to-noise and ghosted signals be taken into account. Figure 5-8 through Figure 5-17 represent the complete design that was developed on the Rapid Prototyping System for the Design I Time-Based Error Analyzer. Two VHDL source files were used in the design, "main2.vhd" and "rsrc_ctl.vhd". Figure 5-10 to Figure 5-12 and Figure 5-16 to Figure 5-17 are flowcharts describing the operation of the two VHDL source files, and the complete VHDL source files are included in the Appendix.

Figure 5-8 is the top level schematic for the Time-Based Error Analyzer circuit. The top level schematic consists of two major blocks called Resource Block and MAIN2 State Machine. The circuitry shown at the bottom of Figure 5-8 synchronizes the input signals H_SYNC and V_SYNC to the system clock SYS_CLK and performs edge detection. The edge detected signals HSYNC and VSYNC are used in the Sync Counters block of Figure 5-9 to provide pixel (TC[4:0]), line (L_CNT[8:0]), and field (F_CNT[4:0]) counts to the rest of the circuit. Each counter in the Sync Counters block has an external synchronous reset.

The algorithm to detect channel changes and VCR playback is controlled by the MAIN2 State Machine block. Therefore, a description of the overall circuit operation will be presented using Figure 5-10 to Figure 5-12, and the implementation details will be described with the discussion of the Resource Block schematic shown in Figure 5-13. From Figure 5-10, the MAIN2 State Machine begins in a state called STARTUP and immediately transitions to a delay state called CC_INTERVAL. The CC_INTERVAL state enables the field counter by deasserting the FC_RST line and is exited when the field counter reaches a count of 15. The purpose of the CC_INTERVAL state is to insure that the tuner has had ample time to stabilize before attempting to analyze time-based errors. The channel change indicator, CC_DET, is set in state SET_CC before entering the state MODE. The MODE state is designed to find the modal or most often occurring pixel count per line value throughout the video. The Resource Block actually does all the work in finding and storing the modal value, and is activated by the assertion of the FIND_MODE signal. Once a modal value is found, the Resource Block signals the MAIN2 state machine that the task is complete via the FOUND_MODE line, and MAIN2 transitions to the state INIT_VERT. The states INIT_VERT and INIT_DONE are designed to initialize the VERT_DET block of the Resource Block. From INIT_DONE, MAIN2 enters the state COUNT, which is the main loop of the state machine.

The state COUNT asserts the signal COUNT_LT_GT, which signals the Resource Block to perform two functions. The first function is to measure variation between the last field and the current field line count, which in turn controls the assertion of the VER_COND signal. The second function is to count the number of times in a field that the pixel count is either greater or less than the modal pixel count

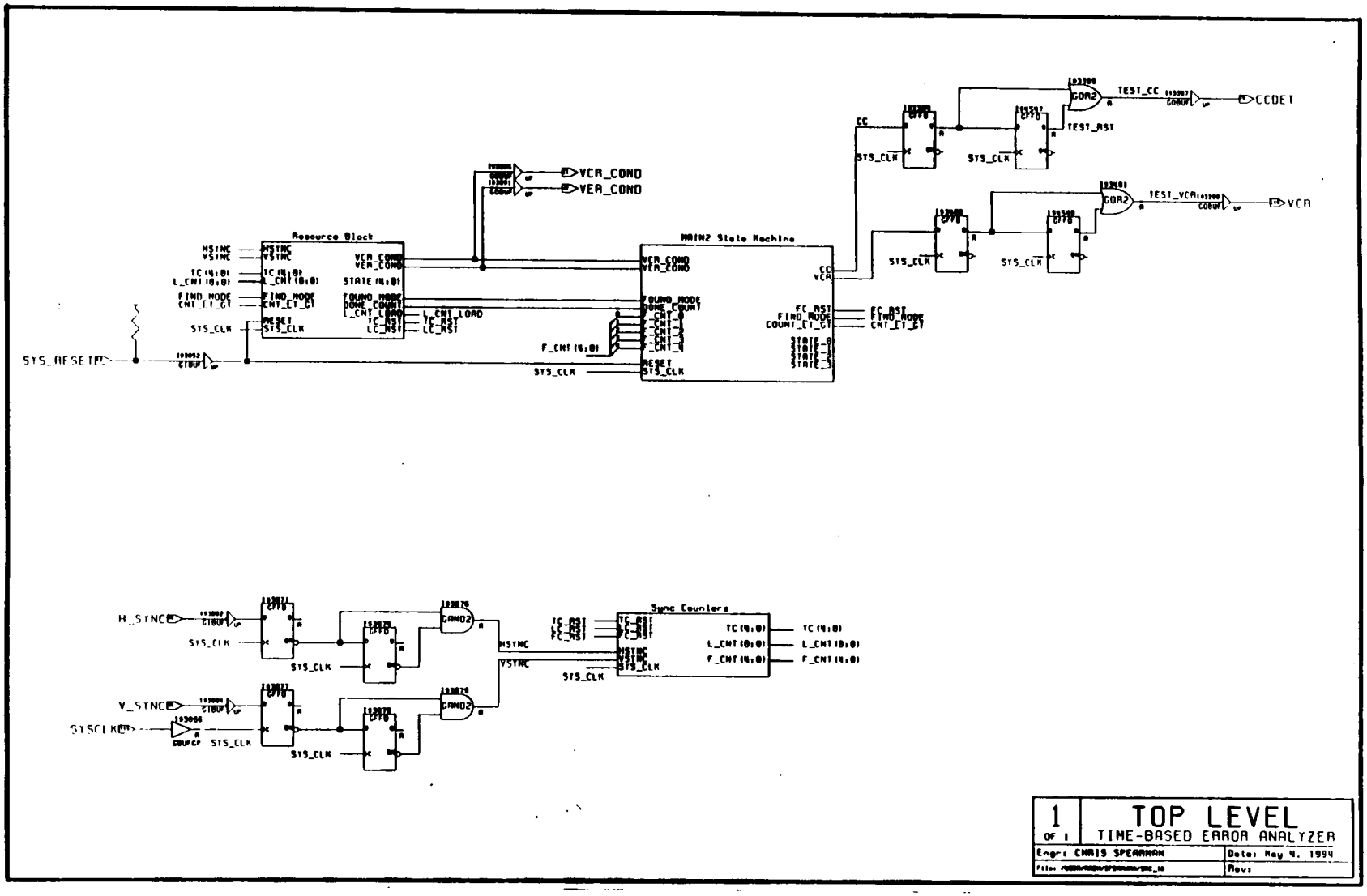

Figure 5-8: Top Level Schematic

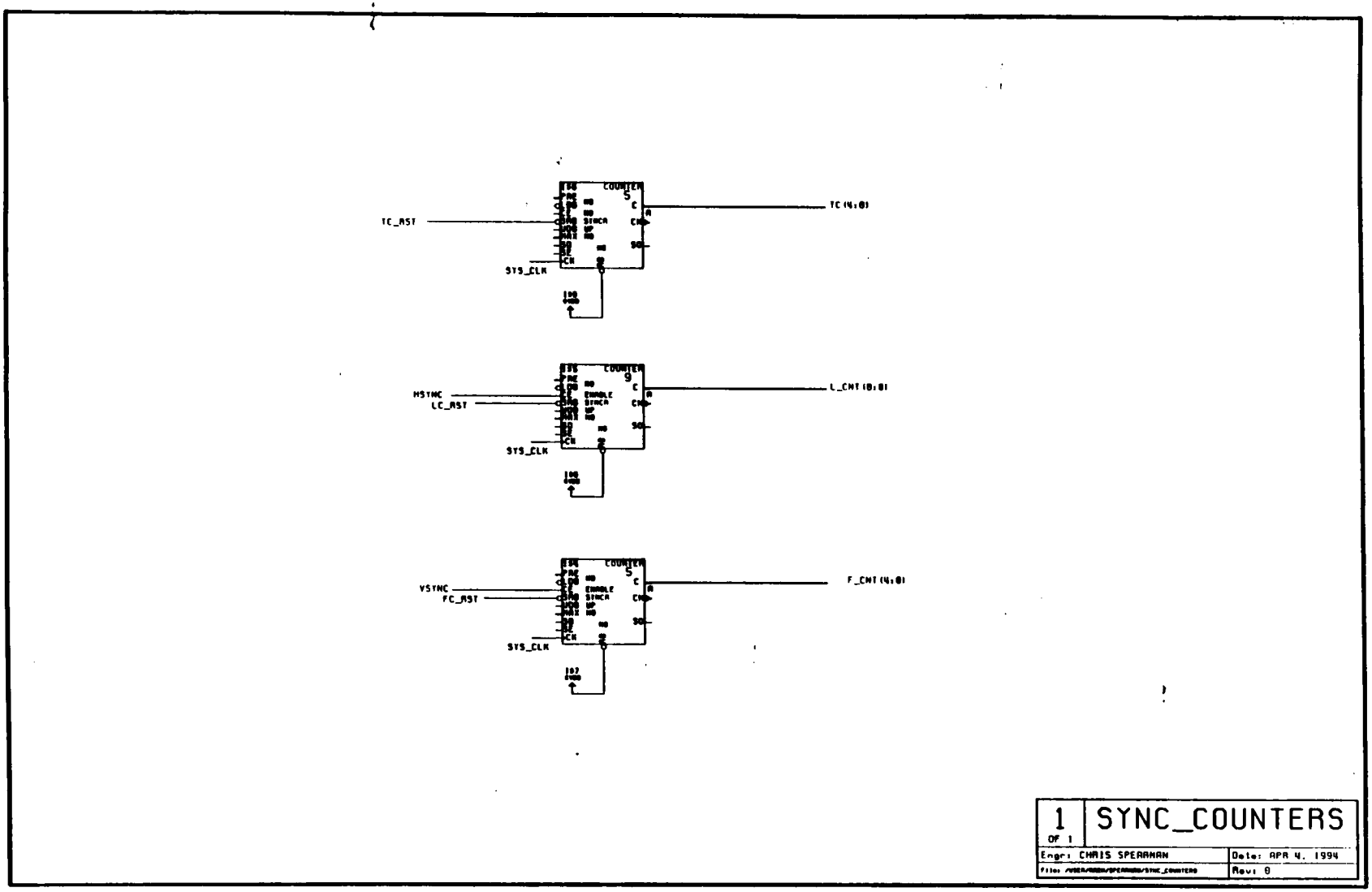

Figure 5-9: Pixel, Line, and Field Counters

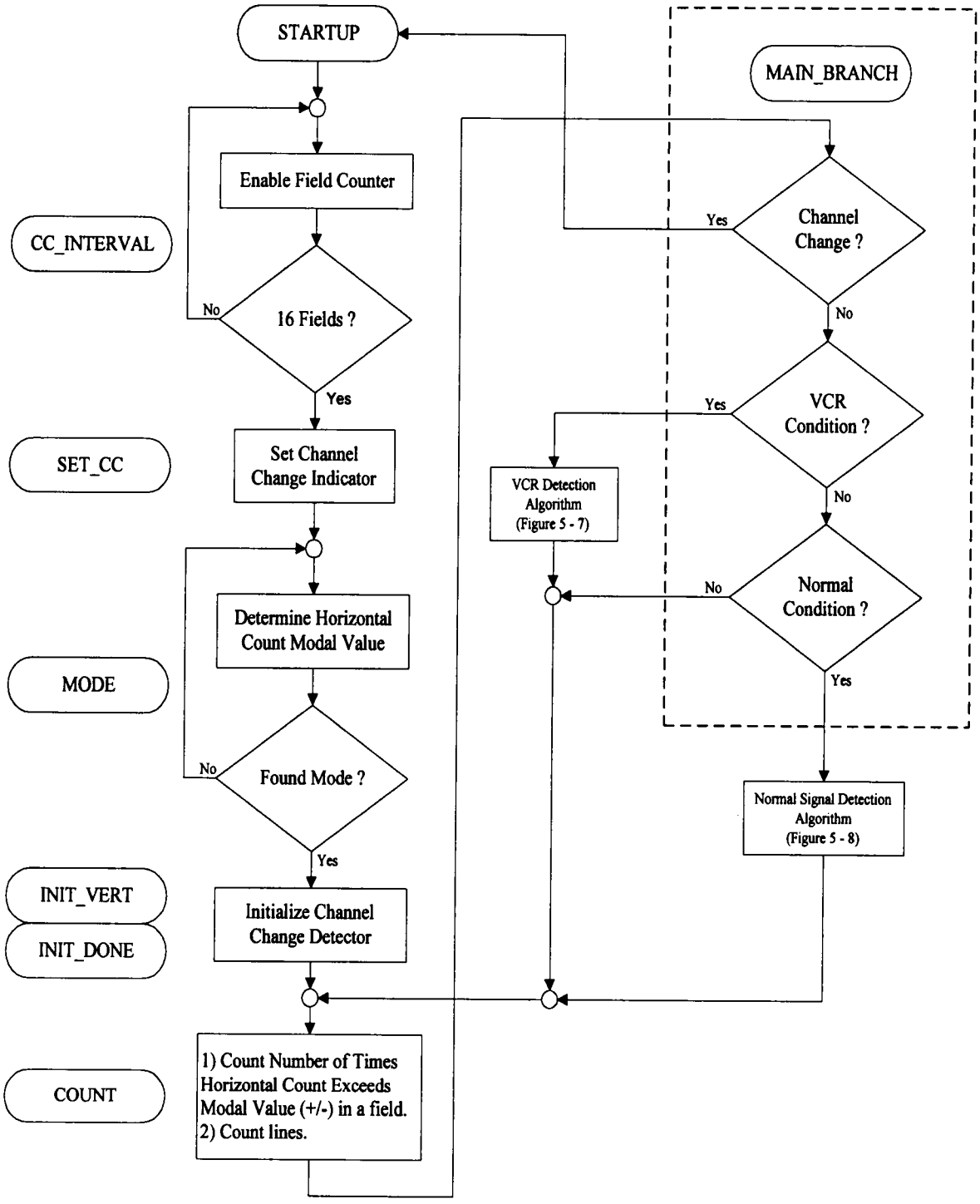

Figure 5-10: MAIN2 State Machine

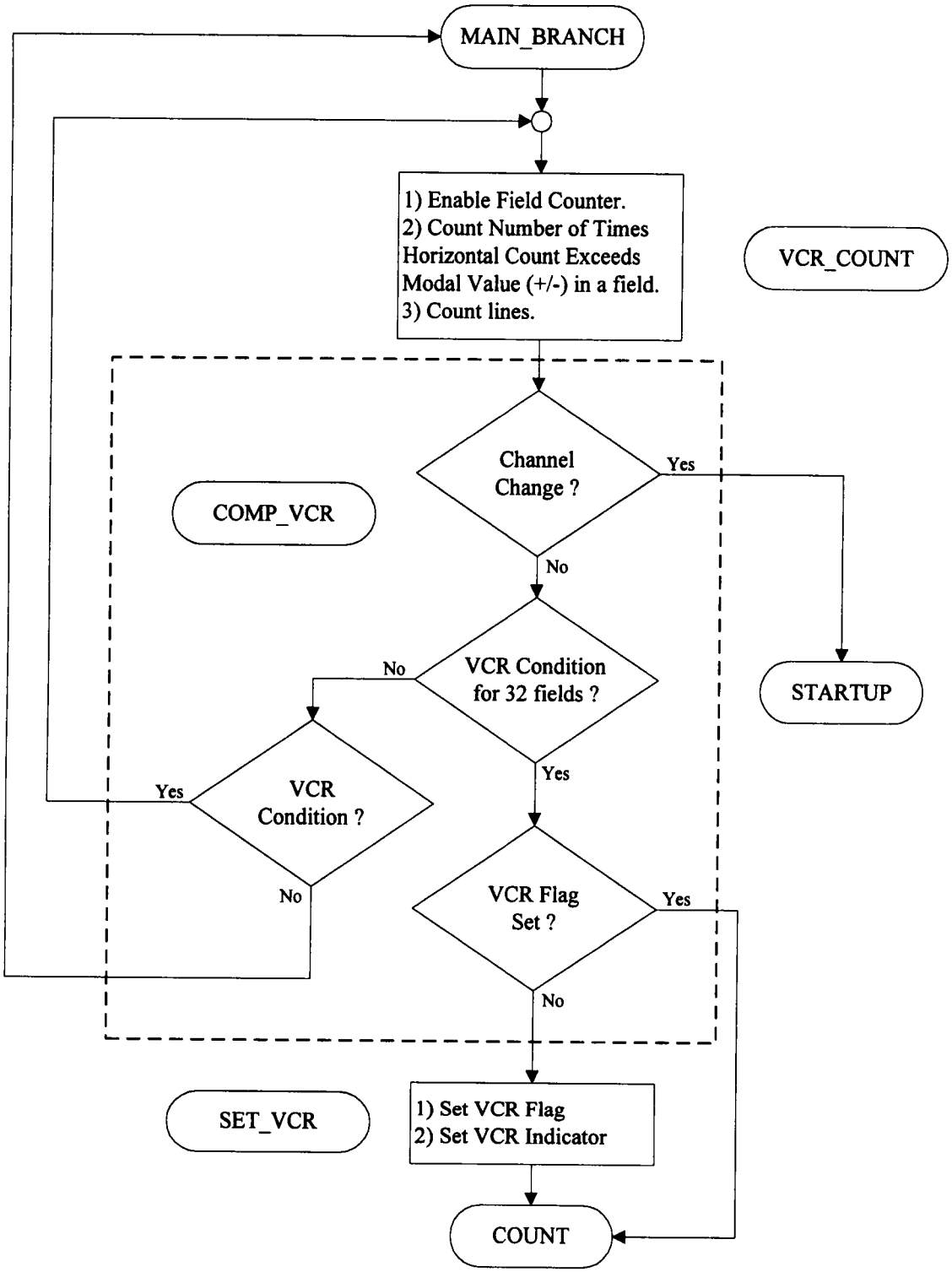

Figure 5-11: VCR Detection Algorithm

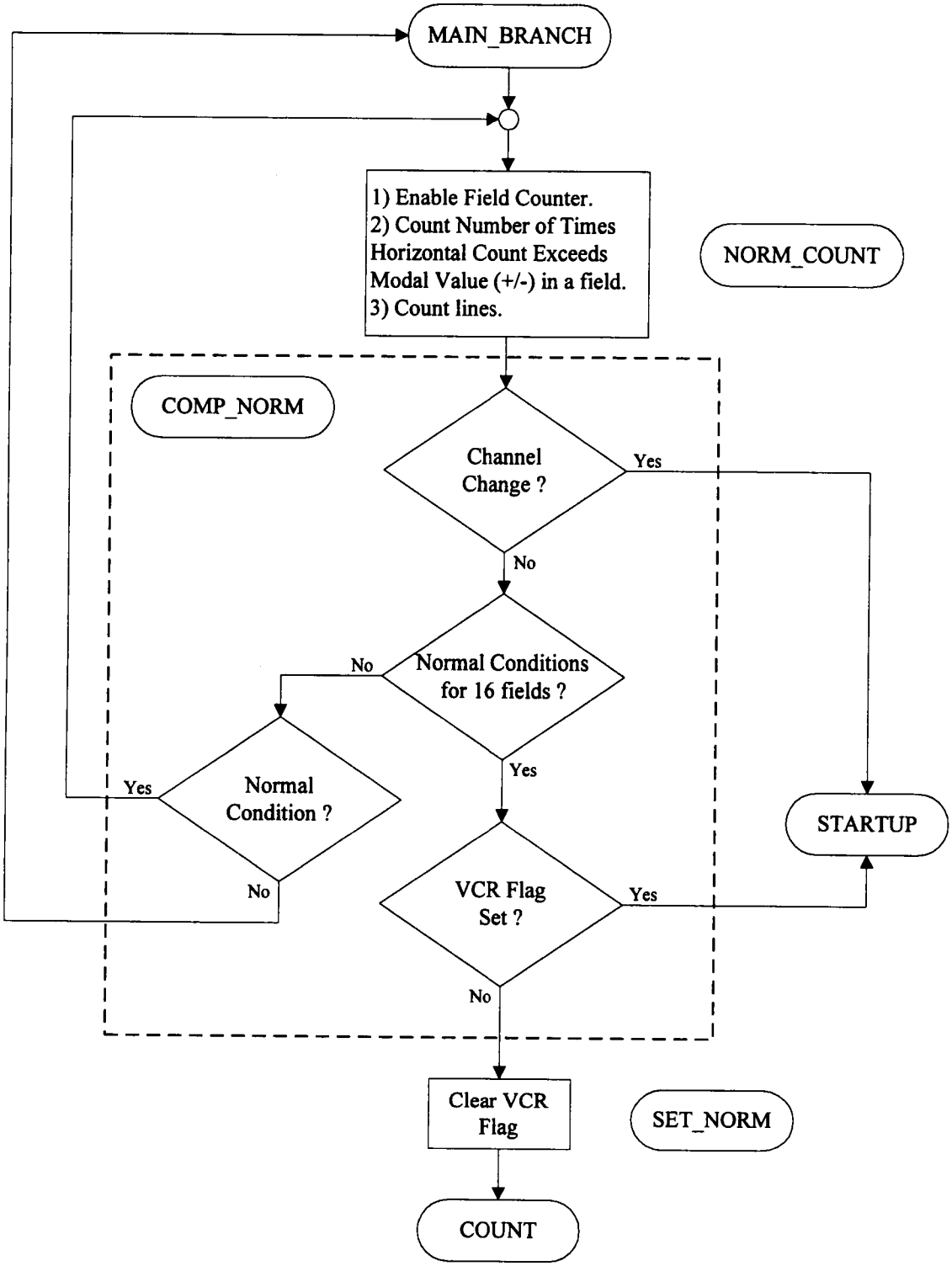

Figure 5-12: Normal Signal Detection Algorithm

value. These "Greater than" and "Less than" counts are used to assert the VCR_COND signal. When the Resource Block completes these two functions, the DONE_COUNT line is asserted, which signals a transition in MAIN2. The interaction between MAIN2 and Resource Block is analogous to a subroutine call, with the call and return being executed via a two signal handshaking protocol. When DONE_COUNT causes state transition from the state COUNT to the state MAIN_BRANCH, three conditions are possible. An active VER_COND is recognized as a channel change and causes a transition to the STARTUP state. If the VCR_COND is active, the VCR detection algorithm depicted in Figure 5-11 executes, otherwise the Normal Signal detection algorithm of Figure 5-12 executes.

The VCR detection algorithm consists of running the COUNT_LT_GT subroutine for 32 fields. If a channel change is detected after any one of the 32 DONE_COUNT returns, MAIN2 transitions to the STARTUP state. For successful VCR detection to occur, the VCR_COND must be asserted for all 32 fields, which causes the VCR indicator to be triggered. Since VCR playback is continually detected, the VCR indicator should only be triggered after the first successful VCR detection. The VCR_FLAG is used to indicate to MAIN2 whether the VCR indicator has been previously signaled. If VCR_COND is not present during any one of the 32 field VCR detection periods, then MAIN2 makes two state transitions. The first transition is to the MAIN_BRANCH state, and since the VCR_COND signal is inactive, the second transition will be to the NORM_COUNT state, which is the initial state in the Normal Signal detection algorithm.

The operation of the Normal Signal detection algorithm is similar to the VCR detection algorithm in that repeated COUNT_LT_GT subroutine calls are made to the Resource Block. Again, if the VER_COND is active after a DONE_COUNT return, the STARTUP state is entered. Normal signal conditions are evaluated 16 times. If 16 consecutive Normal signal conditions are detected, the VCR_FLAG is evaluated. If the VCR_FLAG is set, the assumption is that VCR playback has been previously detected, but the channel change detection circuit did not detect the transition from VCR playback to standard signal. Therefore, the VCR_FLAG is cleared and MAIN2 transitions to STARTUP. If VCR_FLAG is clear, the assumption is that the source is and has been a Normal signal, and MAIN2

transitions through state SET_NORM to state COUNT. If a Normal Condition does not exist after the DONE_COUNT return (i.e. VCR_COND is high), then MAIN2 transitions to state VCR_COUNT through the MAIN_BRANCH state.

The effect of the transition paths between the VCR and Normal Signal detection algorithms provide statistical immunity to spurious VCR or Normal Signal detections. For instance, assume VCR playback has been detected , but during one COUNT_LT_GT call in the VCR detection algorithm, the VCR_COND is not signaled, and the Normal Signal detection algorithm is entered. If the signal source is still a VCR, then one of the next COUNT_LT_GT calls should return with a VCR_COND active, which will cause transitions back to the VCR detection algorithm. Therefore, the push-pull interaction between the VCR and Normal Signal detection algorithms prevented an extra VCR present indication from being signaled. From a system perspective, spurious VCR detects, as well as spurious channel change detects, are a problem because the ghost canceler algorithm flushes its filter coefficients and readapts after VCR and channel change detects. During this time, (ghosted) video is bypassed, which can be very annoying if in fact no change of channel or VCR playback mode was initiated.

The description of the MAIN2 state machine is completely described in the Appendix with the VHDL source file. In the schematic, the CC and VCR outputs of the MAIN2 state machine drive some additional circuitry, which is to condition the one clock cycle long CC and VCR detect pulses to two clock cycles at the request of Zoran. Also of note from the top level diagram is the SYS_RESET line, which is seen to be pulled high. The SYS_RESET line is required by the ASIC implementation of the design; however, the Xilinx XC4005 parts assert a global set/reset line immediately before switching from program mode to normal operation mode, which insures that all flip-flops power on in the zero state, and the SYS_RESET line is not needed for the Xilinx implementation.

The Resource Block implements two separate subroutines, FIND_MODE and COUNT_LT_GT, which share common resources between the two functions. The schematic for the Resource Block is shown in Figure 5-13. Most of the functionality of the Resource Block is implemented in the one level of hierarchy, but two sub-blocks decode VCR_COND (Figure 5-14) and VER_COND (Figure 5-15). Figure

5-16 and Figure 5-17 are state diagrams of the operation of the RSRC_CTL block, which is a state machine implemented using VHDL. The VHDL source code for the RSRC_CTL is included in the Appendix.

RSRC_CTL begins in the WAIT_CMD state, and the two possible transitions are to either the GET_HOLD or the BEGIN_CNT states. From the description of the MAIN2 state machine, FIND_MODE is used to tell the Resource Block to find and store the modal value for the pixel count per line. If the FIND_MODE signal is asserted while the RSRC_CTL is in the WAIT_CMD state, then RSRC_CTL transitions to the state GET_HOLD, which waits for a HSYNC pulse. When the HSYNC pulse is received, the pixel counter is reset by asserting the TC_RST line in the state TC_RESET, and a state transition to HOLD_EN occurs. The state HOLD_EN releases the pixel counter, and when the next HSYNC occurs, RSRC_CTL transitions to the LOAD_HOLD state. The signals HOLD_LOAD and TC_RST are asserted in the LOAD_HOLD state, which perform the tasks of loading the pixel count (TC[4:0]) into the HOLD_REG and clearing the pixel counter respectively. From the LOAD_HOLD state, RSRC_CTL transitions to the COMP_EN state, which is the main loop of the FIND_MODE subroutine. In the main loop, the HIT_CNT_RST signal is deactivated, which allows the HIT_COUNTER to hold its value throughout the main loop, and the LC_RST signal is deactivated, which enables the line counter (L_CNT[8:0]) in the main loop. The COMP_EN state enables the pixel counter until the next HSYNC occurs, which causes a state transition from COMP_EN to LOAD_TC. The state LOAD_TC loads the pixel counter value into the COMP_REG by asserting the NEW_LOAD signal and resets the pixel counter by asserting the TC_RST line. The next state transitions are to ONE_WAIT, which is included to match the number of states in the pixel hold process to the pixel compare process, and then to CHECK_HIT. CHECK_HIT makes a branch to either HIT_YES or HIT_NO based on comparison of the values in HOLD_REG and COMP_REG. The value for COMP_REG applied to the A input of the COMPM5 device is not affected by either the MUX21 or the ADDCI devices because the A input (all zeros) is selected by the low level on the MUX_SRC_SEL line, and the ADDER_CI line is zero. If the values in the HOLD_REG and COMP_REG are equal, then the HIT_YES state is entered which asserts the HIT_CNT_CE signal and increments the HIT_COUNT; otherwise, the HIT_NO state is entered, which

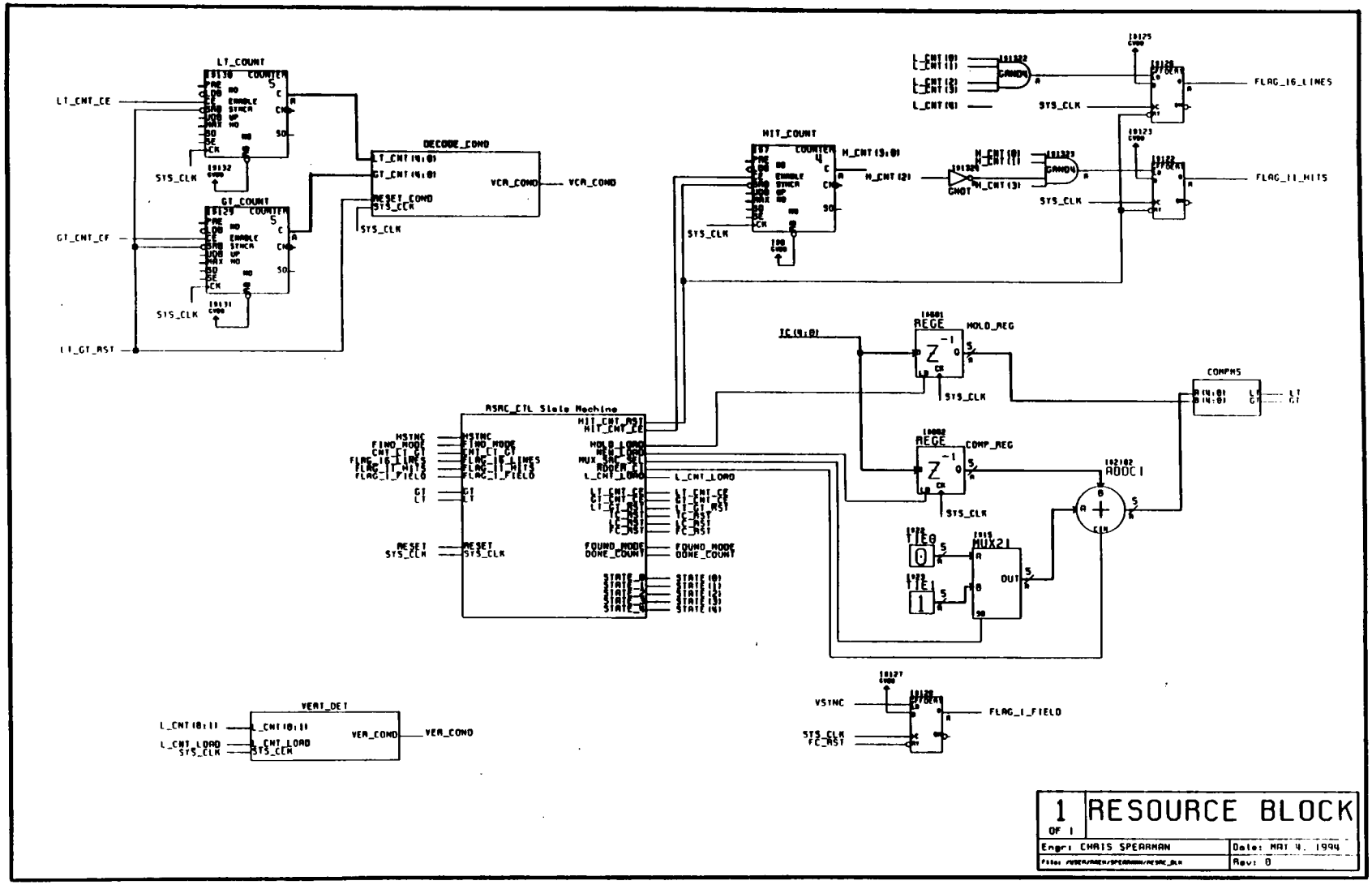

Figure 5-13: Resource Block Schematic

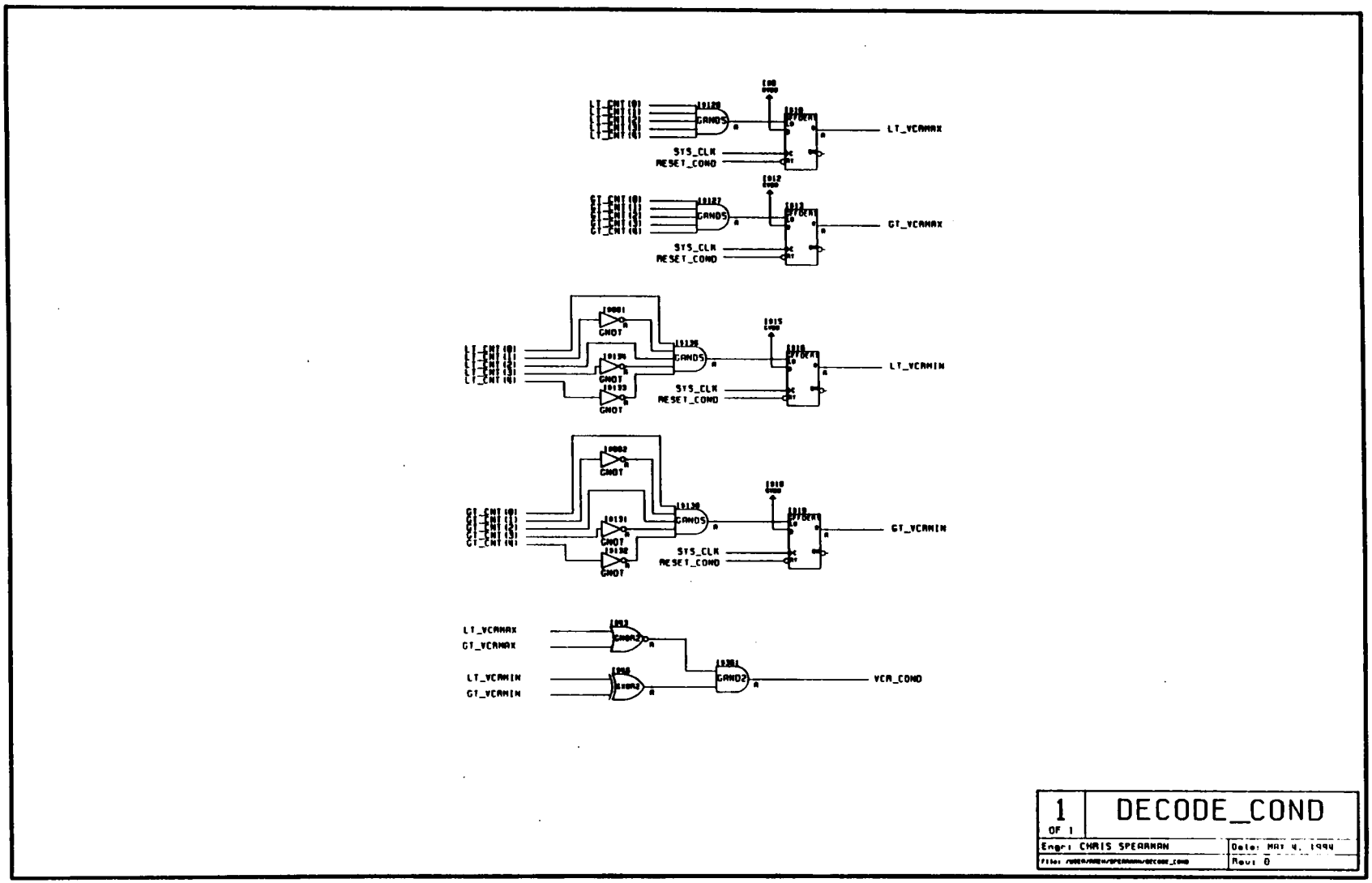

Figure 5-14: VCR Condition Decoder

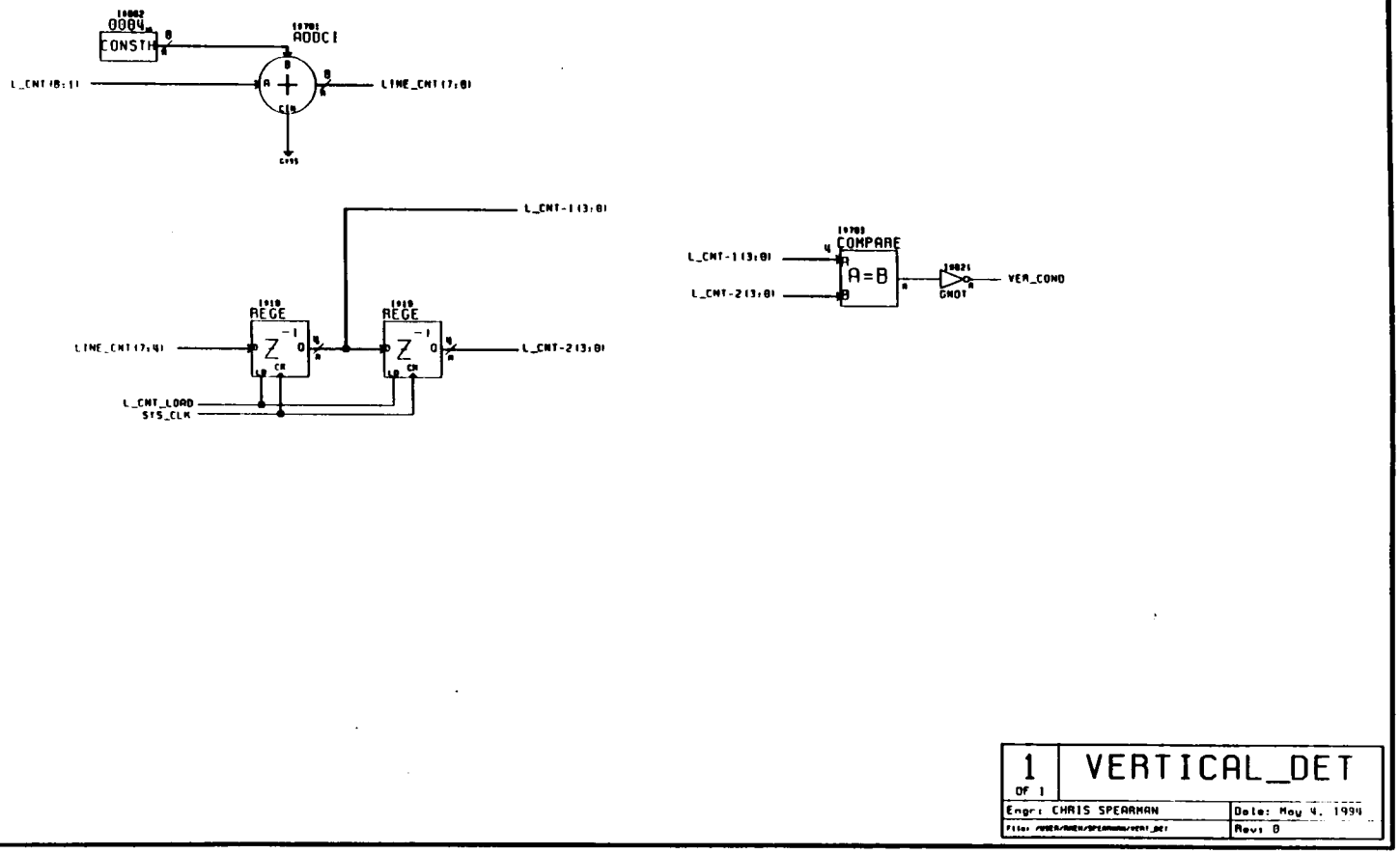

Figure 5-15: Channel Change Condition Decoder

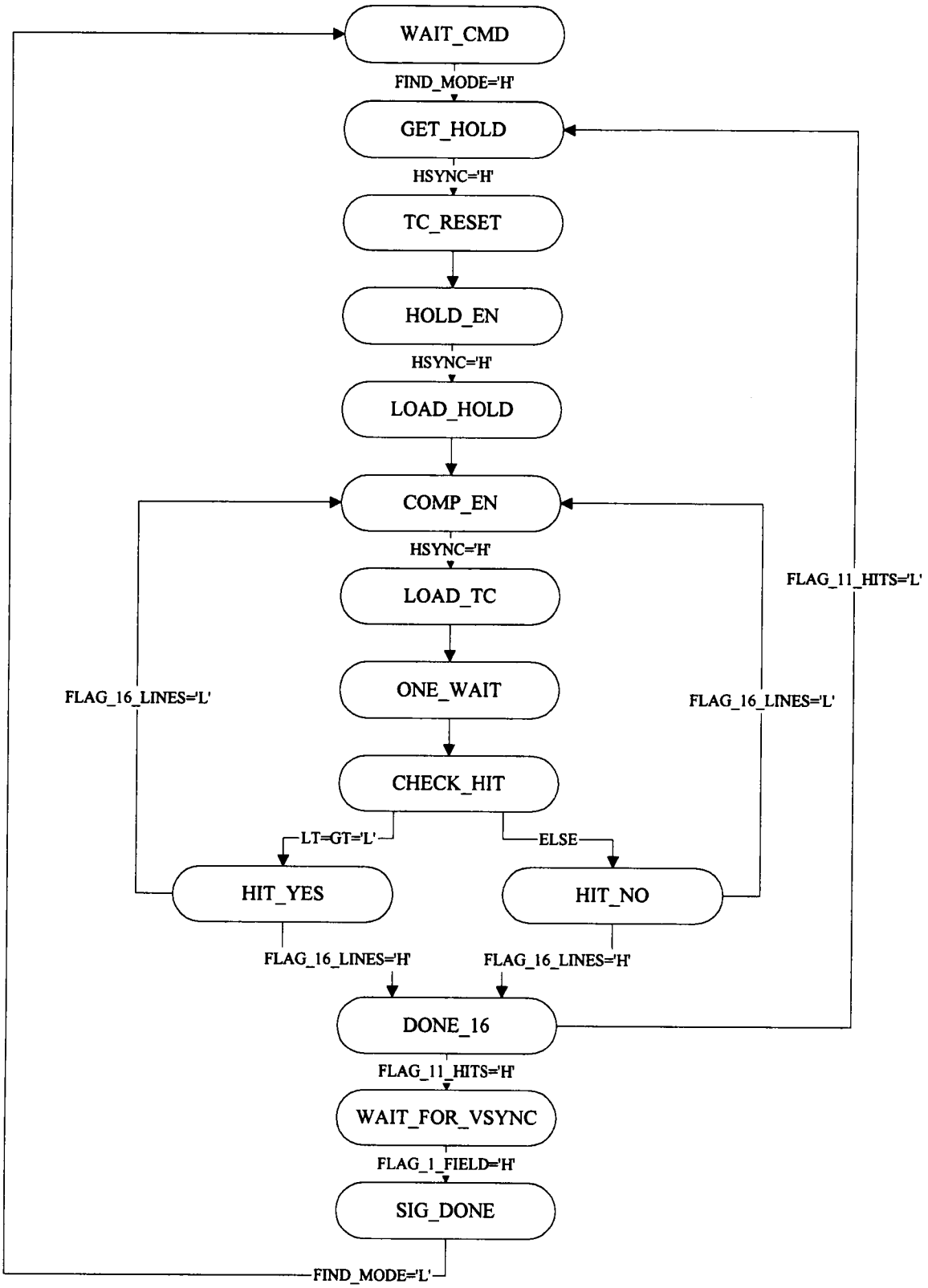

Figure 5-16: FIND_MODE Subroutine State Diagram

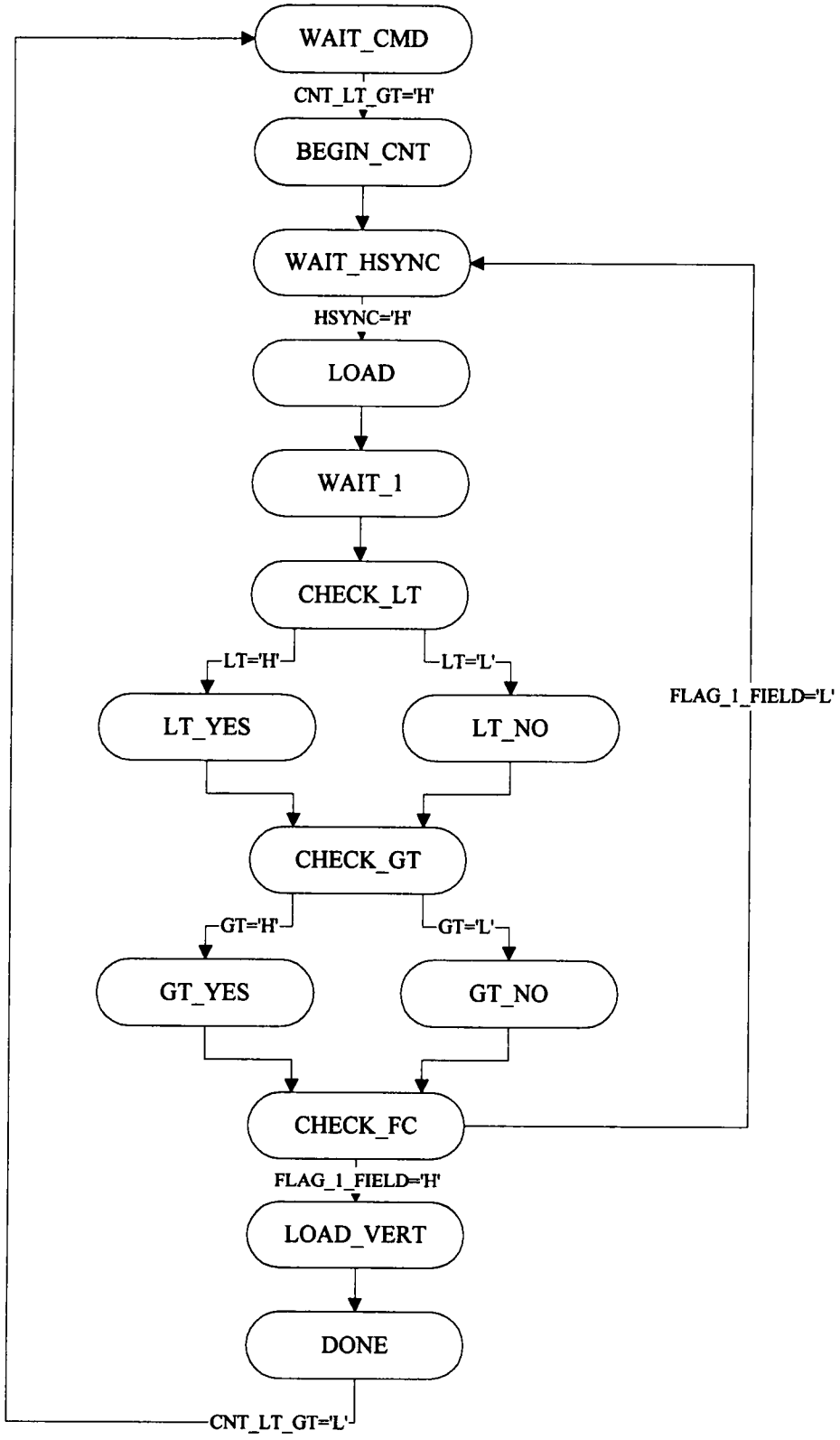

Figure 5-17: CNT_LT_GT Subroutine State Diagram

simply maintains timing. The transition stimulus from both HIT_YES and HIT_NO is the FLAG_16_LINES signal, which is generated by clocking a high level into a flip-flop when the line counter reaches the value of 15. The main loop is executed until the FLAG_16_LINES signal is set, at which point the DONE_16 state is entered. DONE_16 examines the FLAG_11_HITS signal, which is a flag set when the HIT_COUNT reaches a value of 11, and if FLAG_11_HITS is clear, the entire FIND_MODE process is restarted, otherwise RSRC_CTL transitions to WAIT_FOR_VSYNC. The primary function of the WAIT_FOR_VSYNC state is to enable the field counter until the end of the current field, at which point the FLAG_1_FIELD flag is set, and the SIG_DONE state is entered. SIG_DONE asserts the FOUND_MODE line, which is the subroutine return, and when FIND_MODE is deactivated, the WAIT_CMD state is entered.

From the previous discussion of state diagram shown in Figure 5-16, the algorithm for finding the modal pixel count value is deduced. The pixel count between two HSYNC pulses is captured in the HOLD_REG. For the next 16 lines, pixel counts are captured in the COMP_REG and compared to the value in the HOLD_REG. If the values are equal, the HIT_COUNT is incremented, and if 11 of the 16 comparisons are equal, then the value in the HOLD_REG is considered to be the modal pixel count value. If less than 11 hits of 16 lines are counted, then the FIND_MODE subroutine continues until a mode value is found. Once a mode value is determined, the subroutine returns control back to the calling routine.

The second subroutine implemented by the RSRC_CTL is COUNT_LT_GT, which is depicted in Figure 5-17. The BEGIN_CNT state is entered from the WAIT_CMD state by the assertion of CNT_LT_GT, which immediately transitions to the WAIT_HSYNC state. WAIT_HSYNC enables the pixel counter until the activation of HSYNC causes a state transition to LOAD, where the COMP_REG is loaded with the pixel count value by the assertion of NEW_LOAD. Also, the pixel counter is reset in the LOAD state by the assertion of the TC_RST signal. The WAIT_1 state is entered after the LOAD state, which is a wait state inserted to maintain timing, then transitions to CHECK LT. CHECK_LT deasserts the MUX_SRC_SEL line and activates the ADDER_CI line, which effectively adds one to the value in COMP_REG. The (COMP_REG + 1) value is compared to the value in HOLD_REG by the COMPM5

device. If LT is asserted, then the LT_CNT_CE line is asserted in the LT_YES state, which increments the count value in LT_COUNT. The LT_NO state is entered from CHECK_LT if LT is not asserted. The next transition from both LT_YES and LT_NO is to CHECK_GT. CHECK_GT deasserts ADDER_CI and asserts MUX_SRC_SEL to effectively subtract 1 from the COMP_REG value. Based on the comparison HOLD_REG and (COMP_REG - 1), the RSRC_CTL jumps to state GT_YES if GT is asserted otherwise GT_NO. GT_YES asserts GT_CNT_CE, which increments the value in GT_COUNT. CHECK_FC is entered from both GT_NO and GT_YES, and checks the FLAG_1_FIELD indicator to see if a VSYNC has occured. If so, the state LOAD_VERT is entered, otherwise RSRC_CTL loops back to the WAIT_HSYNC state. Transition occurs immediately from the LOAD_VERT state to the DONE state, where the DONE_COUNT line is asserted until CNT_LT_GT is deactivated.

The main loop of the COUNT_LT_GT subroutine begins with the WAIT_HSYNC state, and ends with the CHECK_FC state. Throughout the main loop, the LC_RST and FC_RST lines are deactivated to enable line and field counting. Therefore, in the LOAD_VERT state, the L_CNT_LOAD line is asserted, which registers the current line count as shown in Figure 5-15. From Figure 5-15, a constant of 4 is added to line counter value L_CNT[8:1] to form the signal LINE_COUNT[7:0]. The L_CNT_LOAD is the control signal for a two level shift register, whose input is the LINE_COUNT[7:4] signal. The values in each shift register are equality compared to generate the VER_COND signal. The VCR_COND signal is generated by the DECODE_COND block shown in Figure 5-14. From Figure 5-14, 4 flags are set based on the values in the LT_COUNT and GT_COUNT counters. These flags are LT_VCRMAX, GT_VCRMAX, LT_VCRMIN, and GT_VCRMIN. The values required to set these flags were determined experimentally. The VCR_COND was generated by applying a set of Boolean operators on the flags. The VCR_COND is based on the logic analyzer data shown in Figure 5-3 because the pixel counts were seen to deviate in only one direction from the mode value during VCR playback. Low signal-to-noise ratio and ghosted signals were seen to deviate in both directions as shown in Figure 5-5. Therefore, by XOR'ing the LT_VCRMIN and GT_VCRMIN, noise and ghost immunity are obtained.

The COUNT_LT_GT subroutine previously described is summarized. For an entire field, pixel count values +/- 1 are compared to the modal pixel value in HOLD_REG. The LT_COUNT register is incremented if:

$$COMP\_REG < HOLD\_REG - 1$$

and the GT_COUNT register is incremented if:

$$COMP\_REG > HOLD\_REG + 1$$

Also during the field, lines are counted. At the end of the field, VER_COND and VCR_COND are generated. The VER_COND is generated based on variations in the line count between two successive fields, and the VCR_COND is generated based on the values in LT_COUNT and GT_COUNT.

The operation of Design I of the Time-Based Error Analyzer circuit has been completely described. The design for the Time-Based Error Analyzer presented took approximately two months to develop, prototype, and debug. Since schedule was an important factor in the design, the test methodology had to be implemented at the same time the circuit robustness was evaluated. However, test methodology and robustness turned out to be the two largest obstacles for the design to move forward. The goal of a test methodology is to provide a means to test all nodes in the design for "stuck at" faults, which requires both *controllability and observability* of all nodes in the design. The test methodology that was implemented consisted of breaking the circuit into functional pieces, and writing test vectors that exercised the pieces functionally. The result of this methodology was a much larger circuit, and a set of test vectors that had indeterminate "stuck at" fault coverage.

A second major problem in the Design I Time-Based Error Analyzer circuit was robustness. The channel change detection algorithm had a problem with false channel change detection over long time periods, and the VCR detection algorithm was not able to detect VCR playback when a video tape was played on the same VCR that was used for recording. The cause of the false channel change detections was determined to be at least partially to blame on broadcasters, and partially to blame on the sync processor. On the part of the broadcasters, a study of the video signal over long periods of time revealed that the video was interrupted intermittently; however, the sync processor also seemed to randomly miss

the vertical sync, which also caused the false channel change detection. Ghosting conditions also seemed to increase the frequency at which the false channel changes were detected. The inability to detect VCR playback was because the detection algorithm relied on tape stretch. Since very little tape stretch occurs on the same tape between recording and playback, the horizontal frequency deviation witnessed in Figure 5-3 is not present, and the VCR playback characteristic is very close to the normal signal characteristic shown in Figure 5-2. The Design I Time-Based Error Analyzer, mapped to Mitsubishi gates and including test vector generation, had taken approximately 4 months to complete, which was past the promise date for deliverables to Mitsubishi. However, the lack of testability and robustness forced the decision to redesign the circuit. The Synopsys toolset was chosen as the design flow to reduce design cycle time, and a more classical approach was taken to detection of VCR playback and channel changes using the $4f_{sc}$ burst-locked clock.

Chapter 6

Design II: Time-Based Error Analyzer Using a Burst-Locked Clock

The issue of a free-running vs a burst-locked system clock for the ghost cancellation system has been a topic of debate from the very earliest days of system planning. In past digital signal processing products, a burst-locked system clock was necessary for sync regeneration, color encoding/decoding, etc. The argument against a free-running clock for ghost cancellation is that high frequencies are rolled off in the GCR averaging process because the video samples do not line up vertically and horizontally field to field. However, studies of burst-locked clock performance under ghosting conditions have shown that the burst-locked clock phase jitters significantly, which causes the same high frequency roll off during GCR averaging as a free-running clock. The Design I Time-Based Error Analyzer used the free-running approach to channel change and VCR detection to eliminate the need for a burst-locked clock in the ghost cancellation system. However, the solution did not perform satisfactorily under predicted operating conditions.

The first task was to improve VCR detection performance. From the discussion of VCR playback in Chapter 3, the "color under" modulation system is used in most VCR's on the market. The "color under" modulation scheme produces a non-standard CVBS with no phase relationship between the color burst and horizontal sync. Figure 6-1 is a block diagram of a test circuit that was used to examine the phase relationship between the color burst and the horizontal sync signals. The system clock (SCLK) shown in Figure 6-1 was generated by a burst-locked clock circuit similar to that shown in Figure 3-3. After synchronization of the H_SYNC and V_SYNC signals to SCLK, edge detection is performed on H_SYNC and V_SYNC. The rising edge of H_SYNC was used to load a countdown value of 909 into the 10-bit Down Counter and to count enable the 9-bit Line Counter. The H_SYNC edge also was used to enable a 16-bit Accumulator, which sums the count value in the 10-bit Down Counter. For standard NTSC composite waveforms, the 10-bit Down Counter should be zero at every rising edge of H_SYNC

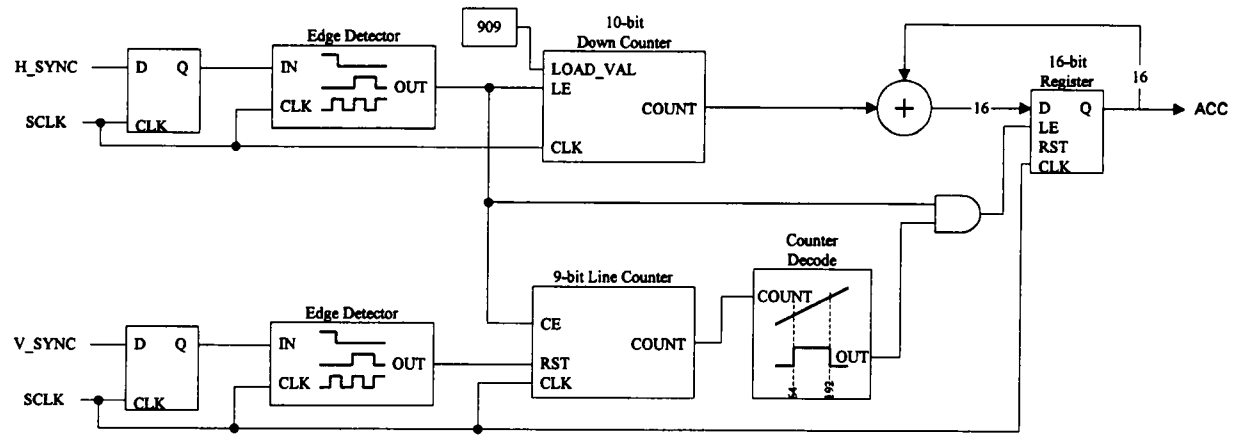

Figure 6-1: Test Circuit

because the fixed relationship between color burst frequency to horizontal frequency, and the 16-bit Accumulator should be zero. If there is phase error between the burst-locked clock and horizontal sync, then a rising edge of H_SYNC will come at a point either greater or less than 909, and the value left in the 10-bit Down Counter is summed in the 16-bit Accumulator. The 9-bit Line Counter and Counter Decode blocks were put into the test circuit of Figure 6-1 to deal with horizontal frequency deviation during the VBI, which was a phenomenon present for all signal sources. The Counter Decode output is high when the 9-bit Line Counter value is between 64 and 192, which is a window of the video outside the VBI. The Counter Decode output is used to gate the rising edge of H_SYNC to the 16-bit Accumulator, which enables phase error accumulation only outside of the VBI.

The test circuit of Figure 6-1 was described in VHDL, which was synthesized to a Xilinx XC4005 part using Synopsys. The complete VHDL source for the test circuit is listed in the Appendix as "err_acc.vhd". The 16-bit Accumulator output was monitored with a Philips PM3585 Logic Analyzer, which was setup to capture data at the rising edge of V_SYNC. Figure 6-2 contains several plots of the logic analyzer data that was collected for different input signal conditions. The accumulator output was relatively flat for normal, low signal-to-noise, and mildly ghosted input signals. VCR Playback was seen to have a relatively steep, monotonic slope. During channel change, the accumulator experienced an

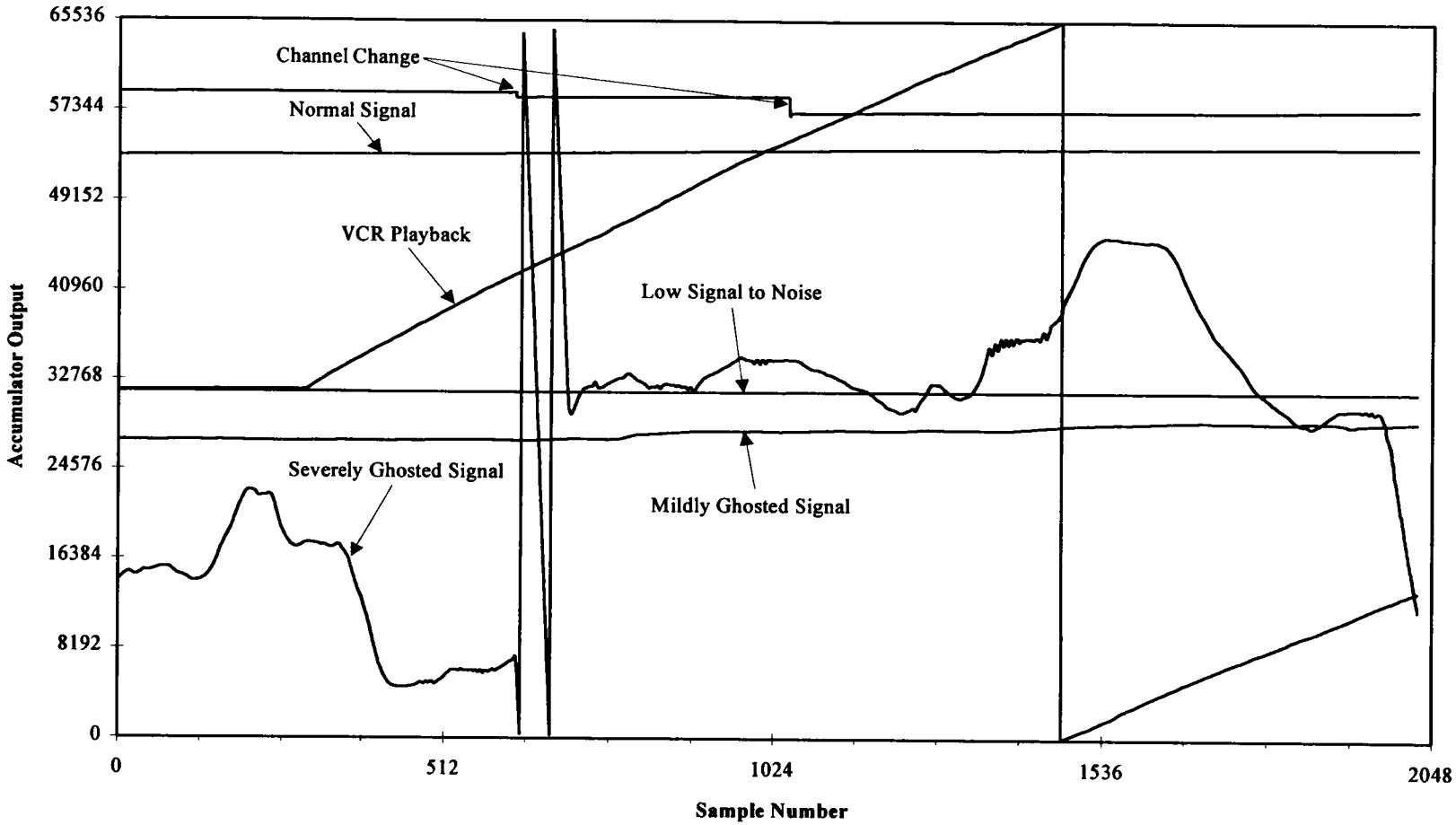

Figure 6-2: Test Circuit Data

abrupt change as both the horizontal PLL and the burst PLL acquired phase lock to the new channel. From Figure 6-2, channel change and VCR detection appears possible by taking a derivative of the phase error accumulator using two different time scales. The phase error accumulation characteristic for channel changes occur within a few fields, suggesting a time scale resolution of one field. The VCR playback accumulation characteristic shown in Figure 6-2 indicates that a longer time base is required for the derivative of the phase error accumulator. Severely ghosted signals can cause the slope of the phase error accumulation to be as large as VCR playback, which is demonstrated in Figure 6-2, and an attempt was made to eliminate the possibility of false VCR detections due to severe ghosting conditions. One distinguishing characteristic of severely ghosted signals from VCR playback is that the phase error accumulator is non-monotonic. Therefore, several smaller time scale derivatives of the accumulator can be used to detect VCR playback, while also preventing false VCR detection due to ghosting. A final note on Figure 6-2 is the extremely sharp spike witnessed on the Severely Ghosted Signal plot, which would most likely be detected as a false channel change. An infinite number of ghosting conditions exist, and the limitations of the sync extraction circuits had to be accepted as a limiting factor in the operation of the ghost cancellation system.

Design II of the Time-Based Error Analyzer circuit grew out of Figure 6-1. Figure 6-3 is a block diagram of the final circuit by function; however, the design was captured purely in VHDL and the exact implementation is not represented by Figure 6-3. The complete VHDL source for Design II Time-Based Error Analyzer is listed in the Appendix. A description of the VCR playback and channel change detection algorithm follows from the circuit description. The flowchart shown in Figure 6-4 describes the operation of the State Machine block of Figure 6-3. The State Machine consists of three states, "main", "cc_delay", and "check_vcr". The "main" state implements the exact functionality of the test circuit in Figure 6-1, with the exception that at each V_SYNC edge, the current accumulated phase error value is compared to the previous field's accumulated phase error value. If the difference is greater than the threshold value CC_THRES, then a change of channel condition is assumed, and a state transition to "cc_delay" occurs. In the state "cc_delay", a delay of 15 fields is incurred to allow time for channel lock before asserting

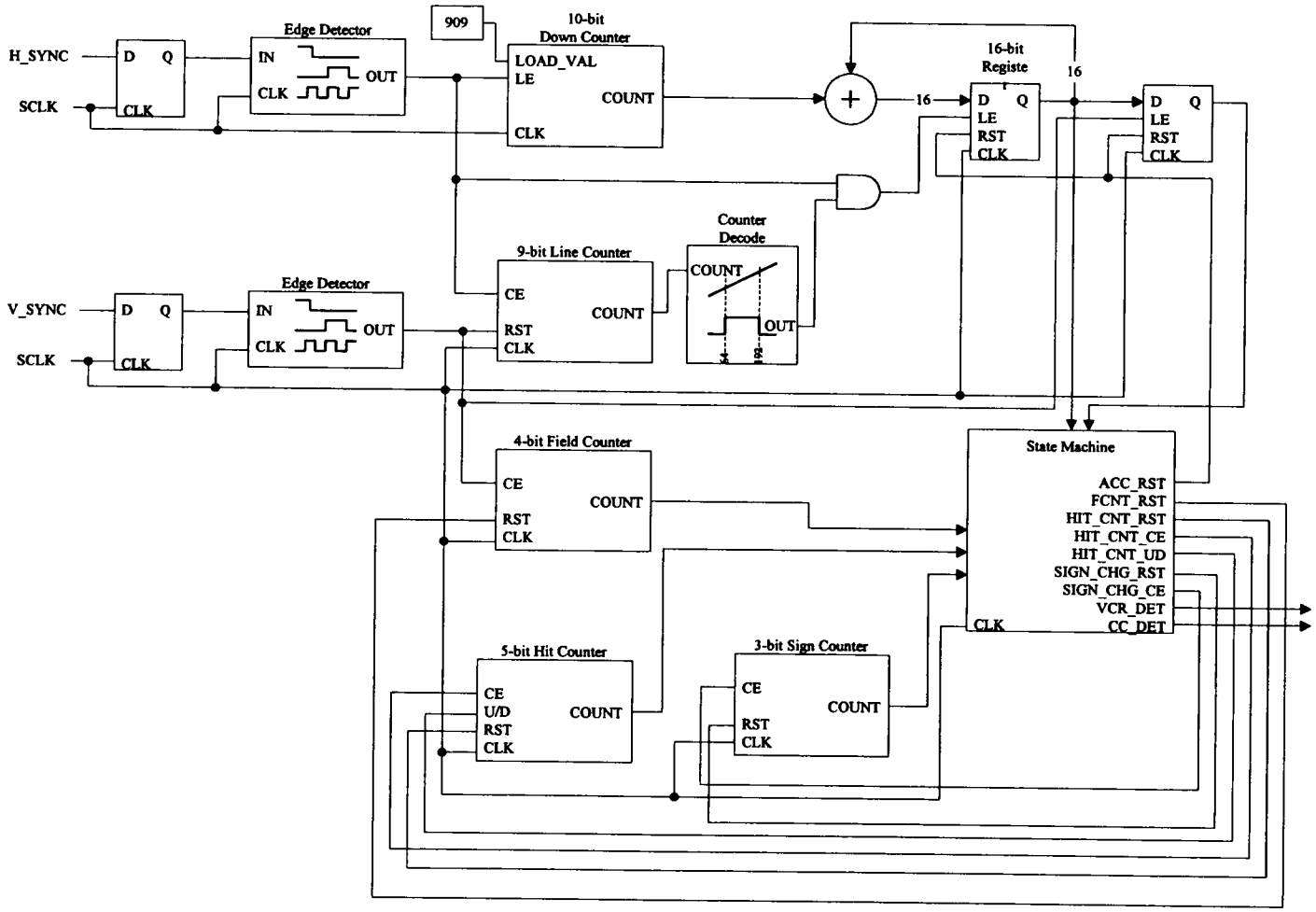

Figure 6-3: Design II Functional Block Diagram

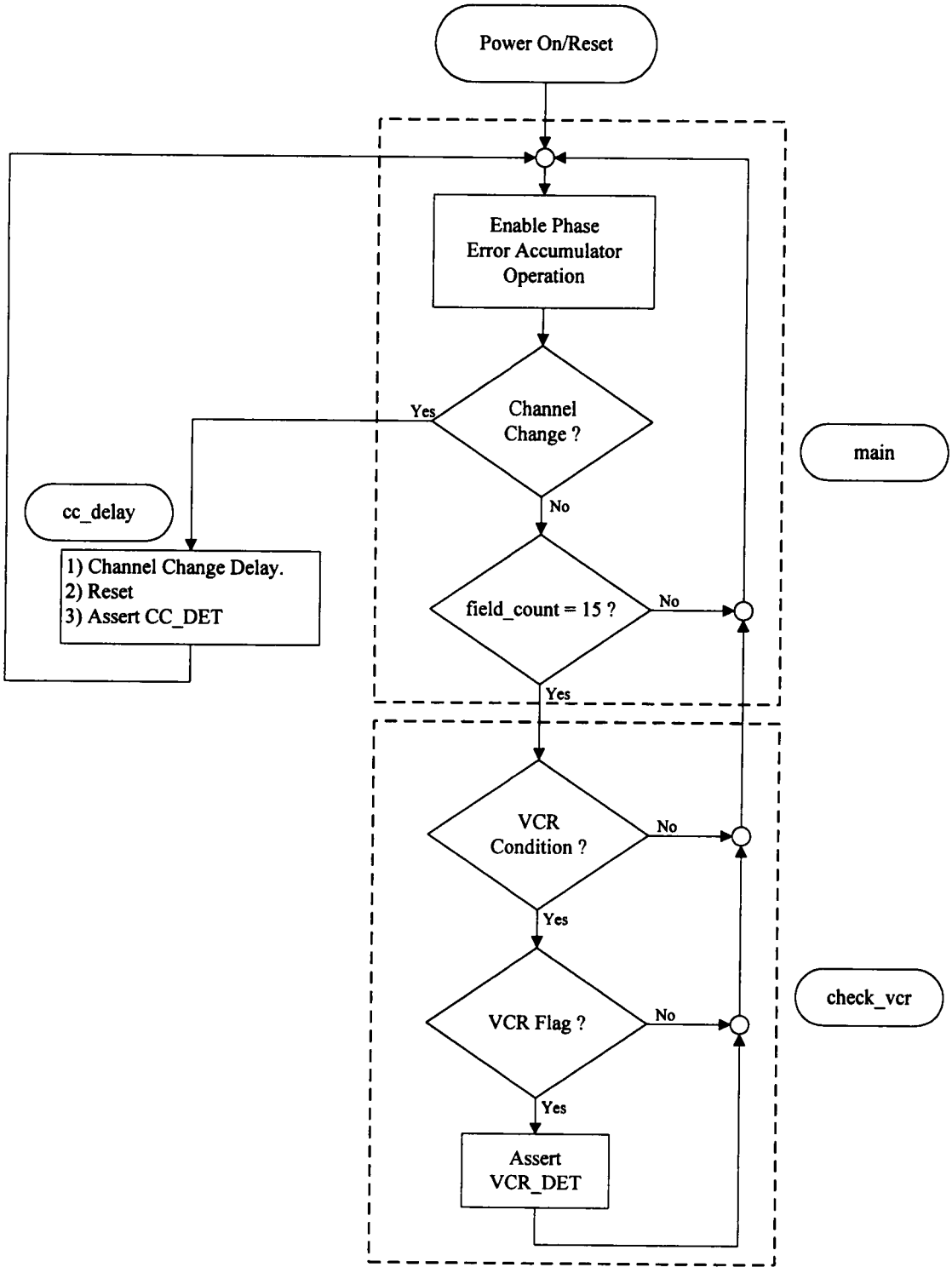

Figure 6-4: Design II Flowchart

before asserting CC_DET. From "cc_delay", the circuit is reinitialized and the "main" state is entered. While in the "main" state, if the field counter reaches a count of 15, the state variable transitions to the "check_vcr" state. The "main" state process allows the phase error accumulator to operate for 16 fields, and since the phase error accumulator always starts from a reset, the value in the accumulator at the end of the 16 fields represents the derivative of the accumulator over a 16 field period. In the "check_vcr" state, the sign of the current 16 field derivative is compared to the previous 16 field derivative. If the signs are different, the 3-bit Sign Counter is incremented, and once the maximum count of 7 is reached, the only way to clear the Sign Counter is by a reset. Next, the accumulator value is compared to a threshold value VCR_THRES, if the accumulator value is greater than VCR_THRES, then a "hit" is registered by incrementing the 5-bit Hit Counter; otherwise, the Hit Counter is decremented. The Hit Counter does not rollover at its maximum or minimum values. The values in the Hit Counter and the Sign Counter are used to determine if the input signal is a VCR playback. First, the Sign Counter has to be less than or equal to one for VCR detection because when the circuit is initialized, there can be at least one sign change if the accumulator sign is different from the initialization value after the first 16 fields. Since VCR playback was seen to have monotonic phase error accumulation, the sign will not change after the first sign comparison. If the input signal is severely ghosted, then the Sign Counter will be incremented up to its maximum value of 7, which prevents false VCR detection.

The next requirement for VCR detection is a value in the Hit Counter of twenty five or greater. The value of twenty five is somewhat arbitrary, but is based on an observation of the phase error accumulator slope being less than the VCR_THRESH value under VCR playback conditions, which occurred infrequently under experimental conditions. The Hit Counter, under VCR playback conditions, will count to its maximum value of 31 and stay as long the 16 field accumulator value is greater than VCR_THRESH. If an accumulator value compares less than VCR_THRESH, the Hit Counter will decrement. Two situations are possible. First, the VCR playback could have been switched off without a channel change being detected, in which case the input signal is a normal signal, and the Hit Counter should count down to below 25, which is interpreted automatically as a channel change. The second

situation is that some anomaly occurred in the VCR playback signal that caused a low phase error accumulation for one or several 16 field derivative(s). As long as the Hit Counter stays above 25, then a false channel change will not occur. This gives the circuit some robustness against spurious VCR and channel change detections caused by unpredictable VCR playback conditions. As with the Design I Time-Based Error Analyzer, the VCR_FLAG variable was used to prevent the signaling of VCR_DET repeatedly. The flowchart of Figure 6-4 describes the interpretation of the VCR_FLAG.

The prototyping results of the circuit of Figure 6-3 were extremely satisfactory. Over 95% of channel changes were detected, and no false channel changes were detected even under some severe ghosting conditions, and VCR detection was far more reliable than the Design I method of VCR detection. The entire design, development, and prototyping of the Design II Time-Based Error Analyzer took approximately four weeks to complete. The next stages of development was to retarget the design from Xilinx XC4000 technology to Mitsubishi cells and development of functional and manufacturing test vectors.

Chapter 7

Design Retargeting and Test Vector Generation

Design Retargeting

The performance of the Design II Time-Based Error Analyzer was determined to be robust enough for implementation in the Mixed-Signal IC. The process of translating the prototyped design to a Mitsubishi gate level netlist required a significant effort. First, the VHDL source file "vcr_det.vhd", which is described in Chapter 6, included some VHDL code that was germane to the Rapid Prototyping System. This required that the prototype specific code be separated out from the Time-Based Error Analyzer code. The result of this process was the hierarchical design depicted in Figure 7-1.

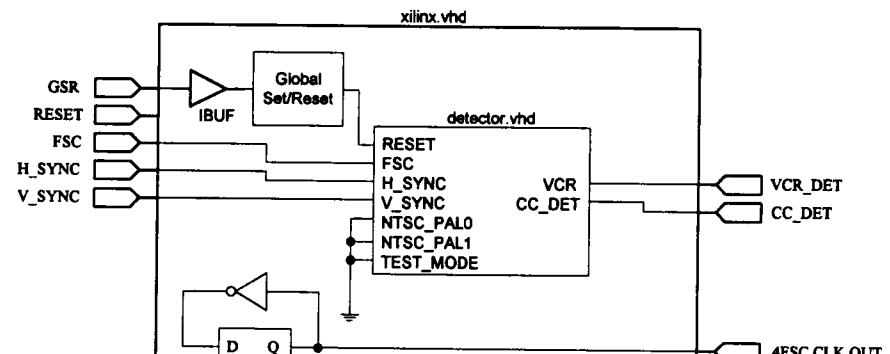

Figure 7-1: Hierarchy

The ghost cancellation system built on the Rapid Prototyping System consisted of multiple Xilinx XC4005 FPGA's and daughter boards. The digital filter board was found to be extremely sensitive to the clock duty cycle. Since the ghost canceler prototype was synchronous, low clock skew between the different XC4005 parts was required. The solution to both clock duty cycle and clock skew was to divide an $8f_{sc}$ clock, which was available from the Analog daughter board, by using a toggle flip-flop. The resultant $4f_{sc}$ clock was used to drive a Fast CMOS Buffer/Clock Driver part from Integrated Device

Technology, Inc. (IDT49FCT805), which can drive up to 10 low skew clock lines from a single clock source. The low skew clock lines were used to drive each of the Xilinx XC4005 parts on the Rapid Prototyping System as well as the filter daughter board. The Global Set/Reset block shown in Figure 7-1 is a Xilinx macro that is instantiated to connect a circuit's reset to the Xilinx part's Global Set/Reset resource, which frees routing resources of the Xilinx part. By adding the new level of hierarchy, the "detector.vhd" VHDL source became a technology independent design, which was described at the Register Transfer Level (RTL), and all the technology dependent components were contained in a single level of hierarchy in the VHDL source "xilinx.vhd". The complete VHDL source files for "xilinx.vhd" and "detector.vhd" are included in the Appendix.

Once the new hierarchical design of Figure 7-1, was verified functionally on the Rapid Prototyping System, the process of retargeting the design from a Xilinx XC4005 netlist to a Mitsubishi gate level netlist was begun. A common method of synthesizing a design in Synopsys is to use compiler scripts. During the prototyping phase, a script was developed for synthesizing the design to Xilinx XC4005 technology. Once a working script was developed, modifications to the design were performed by changing the VHDL source, and the same script was used to synthesize the design. The Xilinx synthesis script is shown below:

```
1.  design = xilinx
2.  read -format vhdl design + ".vhd"
3.  set_attribute design "part" -type string "4005pg156-5"
4.  remove_constraint -all
5.  remove_clock -all
6.  set_operating_conditions WCCOM
7.  set_wire_load "4005-5_avg"
8.  set_port_is_pad "*"
9.  set_attribute "FSC_8"       "pad_location"  -type string "B14"
10. set_attribute "FSC_OUT"     "pad_location"  -type string "C15"
11. set_attribute "FSC"         "pad_location"  -type string "T15"
12. set_attribute "H_SYNC"      "pad_location"  -type string "A1"
13. set_attribute "V_SYNC"      "pad_location"  -type string "A11"
14. set_attribute "CC_DET"      "pad_location"  -type string "L16"
15. set_attribute "VCR"         "pad_location"  -type string "K2"
16. set_pad_type -no_clock "FSC_8"
17. set_pad_type -clock "FSC"
18. uniquify
19. insert_pads
20. dont_touch_network "GSR"
```

21. compile -map_effort med -ungroup_all
22. replace_fpga
23. disconnect_net find(net, all_connected(RESET)) -all
24. xnfout_dont_change_to_dff = 1
25. write -format xnf -hier -output design + ".sxnf"

Lines 1 - 7 read in the analyzed VHDL circuit description and set several compiler environment variables that are used in the synthesis process. Lines 8 - 17 control the placement and type of pads used on the Rapid Prototyping Board. Since the pins of the Xilinx XC4005 part were wirewrapped to the Analog Board, lines 9 through 15 prevent the tools from reassigning pins. Lines 18 tells the Synopsys Compiler how to treat multiple instances of components with the same entity name. Line 19 inserts Xilinx XC4000 pads and buffers to the design database. Lines 20, 23, and 24 perform the operation of connecting the Global Set/Reset block to the RESET line in the design database. Line 25 writes the design database, which is in a Synopsys database, to a Xilinx netlist, which can be processed with the Xilinx place and route tools. The compiler script for the synthesis of the "detector.vhd" source to Mitsubishi technology was more complicated. The Mitsubishi compiler script is listed below:

1. design = detector
2. read -format vhdl design + ".vhd"
3. remove_constraint -all
4. remove_clock -all
5. set_operating_conditions -library "M6005X_6X_5V" "TYPICAL"
6. set_wire_load "M60050" -library "M6005X_6X_5V"
7. include dontuse.list
8. uniquify
9. set_max_area 3000
10. set_drive 10.0 all_inputs()
11. compile -map_effort low
12. set_scan_style multiplexed_flip_flop
13. set_test_methodology full_scan
14. set_signal_type "test_scan_in" H_SYNC
15. set_signal_type "test_scan_out" VCR_OUT
16. set_signal_type "test_scan_enable" TEST_MODE
17. check_test
18. insert_test -scan_chains 1
19. create_clock -name "FSC" -period 70 -waveform {"0" "35"} {"FSC"}
20. set_fix_hold find(clock, "FSC")
21. set_max_fanout 10.0 "FSC"
22. compile -map_effort high
23. set_port_is_pad "*"
24. set_pad_type -clock "FSC"
25. insert_pads

26. create_test_patterns -output detector.vdb -compaction_effort high check_contention_float true -backtrack_effort low -random_pattern_failure_limit 64 -sample 100
27. write_test -input detector.vdb -format verilog -output verilog_vectors
28. write -format vhdl -hierarchy -output detector_struct.vhd {"detector.db:detector"}

Lines 1 - 6, as in the Xilinx script, read the design "detector.vhd" into the Synopsys Design Compiler tool, and set up several synthesis environment variables. Line 7 reads a separate file called "dontuse.list", which flags all the cells in the current library that are not available for use. Since the digital blocks of the Mixed-Signal IC were being implemented in a 1.0 μm BiCMOS process for which no cell library existed, all cells used in the design had to be generated; however, the generated cells were not available in Synopsys format. The 1.0 μm Gate Array library was determined to have similar timing, fanout, etc. characteristics to the 1.0 μm BiCMOS cells, and a Synopsys cell library already existed. Mitsubishi provided PCEC with a list of 1.0 μm Gate Array cells that were 1.0 μm BiCMOS equivalents, which were to be used as the primitives for the implementation of the Time-Based Error Analyzer circuit. The 1.0 μm BiCMOS cell list is included in Table 7-1.

The uniquify command on Line 8 performs the same task as in the Xilinx script. Line 9 sets the maximum area for the compiled design. The value of 3000 here is measured in a unit called grids, and was chosen somewhat arbitrarily. In order to get minimum area, this value should have been set to zero. Line 10 sets the maximum fanout per cell to 10 loads, which performs automatic buffering in the first compile statement in Line 11. Lines 12 - 18 setup and insert scan test circuitry, which provide manufacturing test capability to the circuit. Lines 26 and 27 generate test patterns to test for stuck at faults using the inserted scan test circuitry. The scan test methodology is a well developed means of providing both controllability and observability to all the nodes in the circuit. The details of the scan test methodology will be discussed later in this chapter.

Lines 19 - 22 setup the timing constraints and clock fanout for the design, and recompile the design, with scan test circuitry and buffering included, for maximum timing performance. Lines 23 through 25 insert the pads and pad buffers into the design database. Line 28 writes the compiled design to a structural VHDL netlist, which is a gate level netlist consisting of only the Mitsubishi cells listed in

Table 7-1. Once the Mitsubishi script file was successfully executed, the two files, "verilog_vectors" and "detector_struct.vhd" completed Mitsubishi's requirements for manufacturing test vectors and a gate level netlist of the circuit respectively. However, the scan vectors generated by the Synopsys tools are not sufficient for functional testing of the design, and functional test vector generation was required.

Table 7-1: 1.0 um BiCMOS Cell List

Function	1.0 μm Gate Array Cell Name	1.0 μm Bi-CMOS Cell Name
2 INPUT AND	AN2S	AND2X11
3 INPUT AND	AN3S	AND3X11
CLOCK LINE DRIVER (X2 DRIVE)	K1NS	BUFXX11
CLOCK LINE DRIVER (X4 DRIVE)	K1NQ	BUFXX14
3 STATE DRIVER "L" ENABLE	KL1W	CBFX011
3 STATE DRIVER "L" ENABLE (X2 DRIVE)	K1ZW	CBFX012
Positive edge DFF with SET and RESET	FDES	DFAX441
Positive edge DFF with RESET	FDDS	DFRX241
Positive edge DFF with SET	FDCS	DFSX241
Negative edge DFF with SET and RESET	FD4S	DFAX031
Negative edge DFF with RESET	FD3S	DFRX031
Negative edge DFF with SET	FD2S	DFSX031
NAND type RS LATCH	FLAS	RSLB011
EXCLUSIVE NOR	XNOS	ENR2X11
EXCLUSIVE OR	XORS	EOR2X11
2 INPUT NAND	NO2S	NAD2X11
3 INPUT NAND	NO3S	NAD3X11
4 INPUT NAND	NO4S	NAD4X11
5 INPUT NAND	NO5S	NAD5X11
2 INPUT NOR	RO2S	NOR2X11
3 INPUT NOR	RO3S	NOR3X11
4 INPUT NOR	RO4S	NOR4X11
INVERTER	V01S	NOTXX11
INVERTER (X2 DRIVE)	V01W	NOTXX12
INVERTER (X4 DRIVE)	VO1Q	NOTXX14
2 INPUT OR	OR2S	OR2XX11
3 INPUT OR	OR3S	OR3XX11
4 INPUT OR	OR4S	OR4XX11
HIGH LEVEL GENERATOR	ZHHH	VDDXXX1
LOW LEVEL GENERATOR	ZLLL	VSSXXX1

Test Vector Generation

Manufacturing tests require a test strategy that has a high "stuck at" fault coverage. The test methodology that Synopsys has adopted in their Test Compiler tool is scan test. While Test Compiler provides a variety of scan test implementations, one of the most widely prevalent scan test style is the multiplexed flip-flop, for which a typical implementation is shown in Figure 7-2.

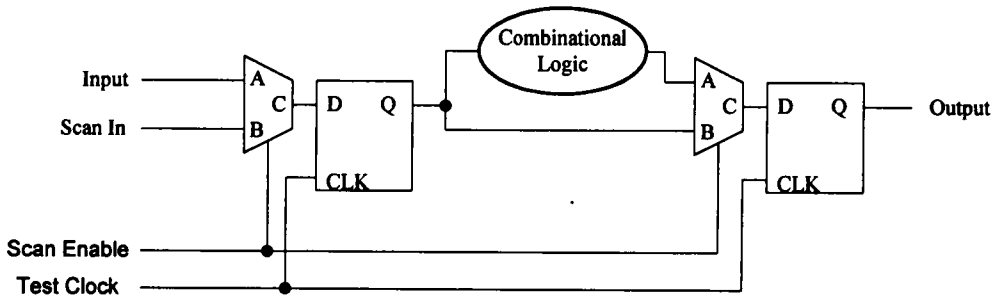

Figure 7-2: Multiplexed Flip Flop Scan Implementation

The operation of the multiplexed flip-flop scan test consists of asserting the Scan Enable line, which transforms the circuit into one long shift register consisting of all the flip-flops in the design. The Scan In line is used to shift in test patterns into the shift pipeline. Once all the flip-flops are loaded with the test pattern, the Scan Enable line is deasserted, and the circuit is clocked once. By controlling the Input line(s) and with the knowledge of the logic function contained within the Combinational Logic block, the value clocked into each flip-flop of the design is predictable. The Scan Enable line is then asserted, and a new test pattern is shifted into the circuit, while at the same time the output vectors are shifted out on the Output line. The scan test methodology has both controllability and observablity required for manufacturing tests, but the generation of many test patterns is required to achieve high fault coverage. For even small designs such as the Time-Based Error Analyzer, test pattern generation is a job most suited to computers, and the Synopsys Test Compiler provides an Automatic Test Pattern Generation (ATPG) tool. With just a few lines in the Mitsubishi compiler script, the scan test implementation of Figure 7-2 was inserted into the design, and the ATPG tool generated a set of test vectors that had over 98 % fault coverage. In Design I,

generation of a manufacturing test strategy took approximately 6 weeks, greatly increased the size of the design, and fault coverage was estimated to be very low. Compared to the Synopsys Test Compiler scan test strategy, virtually no additional time was spent implementing hardware, the size increase was approximately a 20 % larger area per flip-flop, and fault coverage was extremely high.

Functional vectors were required by Mitsubishi in order to check pre-layout and post-layout timing. The job of pre-layout and post-layout timing evaluation is generally the responsibility of the customer and not the vendor. Since the gate level netlist that was provided to Mitsubishi was generated from a 1.0 μm Gate Array library, which was to be translated to the 1.0 μm BiCMOS cell library with more accurate timing, the responsibility of checking pre-layout to post-layout timing was assumed by Mitsubishi. The functional vectors were generated using the "detector.vhd" block instantiated in a testbench VHDL file called "f_vect.vhd", which is included in the Appendix. The Synopsys VSS VHDL simulator was used to generate a Wave Intermediate Format (WIF) file, which was translated to Mitsubishi Intermediate Format (MIF). The resulting .MIF file consisted of input stimulus and expected output vectors, which could be used by the Mitsubishi CAD system to verify layout timing of the Time-Based Error Analyzer circuit.

The problem with functional test vector generation was that a small number of inputs had to be manipulated over long periods just to produce pulses on the outputs of the Time-Based Error Analyzer. At $4f_{sc}$ sampling rates, a single field of video requires 910 x 262.5 = 238875 vectors, and multiple fields of simulation are required to produce any change in the outputs CC_DET and VCR_DET, which translates to a huge number of test vectors if full field simulation is attempted. Therefore, a set of vectors that provided significant functional verification in as few vectors as possible was generated. The functional test strategy was to generate a pulse on the CC_DET output of the Time-Based Error Analyzer circuit. To accomplish this, the conditions for a channel change detection had to be created by the functional vectors. From the discussion of the channel change detection algorithm in Chapter 6, recall that a channel change condition occurs when the derivative of the phase error accumulator between two successive fields is greater than some threshold (CC_THRESH). Once this conditions occurs, the circuit transitions to a delay state and waits for 16 fields to pass before asserting the CC_DET output. Therefore, to create the channel change

condition, the functional vector testbench must initialize the Time-Based Error Analyzer via the RESET line, accumulate enough phase error in one field to generate a channel change condition, and simulate 16 fields. Many of the same circuit elements are used to detect VCR playback conditions, but an extraordinary number of vectors were required. Therefore, in the interest of providing maximum functionality test for the minimum number of vectors, the functional test of VCR playback detection was eliminated.

The simulation of the functional vector stimulus file f_vect.vhd is shown graphically in Figure 7-3 through Figure 7-5. In Figure 7-3, the circuit is initialized by the active low assertion of the RESET line, which causes the Time-Based Error Analyzer circuit to startup in the "main" state. Since phase error is accumulated only in lines 64 through 192 of the video, the number of simulation vectors are reduced by signaling falling edges on H_SYNC every other clock cycle, which enables the line counter once for each falling edge. In Figure 7-3, the number of accelerated H_SYNC falling edges is sixty two, which reduces the number of vectors by over 50,000.

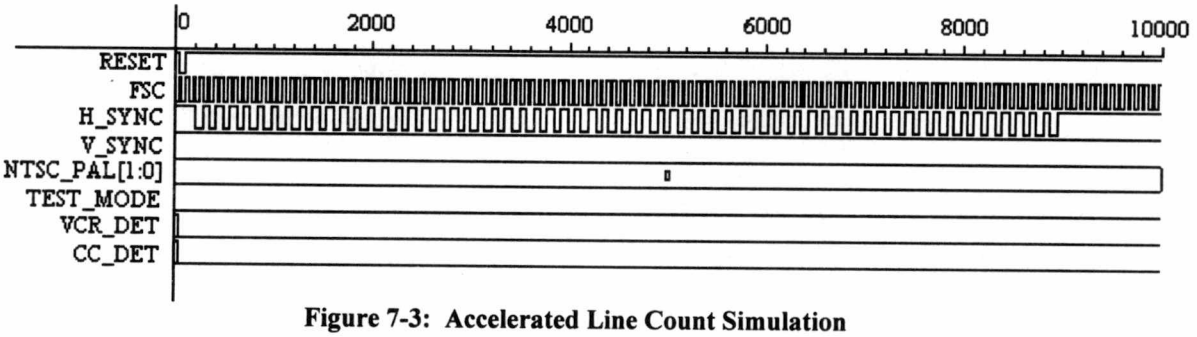

Figure 7-3: Accelerated Line Count Simulation

The majority of the test vectors are shown in Figure 7-4. The time between the falling edges on the H_SYNC line is exactly 911 * $4f_{sc}$, which will cause the accumulation of a -1 every line. The accumulation of phase error will be large enough to be recognized as a channel change condition, which will cause a state transition to the "cc_delay" state once a falling edge is detected on V_SYNC. Once again, since the accumulation process occurs only between lines 64 and 192, the V_SYNC line can be asserted at any point after 192 lines to achieve a savings of over 60,000 vectors.

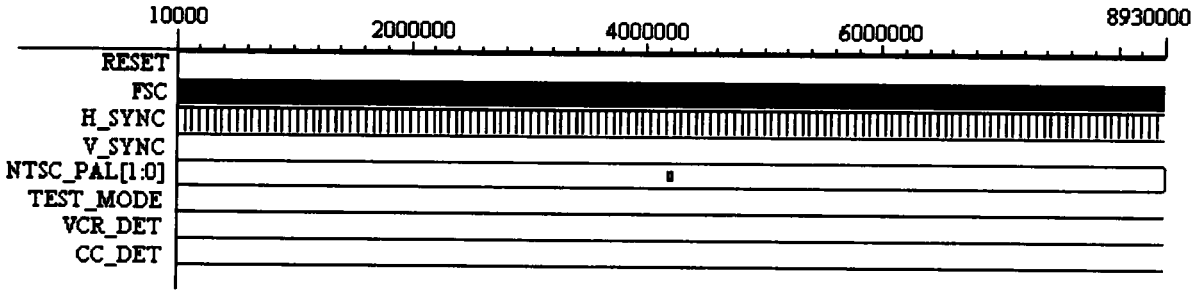

Figure 7-4: Phase Error Accumulator Simulation

After the falling edge of H_SYNC in Figure 7-5, the first falling edge on V_SYNC causes a state transition from the state "main" to "cc_delay". Once in "cc_delay", sixteen falling edges must occur on the V_SYNC line to cause a state transition out of "cc_delay". The falling edges on V_SYNC are produced in rapid succession by the testbench because the only operating component in state "cc_delay" is the field counter. Once the field counter reaches a count of fifteen, the Time-Based Error Analyzer is reinitialized, the CC_DET output is asserted, and a state transition back to "main" occurs. The CC_DET assertion is the desired response; therefore, the functional test is complete.

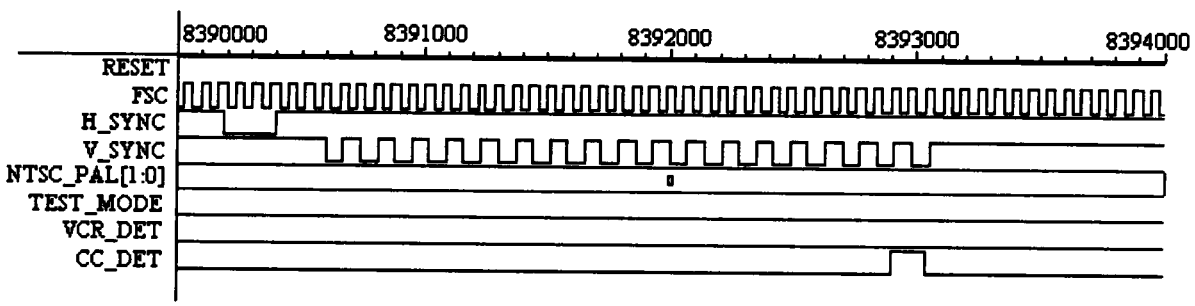

Figure 7-5: State "cc_delay" Simulation

The functional test vectors provided in the f_vect.vhd testbench exercise the following components of the Time-Based Error Analyzer (Figure 6-3):

- Input ports RESET, H_SYNC, V_SYNC, FSC ($4f_{sc}$ clock), NTSC_PAL[1:0] = "00"
- Output port CC_DET
- H_SYNC and V_SYNC edge detectors
- 10-bit Down Counter, 9-bit Line Counter, and 4-bit Field Counter operation
- Counter Decode block
- 16-bit Accumulator and Accumulator register operation

- "main" and "cc_delay" states of state machine

The following circuit functionality is not exercised by the functional test vectors:

- 5-bit Hit Counter, 3-bit Sign Counter
- Output port VCR_DET
- "check_vcr" state of state machine

The functional test vectors totaled to just under 120,000 for the functional test of a channel change. In order to check the VCR detection functionality, approximately 16 fields of vectors similar in length to the one field of video that exercised the channel change detection functionality would be required, which totals to about 2 million vectors. Therefore, the functional vectors provided by the "f_vect.vhd" testbench were a good compromise between functional coverage and total number of vectors. Even with the reduced number of vectors, the full timing gate level simulation took approximately 6 hours to run according to the Mitsubishi engineers.

Chapter 8

IC Performance

The Mitsubishi gate level netlist produced by synthesizing the "detector.vhd" behavioral description of the Time-Based Error Analyzer, along with the manufacturing and functional test vectors, was released to Mitsubishi in late September, 1994. The gate level netlist was translated by Mitsubishi engineers to a Verilog netlist with 1.0 μm BiCMOS cells, and Cadence Ensemble was used for automatic layout of the design. With all steps in this process being automated, the risk of translation error was minimized. Some minor changes were required in the I/O configuration, but no changes were required that necessitated re-releasing either the design or the test vectors.

In preparation for testing of the Mixed-Signal IC, an Analog Test Board was developed. The Analog Test Board consisted of the Mitsubishi Mixed-Signal IC, the Philips Analog-to-Digital Converter, the input anti-aliasing filter, the output smoothing filter, crystals, buffers, etc. Functional testing of the Time-Based Error Analyzer block inside the Mixed-Signal IC was limited to two simple tests. Channel change operation was checked by connecting the Analog Test Board to a standard NTSC source (a VCR tuner), changing channels, and observing the CC_DET line. VCR detection was accomplished in the same manner by playing a video tape on the VCR source and monitoring the VCR_DET line.

First silicon for the Mixed-Signal IC was received in late April, 1995. The performance of the Mitsubishi implementation of the Time-Based Error Analyzer was found to be equivalent to that of the Xilinx prototype, indicating that no translation or layout errors had occurred. Figure 8-1 is the Mixed-Signal IC layout. The layout of the Time-Based Error Analyzer is the large section of rows toward the top of Figure 8-1, which consumes approximately half of the chip area excluding the pad frame. The relative size of the layout of the Time-Based Error Analyzer is attributable to two facts:

1. The Time-Based Error Analyzer cells were implemented in 1.0 μm CMOS as opposed to 0.8 μm for the rest of the circuit.
2. Automatic layout generation is not as area efficient as hand layout.

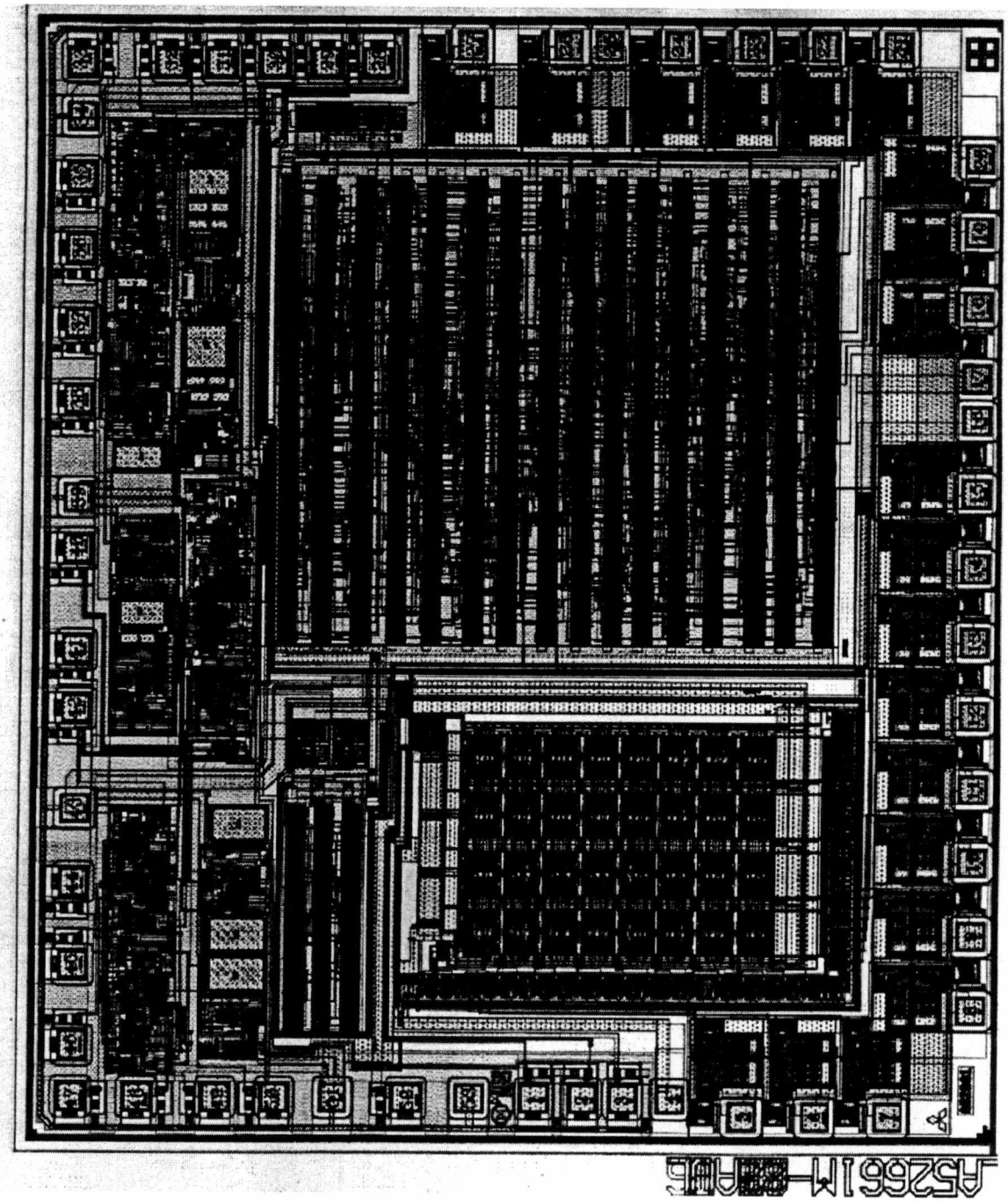

Figure 8-1: Mixed-Signal IC Layout

The size of the Time-Based Error Analyzer became significant when the signal path of video signal at various points on the Analog Test Board were examined. Revision 1.0 of the IC revealed nearly 200 mV_{p-p} of a 28 MHz sinusoid riding on the video signal. The 28 MHz sinusoid was determined to be a strong harmonic of the $4f_{sc}$ clock that was coupling into the analog signal path. Investigation by Mitsubishi engineers revealed that at least part of the problem was that the Digital-to-Analog converter inside the IC was grounded to the analog ground. Once the D/A converter ground was switched to the digital ground, the 28 MHz sinusoid was reduced from 200 mV_{p-p} to approximately 50 mV_{p-p}. The remaining clock cross-talk was assumed to be the result of a large switching current generated by the Time-Based Error Analyzer circuit inside the IC. In addition, the Analog Test Board had a high enough ground impedance to cause the 50 mV_{p-p} voltage to be generated from the switching current across the ground impedance. A noisy isolation barrier around the Time-Based Error Analyzer was also suspected of contributing to the $4f_{sc}$ cross-talk, as well as different transistor geometries between the digital and analog sections. While the clock cross-talk into the analog signal path was not visible when viewed on a television monitor, there was a problem when the resolution of the system was examined. The ghost cancellation system is 8-bit, which requires that for a 2.0 V A/D range, the maximum allowable signal corruption (x) be less than:

$$20 \cdot \log \frac{1}{256} = 48dB \geq 20 \cdot \log \frac{x}{2.0V_{p-p}}$$

$$x \leq 7.8125\,mV$$

at the input to the A/D converter. A considerable amount of effort was expended to meet this system requirement by reducing ground impedance and bandlimiting selected nodes, which was an acceptable compromise for improving the system performance at a reasonable cost. Ideally, the elimination of the cross-talk at the source was the goal; however, completely eliminating digital cross-talk in mixed-mode IC's appears to be a subject worthy of additional research.

Chapter 9

Summary and Conclusions

The problems presented in *The Design and Development of a Digital Time-Based Error Analyzer Circuit for Implementation in a Mixed-Signal Integrated Circuit* were:

1. Design and Development of a circuit to detect channel changes and VCR playback conditions on a Composite Video Baseband Signal.
2. Implementation of the circuit in the Mixed-Signal IC developed by Mitsubishi Electronics, Inc.

The development of a satisfactory solution to the problem of detecting channel changes and VCR playback on a Composite Video Baseband Signal was difficult because of ambiguity in the problem definition. The source of ambiguity in the problem definition stemmed from the fact that the range of acceptable ghosting conditions to the ghost canceler system was never specified. From the discussion of the effects of ghosting in Chapter 3, and considering the system architecture described in Chapter 2, one of the primary limitations on the proposed ghost cancellation system is sync processor performance. Sync failure during ghosting conditions caused a great deal of difficulty in evaluating the performance of the two solutions presented in Chapters 5 and 6. However, the application of a systematic problem solving approach led to the discovery of weaknesses in the Design I circuit, which led to the development of a more robust Design II circuit.

The acquisition of the Synopsys toolset was the solution to the problem of implementing the circuit in the Mixed-Signal IC. The design time for the development of the Design II circuit was less than half of Design I, which indicates that the Synopsys tools speed design time by more than a factor of two. Also, implementation of a test strategy in the Design II circuit was far superior to the Design I circuit because Synopsys Test Compiler provided much higher fault coverage and accelerated the process of test vector generation. Therefore, the Synopsys tools were found to impact significantly both the design time and the design quality.

From a system standpoint, the implementation of channel change and VCR playback detection in the Mixed-Signal IC is not the best solution to the problem. The analog signal path corruption caused by the digital section coupling into the analog section and sync processor limitations under ghosting conditions are sufficient reasons to justify pursuing other methods of implementing channel change and VCR detection. One method of channel change detection that has been examined uses the GCR as a synchronization reference, and channel changes are detected when synchronization with the GCR is lost. Channel change detection using GCR synchronization is advantageous because the technique used to synchronize to the GCR is not susceptible to ghosting conditions. VCR playback detection is possible by analyzing the bandwidth characteristic of the CVBS source, and several methods of bandwidth analysis have been proposed for investigation.

While the Time-Based Error Analyzer circuit may not be best solution to the problem of detecting channel changes and VCR playback conditions on a ghosted CVBS source, it is a good solution as long as the sync processor functions properly. Therefore, an improved problem solution resides at the system performance level. For the first generation ghost canceler, the system performance limitations are considered acceptable. For future generation ghost cancelers, the knowledge gained from the first generation will be used to improve system performance, which is, in essence, the systematic approach to problem solving being applied at the system level. However, the time for a single iteration through the steps described in Chapter 1 is much longer.

List of References

List of References

1. Clapp, Richard G. and Joseph F. Fisher. "Waveforms and Spectra of Composite Video Signals." In Benson, ed., Television Engineering Handbook. New York: McGraw-Hill Book Company,1986. pp. 5.1 - 5.36.

2. GinsBurg, Charles P. et al. "Video Tape Recording." In Benson, ed., Television Engineering Handbook. New York: McGraw-Hill Book Company, 1986. pp. 15.1 - 15.139.

3. Herman, Stephen. "The Complete Ghost Canceler." Presentation to the ICCE. Philips Laboratories, North American Philips. June, 1993.

4. Koo, David. "Properties and Applications of the New Ghost Cancelation Reference Signal." ICCE June, 1993.

5. Shiraishi, Yuma. "History of Home Videotape Recorder Development." SMPTE Journal. December, 1985. pp. 1257-1263.

6. van Zon, Kees et al. "Echo Cancellation Project Phase I Results." VLSI Systems Dept., Philips Laboratories, North American Philips. July, 1990.

7. "Composite Analog Video Signal - NTSC for Studio Applications." ANSI/SMPTE 170M-1994. Oct. 19, 1994.

Appendix

main2.vhd

```
entity main2 is
port(
--  ********** Inputs **********
    SYS_CLK          :in bit;
    RESET            :in bit;
    VCR_COND         :in bit;
    VER_COND         :in bit;
    F_CNT            :in bit_vector (4 downto 0);
    FOUND_MODE       :in bit;
    DONE_COUNT       :in bit;
--  ********** Outputs **********
    STATE            :out bit_vector(3 downto 0);
    CC               :out bit;
    VCR              :out bit;
    FC_RST           :out bit;
    FIND_MODE        :out bit;
    COUNT_LT_GT      :out bit);
end main2;

architecture behavior of main2 is

    constant STARTUP          :bit_vector(3 downto 0) := B"0000";
    constant CC_INTERVAL      :bit_vector(3 downto 0) := B"0001";
    constant SET_CC           :bit_vector(3 downto 0) := B"0010";
    constant MODE             :bit_vector(3 downto 0) := B"0011";
    constant INIT_VERT        :bit_vector(3 downto 0) := B"0110";
    constant INIT_DONE        :bit_vector(3 downto 0) := B"1010";
    constant COUNT            :bit_vector(3 downto 0) := B"0100";
    constant MAIN_BRANCH      :bit_vector(3 downto 0) := B"0101";
    constant VCR_COUNT        :bit_vector(3 downto 0) := B"0111";
    constant COMP_VCR         :bit_vector(3 downto 0) := B"1000";
    constant SET_VCR          :bit_vector(3 downto 0) := B"1001";
    constant NORM_COUNT       :bit_vector(3 downto 0) := B"1011";
    constant COMP_NORM        :bit_vector(3 downto 0) := B"1100";
    constant SET_NORM         :bit_vector(3 downto 0) := B"1101";

begin


    process(SYS_CLK, VER_COND, VCR_COND, F_CNT, FOUND_MODE, DONE_COUNT)

    variable  p_state         :bit_vector(3 downto 0);
    variable  VCR_FLAG        :bit;
    begin

    if( (SYS_CLK = '1') and (SYS_CLK'event) ) then
    STATE <= p_state;
    if(RESET = '0') then
       p_state := STARTUP;
    else
```

```
case p_state is

when STARTUP =>

VCR_FLAG        := '0';
CC              <= '0';
VCR             <= '0';
FC_RST          <= '0';
FIND_MODE       <= '0';
COUNT_LT_GT     <= '0';

p_state := CC_INTERVAL;

when CC_INTERVAL =>

CC              <= '0';
VCR             <= '0';
FC_RST          <= '1';
FIND_MODE       <= '0';
COUNT_LT_GT     <= '0';

if(F_CNT = B"01111") then
   p_state := SET_CC;
end if;

when SET_CC =>

VCR_FLAG        := '0';
CC              <= '1';
VCR             <= '0';
FC_RST          <= '0';
FIND_MODE       <= '0';
COUNT_LT_GT     <= '0';

p_state := MODE;

when MODE=>

CC              <= '0';
VCR             <= '0';
FC_RST          <= '0';
FIND_MODE       <= '1';
COUNT_LT_GT     <= '0';

if(FOUND_MODE = '1') then
   p_state := INIT_VERT;
end if;
```

```
    when INIT_VERT =>

    CC              <= '0';
    VCR             <= '0';
    FC_RST          <= '0';
    FIND_MODE       <= '0';
    COUNT_LT_GT     <= '1';

    if(DONE_COUNT = '1') then
       p_state := INIT_DONE;
    end if;

    when INIT_DONE =>

    CC              <= '0';
    VCR             <= '0';
    FC_RST          <= '0';
    FIND_MODE       <= '0';
    COUNT_LT_GT     <= '0';

    if(DONE_COUNT = '0') then
       p_state := COUNT;
    end if;

    when COUNT =>

    CC              <= '0';
    VCR             <= '0';
    FC_RST          <= '0';
    FIND_MODE       <= '0';
    COUNT_LT_GT     <= '1';

    if(DONE_COUNT = '1') then
       p_state := MAIN_BRANCH;
    end if;

    when MAIN_BRANCH=>

    CC              <= '0';
    VCR             <= '0';
    FC_RST          <= '0';
    FIND_MODE       <= '0';
    COUNT_LT_GT     <= '0';
if(DONE_COUNT = '0') then
    if(VER_COND = '1') then
       p_state := STARTUP;
    elsif(VCR_COND = '1') then
       p_state := VCR_COUNT;
    else
       p_state := NORM_COUNT;
    end if;
end if;
```

```
when VCR_COUNT =>

CC              <= '0';
VCR             <= '0';
FC_RST          <= '1';
FIND_MODE       <= '0';
COUNT_LT_GT     <= '1';

if(DONE_COUNT = '1') then
   p_state := COMP_VCR;
end if;

when COMP_VCR =>

CC              <= '0';
VCR             <= '0';
FC_RST          <= '1';
FIND_MODE       <= '0';
COUNT_LT_GT     <= '0';

if(DONE_COUNT = '0') then
   if(VER_COND = '1') then
      p_state := MAIN_BRANCH;
   elsif(VCR_COND = '1' and F_CNT = B"11111"
            and VCR_FLAG = '0') then
      p_state := SET_VCR;
   elsif(VCR_COND = '1' and F_CNT = B"11111"
            and VCR_FLAG = '1') then
      p_state := COUNT;
   elsif(VCR_COND = '1') then
      p_state := VCR_COUNT;
   else
      p_state := MAIN_BRANCH;
   end if;
end if;

when SET_VCR =>

VCR_FLAG        := '1';
CC              <= '1';
VCR             <= '1';
FC_RST          <= '0';
FIND_MODE       <= '0';
COUNT_LT_GT     <= '0';

p_state := COUNT;
```

```
        when NORM_COUNT =>

        CC              <= '0';
        VCR             <= '0';
        FC_RST          <= '1';
        FIND_MODE       <= '0';
        COUNT_LT_GT     <= '1';

        if(DONE_COUNT = '1') then
           p_state := COMP_NORM;
        end if;

        when COMP_NORM =>

        CC              <= '0';
        VCR             <= '0';
        FC_RST          <= '1';
        FIND_MODE       <= '0';
        COUNT_LT_GT     <= '0';

        if(DONE_COUNT = '0') then
           if(VER_COND = '1') then
              p_state := MAIN_BRANCH;
           elsif(VCR_COND = '0' and F_CNT = B"01111"
                        and VCR_FLAG = '0') then
              p_state := SET_NORM;
           elsif(VCR_COND = '0' and F_CNT = B"01111"
                        and VCR_FLAG = '1') then
              p_state := STARTUP;
           elsif(VCR_COND = '0') then
              p_state := NORM_COUNT;
           else
              p_state := MAIN_BRANCH;
           end if;
        end if;

        when SET_NORM =>

        VCR_FLAG        := '0';
        CC              <= '0';
        VCR             <= '0';
        FC_RST          <= '0';
        FIND_MODE       <= '0';
        COUNT_LT_GT     <= '0';

        p_state      := COUNT;
        end case;
        end if;
        end if;
        end process;

end behavior;
```

rsrc_ctl.vhd

```
entity rsrc_ctl is
port(
--  ********** Inputs **********
    SYS_CLK          :in bit;
    RESET            :in bit;
    HSYNC            :in bit;
    FIND_MODE        :in bit;
    CNT_LT_GT        :in bit;
    FLAG_16_LINES    :in bit;
    FLAG_11_HITS     :in bit;
    FLAG_1_FIELD     :in bit;
    GT               :in bit;
    LT               :in bit;
--  ********** Outputs **********
    STATE            :out bit_vector(4 downto 0);
    HIT_CNT_RST      :out bit;
    HIT_CNT_CE       :out bit;
    HOLD_LOAD        :out bit;
    NEW_LOAD         :out bit;
    L_CNT_LOAD       :out bit;
    MUX_SRC_SEL      :out bit;
    ADDER_CI         :out bit;
    TC_RST           :out bit;
    LC_RST           :out bit;
    FC_RST           :out bit;
    LT_CNT_CE        :out bit;
    GT_CNT_CE        :out bit;
    LT_GT_RST        :out bit;
    FOUND_MODE       :out bit;
    DONE_COUNT       :out bit);
end rsrc_ctl;

architecture behavior of rsrc_ctl is

   constant WAIT_CMD          :bit_vector(4 downto 0) := B"00000";
   constant GET_HOLD          :bit_vector(4 downto 0) := B"00001";
   constant TC_RESET          :bit_vector(4 downto 0) := B"00010";
   constant HOLD_EN           :bit_vector(4 downto 0) := B"00011";
   constant LOAD_HOLD         :bit_vector(4 downto 0) := B"00100";
   constant COMP_EN           :bit_vector(4 downto 0) := B"00101";
   constant LOAD_TC           :bit_vector(4 downto 0) := B"00110";
   constant ONE_WAIT          :bit_vector(4 downto 0) := B"00111";
   constant CHECK_HIT         :bit_vector(4 downto 0) := B"01000";
   constant HIT_YES           :bit_vector(4 downto 0) := B"01001";
   constant HIT_NO            :bit_vector(4 downto 0) := B"01010";
   constant DONE_16           :bit_vector(4 downto 0) := B"01011";
   constant SIG_DONE          :bit_vector(4 downto 0) := B"01100";
   constant BEGIN_CNT         :bit_vector(4 downto 0) := B"01101";
   constant LOAD_VERT         :bit_vector(4 downto 0) := B"01110";
   constant WAIT_FOR_VSYNC :bit_vector(4 downto 0) := B"01111";
```

```
    constant RESET_VERT         :bit_vector(4 downto 0) := B"10000";
    constant COUNT_LINES        :bit_vector(4 downto 0) := B"10001";
    constant VERT_HOLD_LOAD:bit_vector(4 downto 0) := B"10010";
    constant WAIT_HSYNC         :bit_vector(4 downto 0) := B"10011";
    constant LOAD         :bit_vector(4 downto 0) := B"10100";
    constant WAIT_1             :bit_vector(4 downto 0) := B"10101";
    constant CHECK_LT           :bit_vector(4 downto 0) := B"10110";
    constant LT_YES             :bit_vector(4 downto 0) := B"10111";
    constant LT_NO              :bit_vector(4 downto 0) := B"11000";
    constant CHECK_GT           :bit_vector(4 downto 0) := B"11001";
    constant GT_YES             :bit_vector(4 downto 0) := B"11010";
    constant GT_NO              :bit_vector(4 downto 0) := B"11011";
    constant CHECK_FC           :bit_vector(4 downto 0) := B"11100";
    constant DONE         :bit_vector(4 downto 0) := B"11101";

begin

      process(SYS_CLK, HSYNC, GT, LT, FLAG_16_LINES, FLAG_11_HITS,
      FLAG_1_FIELD, FIND_MODE, CNT_LT_GT)
      variable p_state  :  bit_vector(4 downto 0);
      begin

      if ( (SYS_CLK='1') and (SYS_CLK'event) ) then

      STATE <= p_state;

      if(RESET = '0') then
         p_state := WAIT_CMD;
      else
      case p_state is

         when WAIT_CMD  =>

    HIT_CNT_RST       <='0';
    HIT_CNT_CE        <='0';
    HOLD_LOAD         <='0';
    NEW_LOAD          <='0';
    L_CNT_LOAD        <='0';
    MUX_SRC_SEL       <='0';
    ADDER_CI          <='0';
    TC_RST            <='0';
    LC_RST            <='0';
    FC_RST            <='0';
    LT_CNT_CE         <='0';
    GT_CNT_CE         <='0';
    LT_GT_RST         <='1';
    FOUND_MODE        <='0';
    DONE_COUNT        <='0';

        if(FIND_MODE = '1') then
           p_state := GET_HOLD;
        elsif(CNT_LT_GT = '1') then
```

```
            p_state := BEGIN_CNT;
        end if;

        when GET_HOLD  =>

    HIT_CNT_RST       <='0';
    HIT_CNT_CE        <='0';
    HOLD_LOAD         <='0';
    NEW_LOAD          <='0';
    L_CNT_LOAD        <='0';
    MUX_SRC_SEL       <='0';
    ADDER_CI          <='0';
    TC_RST            <='0';
    LC_RST            <='0';
    FC_RST            <='0';
    LT_CNT_CE         <='0';
    GT_CNT_CE         <='0';
    LT_GT_RST         <='0';
    FOUND_MODE        <='0';
    DONE_COUNT        <='0';

        if(HSYNC = '1') then
            p_state := TC_RESET;
        end if;

        when TC_RESET  =>

    HIT_CNT_RST       <='0';
    HIT_CNT_CE        <='0';
    HOLD_LOAD         <='0';
    NEW_LOAD          <='0';
    L_CNT_LOAD        <='0';
    MUX_SRC_SEL       <='0';
    ADDER_CI          <='0';
    TC_RST            <='0';
    LC_RST            <='0';
    FC_RST            <='0';
    LT_CNT_CE         <='0';
    GT_CNT_CE         <='0';
    LT_GT_RST         <='0';
    FOUND_MODE        <='0';
    DONE_COUNT        <='0';

            p_state := HOLD_EN;
```

```
        when HOLD_EN  =>

    HIT_CNT_RST      <='0';
    HIT_CNT_CE       <='0';
    HOLD_LOAD        <='0';
    NEW_LOAD         <='0';
    L_CNT_LOAD       <='0';
    MUX_SRC_SEL      <='0';
    ADDER_CI         <='0';
    TC_RST           <='1';
    LC_RST           <='0';
    FC_RST           <='0';
    LT_CNT_CE        <='0';
    GT_CNT_CE        <='0';
    LT_GT_RST        <='0';
    FOUND_MODE       <='0';
    DONE_COUNT       <='0';

            if(HSYNC = '1') then
               p_state := LOAD_HOLD;
            end if;

        when LOAD_HOLD  =>

    HIT_CNT_RST      <='0';
    HIT_CNT_CE       <='0';
    HOLD_LOAD        <='1';
    NEW_LOAD         <='0';
    L_CNT_LOAD       <='0';
    MUX_SRC_SEL      <='0';
    ADDER_CI         <='0';
    TC_RST           <='0';
    LC_RST           <='0';
    FC_RST           <='0';
    LT_CNT_CE        <='0';
    GT_CNT_CE        <='0';
    LT_GT_RST        <='0';
    FOUND_MODE       <='0';
    DONE_COUNT       <='0';

            p_state := COMP_EN;
```

```
    when COMP_EN  =>

HIT_CNT_RST      <='1';
HIT_CNT_CE       <='0';
HOLD_LOAD        <='0';
NEW_LOAD         <='0';
L_CNT_LOAD       <='0';
MUX_SRC_SEL      <='0';
ADDER_CI         <='0';
TC_RST           <='1';
LC_RST           <='1';
FC_RST           <='0';
LT_CNT_CE        <='0';
GT_CNT_CE        <='0';
LT_GT_RST        <='0';
FOUND_MODE       <='0';
DONE_COUNT       <='0';

      if(HSYNC = '1') then
         p_state := LOAD_TC;
      end if;

      when LOAD_TC  =>

HIT_CNT_RST      <='1';
HIT_CNT_CE       <='0';
HOLD_LOAD        <='0';
NEW_LOAD         <='1';
L_CNT_LOAD       <='0';
MUX_SRC_SEL      <='0';
ADDER_CI         <='0';
TC_RST           <='0';
LC_RST           <='1';
FC_RST           <='0';
LT_CNT_CE        <='0';
GT_CNT_CE        <='0';
LT_GT_RST        <='0';
FOUND_MODE       <='0';
DONE_COUNT       <='0';

         p_state := ONE_WAIT;
```

```
   when ONE_WAIT  =>

HIT_CNT_RST       <='1';
HIT_CNT_CE        <='0';
HOLD_LOAD         <='0';
NEW_LOAD          <='0';
L_CNT_LOAD        <='0';
MUX_SRC_SEL       <='0';
ADDER_CI          <='0';
TC_RST            <='1';
LC_RST            <='1';
FC_RST            <='0';
LT_CNT_CE         <='0';
GT_CNT_CE         <='0';
LT_GT_RST         <='0';
FOUND_MODE        <='0';
DONE_COUNT        <='0';

        p_state := CHECK_HIT;

      when CHECK_HIT  =>

HIT_CNT_RST       <='1';
HIT_CNT_CE        <='0';
HOLD_LOAD         <='0';
NEW_LOAD          <='0';
L_CNT_LOAD        <='0';
MUX_SRC_SEL       <='0';
ADDER_CI          <='0';
TC_RST            <='1';
LC_RST            <='1';
FC_RST            <='0';
LT_CNT_CE         <='0';
GT_CNT_CE         <='0';
LT_GT_RST         <='0';
FOUND_MODE        <='0';
DONE_COUNT        <='0';

      if(LT = '0' and GT='0') then
         p_state := HIT_YES;
      else
         p_state := HIT_NO;
      end if;
```

```
        when HIT_YES  =>

    HIT_CNT_RST      <='1';
    HIT_CNT_CE       <='1';
    HOLD_LOAD        <='0';
    NEW_LOAD         <='0';
    L_CNT_LOAD       <='0';
    MUX_SRC_SEL      <='0';
    ADDER_CI         <='0';
    TC_RST           <='1';
    LC_RST           <='1';
    FC_RST           <='0';
    LT_CNT_CE        <='0';
    GT_CNT_CE        <='0';
    LT_GT_RST        <='0';
    FOUND_MODE       <='0';
    DONE_COUNT       <='0';

        if(FLAG_16_LINES = '1') then
           p_state := DONE_16;
        else
           p_state := COMP_EN;
        end if;

        when HIT_NO  =>

    HIT_CNT_RST      <='1';
    HIT_CNT_CE       <='0';
    HOLD_LOAD        <='0';
    NEW_LOAD         <='0';
    L_CNT_LOAD       <='0';
    MUX_SRC_SEL      <='0';
    ADDER_CI         <='0';
    TC_RST           <='1';
    LC_RST           <='1';
    FC_RST           <='0';
    LT_CNT_CE        <='0';
    GT_CNT_CE        <='0';
    LT_GT_RST        <='0';
    FOUND_MODE       <='0';
    DONE_COUNT       <='0';

        if(FLAG_16_LINES = '1') then
           p_state := DONE_16;
        else
           p_state := COMP_EN;
        end if;
```

```
        when DONE_16  =>

    HIT_CNT_RST       <='1';
    HIT_CNT_CE        <='0';
    HOLD_LOAD         <='0';
    NEW_LOAD          <='0';
    L_CNT_LOAD        <='0';
    MUX_SRC_SEL       <='0';
    ADDER_CI          <='0';
    TC_RST            <='0';
    LC_RST            <='0';
    FC_RST            <='0';
    LT_CNT_CE         <='0';
    GT_CNT_CE         <='0';
    LT_GT_RST         <='0';
    FOUND_MODE        <='0';
    DONE_COUNT        <='0';

        if(FLAG_11_HITS = '1') then
           p_state := WAIT_FOR_VSYNC;
        else
           p_state := GET_HOLD;
        end if;

        when WAIT_FOR_VSYNC =>

    HIT_CNT_RST       <='0';
    HIT_CNT_CE        <='0';
    HOLD_LOAD         <='0';
    NEW_LOAD          <='0';
    L_CNT_LOAD        <='0';
    MUX_SRC_SEL       <='0';
    ADDER_CI          <='0';
    TC_RST            <='0';
    LC_RST            <='0';
    FC_RST            <='1';
    LT_CNT_CE         <='0';
    GT_CNT_CE         <='0';
    LT_GT_RST         <='0';
    FOUND_MODE        <='0';
    DONE_COUNT        <='0';

        if(FLAG_1_FIELD = '1') then
           p_state := SIG_DONE;
        end if;
```

```
      when SIG_DONE  =>

HIT_CNT_RST       <='0';
HIT_CNT_CE        <='0';
HOLD_LOAD         <='0';
NEW_LOAD          <='0';
L_CNT_LOAD        <='0';
MUX_SRC_SEL       <='0';
ADDER_CI          <='0';
TC_RST            <='0';
LC_RST            <='0';
FC_RST            <='0';
LT_CNT_CE         <='0';
GT_CNT_CE         <='0';
LT_GT_RST         <='0';
FOUND_MODE        <='1';
DONE_COUNT        <='0';

        if(FIND_MODE = '0') then
           p_state := WAIT_CMD;
        end if;

      when BEGIN_CNT =>

HIT_CNT_RST       <='0';
HIT_CNT_CE        <='0';
HOLD_LOAD         <='0';
NEW_LOAD          <='0';
L_CNT_LOAD        <='0';
MUX_SRC_SEL       <='0';
ADDER_CI          <='0';
TC_RST            <='0';
LC_RST            <='0';
FC_RST            <='0';
LT_CNT_CE         <='0';
GT_CNT_CE         <='0';
LT_GT_RST         <='0';
FOUND_MODE        <='0';
DONE_COUNT        <='0';

   p_state := WAIT_HSYNC;
```

```
        when WAIT_HSYNC =>

    HIT_CNT_RST     <='0';
    HIT_CNT_CE      <='0';
    HOLD_LOAD       <='0';
    NEW_LOAD        <='0';
    L_CNT_LOAD      <='0';
    MUX_SRC_SEL     <='0';
    ADDER_CI        <='0';
    TC_RST          <='1';
    LC_RST          <='1';
    FC_RST          <='1';
    LT_CNT_CE       <='0';
    GT_CNT_CE       <='0';
    LT_GT_RST       <='1';
    FOUND_MODE      <='0';
    DONE_COUNT      <='0';


        if(HSYNC = '1') then
           p_state := LOAD;
        end if;

        when LOAD =>

    HIT_CNT_RST     <='0';
    HIT_CNT_CE      <='0';
    HOLD_LOAD       <='0';
    NEW_LOAD        <='1';
    L_CNT_LOAD      <='0';
    MUX_SRC_SEL     <='0';
    ADDER_CI        <='1';
    TC_RST          <='0';
    LC_RST          <='1';
    FC_RST          <='1';
    LT_CNT_CE       <='0';
    GT_CNT_CE       <='0';
    LT_GT_RST       <='1';
    FOUND_MODE      <='0';
    DONE_COUNT      <='0';


        p_state := WAIT_1;
```

```
        when WAIT_1 =>

    HIT_CNT_RST     <='0';
    HIT_CNT_CE      <='0';
    HOLD_LOAD       <='0';
    NEW_LOAD        <='0';
    L_CNT_LOAD      <='0';
    MUX_SRC_SEL     <='0';
    ADDER_CI        <='1';
    TC_RST          <='1';
    LC_RST          <='1';
    FC_RST          <='1';
    LT_CNT_CE       <='0';
    GT_CNT_CE       <='0';
    LT_GT_RST       <='1';
    FOUND_MODE      <='0';
    DONE_COUNT      <='0';


        p_state := CHECK_LT;

        when CHECK_LT =>

    HIT_CNT_RST     <='0';
    HIT_CNT_CE      <='0';
    HOLD_LOAD       <='0';
    NEW_LOAD        <='0';
    L_CNT_LOAD      <='0';
    MUX_SRC_SEL     <='0';
    ADDER_CI        <='1';
    TC_RST          <='1';
    LC_RST          <='1';
    FC_RST          <='1';
    LT_CNT_CE       <='0';
    GT_CNT_CE       <='0';
    LT_GT_RST       <='1';
    FOUND_MODE      <='0';
    DONE_COUNT      <='0';


        if(LT = '1' and GT = '0') then
            p_state := LT_YES;
        else
            p_state := LT_NO;
        end if;
```

```
        when LT_YES =>

    HIT_CNT_RST     <='0';
    HIT_CNT_CE      <='0';
    HOLD_LOAD       <='0';
    NEW_LOAD        <='0';
    L_CNT_LOAD      <='0';
    MUX_SRC_SEL     <='1';
    ADDER_CI        <='0';
    TC_RST          <='1';
    LC_RST          <='1';
    FC_RST          <='1';
    LT_CNT_CE       <='1';
    GT_CNT_CE       <='0';
    LT_GT_RST       <='1';
    FOUND_MODE      <='0';
    DONE_COUNT      <='0';


        p_state := CHECK_GT;

        when LT_NO =>

    HIT_CNT_RST     <='0';
    HIT_CNT_CE      <='0';
    HOLD_LOAD       <='0';
    NEW_LOAD        <='0';
    L_CNT_LOAD      <='0';
    MUX_SRC_SEL     <='1';
    ADDER_CI        <='0';
    TC_RST          <='1';
    LC_RST          <='1';
    FC_RST          <='1';
    LT_CNT_CE       <='0';
    GT_CNT_CE       <='0';
    LT_GT_RST       <='1';
    FOUND_MODE      <='0';
    DONE_COUNT      <='0';


        p_state := CHECK_GT;
```

```
        when CHECK_GT =>

    HIT_CNT_RST     <='0';
    HIT_CNT_CE      <='0';
    HOLD_LOAD       <='0';
    NEW_LOAD        <='0';
    L_CNT_LOAD      <='0';
    MUX_SRC_SEL     <='1';
    ADDER_CI        <='0';
    TC_RST          <='1';
    LC_RST          <='1';
    FC_RST          <='1';
    LT_CNT_CE       <='0';
    GT_CNT_CE       <='0';
    LT_GT_RST       <='1';
    FOUND_MODE      <='0';
    DONE_COUNT      <='0';


        if(GT = '1' and LT = '0') then
           p_state := GT_YES;
        else
           p_state := GT_NO;
        end if;

        when GT_YES =>

    HIT_CNT_RST     <='0';
    HIT_CNT_CE      <='0';
    HOLD_LOAD       <='0';
    NEW_LOAD        <='0';
    L_CNT_LOAD      <='0';
    MUX_SRC_SEL     <='0';
    ADDER_CI        <='0';
    TC_RST          <='1';
    LC_RST          <='1';
    FC_RST          <='1';
    LT_CNT_CE       <='0';
    GT_CNT_CE       <='1';
    LT_GT_RST       <='1';
    FOUND_MODE      <='0';
    DONE_COUNT      <='0';


        p_state := CHECK_FC;
```

```
        when GT_NO =>

    HIT_CNT_RST     <='0';
    HIT_CNT_CE      <='0';
    HOLD_LOAD       <='0';
    NEW_LOAD        <='0';
    L_CNT_LOAD      <='0';
    MUX_SRC_SEL     <='0';
    ADDER_CI        <='0';
    TC_RST          <='1';
    LC_RST          <='1';
    FC_RST          <='1';
    LT_CNT_CE       <='0';
    GT_CNT_CE       <='0';
    LT_GT_RST       <='1';
    FOUND_MODE      <='0';
    DONE_COUNT      <='0';


        p_state := CHECK_FC;

        when CHECK_FC =>

    HIT_CNT_RST     <='0';
    HIT_CNT_CE      <='0';
    HOLD_LOAD       <='0';
    NEW_LOAD        <='0';
    L_CNT_LOAD      <='0';
    MUX_SRC_SEL     <='0';
    ADDER_CI        <='0';
    TC_RST          <='1';
    LC_RST          <='1';
    FC_RST          <='1';
    LT_CNT_CE       <='0';
    GT_CNT_CE       <='0';
    LT_GT_RST       <='1';
    FOUND_MODE      <='0';
    DONE_COUNT      <='0';


        if(FLAG_1_FIELD = '0') then
            p_state := WAIT_HSYNC;
        else
            p_state := LOAD_VERT;
        end if;
```

```
            when LOAD_VERT =>

        HIT_CNT_RST     <='0';
        HIT_CNT_CE      <='0';
        HOLD_LOAD       <='0';
        NEW_LOAD        <='0';
        L_CNT_LOAD      <='1';
        MUX_SRC_SEL     <='0';
        ADDER_CI        <='0';
        TC_RST          <='0';
        LC_RST          <='1';
        FC_RST          <='0';
        LT_CNT_CE       <='0';
        GT_CNT_CE       <='0';
        LT_GT_RST       <='1';
        FOUND_MODE      <='0';
        DONE_COUNT      <='0';

          p_state := DONE;

            when DONE =>

        HIT_CNT_RST     <='0';
        HIT_CNT_CE      <='0';
        HOLD_LOAD       <='0';
        NEW_LOAD        <='0';
        L_CNT_LOAD      <='0';
        MUX_SRC_SEL     <='0';
        ADDER_CI        <='0';
        TC_RST          <='0';
        LC_RST          <='0';
        FC_RST          <='0';
        LT_CNT_CE       <='0';
        GT_CNT_CE       <='0';
        LT_GT_RST       <='1';
        FOUND_MODE      <='0';
        DONE_COUNT      <='1';


            if(CNT_LT_GT = '0') then
               p_state := WAIT_CMD;
            end if;

          end case;
          end if;
          end if;
          end process;

end behavior;
```

err_acc.vhd

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity err_acc is
port(   SCLK                    :in std_logic;
        H_SYNC_EXT              :in std_logic;
        V_SYNC_EXT              :in std_logic;
        ACC                     :out std_logic_vector(15 downto 0);
        TP_HSYNC                :out std_logic;
        TP_VSYNC                :out std_logic);
end err_acc;

architecture behavior of err_acc is

signal H_SYNC           :std_logic;
signal V_SYNC           :std_logic;
signal l_hsync          :std_logic;
signal l_vsync          :std_logic;
signal HSYNC_EDGE       :std_logic;
signal VSYNC_EDGE       :std_logic;
signal PIX_CNT          :std_logic_vector(9 downto 0);
signal PIX_EXT          :std_logic_vector(5 downto 0);
signal PIX              :std_logic_vector(15 downto 0);
signal LINE_CNT         :std_logic_vector(8 downto 0);
signal ACC_CE           :std_logic;
signal tie_high         :std_logic;
signal tie_low          :std_logic;

component acc16
port(   CI                                      :in std_logic;
        B15,B14,B13,B12,B11,B10,B9,B8           :in std_logic;
        B7,B6,B5,B4,B3,B2,B1,B0                 :in std_logic;
        D15,D14,D13,D12,D11,D10,D9,D8           :in std_logic;
        D7,D6,D5,D4,D3,D2,D1,D0                 :in std_logic;
        L                                       :in std_logic;
        ADD                                     :in std_logic;
        CE                                      :in std_logic;
        C                                       :in std_logic;
        R                                       :in std_logic;
        Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8           :out std_logic;
        Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0                 :out std_logic;
        CO                                      :out std_logic;
        OFL                                     :out std_logic);
end component;
```

```
begin

HSYNC_EDGE          <= not H_SYNC and l_hsync;
VSYNC_EDGE          <= not V_SYNC and l_vsync;
TP_HSYNC            <= HSYNC_EDGE;
TP_VSYNC            <= VSYNC_EDGE;
PIX_EXT             <=  PIX_CNT(9) & PIX_CNT(9) & PIX_CNT(9)
                        & PIX_CNT(9) & PIX_CNT(9) & PIX_CNT(9);
PIX                 <= PIX_EXT & PIX_CNT;
tie_high            <= '1';
tie_low             <= '0';
ACC_CE              <= '1' when ( (HSYNC_EDGE = '1') and
                        (LINE_CNT > 64 and LINE_CNT < 192) ) else '0';

        process(SCLK)
        begin
                if(SCLK = '1' and SCLK'event) then
                        l_hsync <= H_SYNC;
                        l_vsync <= V_SYNC;
                        PIX_CNT         <= PIX_CNT;
                        LINE_CNT        <= LINE_CNT;
                        if(VSYNC_EDGE = '1') then
                                LINE_CNT <= "000000000";
                        end if;
                        if(HSYNC_EDGE = '1') then
                                PIX_CNT         <= "1110001101";
                                LINE_CNT <= LINE_CNT + 1;
                        else
                                PIX_CNT         <= PIX_CNT - 1;
                        end if;
                end if;
        end process;

        process(SCLK)
        begin
                if(SCLK = '1' and SCLK'event) then
                        H_SYNC <= H_SYNC_EXT;
                        V_SYNC <= V_SYNC_EXT;
                end if;
        end process;

I01:  acc16 port map(tie_low,PIX(15),PIX(14),PIX(13),PIX(12),PIX(11),PIX(10),PIX(9) ,PIX(8) ,
                PIX(7) ,PIX(6) ,PIX(5) ,PIX(4) ,PIX(3) ,PIX(2) ,PIX(1) ,PIX(0) , tie_low,tie_low,
                tie_low,tie_low,tie_low,tie_low,tie_low,tie_low,tie_low, tie_low,tie_low,tie_low,tie_low,
                tie_low,tie_low,tie_low,tie_low,tie_low,tie_high,ACC_CE,SCLK,tie_low,
                ACC(15),ACC(14),ACC(13),ACC(12),ACC(11),ACC(10),ACC(9) ,ACC(8) ,
                ACC(7) ,ACC(6) ,ACC(5) ,ACC(4) ,ACC(3) ,ACC(2), ACC(1), ACC(0) ,open,open);

end behavior;
```

vcr_det.vhd

```
Library IEEE,synopsys;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use synopsys.attributes.all;

entity vcr_det is
port(
-- The following ports are required for the Mitsubishi IC.
-- NTSC_PAL and TEST_MODE inputs are not implemented in the
-- Xilinx Prototype at this time.

        RESET           :in std_logic;
        FSC             :in std_logic;
        H_SYNC          :in std_logic;
        V_SYNC          :in std_logic;
--      NTSC_PAL        :in std_logic_vector(1 downto 0);
--      TEST_MODE       :in std_logic;
        VCR             :out std_logic;
        CC_DET                  :out std_logic;

-- The following ports are required for the Xilinx prototype
-- in order to divide the 8FSC clock downto 4FSC on pin T15.
-- and to instantiate the STARTUP block

        GSR             :in std_logic;
        FSC_8           :in std_logic;
        FSC_OUT         :buffer std_logic;

-- The following ports are used for testing of the Xilinx Prototype.

        TP_SIGN_CHG  :out integer range 0 to 7;
        TP_VCR_FLAG :out std_logic;
        TP_HITS         :out integer range 0 to 31;
        TP_STATE        :out std_logic_vector(1 downto 0);
        TP_WIN_EN       :out std_logic;
        TP_SUM          :out std_logic_vector(15 downto 0);
        TP_HSYNC        :out std_logic;
        TP_VSYNC        :out std_logic
   );
end vcr_det;

architecture behavior of vcr_det is

-- Main Circuit Declarations
component edge_det
port(   INPUT           :in std_logic;
        RESET           :in std_logic;
        CLK             :in std_logic;
        OUTPUT          :out std_logic);
end component;
```

```
constant main           :std_logic_vector(1 downto 0) := "00";
constant cc_delay       :std_logic_vector(1 downto 0) := "01";
constant check_vcr      :std_logic_vector(1 downto 0) := "10";
constant CC_THRES       :std_logic_vector(15 downto 0) := "0000000001000000";
constant VCR_THRES      :std_logic_vector(15 downto 0) := "0000000000010000";

signal WIN_EN           :std_logic;
signal HSYNC            :std_logic;
signal VSYNC            :std_logic;
signal COUNTDOWN        :std_logic_vector(11 downto 0);
signal V_SYNC_INV       :std_logic;

-- Xilinx Prototpye component and signal declarations

component STARTUP
port(   GSR,GTS,CLK             :in std_logic;
        Q2,Q3,Q1Q4,DONEIN       :out std_logic);
end component;
attribute dont_touch of U1:label is true;
signal FOO              :std_logic;
signal NTSC_PAL                 :std_logic_vector(1 downto 0);

begin

-- Signal assignments for Xilinx Prototype

TP_HSYNC <= HSYNC;
TP_VSYNC <= VSYNC;
TP_WIN_EN <= WIN_EN;
NTSC_PAL <= "00";       -- Set to NTSC
FOO     <= '0';

-- STARTUP block instantiation for Xilinx Prototype

U1:     STARTUP port map(GSR, FOO, FOO, open, open, open, open);

-- Concurrent Signal Assignments for Main Circuit

V_SYNC_INV <= not V_SYNC;

-- The following code is used for the Xilinx prototype ONLY!

process(FSC_8)
begin
if(FSC_8 = '1' and FSC_8'event) then
        FSC_OUT <= not FSC_OUT;
end if;
end process;



-- End of Xilinx Specific code
```

```
-- Mux the appropriate COUNTDOWN value based on NTSC_PAL input.

COUNTDOWN <= "001110001101" when NTSC_PAL = "00" else     -- NTSC
             "001110001100" when NTSC_PAL = "01" else     -- PAL/M
             "010001101110" when NTSC_PAL = "10" else     -- PAL/B,D,G,H,N,I
             "001110001101";                              -- NTSC

accummulator:process(RESET,FSC)
variable pixel_count    :std_logic_vector(11 downto 0);
variable sum            :std_logic_vector(15 downto 0);
variable sum_1          :std_logic_vector(15 downto 0);
variable state          :std_logic_vector(1 downto 0);
variable field_count          :integer range 0 to 15;
variable vcr_flag  :std_logic;
variable hits                 :integer range 0 to 31;
variable sign_chg :integer range 0 to 7;
variable last_sign :std_logic;
begin
  if(RESET = '0') then
        pixel_count       := "000000000000";
        sum               := "0000000000000000";
        sum_1             := "0000000000000000";
        state             := "00";
        vcr_flag          := '0';
        field_count       := 0;
        hits              := 0;
        sign_chg          := 0;
        last_sign         := '0';
  else
        if(FSC = '1' and FSC'event) then
                CC_DET          <= '0';
                VCR             <= '0';
                case state is

                when main =>
                if(VSYNC = '1') then
                        field_count := field_count + 1;
                        if( abs(sum - sum_1) > CC_THRES) then
                                state := cc_delay;
                                field_count := 0;
                        elsif(field_count = 15) then
                                field_count := 0;
                                state := check_vcr;
                        end if;
                        sum_1 := sum;
                end if;
                if(HSYNC = '1') then
                        if(WIN_EN = '1') then
                          sum := sum + pixel_count;
                        end if;
                        pixel_count := COUNTDOWN;
                else
```

```
            pixel_count := pixel_count - 1;
end if;

when cc_delay =>
sum             := "0000000000000000";
sum_1           := "0000000000000000";
hits            := 0;
vcr_flag        := '0';
sign_chg        := 0;
if(VSYNC = '1') then
            field_count := field_count + 1;
            if(field_count = 15) then
                        field_count := 0;
                        CC_DET <= '1';
                        state := main;
            end if;
end if;

when check_vcr =>
if(sum(15) /= last_sign) then
            if(sign_chg = 7) then
                        sign_chg := 7;
            else
                        sign_chg := sign_chg + 1;
            end if;
end if;
last_sign := sum(15);
if( abs(sum) > VCR_THRES ) then
            if(hits = 31) then
                        hits := 31; else
                        hits := hits + 1;
            end if;
else
            if(hits = 0) then
                        hits := 0; else
                        hits := hits - 1;
            end if;
end if;
if(hits >= 25 and sign_chg <= 1) then
            if(vcr_flag = '0') then
                        vcr_flag := '1';
                        VCR     <= '1';
                        CC_DET  <= '1';
            end if;
else
            if(vcr_flag ='1') then
                        hits := 0;
                        vcr_flag := '0';
                        sign_chg := 0;
                        CC_DET  <= '1';
            end if;
end if;
```

```
                sum     := "0000000000000000";
                sum_1   := "0000000000000000";
                state   := main;

        when OTHERS =>
                state := main;
                end case;
                TP_VCR_FLAG             <= vcr_flag;
                TP_STATE                <= state;
                TP_SUM(14 downto 0)     <= sum(14 downto 0);
                TP_SUM(15)              <= not sum(15);
                TP_HITS                 <= hits;
                TP_SIGN_CHG             <= sign_chg;
        end if;
  end if;
end process;

windower:
process(RESET,FSC)
variable line_count     :integer range 0 to 511;
variable window         :std_logic;
begin
        if(RESET = '0') then
                line_count      := 0;
                window          := '0';
        else
                if(FSC = '1' and FSC'event) then
                        if(VSYNC = '1') then
                                window := '0';
                                line_count := 0;
                        elsif(HSYNC = '1') then
                                line_count := line_count + 1;
                                if(line_count = 64) then
                                        window := '1';
                                elsif(line_count = 192) then
                                        window := '0';
                                end if;
                        end if;
                        WIN_EN <= window;
                end if;
        end if;
end process windower;

ED01:   edge_det port map(H_SYNC,RESET,FSC,HSYNC);
ED02:   edge_det port map(V_SYNC_INV,RESET,FSC,VSYNC);

end behavior;
```

xilinx.vhd

```
Library IEEE,synopsys;
use IEEE.std_logic_1164.all;
use synopsys.attributes.all;

entity xilinx is
port(   RESET           :in std_logic;
        FSC             :in std_logic;
        H_SYNC          :in std_logic;
        V_SYNC          :in std_logic;
        VCR             :out std_logic;
        CC_DET          :out std_logic;

-- The following ports are required for the Xilinx prototype
-- in order to divide the 8FSC clock downto 4FSC on pin T15.
-- and to instantiate the STARTUP block

        GSR             :in std_logic;
        FSC_8           :in std_logic;
        FSC_OUT         :buffer std_logic);

end xilinx;

architecture behavior of xilinx is

-- Main Circuit Declarations

component detector
port(   RESET           :in std_logic;
        FSC             :in std_logic;
        H_SYNC          :in std_logic;
        V_SYNC          :in std_logic;
        NTSC_PAL        :in std_logic_vector(1 downto 0);
        TEST_MODE       :in std_logic;
        VCR_OUT         :out std_logic;
        CC_DET_OUT      :out std_logic);
end component;

-- Xilinx Prototpye component and signal declarations

component STARTUP
port(   GSR,GTS,CLK             :in std_logic;
        Q2,Q3,Q1Q4,DONEIN       :out std_logic);
end component;

attribute dont_touch of U1:label is true;

signal FOO              :std_logic;
signal NTSC_PAL         :std_logic_vector(1 downto 0);
signal TEST_MODE        :std_logic;
signal H_SYNC_INV       :std_logic;
```

```
signal V_SYNC_INV       :std_logic;

begin

-- Signal assignments for Xilinx Prototype

H_SYNC_INV   <= not H_SYNC;
V_SYNC_INV   <= not V_SYNC;
NTSC_PAL     <= "00";-- Set to NTSC
TEST_MODE    <= '0';
FOO          <= '0';

-- STARTUP block instantiation for Xilinx Prototype

U1:     STARTUP port map(GSR, FOO, FOO, open, open, open, open);
U2:     detector port map(RESET,FSC,H_SYNC_INV,V_SYNC_INV,NTSC_PAL,TEST_MODE,
                          VCR,CC_DET);

-- The following code is used for the Xilinx prototype ONLY!

process(FSC_8)
begin
if(FSC_8 = '1' and FSC_8'event) then
        FSC_OUT <= not FSC_OUT;
end if;
end process;

-- End of Xilinx Specific code

end behavior;
```

detector.vhd

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;

entity detector is
port(   RESET           :in std_logic;
        FSC             :in std_logic;
        H_SYNC          :in std_logic;
        V_SYNC          :in std_logic;
        NTSC_PAL        :in std_logic_vector(1 downto 0);
        TEST_MODE       :in std_logic;
        VCR_OUT         :out std_logic;
        CC_DET_OUT      :out std_logic);
end detector;

architecture behavior of detector is

-- Main Circuit Declarations

component edge_det
port(   INPUT           :in std_logic;
        RESET           :in std_logic;
        CLK             :in std_logic;
        OUTPUT          :out std_logic);
end component;

component output_proc
port(   INPUT           :in std_logic;
        RESET           :in std_logic;
        CLK             :in std_logic;
        OUTPUT          :out std_logic);
end component;

constant main           :std_logic_vector(1 downto 0) := "00";
constant cc_delay       :std_logic_vector(1 downto 0) := "01";
constant check_vcr      :std_logic_vector(1 downto 0) := "10";
constant CC_THRES       :std_logic_vector(15 downto 0) := "0000000001000000";
constant VCR_THRES      :std_logic_vector(15 downto 0) := "0000000000010000";

signal WIN_EN           :std_logic;
signal HSYNC            :std_logic;
signal VSYNC            :std_logic;
signal COUNTDOWN        :std_logic_vector(11 downto 0);
signal V_SYNC_INV       :std_logic;
signal H_SYNC_INV       :std_logic;
signal CC_DET           :std_logic;
signal VCR              :std_logic;

begin
```

```
H_SYNC_INV <= not H_SYNC;
V_SYNC_INV <= not V_SYNC;


-- Mux the appropriate COUNTDOWN value based on NTSC_PAL input.

COUNTDOWN <= "001110001101" when NTSC_PAL = "00" else     -- NTSC
             "001110001100" when NTSC_PAL = "01" else     -- PAL/M
             "010001101110" when NTSC_PAL = "10" else     -- PAL/B,D,G,H,N,I
             "001110001101";                              -- NTSC

accummulator:
process(RESET,FSC)
variable pixel_count      :std_logic_vector(11 downto 0);
variable sum              :std_logic_vector(15 downto 0);
variable sum_1            :std_logic_vector(15 downto 0);
variable state            :std_logic_vector(1 downto 0);
variable field_count      :integer range 0 to 15;
variable vcr_flag         :std_logic;
variable hits             :integer range 0 to 31;
variable sign_chg         :integer range 0 to 7;
variable last_sign        :std_logic;
begin
  if(RESET = '0') then
        pixel_count       := "000000000000";
        sum               := "0000000000000000";
        sum_1             := "0000000000000000";
        state             := "00";
        vcr_flag          := '0';
        field_count       := 0;
        hits              := 0;
        sign_chg          := 0;
        last_sign         := '0';
  else
        if(FSC = '1' and FSC'event) then
                CC_DET  <= '0';
                VCR     <= '0';
                case state is
                when main =>
                if(VSYNC = '1') then
                        field_count := field_count + 1;
                        if( abs(sum - sum_1) > CC_THRES and vcr_flag = '0') then
                                state := cc_delay;
                                field_count := 0;
                        elsif(field_count = 15) then
                                field_count := 0;
                                state := check_vcr;
                        end if;
                        sum_1 := sum;
                end if;
                if(HSYNC = '1') then
                        if(WIN_EN = '1') then
```

```
            sum := sum + pixel_count;
        end if;
        pixel_count := COUNTDOWN;
else
        pixel_count := pixel_count - 1;
end if;

when cc_delay =>
sum     := "0000000000000000";
sum_1   := "0000000000000000";
hits    := 0;
vcr_flag := '0';
sign_chg := 0;
if(VSYNC = '1') then
        field_count := field_count + 1;
        if(field_count = 15) then
                field_count := 0;
                CC_DET <= '1';
                state := main;
        end if;
end if;

when check_vcr =>

if(sum(15) /= last_sign) then
        if(sign_chg = 7) then
                sign_chg := 7;
        else
                sign_chg := sign_chg + 1;
        end if;
end if;
last_sign := sum(15);
if( abs(sum) > VCR_THRES ) then
        if(hits = 31) then
                hits := 31; else
                hits := hits + 1;
        end if;
else
        if(hits = 0) then
                hits := 0; else
                hits := hits - 1;
        end if;
end if;
if(hits >= 25 and sign_chg <= 1) then
        if(vcr_flag = '0') then
                vcr_flag  := '1';
                VCR       <= '1';
                CC_DET    <= '1';
        end if;
else
        if(vcr_flag ='1') then
                hits      := 0;
```

```
                                vcr_flag  := '0';
                                sign_chg := 0;
                                CC_DET <= '1';
                        end if;
                end if;
                sum   := "0000000000000000";
                sum_1 := "0000000000000000";
                state := main;

                when OTHERS =>
                state := main;
                end case;
        end if;
  end if;
end process;

windower:
process(RESET,FSC,HSYNC,VSYNC)
variable line_count     :integer range 0 to 1023;
variable window         :std_logic;
begin
        if(RESET = '0') then
                line_count      := 0;
                window          := '0';
        else
                if(FSC = '1' and FSC'event) then
                        if(VSYNC = '1') then
                                window := '0';
                                line_count := 0;
                        elsif(HSYNC = '1') then
                                line_count := line_count + 1;
                                if(line_count = 64) then
                                        window := '1';
                                elsif(line_count = 192) then
                                        window := '0';
                                end if;
                        end if;
                        WIN_EN <= window;
                end if;
        end if;
end process windower;

ED01:   edge_det port map(H_SYNC_INV,RESET,FSC,HSYNC);
ED02:   edge_det port map(V_SYNC_INV,RESET,FSC,VSYNC);
OP01:output_proc port map(CC_DET,RESET,FSC,CC_DET_OUT);
OP02:output_proc port map(VCR,RESET,FSC,VCR_OUT);

end behavior;
```

edge_det.vhd

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity edge_det is
        port(   INPUT           :in std_logic;
                RESET           :in std_logic;
                CLK             :in std_logic;
                OUTPUT          :out std_logic);
end edge_det;

architecture behavior of edge_det is
signal  Q1,Q2       :std_logic;
begin
        OUTPUT <= Q1 and (not Q2);
        FF1: process(CLK,INPUT,RESET)
        begin
                if (RESET = '0') then
                        Q1 <= '0';else
                        if(CLK = '1' and CLK'event) then
                                Q1 <= INPUT;
                        end if;
                end if;
        end process;
        FF2: process(CLK,Q1,RESET)
        begin
                if (RESET = '0') then
                        Q2 <= '0';
                else
                        if(CLK = '1' and CLK'event) then
                                Q2 <= Q1;
                        end if;
                end if;
        end process;
end behavior;
```

output_proc.vhd

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity output_proc is
        port(   INPUT           :in std_logic;
                RESET           :in std_logic;
                CLK             :inout std_logic;
                OUTPUT          :out std_logic);
end output_proc;

architecture behavior of output_proc is
signal  Q1,Q2       :std_logic;
begin
        OUTPUT <= Q1 or Q2;
        FF1: process(CLK,INPUT,RESET)
        begin
        if (RESET = '0') then
                Q1 <= '0';else
                if(CLK = '1' and CLK'event) then
                        Q1 <= INPUT;
                end if;
        end if;
        end process;

        FF2: process(CLK,Q1,RESET)
        begin
                if (RESET = '0') then
                        Q2 <= '0';else
                if(CLK = '1' and CLK'event) then
                        Q2 <= Q1;
                end if;
        end if;
        end process;
end behavior;
```

f_vect.vhd

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity f_vectors is
end f_vectors;

architecture vectors of f_vectors is
signal RESET            :std_logic;
signal FSC              :std_logic;
signal H_SYNC           :std_logic;
signal V_SYNC           :std_logic;
signal NTSC_PAL         :std_logic_vector(1 downto 0);
signal TEST_MODE        :std_logic;
signal VCR_OUT          :std_logic;
signal CC_DET_OUT       :std_logic;
signal TRUE             :std_logic;
signal FALSE            :std_logic;

component detector
port(   RESET           :in std_logic;
        FSC             :in std_logic;
        H_SYNC          :in std_logic;
        V_SYNC          :in std_logic;
        NTSC_PAL        :in std_logic_vector(1 downto 0);
        TEST_MODE       :in std_logic;
        VCR_OUT         :out std_logic;
        CC_DET_OUT      :out std_logic);
end component;

begin

TRUE   <= '1';
FALSE  <= '0';
U01: detector port
map(RESET,FSC,H_SYNC,V_SYNC,NTSC_PAL,TEST_MODE,VCR_OUT,CC_DET_OUT);

NTSC_PAL      <= "00";
TEST_MODE     <= '0';

 process
 begin
 RESET <= '1';
 wait for 50 ns;
 RESET <= '0';
 wait for 50 ns;
 RESET <= '1';
 wait;
 end process;
```

```
 CLOCK:process
   begin
     FSC <= '0';
     wait for 35 ns;
     FSC <= '1';
     wait for 35 ns;
   end process CLOCK;

   HORIZONTAL_SYNC:process
    begin
          H_SYNC <= '1';
          wait for 200 ns;
          for i in 0 to 62 loop
            H_SYNC <= '0';
            wait for 70 ns;
            H_SYNC <= '1';
            wait for 70 ns;
          end loop;
          for i in 0 to 130 loop
            H_SYNC <= '1';
            wait for 911 * 70 ns;
            H_SYNC <= '0';
            wait for 3 * 70 ns;
          end loop;
          H_SYNC <= '1';
          wait;
   end process;

   VERTICAL_SYNC:process
   begin
          V_SYNC <= '1';
          wait for 200 ns;
          wait for 8390400 ns;
          for i in 0 to 17 loop
            V_SYNC <= '0';
            wait for 70 ns;
            V_SYNC <= '1';
            wait for 70 ns;
          end loop;
          wait;
    end process;
end vectors;

configuration simulation of f_vectors is
for vectors
  for U01: detector use entity work.detector(behavior);
  end for;
end for;
end simulation;
```

Vita

Christopher L. Spearman was born in Knoxville, Tennessee on July 25, 1966. He attended elementary school at Chilhowee and First Lutheran schools and graduated from Knoxville Catholic High School in May, 1984. After attending State Technical Institute of Knoxville, he enlisted as an airborne communications specialist in the U.S. Army, where he served on active duty for four years at Fort Bragg, N.C. and in the National Guard for four years in Oak Ridge, TN. Upon completing his active duty service, he entered The University of Tennessee, Knoxville in August, 1989, and received his undergraduate degree in Electrical and Computer Engineering in May, 1993. He continued his studies at the University of Tennessee, Knoxville as a graduate student, and received his Master's Degree in Electrical Engineering in December 1995. He has been employed as a Digital Signal Processing Engineer at Philips Consumer Electronics, Knoxville, TN since March, 1994.