A FLEXIBLE DIGITAL COMMUNICATIONS SYSTEM FOR EDUCATION

BY

JACOB WILLIAM JANOVETZ

B.S., University of Illinois at Urbana-Champaign, 1996

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1999

Urbana, Illinois

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

SEPTEMBER 1999
(date)

WE HEREBY RECOMMEND THAT THE THESIS BY

JACOB WILLIAM JANOVETZ

ENTITLED FLEXIBLE DIGITAL COMMUNICATIONS SYSTEM FOR EDUCATION

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE

Douglas L. Jones
Director of Thesis Research

Head of Department

Committee on Final Examination†

Chairperson

† Required for doctor's degree but not for master's.

O-517

A FLEXIBLE DIGITAL COMMUNICATIONS SYSTEM FOR EDUCATION

BY

JACOB WILLIAM JANOVETZ

B.S., University of Illinois at Urbana-Champaign, 1996

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1999

Urbana, Illinois

To all those with a child-like curiosity, an explorative spirit, and the will to enact them.

ACKNOWLEDGMENTS

I would especially like to thank my friend Dr. Ricardo Uribe for our endless discussions, his unique perspectives, and the environment and atmosphere he has cultivated in the Advanced Digital Systems Lab. Not only would this work not have been possible without the ADSL, but my motivations may have faltered without the people there.

Thanks to Professors Doug Jones and Steve Franke as advisors; they have welcomed and encouraged my involvement in the DSP and Digital Communications Labs and have offered many good suggestions for the final form of this work.

I also want to thank Mike Kramer, Mike Palac, and Tony Keiser for discussions and suggestions regarding ERITHACUS.

Thanks, finally, to Joel Sherrill, Eric Norum, and all the RTEMS support people for their contributions to the free RTEMS operating system. All in all, they have more source code running on ERITHACUS than I do.

TABLE OF CONTENTS

CHAPTER PAGE

1 INTRODUCTION 1
1.1 Hardware and Software 1
1.2 Communication Labs 3
1.3 Organization 3

2 SYSTEM OVERVIEW 4
2.1 Design Goals 4
2.1.1 Present lab sessions 5
2.1.2 Possible extensions 6
2.1.3 Effective and portable interface 6
2.1.4 Programmability 6
2.2 Technology Introduction 7
2.2.1 Digital signal processors 7
2.2.2 Field programmable gate arrays 8
2.2.3 VHSIC hardware description language 9
2.2.4 RTEMS 10
2.3 System Overview 11
2.3.1 FPGA interface 12
2.3.2 DSP interface 12

3 TRANSMITTER ARCHITECTURE 14
3.1 System Architecture 15
3.1.1 Wavetable synthesis 15
3.1.2 Waveform synthesis with memory 16
3.1.2.1 Channel coding 17
3.2 Hardware Architecture 17

4 RECEIVER ARCHITECTURE 20
4.1 System Architecture 20
4.2 Hardware Receiver Architecture 22

5 CONCLUSION 26

APPENDIX A TRANSMITTER TECHNICAL INFORMATION 28
A.1 Transmitter FPGA Registers 28
A.1.1 Register descriptions 28

A.2 DSP/Wavetable Interface 31
A.3 Output Buffering 32
A.4 Sample Rate DDS 32
A.4.1 DDS configuration sample code 33

APPENDIX B RECEIVER TECHNICAL INFORMATION 34
B.1 Software Receiver FPGA Registers 34
B.1.1 Register descriptions 34
B.2 Hardware Receiver FPGA Registers 35
B.2.1 Register descriptions 35
B.3 Analog Inputs 39
B.4 Analog Outputs 39
B.5 Sample Rate DDS 40
B.6 PLL Coefficient 40

APPENDIX C SOFTWARE SUPPORT 43
C.1 Device-Side Software 43
C.1.1 FTPD 43
C.1.2 Telnet 44
C.1.3 HTTPD 46
C.2 Host-Side Software 46
C.2.1 Libraries 46
C.2.2 MATLAB commands 48
C.2.3 MATLAB user interfaces 49

APPENDIX D SCHEMATICS 52

APPENDIX E BILL OF MATERIALS 69

APPENDIX F REVISION 1.0 ERRATA 74

REFERENCES 76

LIST OF TABLES

Table		Page
2.1	ECE 363 Lab Sessions	5
2.2	Logic Block Usage on Xilinx 4000 Devices	9
A.1	Transmitter FPGA Registers	29
A.2	Transmitter Data Sources	30
B.1	Software Receiver FPGA Registers	34
B.2	Hardware Receiver FPGA Registers	36
B.3	RX_MODE Register	37
B.4	DAC_SELECT Register	38
C.1	Supported FTPD Directives	44
C.2	ERITHACUS FTPD Files	45
C.3	ERITHACUS Telnet Commands	45
C.4	COMMSETTXPARAM parameters	49
C.5	Receiver DAC Output Options	50
E.1	Bill of Materials	69

LIST OF FIGURES

Figure **Page**

2.1 ERITHACUS Block Diagram. 12

3.1 Quadrature Modulation. 14
3.2 Wavetable Synthesis. 15
3.3 Segmented Bandlimited Pulse. 16
3.4 Transmission Sequence with ISI. 17
3.5 Transmitter Block Diagram. 18
3.6 Transmitter Source Generation. 18
3.7 Wavetable Address Generation. 19

4.1 A Complete Digital Receiver Using ERITHACUS. 20
4.2 Block Diagram of ERITHACUS Receiver Section. 21
4.3 Block Diagram of the Hardware Receiver. 22
4.4 Receiver Input Stage and Phase Detector. 23
4.5 Receiver Loop Filter. 24
4.6 Receiver Output Stage. 25

C.1 Receiver Control Panel. 50
C.2 Auto BERT. 51

D.1 Design Notes. 53
D.2 Processor. 54
D.3 Processor FPGA and Memory. 55
D.4 Ethernet Interface. 56
D.5 Accessory DSP and Memory. 57
D.6 Audio Interface. 58
D.7 Transmitter FPGA and FPGA Memory. 59
D.8 Transmitter DSP. 60
D.9 Wavetable Memory and Analog Outputs. 61
D.10 Clock Generation. 62
D.11 Receiver FPGA. 63
D.12 Receiver DSP 1 and Shared Memory. 64
D.13 Receiver DSP 2. 65
D.14 Receiver Analog Outputs. 66
D.15 Analog Inputs. 67
D.16 Power Supply. 68

LIST OF ABBREVIATIONS

ABC - Already Been Chewed

ABS - Anti-lock Braking System

ADC - Analog-to-Digital Converter

ALU - Arithmetic Logic Unit

ATM - Asynchronous Transfer Mode

BDM - Background Debug Mode

BER - Bit Error Rate

BPSK - Binary Phase Shift Keying

BSP - Board Support Package

CDMA - Code Division Multiple Access

CDS - Christopher Dale Schmitz

CPLD - Complex Programmable Logic Device

CPU - Central Processing Unit

DAC - Digital-to-Analog Converter

DDS - Direct Digital Synthesizer (or Synthesis)

DRAM - Dynamic Random Access Memory

DSP - Digital Signal Processor

DSSS - Direct Sequence Spread Spectrum

FIFO - First-In First-Out

FPGA - Field Programmable Gate Array

FSK - Frequency Shift Keying

GNU - GNU is Not UNIX

IF - Intermediate Frequency

IIR - Infinite Impulse Response

IMFS - In-memory File System

IP - Internet Protocol

ISI - Intersymbol Interference

JWJ - Jacob William Janovetz

MAC - Multiply and Accumulate

MB - Megabytes

MSB - Most Significant Bit

NCO - Numerically Controlled Oscillator

NSF - National Science Foundation

OAR - On-line Applications Research Corporation

PAM - Pulse Amplitude Modulation

PCMCIA - People Can't Memorize Computer Industry Acronyms

PLL - Phase-locked Loop

PN - Pseudorandom Noise

QPSK - Quadrature Phase Shift Keying

QAM - Quadrature Amplitude Modulation

RISC - Reduced Instruction Set Computer

RTEMS - Real Time Executive for Military Systems

RTOS - Real Time Operating System

SIMM - Single Inline Memory Module

SRAM - Static Random Access Memory

SRRC - Square Root Raised Cosine

TCP - Transmission Control Protocol

USB - Universal Serial Bus

VCO - Voltage-controlled Oscillator

VHDL - VHSIC Hardware Description Language

VHSIC - Very High Speed Integrated Circuits

CHAPTER 1

INTRODUCTION

This thesis presents the design for a flexible digital communications system based on reprogrammable special-purpose microprocessors and reconfigurable hardware devices for use in the Digital Communications Laboratory (ECE 363) at the University of Illinois. The new design will serve as a replacement for the analog system currently in use, as well as provide a platform for further course development.

The ECE 363 coursework involves BER measurements under different system configurations and parameters. The relevance of this work is that students acquire an intuition for which parts of the system are critical paths to overall system performance. A vital component to building this intuition is the ability to reconfigure the system to make reasonable comparisons between BER curves. The original lab equipment was designed in blocks to allow physical reconfiguration of the system. Unfortunately, the digital receiver is one of those blocks and is hardwired for its task, making receiver configuration difficult. A software-based configurable system is one alternative, but software systems are often too slow to run at high bitrates[1]. Therefore, a system has been designed which leverages the advantages of both hardware and software design.

1.1 Hardware and Software

Conventional wisdom has it that hardware is fast while software is flexible. The original lab equipment for ECE 363 was designed as rather inflexible analog hardware to meet the bit-rate demands of the course. A flexible software solution was not available at the time. Recently however, a new class of programmable hardware devices has lessened the extent to

[1]High bit-rates are necessary to get adequate resolution (BER of 10^{-6}) on BER measurements in a reasonable amount of time (20 seconds).

which this conventional wisdom is true. FPGAs and other programmable logic devices provide a new architecture with performance near that of dedicated hardware and flexibility near that of software.

FPGAs have been around since the mid-1980's, but not until recently have their density and speeds met the needs of general architectures. Driven by requirements of reduced time to market and aftermarket flexibility, FPGAs have become fast and dense, sponsoring their use in high-speed busses and reconfigurable computing tasks. Recently, FPGAs have grown to accomodate large amounts of logic (10,000 gates to over 1 million gates) and operate at high speeds (40 MHz to over 200 MHz [1]). This advancement in their capability has made them viable solutions for many signal processing and communications tasks. FPGA use in communications has even triggered research towards more "special purpose" architectures such as Tsutsui's FPGA optimized for digital telecommunications systems such as ATM implementations [2]. A well-written, thorough review of FPGA technology is presented in [3].

One of the most common ways to program FPGAs is through the use of a hardware description language such as Verilog or VHDL. Verilog and VHDL provide a means for the designer to describe hardware textually rather than through more traditional schematics. The use of HDLs represents a convergence of hardware and software methodologies. Both FPGAs and VHDL have been used extensively in DSP and communicaions applications because it is these applications which often require high processing speeds and configurability. McCloskey [4] discusses the application of VHDL to software radio and overviews general software radio architectures. Vaupel et al. [5] present COMBOX, a library of VHDL module generators for communications.

The system presented here fits the genre of software radio testbeds that have been proposed in the literature [6], [7], [8], [9]. Typically, these systems are composed of a centralizing CPU (or host workstation) and one or more FPGAs for computation. Some systems include specialized ASIC modules to provide fast, dedicated functional units (in [8], an ASIC provides common CORDIC functionality while the FPGA provides reconfigurable control and datapath elements). Nearly all existing software radio designs contain elements of both hardware and software design. Even after the system has been built, hardware design continues in the form of hardware descriptions for the FPGAs on board, and software design continues in the form of software or firmware upgrades. In both cases, the platform is not particularly limited by the original hardware design.

1.2 Communication Labs

Several universities offer lab courses in digital communications and a few authors have published their laboratory setups. Kamali [10] presents a setup using off-the-shelf components such as spectrum analyzers, signal generators, and optical sensors to introduce students to practical measurements of communications systems. Overstreet and Austen [11] use a DSP, adding flexibility to the lab equipment and allowing the generation of several modulation schemes. Unfortunately, most of the work of other universities remains largely unpublished. However, to the author's knowledge, no other university uses reconfigurable hardware in its digital communications courses.

Although not directly related to digital communication, [12] introduces a rapid prototype design course funded by the NSF and taught at the Georgia Institute of Technology. The course allows students to prototype a number of different types of processors using VHDL and FPGAs. The University of Illinois also has a course which introduces VHDL for use on FPGAs [13]. These offerings show that there is a concentration on VHDL and reconfigurable hardware in education.

1.3 Organization

The remainder of this thesis is presented as follows: Chapter 2 discusses the overview of the system which was designed. Chapters 3 and 4 present the transmitter and receiver architectures, respectively. Chapter 5 provides conclusions of the project and discusses future directions. Finally, the appendices contain more technical information related to the project.

CHAPTER 2

SYSTEM OVERVIEW

ERITHACUS is a large system composed of many smaller blocks which, in turn, are composed of several different technologies. This chapter presents the design goals of the system and introduces the various technologies used. This chapter also includes a brief overview of ERITHACUS.

2.1 Design Goals

The University of Illinois at Urbana-Champaign (UIUC) hosts two separate laboratory courses concerned with digital communications and digital signal processing (DSP). The Digital Signal Processing Lab (ECE 320) is concerned with teaching the basics of DSP implementation on special-purpose processors called DSPs. The primary purpose of ECE 320 is to build insight into DSP and uses DSP programming as a vehicle for achieving this goal. The Digital Communications Lab (ECE 363) is concerned with building an intuition for digital communications systems. It covers the construction and measurement of various pieces of a communications system such as matched filtering, carrier recovery, and timing recovery. ECE 363 is not a programming course. Instead, it concentrates on block-based physical implementations.

The hardware described in this thesis (henceforth referred to as ERITHACUS) is designed as a replacement for the aging analog equipment currently used in ECE 363. The general philosophy is to design a highly configurable digital signal processing system geared toward the implementation of communications systems and capable of operating at high bit rates (above 1 Mbps).

The requirements placed on the new system were:

- Capability of replacing the present analog equipment in every lab situation.

- Provision of flexibility to allow enhancing the present labs.
- Provision of a simple, effective, and portable interface to allow for changing lab equipment.
- Provision of programmability to allow students in the DSP course to test algorithms.

These four requirements are addressed in the following sections.

2.1.1 Present lab sessions

Table 2.1 lists the current lab sessions in ECE 363. Labs 1 and 2 are introductory and characterization labs which do not use the digital receiver. Labs 3 and 4 work on baseband signals only. The remaining labs all operate on passband signals.

Table 2.1 ECE 363 Lab Sessions.

Lab	Title
Lab 1	Introduction to the Spectrum Analyzer
Lab 2	PN, Mixer, and Noise Source Characterization
Lab 3	Binary Antipodal Signaling
Lab 4	Suboptimal Demodulation (Midsymbol Sampling)
Lab 5	Binary Phase Shift Keying
Lab 6	Carrier Recovery
Lab 7	Quadrature Phase Shift Keying
Lab 8	Differential Phase Shift Keying
Lab 9	Clock Recovery
Lab 10	Communications in a Hostile Channel

All labs currently operate at a bitrate of 1 Mbps on an analog hardware receiver operating on baseband signals. Supporting these present lab sessions placed the following requirements on the new hardware:

- Support for high-speed baseband binary antipodal signaling using matched filter (optimal) reception and midsymbol sampling (suboptimal) reception.
- Passband signaling using external components for upconversion, downconversion, and quadrature separation.
- Closed-loop control of external components (e.g., VCOs for carrier recovery).

2.1.2 Possible extensions

The new hardware also has to support the possibility of future lab sessions. Because the old hardware is a static, analog design, its ability to adapt to the changing needs of the course is limited. The new hardware has to be programmable enough to support enhancements such as

- Source and channel coding experiments using simple codes such as Hamming codes or more advanced codes such as Reed-Solomon codes. Trellis-coded modulation and maximum-likelihood sequence estimation should be possible.
- Direct sequence-spread spectrum labs in which the signal bandwidth can be very large relative to the bitrate.
- Transmission and reception of real, useful data with a simple method of transferring that real data to a workstation.

The first goal is met with a highly programmable system using FPGAs and DSPs. The second goal is met with high-bandwidth components such as ADCs and DACs in conjunction with high-performance computation units such as FPGAs. The third requirement is discussed in the following section.

2.1.3 Effective and portable interface

In order that the new hardware be usable for years to come, it was necessary to incorporate a standard and portable interface. The interface also had to provide enough bandwidth for real-time transfer of data. After considering many current technologies such as IEEE 1394, USB, and ethernet, ethernet was chosen. Ethernet provides a remarkably portable interface while still providing transfer rates that are more than adequate. Leveraging this technology allows the new hardware to be used with a variety of workstations and with almost no bounds on location.

2.1.4 Programmability

One shortcoming of the ECE 320 course is that the lab equipment is poorly matched to digital communications projects. The course currently uses DSP evaluation modules which have

audio-grade converters and operate at common audio sample rates. The new system should provide an alternative development platform for these students that is appropriate for a variety of digital communications projects and has a short learning curve for students of ECE 320. This means that the sample rate should be programmable to almost any frequency in the range of the conversion system and that the conversion system be given a bandwidth larger than the conversion rate to allow for bandpass sampling. Antialias and reconstruction filters should be external to the system.

2.2 Technology Introduction

2.2.1 Digital signal processors

DSPs are special-purpose microprocessors designed for operating on signals. Their architecture has been tuned to operate very well in control, communication, audio, and other areas where low power consumption and programmable signal processing is needed. The differences between DSPs and general microprocessors are subtle and vary from one DSP architecture to another, but all DSPs contain some architectural optimizations for performing Fast Fourier Transforms (FFTs) and general filtering tasks (FIR or IIR).

The DSP used in ERITHACUS is a Motorola DSP56302 that runs at 66 MHz and can perform one instruction per cycle. The 56302 is very well suited to high-end audio applications due to its 24-bit data bus, but performs well in communications applications as well. In addition, the 56302 is easier to program in assembly than most of its competitors. Although the width of the 56302 datapath exceeds requirements for most communications tasks, it was chosen because current courses already use the 56302 and because of its ease of programming. For the target application and low quantity, competing DSPs offer no advantages to the 56302.

Most DSPs are characterized by several architectural features that make them different from general-purpose architectures. These differences include differences in bus architecture, ALU style, registers, and the control unit.

Most DSPs have a Harvard (as opposed to von Neumann) architecture bus which provides a separation between code and data sections of memory. Although recent DSPs have integrated small instruction caches, no current DSP has data cache. DSPs are often used in real-time situations and almost always operate intensively on constantly changing data. Therefore, a

data cache would provide limited usefulness. The Harvard architecture and instruction cache are provided in the interest of getting data into the DSP's ALU(s) as quickly as possible.

The DSPs that are most commonly used use a fixed-point representation. Although many DSPs contain floating-point ALUs, DSPs used for communications and digital audio are almost exclusively fixed-point. A fixed-point ALU has many advantages over a floating-point ALU, among these being lower complexity, lower power consumption, smaller die size, and faster operation. All DSPs contain a complete multiply-and-accumulate unit which can produce at least one result per instruction cycle. Often, these MAC units are pipelined to provide faster operation and eliminate critical paths at high clock speeds.

Unlike typical RISC microprocessors, which provide a large number of general-purpose registers, DSPs have many special-use registers that can be used for specific operations. For example, the Motorola DSP has four register types that correspond to address registers, offset registers, modulo registers, and ALU registers. Even among the ALU registers, segregation is present.

The control unit of a DSP is simplified compared to a full microprocessor, shifting the cost and power consumption of the DSP towards the ALU. The result is a simplified interrupt structure with very predictable results, but poorer performance in non-computationally-intensive situations. On the other hand, a few features are added, such as hardware support for loops, modulo addressing, bit-reversed addressing, and instruction set support for parallel memory access and ALU operations.

2.2.2 Field programmable gate arrays

FPGAs are large monolithic devices which contain hundreds or thousands of logic blocks. Each logic block contains a certain amount of programmable logic and/or memory depending on the FPGA architecture. Specifically, in the Xilinx 4000 series devices used on ERITHACUS, each block contains two D flip-flops, several control multiplexors, and two blocks that can function as either look-up tables, single- or dual-ported SRAM, or fast-carry logic for simple ALUs [14]. These blocks are arranged in regular patterns on the die and contain a large amount of programmable interconnect to perform the task of routing between logic blocks and to the input/output sections of the chip.

FPGAs have many advantages over DSPs for high-speed signal processing. Because the hardware is programmable, the architecture of the device is flexible. That is, although a DSP may be restricted to one or two multipliers, an FPGA can have many. Indeed, the precision of each multiplier can be tuned to fit the algorithm. Many communications tasks require the high-speed execution of a simple algorithm such as a 16-tap filter. An FPGA programmed with eight or 16 multipliers would be well suited to this type of work.

Unfortunately, the advantages of FPGAs come at a high cost. Large FPGAs are expensive due to the large amount of logic required to support flexibility. FPGAs are also more difficult to program; while a DSP algorithm can be written fairly quickly in the native assembly, the FPGA is still a hardware design platform. Therefore, a hardware designer must carefully architect hardware components together to form a datapath for the algorithm. Even for simple algorithms such as an IIR filter, FPGA design is far more difficult and time-consuming than DSP code development.

The FPGA architecture used in ERITHACUS is the Xilinx 4000. Specifically, a 4013XLA (576 logic blocks) is used in the transmitter and a 4062XLA (2304 logic blocks) is used in the receiver. Table 2.2 [15] shows the number of logic blocks required to implement some common algorithms.

Table 2.2 Logic Block Usage on Xilinx 4000 Devices.

Algorithm / Hardware Block	CLBs
16-bit 2's Complement Adder	9
8x10 Fixed-Point Multiplier	65
10-bit 24-tap Symmetric FIR Filter	101
1024-point FFT	532
Reed-Solomon Encoder	105
Reed-Solomon Decoder (8 bits per symbol)	941
Viterbi Decoder	425

2.2.3 VHSIC hardware description language

VHSIC Hardware Description Language (VHDL) is a language developed as a standard way to describe hardware components, their entity, and their functionality (either behaviorally or structurally) [16]. Originally developed as a language used for simulating hardware, VHDL has developed into a synthesis language. That is, the hardware designer can describe hardware and

have a compiler synthesize the design into a collection of small logic blocks. These logic blocks depend heavily on the target architecture (for example, ASIC and FPGA).

The main drawback to using VHDL for a description methodology is that the present synthesis technology is fairly young. The resulting design is almost always larger and slower than a hand-made design. Fortunately, the hardware design market is quickly driving VHDL synthesis tools to become better and more complete. Presently, however, a good trade-off must be made between using VHDL and customized blocks. In a hybrid design, VHDL is used to describe simple, well-synthesized hardware and custom blocks are provided which take full advantage of the target hardware. Several such blocks are described in [15].

Although there are many other ways to develop FPGA hardware descriptions, VHDL was chosen for the following reasons:

- Text-based description promotes portability to other systems and synthesis tools.
- Text-based description promotes self-documentation.
- Text-based descriptions are more flexible than schematic-based descriptions.
- Although synthesis tools are still somewhat immature, the design cycle with synthesis is much shorter than with custom schematics.

2.2.4 RTEMS

RTEMS is an RTOS originally commissioned by the United States Army for use in military missile systems. The RTEMS code base was placed into the public domain, but remains supported by OAR. Like any RTOS, RTEMS allows the programmer to start multiple tasks running at the same time. The operating system is responsible for scheduling tasks to execute and also handles any incoming and outgoing events that occur, such as interrupts. Full documentation for RTEMS is available in [17]. The following features highlight RTEMS's capabilities:

- Task manager for setting up multiple tasks with different scheduling strategies.
- Semaphore manager for managing resources.
- Message, event, and signal managers for passing messages between tasks.
- Interrupt, clock, and timer managers for coordinating time-sensitive events.

RTEMS is used on ERITHACUS as the support hub for system control. The CPU on ERITHACUS (Motorola 68EN360) is used to communicate with the various DSP and FPGA processors on-board as well as communicate with other devices through ethernet. Due to the CPU's many responsibilities, a multitasking OS was required. RTEMS was chosen for several reasons:

- Free software tools (compiler, assembler, etc.) available from GNU are mature, stable, and reliable.
- Source code for RTEMS is freely available (Open Source) and still under evolutionary development.
- Availability of a free integrated TCP/IP stack (ported from FreeBSD).

Using RTEMS on a target board such as ERITHACUS only requires the development of a board support package (BSP) that gives the OS access to the hardware through various initialization and interface routines. Once complete, the programmer has access to many operating system perks such as memory allocation, multithreading, and message passing. Development under RTEMS can be done in C, C++, or Ada. Finally, because the TCP/IP stack was ported from a UNIX variant, porting applications to the stack is relatively simple.

2.3 System Overview

ERITHACUS is a system of many interconnected blocks and as such requires something to tie it all together. The CPU (a Motorola 68EN360) controls all of the smaller components of the system. It interfaces to ethernet on one side, and to the system on the other. It can reset and download programs to all of the DSPs and FPGAs on the system. It can also communicate with any of these devices during operation to update coefficients, change waveforms, capture data, etc. Figure 2.1 shows a simplified block diagram of the entire ERITHACUS system and how the CPU is interconnected.

The CPU is directly supported by a standard 72-pin DRAM SIMM (up to 16 MB), two 1 MB flash memory devices, and an ethernet physical interface. The DRAM provides working memory while the flash memory provides nonvolatile storage for things such as boot code, configuration parameters, common DSP or FPGA images, and transmission waveforms. The CPU itself contains support circuitry for the ethernet standard, but relies on the Motorola

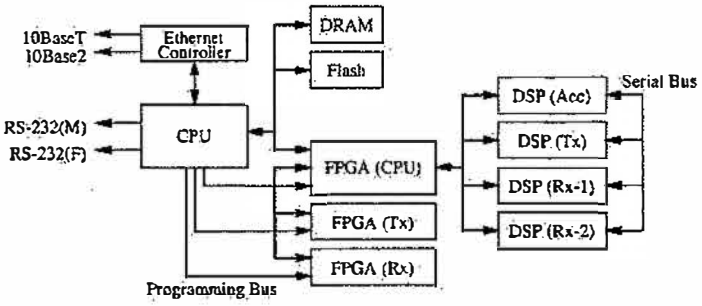

Figure 2.1 ERITHACUS Block Diagram.

68160 for the line drivers and receivers. Two RS-232 serial interface ports are provided. One is used as a monitor interface. The other can be used to connect a serial mouse or keyboard.

2.3.1 FPGA interface

There are three FPGAs on ERITHACUS. One (a Xilinx 4008E with 324 logic blocks) serves as a CPU interface block. Presently, this device does little more than address decoding for the rest of the system. The transmitter section has a Xilinx 4013XLA with 576 logic blocks to perform transmitter algorithms and flexible address generation for the wavetable synthesis. The third FPGA is a Xilinx 4062XLA with 2304 logic blocks located in the receiver section for hardware receiver functionality.

The three FPGAs are connected to the CPU through two separate interfaces, one for general communication and one for programming. A bus of interconnected programmable pins between the FPGAs serves as the general-purpose interconnect. Because the function of these pins is fully programmable, their use is determined by the specific images loaded onto the FPGAs.

The programming interface to each FPGA is a synchronous serial interface connected to the CPU general-purpose input/output pins. When an FPGA is reset, it expects an FPGA image to be sent as a bitstream through the synchronous interface. Once programming is complete, the programming interface is not used until there is a reset of the FPGA.

2.3.2 DSP interface

In addition to the FPGAs, the system has four Motorola 56302 DSPs. One is located in the transmitter section, two are connected through shared memory in the receiver section,

and the fourth is an "accessory" DSP with its own audio-grade CODEC system. The CPU communicates with four DSPs through a common 8-bit parallel bus. On the CPU side, this bus is mapped to the CPU's address space and on the DSP side, it corresponds to the DSP Host Interface. The bus provides both programming and communication capability. When a DSP is reset (by the CPU), it expects boot code to be sent through the host interface. The CPU provides this boot code. Once the DSP is operating, the CPU can interrupt the DSP to change parameters or request data. More elaborate communication protocols can be developed into software on the CPU and DSP to provide a continuous interchange of data.

The DSPs are also interconnected through a synchronous serial interface. This interface provides a means for the DSPs to communicate without going through the CPU. One situation in which this may be useful is if the accesory DSP takes samples from the audio CODEC, encodes them for transmission and sends the encoded bitstream to the transmitter DSP, then receives a similar bitstream from the receiver section and decodes it to be sent out the audio CODEC.

CHAPTER 3

TRANSMITTER ARCHITECTURE

The primary goal for the transmitter architecture is flexibility. Because a wide variety of modulation formats are available to the communications designer [18], the transmitter must be able to encode data in many ways, including baseband and passband formats.

A complex baseband design was chosen for ERITHACUS. This means that the transmitter output two baseband signals, $s_I(t)$ and $s_Q(t)$, which can be jointly modulated to form a passband signal $s_m(t)$ as in

$$s_m(t) = s_I(t)\cos(2\pi f_o t) + s_Q(t)\sin(2\pi f_o t) \tag{3.1}$$

where f_o is the carrier frequency. A block diagram of this quadrature modulation is shown in Figure 3.1. The quadrature modulator is a simple circuit with three inputs (carrier input, I input, and Q input) and a single output. Complex baseband signal generation is flexible enough to generate binary antipodal signals, BPSK, QPSK, n-QAM, FSK, DSSS, CDMA, AM, FM, and other modulation formats as long as the hardware to generate the baseband signal is flexible enough.

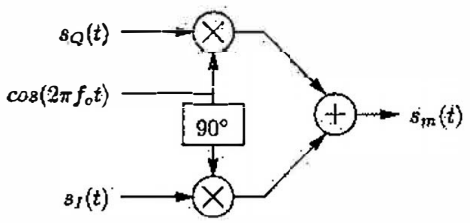

Figure 3.1 Quadrature Modulation.

3.1 System Architecture

3.1.1 Wavetable synthesis

Due to the flexibility requirements placed on the design, wavetable synthesis was chosen as the transmission method for ERITHACUS. In conjunction with quadrature modulation, many transmission schemes can be synthesized. The block diagram for a basic wavetable synthesizer is shown in Figure 3.2. The address generator produces consecutive addresses with some base address offset at the clock rate f_{clk}. These addresses select a word from the wavetable RAM that is fed to the DAC to produce an analog voltage. The base address offset changes at a clock rate lower than f_{clk} to select which waveform is being sent. In a digital transmitter, the waveform select signal acts as a symbol select (to determine which symbol is sent).

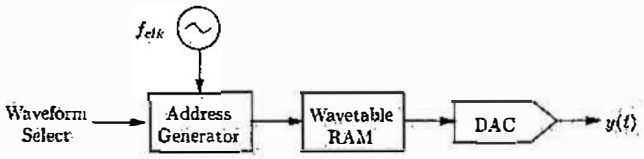

Figure 3.2 Wavetable Synthesis.

Given that the wavetable must produce an M-ary signal with N samples per symbol, the wavetable must be large enough to store $N \cdot M$ samples. The symbol rate is then limited to $f_{sym} = f_{clk}/N$. Therefore, as the target symbol rate is increased and the sample rate remains constant, the temporal resolution of the waveform must degrade. This is a basic trade-off present in any wavetable synthesis method. Fortunately, with proper reconstruction filters, as long as the bandwidth of the waveform does not exceed the Nyquist bandwidth of the conversion system, temporal resolution is not an issue.

The agility of wavetable synthesis methods lies in the programmability of the wavetable memory, but even more flexibility can be gained with a programmable address generator. A programmable address generator can be used to sequence transmitted waveforms in ways beneficial to digital communications. Such sequencing can be described as memory in the transmitter.

3.1.2 Waveform synthesis with memory

In the digital communications context, memory is usually used to describe the characteristics of a channel which 'remembers' past symbols. That is, the waveform that appears at the receiver is not only a product of the current symbol, but also some number, L, of previous symbols.

There are many communications applications in which symbol memory can prove useful in the synthesis of waveforms. At the transmitter, memory is useful in two areas: signal generation and coding. Some signaling waveforms are designed to last longer than a single symbol time. Examples are the SRRC pulse and the duobinary pulse [18]. Both pulses are designed to work in bandlimited channels and both have memory. Coding can be used in digital communication systems as a means of compressing the data before transmission, introducing redundancy to help detect or correct errors, or as a means of shaping the spectrum of the signal. In general, coding works on a sequence of symbols and therefore introduces memory into the transmitted signal. Although there are hardware-dependent limitations, many coding schemes can be implemented easily by organizing the wavetable.

The effects of a channel can also be described using memory. In these situations, a wavetable synthesizer with memory can emulate the channel characteristics, a valuable ability for a system used as a testbed for communications work. Figure 3.3 illustrates the channel emulation concept for a channel with a low-pass characteristic (assuming the signal is baseband). In this example, a square pulse has been broadened by the channel. The pulse now has four times its original temporal duration. Therefore, a wavetable synthesizer with $L = 3$ can be used to emulate this channel.

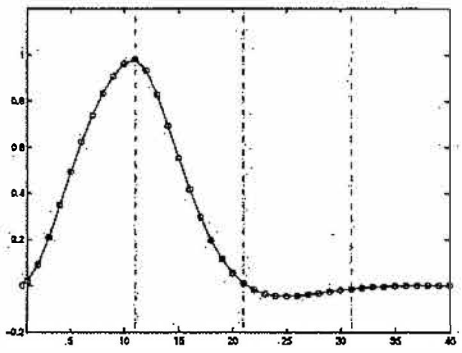

Figure 3.3 Segmented Bandlimited Pulse.

Figure 3.4 shows how a sequence of bits are affected by the channel discussed above. Shown below the filtered waveform are channel memory for each symbol epoch. In order to synthesize this waveform, the transmitter must superimpose segments from Figure 3.3 corresponding to the memory of the channel at each epoch. With wavetable synthesis methods, all possible superpositions must be calculated *a priori*.

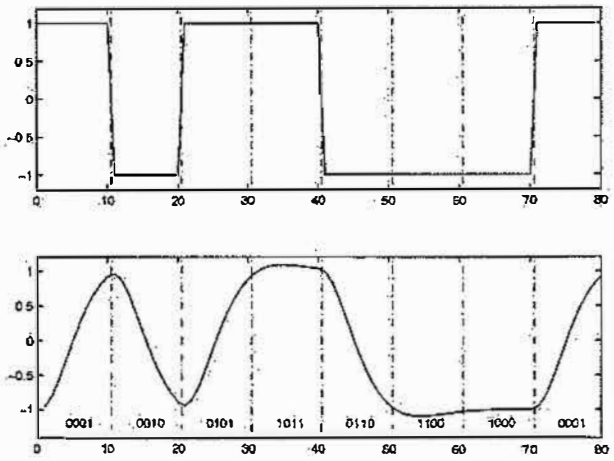

Figure 3.4 Transmission Sequence with ISI.

3.1.2.1 Channel coding

Many channel coding schemes can be described in terms of a simple state machine. These types of codes lend themselves to VHDL implementation due to the ease with which state machines can be implemented in VHDL. With this technique, input symbols are considered the input to the state machine and the current state corresponds to the symbol being sent.

3.2 Hardware Architecture

Figure 3.5 contains the block diagram of the system transmitter along with FPGA internals specifically designed for the ECE 363 course. The FPGA design currently contains two major blocks for wavetable synthesis: the source generator and the address generators. The source generator is responsible for synthesizing the bits to be sent out the transmitter. The address generators are responsible for sequencing wavetable addresses depending on the type of modulation for which they have been programmed.

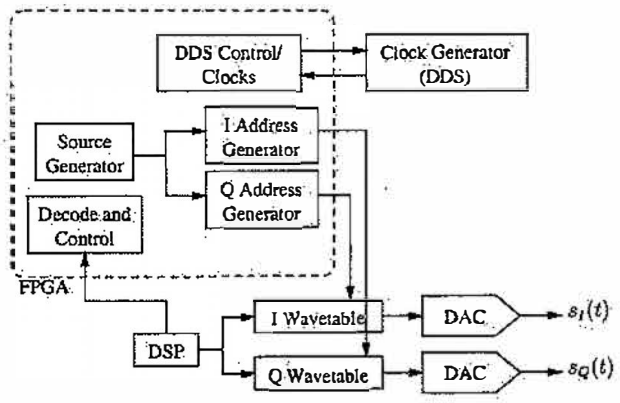

Figure 3.5 Transmitter Block Diagram.

Figure 3.6 shows the transmitter source generation hardware. Presently, four datasources are available: PN7, PN15, PN23, and a data FIFO. PN7, PN15, and PN23 are pseudorandom noise generators which use generator polynomials x^7+x^6+1, $x^{15}+x^{14}+1$, and $x^{23}+x^5+1$, respectively. The actual implementation makes better use of the available hardware by collapsing all three PN generators into a single 23-bit register. The next bit is then computed by selecting which modulo-2 addition is required (taps 6 and 7, taps 14 and 15, or taps 5 and 23). The data FIFO is a 16x16 bit FIFO that takes data from the DSP and syncronizes the data to the output bitrate. The DSP must be able to keep up with the rate or data will be lost. This datasource can be used to transmit real data rather than test sequences.

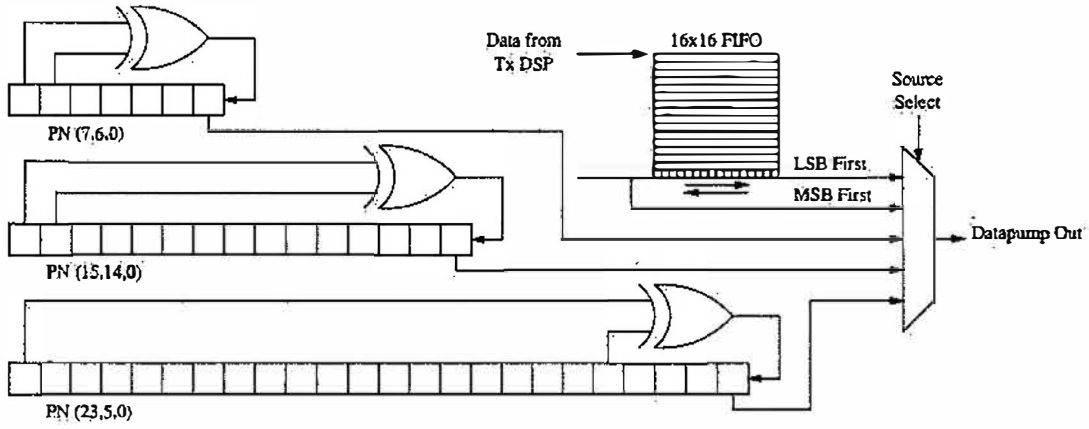

Figure 3.6 Transmitter Source Generation.

Figure 3.7 contains a simplified block diagram of the wavetable address generators. The word used to address the wavetable is composed of two parts: a sample index and a waveform offset. The sample index is k bits used to index into a particular waveform. All waveforms must be composed of the same number of bits. The waveform offset is constructed from the current data symbol and L previous data symbols. New data is shifted into the datapump every $N \cdot W$ samples and therefore the waveform offset changes every $N \cdot W$ samples. The variable W is used to indicate how many times a particular waveform should be repeated before shifting in another symbol. While this is not particularly useful in baseband transmission, it serves a purpose in passband signals.

For example, consider passband synthesis of BPSK with a symbol rate of 1 kHz and carrier at 130 MHz. We would like to use simple filters and therefore as high a sample rate as possible. Using a simple wavetable without repetition and a sample rate of 40 MHz, 40,000 samples would need to be stored for each symbol. With repetition, however, we can program a full waveform to be four samples. Therefore, $N = 4$ and $W = 10{,}000$. Without reconstruction filters, our signal appears at 10 MHz, 30 MHz, 50 MHz, 70 MHz, 90 MHz, 110 MHz, 130 MHz, and so on. We build a reconstruction filter to be centered at 130 MHz.

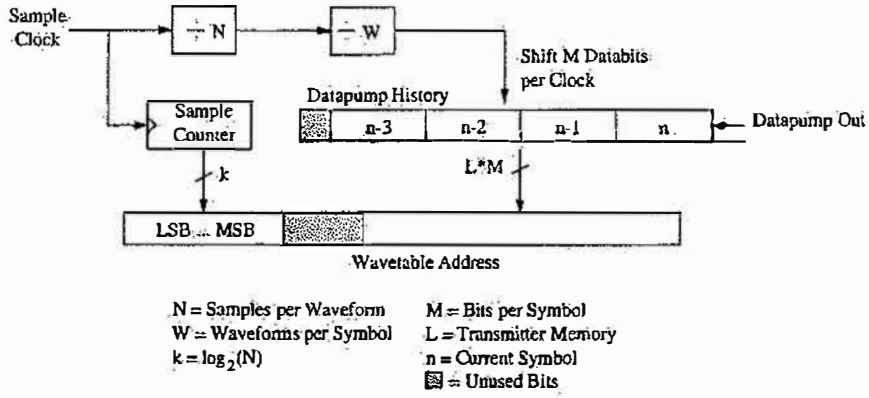

Figure 3.7 Wavetable Address Generation.

CHAPTER 4

RECEIVER ARCHITECTURE

This chapter discusses the final receiver architecture and how it can be used to construct a variety of receivers. Two levels of architecture are discussed: the system and the FPGA hardware receiver. The flexible system architecture is typically fairly simple because it is composed of reconfigurable computing blocks. The FPGA hardware receiver is somewhat more complex because it consists of the design elements of an actual receiver, with its datapath optimized for the FPGA architecture.

4.1 System Architecture

The digital receiver section of ERITHACUS was designed as a baseband (or low IF) receiver and is expected to be operated in a system similar to that shown in Figure 4.1. It samples two bandlimited channels, In-phase (I) and Quadrature (Q), and uses those samples to perform carrier recovery and make symbol decisions.

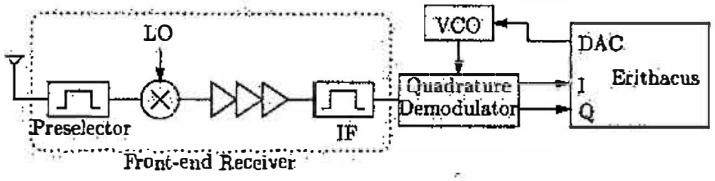

Figure 4.1 A Complete Digital Receiver Using ERITHACUS.

A block diagram of the entire receiver section is shown in Figure 4.2. The receiver contains two 10-MSPS ADCs for I and Q sampling, a Xilinx XC4062XLA FPGA, two Motorola 56302

DSPs sharing 16 kwords of dual-ported SRAM, a DDS-based clock generation circuit, and four 100-MSPS DACs for various output tasks.

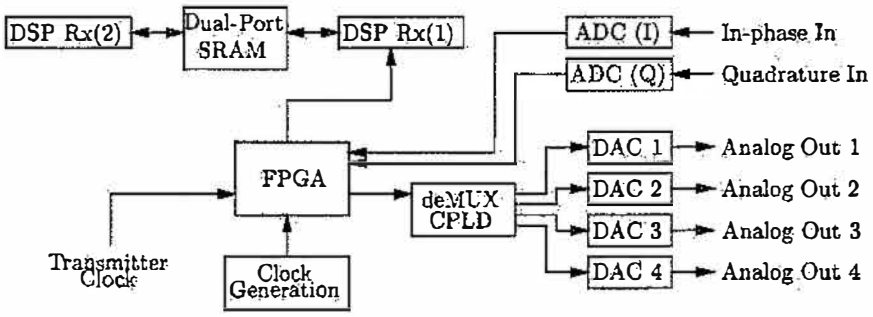

Figure 4.2 Block Diagram of ERITHACUS Receiver Section.

The sample clock for the receiver section may come from either the transmitter section or the dedicated clock generation circuit in the receiver. In some lab setups, it is necessary to synchronize the transmitter and receiver in order to eliminate the need for timing recovery. If this synchrony is not required, the receiver can generate its own sampling clock up to 42 MHz with 30-mHz resolution. The ADCs in the receiver, however, are limited to 10 MSPS. Although limited to 10 MSPS, the ADCs have an input bandwidth of 60 MHz [19]. As a result, the ADCs can perform bandpass sampling as long as the input signal does not exceed the Nyquist criterion.

Four DACs are present in the receiver design for several reasons. They provide a simple method of getting internal signals out of the digital domain for observation on an oscilloscope or spectrum analyzer. The DACs can also output signals for closed-loop control as in PLL applications. Due to pin-count restraints on the FPGA, the four DACs are multiplexed to a single bus on the FPGA. In order to output a sample to all four DACs, the FPGA requires four clock periods. The design intention was that the clock generation would run at four times the sample rate and be divided down by the FPGA to input and output samples synchronously.

The computational units of the receiver are the two DSPs and the FPGA. The partitioning of any particular receiver is very open and flexible. The two DSPs are more appropriate for designs at lower rates (300 ksps and below) or high complexity because their architecture is a good-enough fit for these rates. In addition, they are much easier to target than the FPGA,

significantly reducing implementation time. At higher rates, the FPGA is better suited because multiple functional units can be instantiated that operate concurrently on the incoming data. In many cases, a hybrid design may work very well. In a hybrid, the DSPs take care of the complex, lower-rate computations such as AGC or decoding while the FPGA handles the higher-rate problems such as matched filtering, carrier recovery, and timing recovery.

4.2 Hardware Receiver Architecture

The ECE 363 lab equipment runs at a symbol rate of 1 MHz and sample rate of 10 MSPS. For these rates, a hardware design is necessary. This section presents the particular hardware receiver used in the ECE 363 course. Because the FPGA is reprogrammable, this is certainly not the only hardware receiver possible. It is, however, tuned for the uses of the course and is a good starting point for other hardware receiver designs.

The overall block diagram is shown in Figure 4.3. The in-phase samples come from the I ADC and the quadrature samples come from the Q ADC. The output stage controls the DAC demultiplexor which sends samples to the DAC. Not shown in the diagram, but implicit to any FPGA receiver design, is the interface to the DSP which, in this case, performs little more than a supervisory task. The DSP configures certain parameters of the architecture and queries other parameters such as error count, bit count, etc. Technical details on these parameters are given in Appendix B.

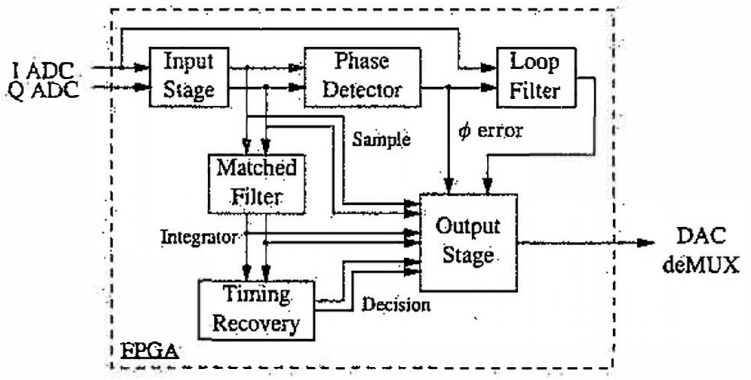

Figure 4.3 Block Diagram of the Hardware Receiver.

After samples are acquired by the ADCs, they are sent directly to the FPGA. Here, they enter the input stage shown in Figure 4.4. The ADCs are DC-coupled to allow low-frequency receiver designs. As a result, the ADCs will add a small amount of DC offset to the input signal which must be removed to avoid demodulation errors. A simple adder as shown in the figure is adequate. Following offset adjustment is a gain adjustment (10x8 multiplier) for automatic gain control. The I Gain and Q Gain values can come from an external source such as the DSP.

Following the offset and gain adjustment, I and Q samples are sent to a modified Costas phase detector [20]. The modified form computes the equation

$$y_n = x_{n,Q} \cdot \text{sign}\, x_{n,I} \tag{4.1}$$

for BPSK mode and

$$y_n = x_{n,Q} \cdot \text{sign}\, x_{n,I} - x_{n,I} \cdot \text{sign}\, x_{n,Q} \tag{4.2}$$

for QPSK mode, where $x_{n,I}$ and $x_{n,Q}$ are the current samples from the in-phase and quadrature ADCs, respectively, y_n is the output of the phase detector, and sign $x = +1$ for $x > 0$ and -1 for $x < 0$.

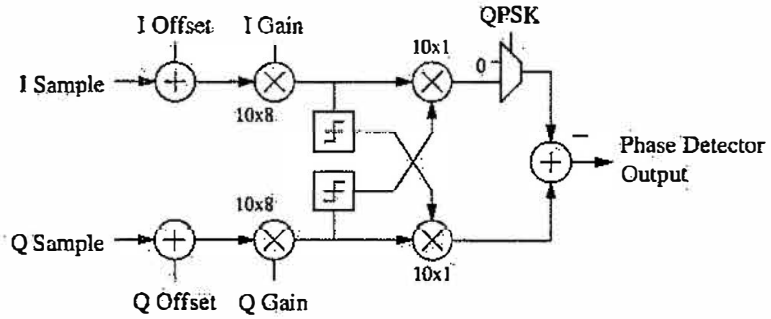

Figure 4.4 Receiver Input Stage and Phase Detector.

The phase detector output is sent to the loop filter. For testing purposes, the loop filter can derive its input from the I input as well, as seen in Figure 4.5. The loop filter is composed of three multipliers and three adders. The block is implemented in fixed-point arithmetic with the precisions of each signal shown above a hatch on that signal. The filter implements the first-order IIR filter with transfer function

$$H(z) = \frac{a}{1 - (1-a)z^{-1}} \tag{4.3}$$

where the constant a is between 0 and 1.0. The time-domain recursion for this filter is

$$y_n = ax_n + (1-a)y_{n-1} \tag{4.4}$$

Due to pipelining constraints (see Section B.6), the recursion that is actually implemented is

$$y_n = ax_n + (a-a^2)x_{n-1} + (1-2a+a^2)y_{n-2} \tag{4.5}$$

$$y_n = ax_n + bx_{n-1} + (1-c)y_{n-2} \tag{4.6}$$

where $b = (a-a^2)$ and $c = (2a-a^2)$. The relationship between this recursion and the block diagram should be evident. The constants in the diagram (A, B, and C) correspond to fixed-point versions of the constants a, b, and c, respectively.

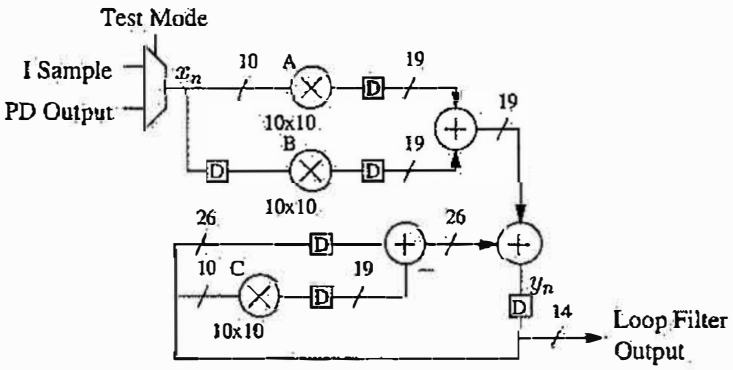

Figure 4.5 Receiver Loop Filter.

A note about the multiplier implementation is appropriate here. To be precise, a multiplier with two 10-bit inputs will require 21 bits to fully represent the result. All 21 bits are required for only one pair of inputs (-512 and -512). With a restriction on the PLL coefficients, this pair will never occur. Therefore, the MSB of the result is discarded.

The matched filter and timing recovery components from Figure 4.3 are difficult to describe in block diagram form. The matched filter is implemented as an integrate and dump (or, more appropriately, a sum and clear) design. The receiver can also be placed in a midsymbol sampling mode which bypasses the integrate and dump filter. The symbol decision is based on a single I/Q sample. Timing recovery is implemented as a common early/late gate [18], [21].

The last major component of the hardware receiver is the output stage shown in Figure 4.6. It consists of four multiplexors, one for each DAC signal. The select signals to the multiplexors

can be changed at any time. Software has been written to configure these multiplexors so that students can view different signals in the signal path. See Appendix C for more details.

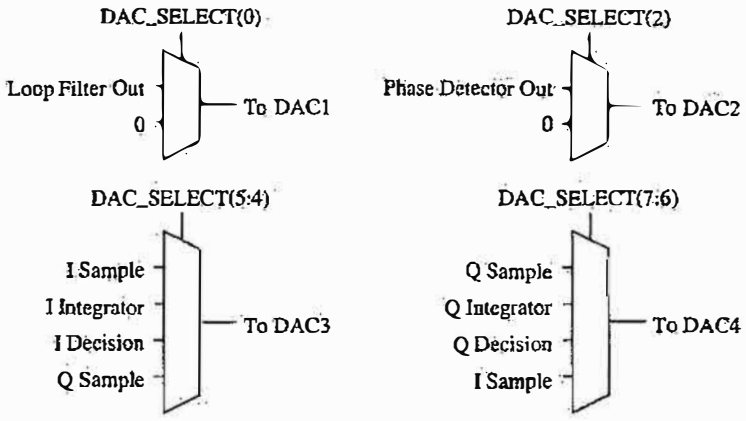

Figure 4.6 Receiver Output Stage.

CHAPTER 5

CONCLUSION

A reconfigurable system, designed for use in digital signal processing and digital communications, has been presented. The system has already been found capable of replacing the current analog-based hardware used in the educational lab for which it was designed. Due to its digital design, the system should prove to be far more reliable than the current hardware.

A hybrid design approach of both software-based components (DSPs) and hardware-based components (FPGAs) was taken to provide both high-speed capability at the cost of design time (hardware) and ease of programmability at the cost of operation speed (software). Thus far, the approach has paid off; both software and hardware receivers have been developed on the system.

Additional labs will be developed to take advantage of the new hardware. An example of this is a coding lab which will introduce students to the necessity for channel coding in a useful communications link. The analog-based receiver does not easily enable the creation of such a design, whereas the new hardware makes this development somewhat painless. The implementation of a spread spectrum transmitter and receiver would also be possible and is a likely extension given the importance of spread spectrum technology.

Ethernet capability has been a remarkably useful aspect of the ERITHACUS design. While it not only provides rapid reconfiguration from well-known and ubiquitous software protocols such as ftp and telnet, it also allows control of remotely located ERITHACUS devices by Matlab or Java clients. In the two years since ethernet was chosen for ERITHACUS, the Internet explosion has lead to an unparalleled ubiquity of ethernet presence, further assuring that the proper choice had been made.

In addition to applications in digital communication education, ERITHACUS should prove useful in research activities. Palac [22] presents a communications testbed for use in joint source-channel algorithm research. Such equipment could be used jointly with ERITHACUS for a more thorough coverage of requirements.

APPENDIX A

TRANSMITTER TECHNICAL INFORMATION

A.1 Transmitter FPGA Registers

Although the hardware is reprogrammable, the specific transmitter FPGA image developed for the Digital Communications Lab is discussed here. Not only does this FPGA image provide a good starting point for other designs, but it is also flexible enough to be useful in most transmitter configurations. This section describes the parameters that can be modified in the transmitter FPGA. Table A.1 lists the registers available to the transmitter DSP. All registers are write-only.

The DSP maps the FPGA to a certain address space. This is typically the block starting at x:$20000, but is reprogrammable. It is set by the DSP boot code. When writing to the FPGA, the 16 least significant bits of the source are written. The eight most significant bits are ignored. An example follows.

```
BITS_PER_SYMBOL equ     $20008
...
; Set the transmitter for 4 bits per symbol.
        move    #$000004,x0
        move    x0,x:BITS_PER_SYMBOL
```

A.1.1 Register descriptions

DDS_RESET

This register is used to reset the transmitter DDS. The data written to this register are ignored, but the DDS is immediately reset. The DDS should be reset prior to programming.

Table A.1 Transmitter FPGA Registers.

Address	Register	Description
$00	DDS_RESET	Write causes DDS reset.
$01	DDS_16BIT	Write 16 bits to DDS.
$02	DDS_8BIT	Write 8 bits to DDS and update.
$03	WAVE_CE	Chip enable for wavetable outputs.
$04	DATAPUMP_RESET	Write causes datapump logic to reset.
$05	SOURCE_SELECT	Selects data source for transmitter.
$06	COUNTER_MASK	Mask used for sample counter (related to samples per waveform).
$07	DATA_MASK	Mask used for data (related to bits per symbol).
$08	SAMPLES_PER_WAVEFORM	Number of samples per waveform.
$09	WAVEFORMS_PER_SYMBOL	Number of waveforms per symbol.
$0A	BITS_PER_SYMBOL	Number of bits per symbol.
$0B	I_BASE_ADDRESS	Base address of I lookup table.
$0C	Q_BASE_ADDRESS	Base address of Q lookup table.
$10	FIFO_DATAIN	Writes a 16-bit data word to the datapump FIFO.

DDS_16BIT

This register is used to shift 16 bits of update information into the DDS. Once written, a state machine is started to transfer the data to the DDS. Therefore, the DSP must wait until the state machine is complete before writing to the DDS again. This time is typically 60-100 DSP clock cycles.

DDS_8BIT

This register is used to shift 8 bits of update information into the DDS. Once written, a state machine is started to transfer the data to the DDS. When the state machine is complete, the DDS is instructed to update its frequency and phase registers. The DSP must wait until the state machine is complete before writing to the DDS again.

WAVE_CE

The least significant bit of this register is used to control the chip enable ($\overline{CE}$) of both wavetables. Because the wavetables are dual-ported, a simultaneous read and write at the

same address is not allowed. Therefore, for proper wavetable updates, the DSP must either guarantee that it is not writing to the same addresses that are being read out (as transmission symbols) or the DSP must disable the wavetables for a short time while writing.

Writing a '0' to this register enables the wavetables. Writing a '1' disables them.

DATAPUMP_RESET

This register is used to reset the transmitter datapump. Data written to this register are ignored. When written, the FPGA resets the PN registers and clears all counters used to generate symbols. In addition, a state machine is started that clears the datapump FIFO.

SOURCE_SELECT

This register is used to select the source for bits sent to the datapump. See Figure 3.6. Table A.2 shows the available sources and their corresponding settings for this register.

Table A.2 Transmitter Data Sources.

Data Source	SOURCE_SELECT
PN (7, 6, 0)	$00
PN (15, 14, 0)	$01
PN (23, 5, 0)	$02
FIFO (LSB first)	$04
FIFO (MSB first)	$05

COUNTER_MASK

The value set in this register is used as a mask to transfer the sample count to the wavetable address. In most cases, COUNTER_MASK will contain $\log_2$(SAMPLES_PER_WAVEFORM) ones. For example, if SAMPLES_PER_WAVEFORM=32, COUNTER_MASK=$001F so that five bits of the counter are transferred to the wavetable address.

DATA_MASK

The value set in this register is used as a mask to transfer the datapump history to the wavetable address. In most cases, DATA_MASK will contain $(L+1) \cdot M$ ones where L is the

transmitter memory (in symbols) and M is the number of bits per symbol. For example, with two bits per symbol and transmitter memory of four symbols, DATA_MASK=$00FF.

SAMPLES_PER_WAVEFORM

This register specifies the number of samples per waveform. An actual symbol can be composed of many repeated waveforms.

WAVEFORMS_PER_SYMBOL

This register specifies the number of waveforms used to construct a symbol. If this value is greater than one, the waveform must be periodic because the same waveform will be repeated WAVEFORMS_PER_SYMBOL times.

BITS_PER_SYMBOL

This register specifies how many bits will be taken from the transmitter data source for each symbol. Closely related to this is the constellation size. Due to the design of the transmitter, SAMPLES_PER_WAVEFORM must be greater than BITS_PER_SYMBOL.

I_BASE_ADDRESS and Q_BASE_ADDRESS

This register sets an offset into the wavetable for use in address generation. This offset is most useful when real-time changes of the transmission waveforms are necessary. Two complete wavetables can be stored in memory. While the DSP is updating one wavetable, the transmitter is using the other. Writing to this register swaps the wavetables.

FIFO_DATAIN

This register is used to enqueue a single 16-bit word into the transmitter FIFO. The FIFO can contain up to 16 words. When the FIFO is half empty, the FPGA generates an interrupt to the DSP. The DSP must then write eight words to this register.

A.2 DSP/Wavetable Interface

The I and Q wavetables are mapped as write-only to DSP memory at `X:$10000` and `Y:$10000`, respectively. Specifically, bits 9-22 of the DSP databus are mapped to the 14 most

significant bits of the 16-bit wide dual-port SRAMs. This is indicated in the schematic sheet in Figure D.9. Note that the most significant bit of the DSP bus is not connected to the wavetables. The DSP represents numbers in two's complement. Therefore, this bit is a sign bit. The DACs, however, use a linear mapping from negative full-scale to positive full-scale. As a result of this wiring, DSP words should be formatted between 0.0 and 1.0 (fixed point) to correspond to the full DAC range.

Numbers represented with the full DSP range (-1.0 to 1.0) can easily be converted into the range required by the DACs by dividing by two (right-shift by one bit) and then adding 0.5.

A.3 Output Buffering

Figure D.9 shows the schematics for the output buffering circuitry. The DAC outputs are fed into a current-feedback amplifier for buffering. The output stage is DC-coupled to allow for very low bit-rate transmission. When coupled to a 50-Ω load, the load should see a +/- 1.0 voltage swing corresponding with the +/- full-scale output of the DAC.

Because the outputs are dc-coupled, a bias adjustment is provided to zero the offset voltage when the DAC output is zero. This bias adjustment is provided as a ten-turn potentiometer. To adjust the bias, the transmitter is configured to output constant zero, and the output stage is terminated with a 50 Ω load. Finally, the bias potentiometer is adjusted until the voltage at the output is zero.

A.4 Sample Rate DDS

The DDS devices in the transmitter and receiver are fed from the same 125-MHz clock source. The output clock frequency (f_{out}) is adjustable in 29-mHz increments from DC (no clock) to about 42 MHz by the word written to the DDS_16BIT and DDS_8BIT registers. Thirty-two bits of this 40-bit word determine the sample rate. The other eight bits determine phase and special functions (see [23] for details). The 32-bit word F_{val} is related to f_{out} by the following equation:

$$f_{out} = 125000000 \cdot \frac{F_{val}}{2^{32}} \text{ Hz} \tag{A.1}$$

The code in Section A.4.1 sets the transmitter sample rate to 40 MHz. Appropriate delays are inserted to allow the DDS programming state machines to complete after each write to

the DDS registers. Immediately after a reset, the DDS must receive a FQ_UD. Although not documented, this requirement has been consistent. Note that the 16-bit values written to the DDS registers are right-justified in the DSP's 24-bit operands.

A.4.1 DDS configuration sample code

```
DDS_RESET       equ     $20000
DDS_16BIT       equ     $20001
DDS_8BIT        equ     $20002

; Configure the DDS frequency (40 MHz)
        move    #>$000000,x0            ; Reset the DDS
        move    x0,x:DDS_RESET
        rep     #30
        nop
        move    #>$000000,x0            ; Flip the FQ_UD (must do after reset)
        move    x0,x:DDS_8BIT
        rep     #200
        nop

        move    #>$008520,x0            ; Setup the DDS for 40 MHz
        move    x0,x:DDS_16BIT
        rep     #200
        nop
        move    #>$0051eb,x0
        move    x0,x:DDS_16BIT
        rep     #200
        nop
        move    #>$000000,x0
        move    x0,x:DDS_8BIT
        rep     #200
        nop
```

APPENDIX B

RECEIVER TECHNICAL INFORMATION

B.1 Software Receiver FPGA Registers

Table B.1 lists the registers available to the receiver DSP.

Table B.1 Software Receiver FPGA Registers.

Address	Register	R/W	Description
$00	DDS_RESET	W	Write causes DDS reset.
$01	DDS_16BIT	W	Write 16 bits to DDS.
$02	DDS_8BIT	W	Write 8 bits to DDS and update.
$10	DACOUT1	W	Digital to analog converter 1.
$11	DACOUT2	W	Digital to analog converter 2.
$12	DACOUT3	W	Digital to analog converter 3.
$13	DACOUT4	W	Digital to analog converter 4.
$14	I_INPUT	R	I analog to digital converter.
$15	Q_INPUT	R	Q analog to digital converter.

B.1.1 Register descriptions

DDS_RESET

This register is used to reset the receiver DDS. The data written to this register are ignored, but the DDS is immediately reset.

DDS_16BIT

This register is used to shift 16 bits of update information into the DDS. Once written, a state machine is started to transfer the data to the DDS. Therefore, the DSP must wait until

the state machine is complete before writing to the DDS again. This time is typically about 100 DSP clock cycles.

DDS_8BIT

This register is used to shift 8 bits of update information into the DDS. Once written, a state machine is started to transfer the data to the DDS. When the state machine is complete, the DDS is instructed to update its frequency and phase registers. The DSP must wait until the state machine is complete before writing to the DDS again.

DACOUT1, DACOUT2, DACOUT3, DACOUT4

The values written to these registers will appear at the output of DAC1, DAC2, DAC3, and DAC4, respectively. The values take effect on the next sample clock.

I_INPUT, Q_INPUT

The values read from these registers are the current samples from the in-phase and quadrature ADCs, respectively.

B.2 Hardware Receiver FPGA Registers

The hardware receiver FPGA image is used when high data rates must be accomodated. This image provides complete demodulation including carrier recovery, matched filtering, and timing recovery. Although the DSP does little computation in hardware receiver mode, it is used as a communication link between the CPU and FPGA. Table B.2 lists the registers available to the receiver DSP. The next section describes these registers.

B.2.1 Register descriptions

DDS_RESET

This register is used to reset the receiver DDS. The data written to this register is ignored, but the DDS is immediately reset.

Table B.2 Hardware Receiver FPGA Registers.

Address	Register	R/W	Description
$00	DDS_RESET	W	Write causes DDS reset.
$01	DDS_16BIT	W	Write 16 bits to DDS.
$02	DDS_8BIT	W	Write 8 bits to DDS and update.
$04	BER_RESET	W	Write causes BER counters to reset.
$05	BER_ENABLE	W	Enables or disables the BERT.
$06	RX_MODE	W	Determines the receiver mode.
$07	DAC_SELECT	W	Select bits for receiver DAC outputs.
$08	LOOPF_SELECT	W	Loop filter input select.
$10	I_OFFSET	W	I ADC offset adjust.
$11	Q_OFFSET	W	Q ADC offset adjust.
$12	I_GAIN	W	I input gain.
$13	Q_GAIN	W	Q input gain.
$14	PLL_COEF1	W	PLL Coefficient.
$15	PLL_COEF2	W	PLL Coefficient.
$16	PLL_COEF3	W	PLL Coefficient.
$20	BIT_COUNT_H	R	Received bit count (bits 31:16)
$21	BIT_COUNT_L	R	Received bit count (bits 15:0)
$22	ERROR_COUNT_H	R	Error count (bits 23:16)
$23	ERROR_COUNT_L	R	Error count (bits 15:0)
$28	I_SAMPLE	R	I sample (2s complement after offset adjust).
$29	Q_SAMPLE	R	Q sample (2s complement after offset adjust).

DDS_16BIT

This register is used to shift 16 bits of update information into the DDS. Once written, a state machine is started to transfer the data to the DDS. Therefore, the DSP must wait until the state machine is complete before writing to the DDS again. This time is typically about 100 DSP clock cycles.

DDS_8BIT

This register is used to shift 8 bits of update information into the DDS. Once written, a state machine is started to transfer the data to the DDS. When the state machine is complete, the DDS is instructed to update its frequency and phase registers. The DSP must wait until the state machine is complete before writing to the DDS again.

BER_RESET

When written to, the bit error rate tester is reset. The bit count, error count, and detector registers are cleared.

BER_ENABLE

The least significant bit of this register determines if the BER tester is enabled. When set, the BERT is turned off and no counts are updated. When clear, the BERT is enabled.

RX_MODE

The least significant bits in this register determine the operating mode of the receiver, as shown in Table B.3.

Table B.3 RX_MODE Register.

RX_MODE	Receiver Mode
xxx0	BPSK phase detector.
xxx1	QPSK phase detector.
xx0x	Matched filter disabled.
xx1x	Matched filter enabled.
x0xx	Timing recovery disabled.
x1xx	Timing recovery enabled.
0xxx	Receiver clock source is DDSTX.
1xxx	Receiver clock source is DDSRX.

DAC_SELECT

The bits in this register determine what is sent to the four DACs in the receiver section, as shown in Table B.4.

LOOPF_SELECT

The least significant bit of this register determines the input for the loop filter. When clear, the loop filter input comes from the phase detector. When set, the loop filter gets its input from the in-phase ADC.

Table B.4 DAC_SELECT Register.

DAC_SELECT	DAC output
xxxxxxx0	DAC1 = Loop filter output.
xxxxxxx1	DAC1 = 0
xxxxx0xx	DAC2 = Phase detector output.
xxxxx1xx	DAC2 = 0
xx00xxxx	DAC3 = Quadrature ADC sample.
xx01xxxx	DAC3 = Quadrature matched-filter output.
xx10xxxx	DAC3 = Quadrature decision.
xx11xxxx	DAC3 = In-phase ADC sample.
00xxxxxx	DAC4 = In-phase ADC sample.
01xxxxxx	DAC4 = In-phase matched-filter output.
10xxxxxx	DAC4 = In-phase decision.
11xxxxxx	DAC4 = Quadrature ADC sample.

I_OFFSET, Q_OFFSET

The values written to these registers specify the values to be added to incoming in-phase and quadrature ADC samples, respectively. This can be used to adjust for the DC-coupled offset error of the ADCs.

I_GAIN, Q_GAIN

The values written to these registers specify the constant gain multipliers applied to the incoming in-phase and quadrature, respectively. The gain is applied after offset adjustment.

PLL_COEF1, PLL_COEF2, PLL_COEF3

These three values specify the loop filter coefficients used in the phase-locked loop. For more details on programming the loop filter, see Section B.6.

BIT_COUNT_H, BIT_COUNT_L

These two registers can be read to fetch the upper and lower 16 bits of the 32-bit bit count register, respectively.

ERROR_COUNT_H, ERROR_COUNT_L

These two registers can be read to fetch the upper and lower 16 bits of the 32-bit error count register, respectively. Because of the nature of pseudorandom sequence error counting, the result here is approximately three times the true error count. Host software is responsible for dividing this value by three to determine bit error rate.

I_SAMPLE, Q_SAMPLE

These two registers can be read to fetch the current 12-bit sample from the in-phase and quadrature ADCs, respectively. The values are left-justified to fit the fractional representation of the DSP.

B.3 Analog Inputs

The two analog inputs on ERITHACUS are dc-coupled 50-Ω inputs with a specified maximum sampling rate of 10 MSPS and a bandwidth of approximately 100 MHz to allow for bandpass sampling. The ADCs have been successfully driven past 20 MSPS with no noticeable degradation.

Due to the dc-coupling, some offset error may be present on the digital value read from the ADCs. Therefore, offset adjustment is provided by the FPGA and should be performed when the inputs are terminated with a 50-Ω load.

B.4 Analog Outputs

Because the receiver design is configurable, its intermediate results are only available inside the DSP or FPGA. This makes debugging and evaluating the receiver slightly difficult. Therefore, four analog outputs are provided as virtual testpoints for the receiver. The design of these analog outputs is identical to those in the transmitter portion of ERITHACUS. In software receiver mode, the DSP can write a fixed-point value to any of the DAC registers and have it appear as a voltage at the output (-1.0 corresponds to -1.0 V, +0.125 corresponds to +0.125 V, and so on). In hardware receiver mode, the FPGA controls what is sent to each DAC. Flexible routing inside the FPGA allows the DAC outputs to be configured dynamically by the user.

DACOUT1 has 14-bit resolution, DACOUT2 has 13-bit resolution, and DACOUT3 and DACOUT4 each have 12-bit resolution. The differences in resolution were an outcome of physical design constraints (pin count limitations). As a result, DACOUT1 should be used when higher resolution is beneficial, such as when controlling a VCO. DACOUT3 and DACOUT4 can be used for less critical outputs such as observing internal intermediate signals.

B.5 Sample Rate DDS

The receiver section has a DDS to control sample rate, similar to the one in the transmitter section. For more details, see Section A.4.

Programming the receiver DDS is similar to programming the transmitter DDS (Section A.4.1). However, while words written to the transmitter FPGA need to be right-justified, words written to the receiver FPGA need to be left-justified. Therefore, a sequence such as

```
move    #>$0051eb,x0
move    x0,x:DDS_16BIT
```

would be replaced by:

```
move    #>$51eb00,x0
move    x0,x:DDS_16BIT
```

B.6 PLL Coefficient

The loop filter in the hardware receiver PLL is a first-order IIR filter with transfer function

$$H(z) = \frac{a}{1-(1-a)z^{-1}} \tag{B.1}$$

The 3-dB point is defined as

$$|H(e^{j\omega})|^2_{\omega=\omega_{3dB}} = \frac{1}{2}|H(0)|^2 \tag{B.2}$$

Solving for ω, we get

$$|H(e^{j\omega})|^2 = \frac{a^2}{1-(1-a)e^{-j\omega})(1-(1-a)e^{j\omega})} \tag{B.3}$$

$$= \frac{a^2}{1-2(1-a)\cos\omega+(1-a)^2} \tag{B.4}$$

$$= \frac{1}{2} \text{ (at the 3 dB point)} \tag{B.5}$$

$$2a^2 = 1-2(1-a)\cos\omega+(1-a)^2 \tag{B.6}$$

$$\omega = \cos^{-1}[-\frac{2a^2-1-(1-a)^2}{2(1-a)}] \tag{B.7}$$

Therefore, an expression for the 3-dB point (in Hz) in terms of the single degree of freedom (a) is given by

$$f_{3dB} = \frac{F_s}{2\pi}\cos^{-1}[-\frac{2a^2-1-(1-a)^2}{2(1-a)}] \tag{B.8}$$

The time-domain recursion for the filter in (B.1) is

$$y_n = ax_n + (1-a)y_{n-1} \tag{B.9}$$

where x_n is the input sequence, y_n is the output sequence at time n, and a is the PLL coefficient $0 < a < 1.0$. The multipliers realized in the FPGA implementation are 10-bit by 10-bit multipliers from the Xilinx CORE library [15] with a latency of five cycles. The multipliers run at a clock rate that is four times that of the sample rate. Therefore, their effective latency is two cycles. As a result, look-ahead [24], [25] must be applied once, resulting in the following transformed equation:

$$y_n = ax_n + (a-a^2)x_{n-1} + (1-2a+a^2)y_{n-2} \tag{B.10}$$

$$y_n = ax_n + bx_{n-1} + (1-c)y_{n-2} \tag{B.11}$$

where $b = (a-a^2)$ and $c = (2a-a^2)$. These coefficients are represented internally as 10-bit signed fixed-point numbers. Because a is typically very small, its value is prescaled by 128 to avoid requiring a large multiplier. Once an appropriate value for the coefficient a has been found using (B.8), b and c can be determined. Finally, conversion to fixed-point notation is required.

As an example, suppose we want the loop filter to have a cutoff frequency of 500 Hz. Then $a \approx 3.0510^{-4}$. Prescaling by 128 and converting to 10-bit signed fixed-point yields

$A = 3.0510^{-4} \cdot 128 \cdot 512 \approx 20$. Therefore, $A = 20$, $B = 20$, and $C = 40$. Here, A, B, and C are the integer coefficients representing a, b, and c, respectively. Typically, for the range of interest in ECE 363, $A = B = \frac{1}{2} \cdot C$. Therefore, only the value for A can be entered in the Receiver Control Panel.

APPENDIX C

SOFTWARE SUPPORT

C.1 Device-Side Software

The device-side software is software written to interact under the RTEMS operating system. The code is written in C and is a collection of procedures that run as threads under the OS. All of the software is loaded at one time during boot.

C.1.1 FTPD

A subset of a UNIX file transport protocol daemon (FTPD) has been written for ERITHACUS. The ERITHACUS FTPD[1] supports nearly all of the most-used capabilities of UNIX FTPD and can interface with standard FTP clients. The FTPD written for this thesis has been submitted for inclusion in future releases of RTEMS and will be available in RTEMS 4.1.0.

A standard FTP daemon merely receives files from a client and places those files on the server (or vice versa). In addition, some file manipulation capabilities are provided such as remove, make directory, etc. A list of the supported FTP directives is shown in Table C.1.

To better facilitate interoperability with the programmable hardware on ERITHACUS, FTPD "hooks" are provided. A hook is a filename with a special meaning attached. When a file is PUT to a recognized hook, FTPD performs a special operation on that data. The hooks provided for the standard ERITHACUS firmware are shown in Table C.2.

The format of the file depends on its destination on ERITHACUS. ERITHACUS allows the Xilinx FPGAs in the transmitter and receiver to be programmed through FTP. The file is a standard Xilinx bitfile which is generated from the Xilinx back-end toolset. The bitfile must have been generated for the exact device on the system.

[1] It is common to refer to the server-side program of a protocol as a daemon.

Table C.1 Supported FTPD Directives.

FTP Directive	Description
RETR x	Retrieve file x from the server.
STOR x	Send file x to the server.
LIST x	Retrieve a file list.
USER	Currently a null op.
PASS	Currently a null op.
SYST	Replies with the system type ('RTEMS').
DELE x	Remove file x from the server.
MKD x	Create a directory x on the server.
RMD x	Remove directory x from the server.
PWD	Print current (remote) working directory.
CWD x	Change current (remote) working directory to x.
SITE CHMOD x y	Change permissions on file y to x (in octal).
PORT a,b,c,d,x,y	Setup a data port to IP address a.b.c.d with port x*256 + y.

ERITHACUS also allows the four DSPs to be programmed through FTP. Two versions of code are allowed for each DSP. The first is boot code, which is downloaded immediately after resetting the device. Boot code must contain only program and no data. The boot code is typically short and allows for downloading larger programs in a second pass. Boot code files are specified with the prefix `dspboot`. The second pass of code can contain more program and can also contain data segments. This pass is specified with the prefix `dspcode`. All DSP hooks expect a Motorola `.cld` file as input, which is generated from Motorola's DSP assembler.

Finally, FTPD allows raw data to be written to the transmitter's wavetable memory. This procedure actually writes the data through the transmitter DSP. Therefore, standard boot code must be running on the DSP. Typically, a MATLAB program will generate and download these files.

C.1.2 Telnet

A programmable telnet daemon exists on ERITHACUS to allow users (and host-side software) to configure the system. Two modes of operation exist: interactive and automated. The automated mode is a less verbose mode used for applications which must configure ERITHACUS without user input, including MATLAB scripts and Java programs.

Table C.2 ERITHACUS FTPD Files.

Filename	Expected Format	Description
`untar`	UNIX tar file.	Untars the file to the current directory.
`fpga.tx`	Xilinx `.bit` file.	Image for transmitter FPGA.
`fpga.rx`	Xilinx `.bit` file.	Image for receiver FPGA.
`waveram.i`	Raw data (up to 8kx16).	In-phase wavetable.
`waveram.q`	Raw data (up to 8kx16).	Quadrature wavetable.
`dspboot.tx`	Motorola `.cld` file.	Boot code for transmitter DSP.
`dspboot.rx1`	Motorola `.cld` file.	Boot code for receiver DSP #1.
`dspboot.rx2`	Motorola `.cld` file.	Boot code for receiver DSP #2.
`dspboot.acc`	Motorola `.cld` file.	Boot code for accessory (audio) DSP.
`dspcode.tx`	Motorola `.cld` file.	Executable for transmitter DSP.
`dspcode.rx1`	Motorola `.cld` file.	Executable for receiver DSP #1.
`dspcode.rx2`	Motorola `.cld` file.	Executable for receiver DSP #2.
`dspcode.acc`	Motorola `.cld` file.	Executable for accessory (audio) DSP.

The interactive version of the telnet daemon provides a simple command language to set and query parameters. For the most part, the language provides a simple set of instructions to communicate with the DSPs. Configuration of the system is mainly done through this interface.

Table C.3 lists the supported commands and a brief description of each. Surprisingly, this short list of commands provides a good deal of flexibility. Valid DSP IDs are `tx`, `acc`, `rx1`, and `rx2`.

Table C.3 ERITHACUS Telnet Commands.

Command	Description
`dspcommand ID VECTOR`	Performs a host command.
`dspwrite ID DATA`	Writes data to a DSP host interface.
`dspwrite ID MEM:ADDR DATA`	Writes data to a location in DSP memory.

dspcommand ID VECTOR

This command performs a Host Interface Command Vector 'VECTOR' on DSP 'ID'. After this command, the DSP calls the interrupt at the given vector location. This command can be used to invoke certain events on the DSP. It can also be used to change a single parameter if a `dspwrite` precedes the command. More information on Host Command Vectors can be found in [26].

Example: `dspcommand tx $32` causes the interrupt at vector $32 on the transmitter DSP to be called.

dspwrite ID DATA

This command writes a single word to the host interface. When followed by a `dspcommand`, the sequence can be used to change single parameters on the DSP.

Example: `dspwrite rx1 $1234af` writes the hexadecimal value $1234af to the host interface. Subsequently, a host command could be issued invoking an interrupt service routine to read the data from the host interface.

dspwrite ID MEM:ADDR DATA

This command can only be issued when the boot code is present on the DSP. The boot code provides a method for writing data to arbitrary locations in DSP memory.

Example: `dspwrite acc x:$002000 $000000` clears the memory location `x:$002000`.

C.1.3 HTTPD

A very simple version of an HTTP daemon is provided to serve static hypertext information to any standard web browser, such as Lynx or Netscape. The purpose of this support is to provide on-line documentation in a simple, portable, ubiquitous format. In addition, ERITHACUS control programs written in Java can be served from the device itself, providing a highly portable environment with which to control ERITHACUS.

C.2 Host-Side Software

The host-side software is a collection of routines written to allow higher-level systems (such as MATLAB) to control ERITHACUS. These routines provide communication through the telnet and FTP daemons running on the system.

C.2.1 Libraries

Communication with ERITHACUS is done through two protocols: FTP and telnet. Although the user can control the system through appropriate clients on a host workstation (`ftp` and

telnet), more flexible interfaces can be built into programs such as MATLAB. To facilitate communication with ERITHACUS through such programs, a library has been developed that provides the necessary commands. What follows is a list of these commands, their description, and how to use them. Examples can be found in the ERITHACUS source code at [27].

FTPLIB_OpenConnection(char *hostname, int showMessages)

This function opens a connection to the ERITHACUS with the hostname given in the first argument. The second argument specifies whether progress messages should be displayed on standard output.

Example: `FTPLIB_OpenConnection("erithacus0.ece.uiuc.edu", 0)`

FTPLIB_CloseConnection()

This function closes a connection previously opened with `FTPLIB_OpenConnection(...)`. Connections should be closed before exiting the program.

Example: `FTPLIB_CloseConnection()`

FTPLIB_SendFile(char *localFilename, char *remoteFilename)

This function sends a file over a previously opened connection. The local filename is the name of the file on the host workstation. The remote filename could be one of those listed in Table C.2. If the filename is not a recognized filename, then the file will be stored in the RTEMS IMFS.

Example: `FTPLIB_SendFile("MyRxCode.cld", "dspcode.rx1")`

FTPLIB_SendData(long size, unsigned char *bufr, char *remoteFilename)

This is identical to `FTPLIB_SendFile(...)` except that the source of data is a buffer rather than a file on the host.

Example: `FTPLIB_SendData(8192, IWaveBufr, "waveram.i")`

C.2.2 MATLAB commands

MATLAB provides the front end for most of the educational labs using ERITHACUS. The following commands are provided to configure the transmitter and receiver as well as extract information from the receiver during operation.

RESULT = COMMECPDIRECT(HOSTNAME, COMMAND)

This command is equivalent to sending the string COMMAND through the telnet interface to ERITHACUS. A single-line result string is returned in RESULT. The command string can be any of those listed in Table C.3. This command is commonly used by the MATLAB interfaces to query bit error rate and other parameters from the receiver.

Y = COMMMAKEWAVETABLE(WAVEFORM, N)

The function of this command is somewhat elusive, but important. Given a matrix of waveform vectors (WAVEFORM) and the length of a single bit period (N), this command generates a matrix which can be downloaded into the ERITHACUS wavetables. Each column is a single bit period which makes up a waveform in the wavetable. In general, the waveforms specified in WAVEFORM are longer than a single bit period representing memory in the transmitter. COMMMAKEWAVETABLE breaks the waveform into sections of length N and computes every combination of these sections representing every combination of bits sent through the transmitter.

COMMDELAY(X)

This command causes the MATLAB program to delay for X seconds with microsecond resolution. This function is used when measuring BER to wait for a certain amount of time.

COMMSETTXPARAM(HOSTNAME, Parameter1, Value1, Parameter2, Value2, ...)

This command is used to configure the transmitter. Any number of parameters and their values can be specified at a time. Supported parameters are listed in Table C.4.

Table C.4 COMMSETTXPARAM parameters.

Parameter	Description
SampleRate	The transmitter sample rate (in MHz).
SamplesPerWaveform	Number of samples per waveform.
WaveformsPerSymbol	Number of waveforms per symbol period.
BitsPerSymbol	The number of bits per symbol.
DataSource	One of 'PN7', 'PN15', 'PN23', 'User', or 'UserScrambled'.
IWaveform	The data to be written to the I wavetable.
QWaveform	The data to be written to the Q wavetable.

The following example configures the transmitter for a QPSK constellation. The waveforms were previously generated by calls to COMMMAKEISI. The sample rate is 10 MHz, the symbol rate is 1 MHz.

```
commsettxparam('dcomm0.ece.uiuc.edu', 'samplesperwaveform', 10, ...
                                      'waveformspersymbol', 1, ...
                                      'samplerate', 10.0, ...
                                      'bitspersymbol', 2, ...
                                      'datasource', 'PN15', ...
                                      'iwaveform', [iwave iwave], ...
                                      'qwaveform', [qwave qwave]);
```

C.2.3 MATLAB user interfaces

Two graphical user interfaces have been developed for use in the ECE 363 lab: the receiver control panel (Figure C.1) is used to configure the receiver, and AutoBERT (Figure C.2) is used to take automated bit-error-rate measurements.

The receiver control panel is started with the MATLAB command COMMRXCONTROLPANEL. Along the bottom of the window, four selections appear to configure the outputs to the DACs. The available options are shown in Table C.5. Note that at this time DAC3 and DAC4 only have one output available. The buttons under "ADC Offset Calibration" instruct ERITHACUS to perform calibration on the I and Q inputs. The inputs must be terminated for the calibration to be correct. Under "Receiver Mode," BPSK or QPSK can be selected, which alters the way the phase detector operates. Under "Matched Filter," matched filtering ("enabled") or midsymbol sampling ("disabled") can be selected. Also, timing recovery can be

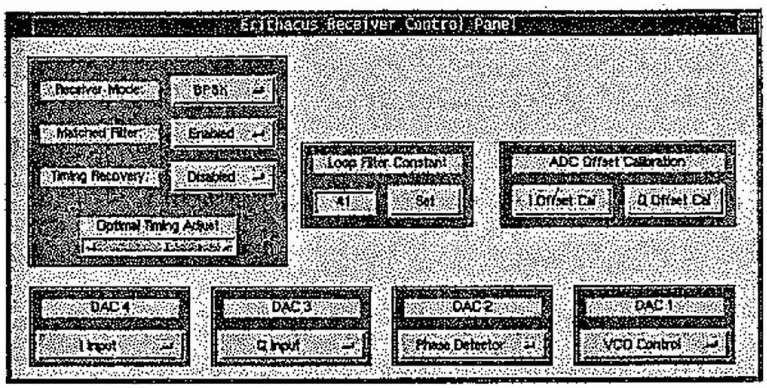

Figure C.1 Receiver Control Panel.

enabled or disabled. A slider is provided as "Optimal Timing Adjust" to select the sampling time for symbol decision. Finally, under "Loop Filter Constant," the characteristics of the loop filter can be adjusted. See Section B.6 for information on this value.

Table C.5 Receiver DAC Output Options.

DAC	Outputs Available
DAC1	I Input, I Integrator, I Decision, Q Input
DAC2	Q Input, Q Integrator, Q Decision, I Input
DAC3	Phase Detector Output
DAC4	VCO Control Output

The BERT is started with the MATLAB command COMMAUTOBERT. COMMBERT starts a similar GUI that is manually driven. The AutoBERT (shown in Figure C.2) takes automated measurements through a range of SNR (specified by "SNR Start," "SNR Stop," and "SNR Step") by adjusting the output signal power referred to the specified noise power ("N_o"). Once a measurement starts, total bits and errors are counted by the receiver. The measurement stops when at least "Min errors" have been counted or the "Max dwell" time has been exceeded, whichever comes first. During a measurement, the total bits, total errors, and the BER are updated once per second.

For square pulses and a given SNR and N_o, the peak power is computed as

$$P_{peak} = SNR + N_o + 60 \tag{C.1}$$

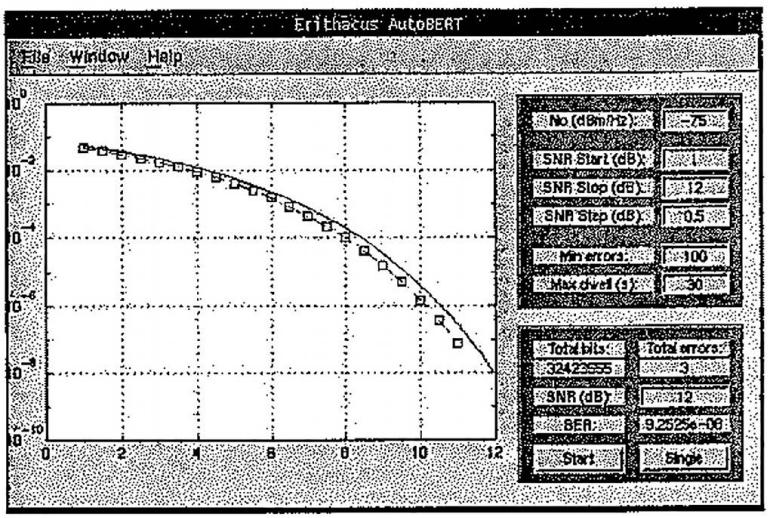

Figure C.2 Auto BERT.

The factor of 60 is included to adjust for a bit-rate of 10^6 because $SNR = E_b/N_o$, $E_b = P_{rms}/T_b$, and for square pulses, $P_{rms} = P_{peak}$.

The results of the measurement are plotted in the same window. The theoretical curve for BPSK is shown solid. The measurement data is shown dashed and boxed. Theoretical results are computed as

$$P_e = Q(\sqrt{2 \cdot 10^{SNR_{dB}/10}}) \tag{C.2}$$

APPENDIX D

SCHEMATICS

This appendix contains the schematics for ERITHACUS. The entire board was entered into MicroSim's schematic capture program Schematics Version 7.1. The board layout was also done in MicroSim using PCBoards Version 7.1.

The board consists of six layers: four signal layers and two supply layers. The signal layers are the two outer layers and the two inner layers. Although this distribution makes debugging and trace alterations more difficult (because the two inner layers are under the supply layers), impedance is better equalized between the outer signal and inner signal layers. This is essential because sections of the board can operate to 66 MHz.

The ground plane is continuous through the digital section of the board. However, separate ground planes are provided under each analog section. These independent ground planes are connected via a ferrite bead to the digital ground plane.

The power plane is separated into several sections to provide both 3.3- and 5.0-V sections to the digital logic as well as to provide separate analog power planes to each analog section. The supply plane partitioning is explained in Figure D.1.

Gerber files were produced and sent to Paramount Circuits. Paramount performed the board fabrication and testing from these files. Finally, the board was assembled by hand by the author in the Advanced Digital Systems Lab at the University of Illinois.

Figure D.1 contains design notes for the system. Figures D.2-D.4 contain schematics for the CPU section of the system. Figures D.5 and D.6 contain schematics for the accessory DSP portion. Figures D.7-D.9 contain schematics for the transmitter section. Figure D.10 contains the clock generator schematic. Figures D.11-D.15 contain schematics for the receiver section. Figure D.16 contains the schematic for the board power supply.

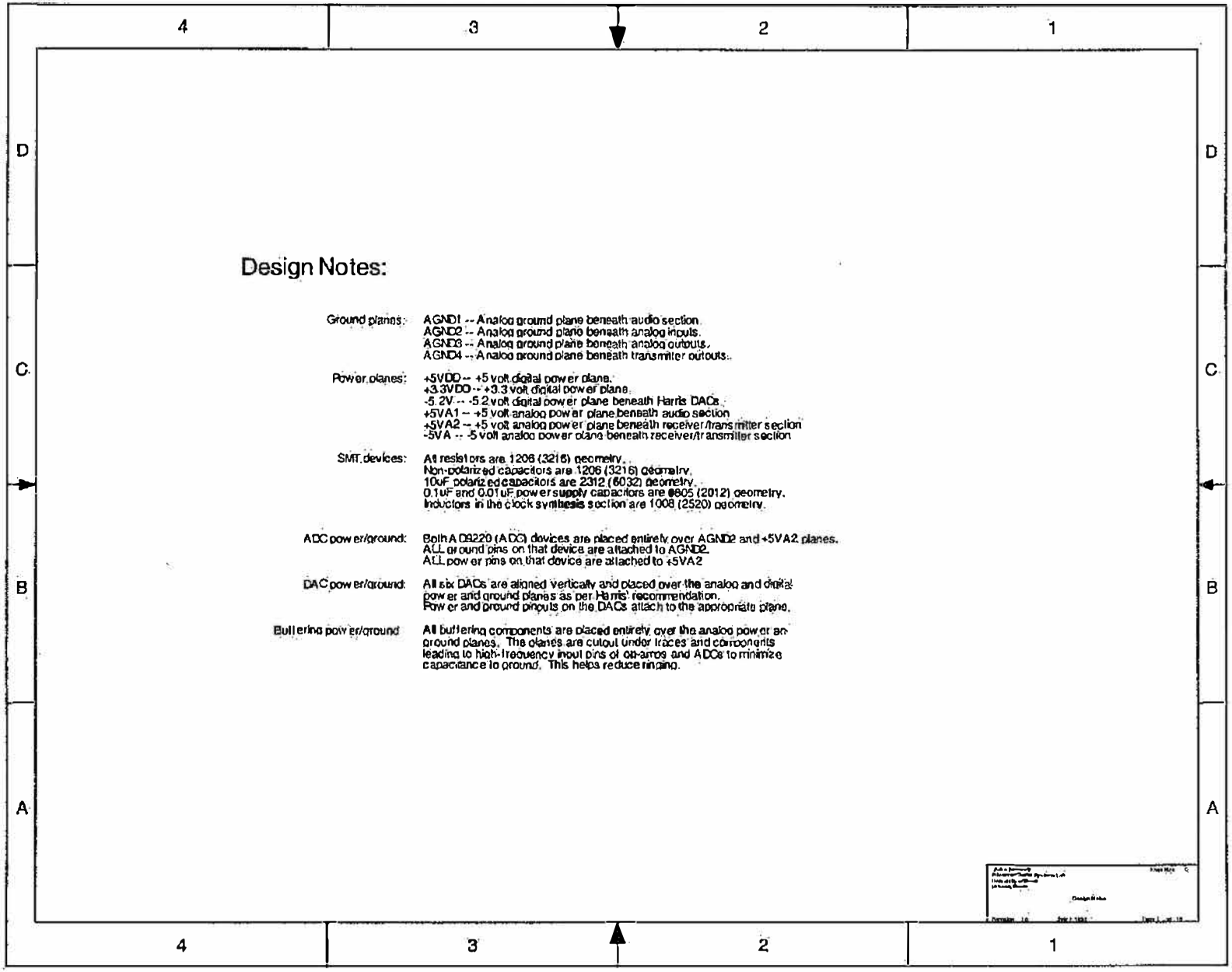

Figure D.1 Design Notes.

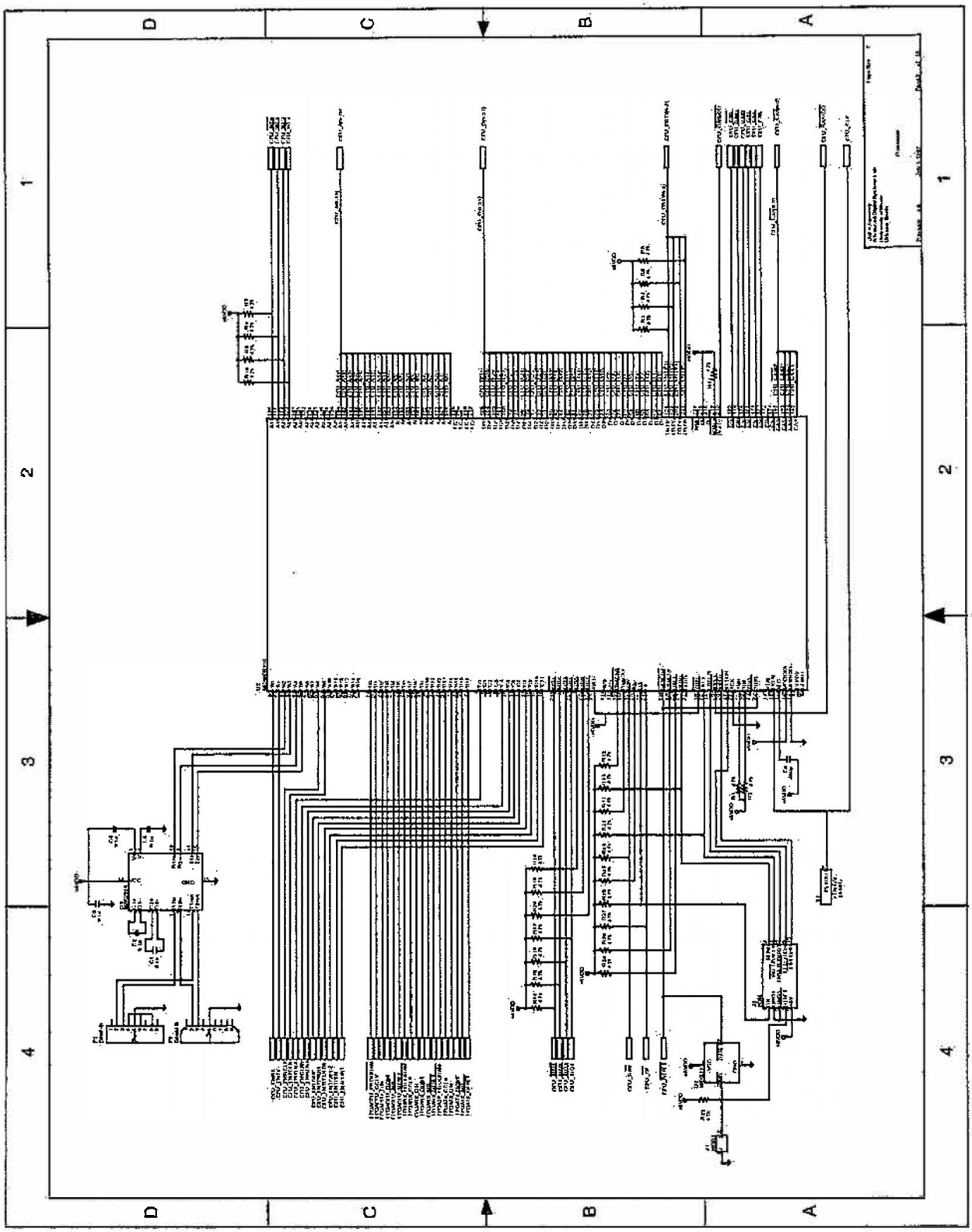

Figure D.2 Processor.

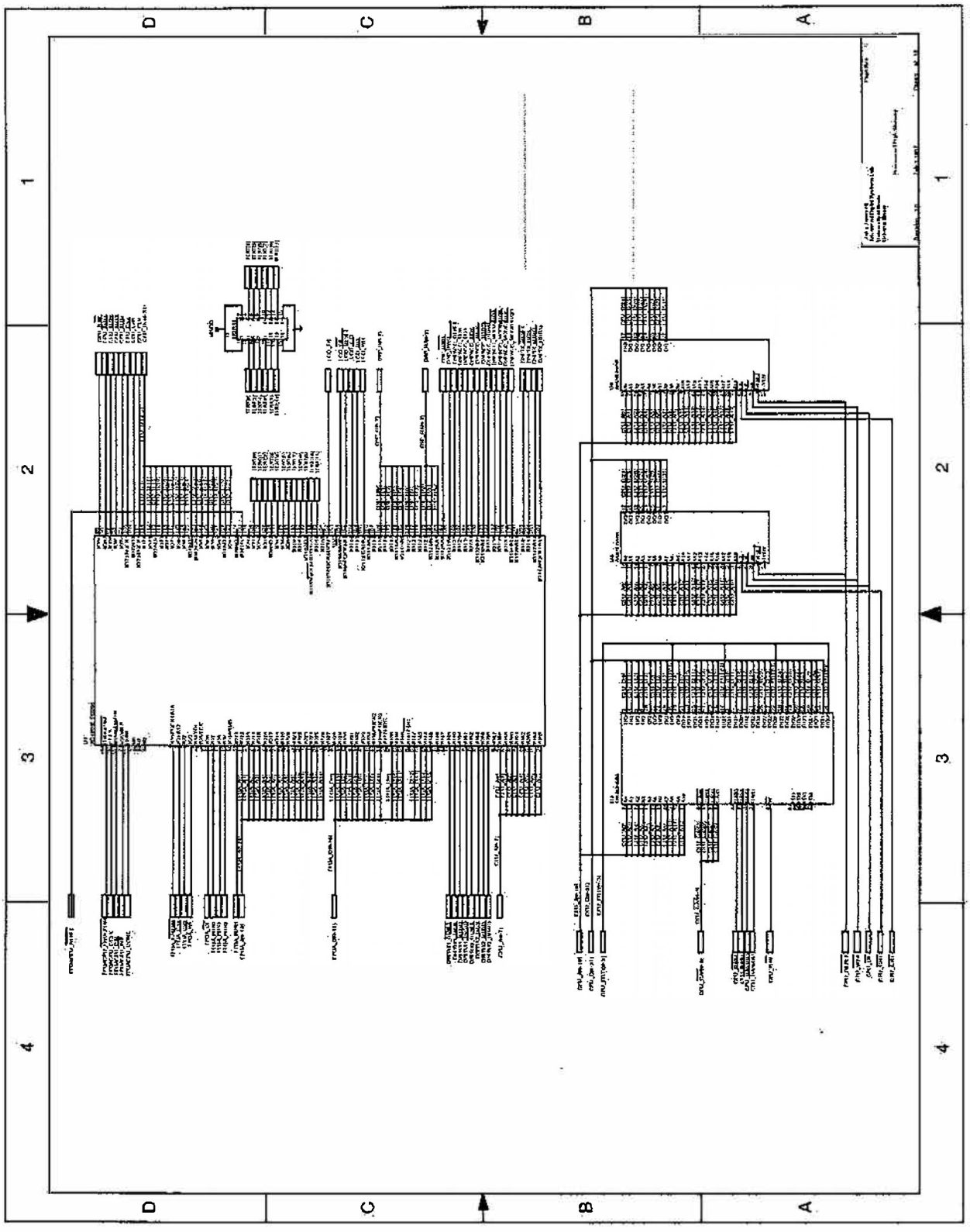

Figure D.3 Processor FPGA and Memory.

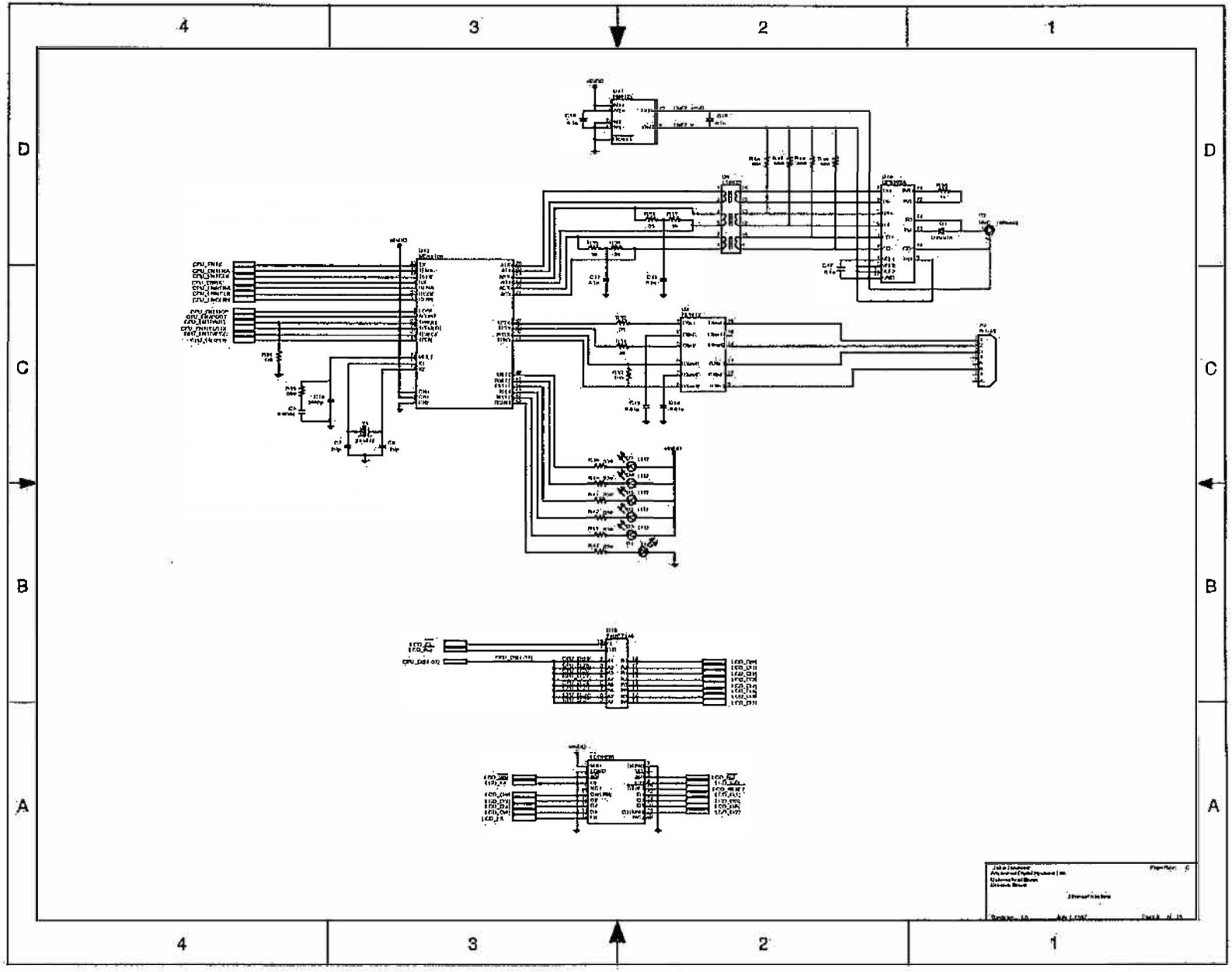

Figure D.4 Ethernet Interface.

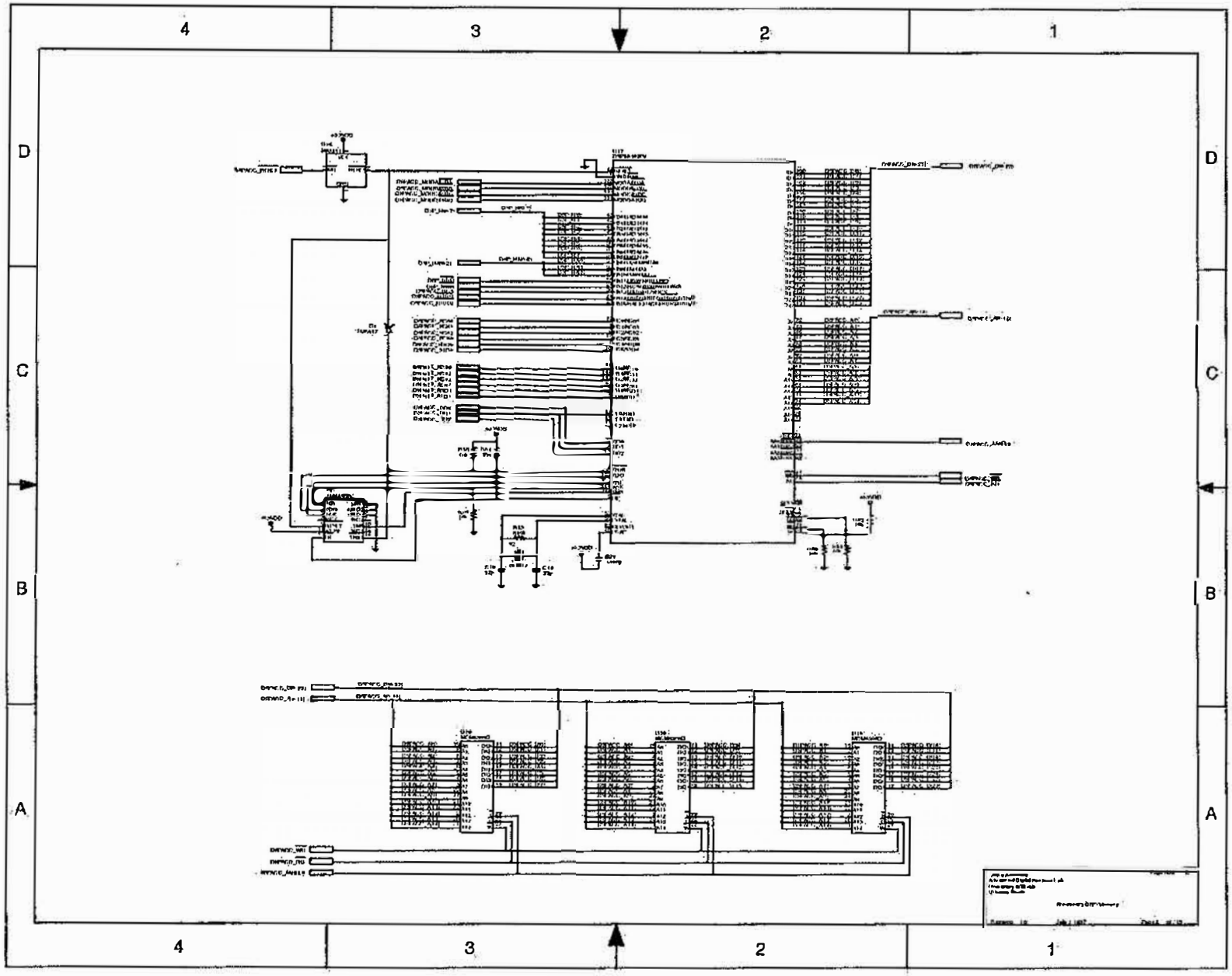

Figure D.5 Accessory DSP and Memory.

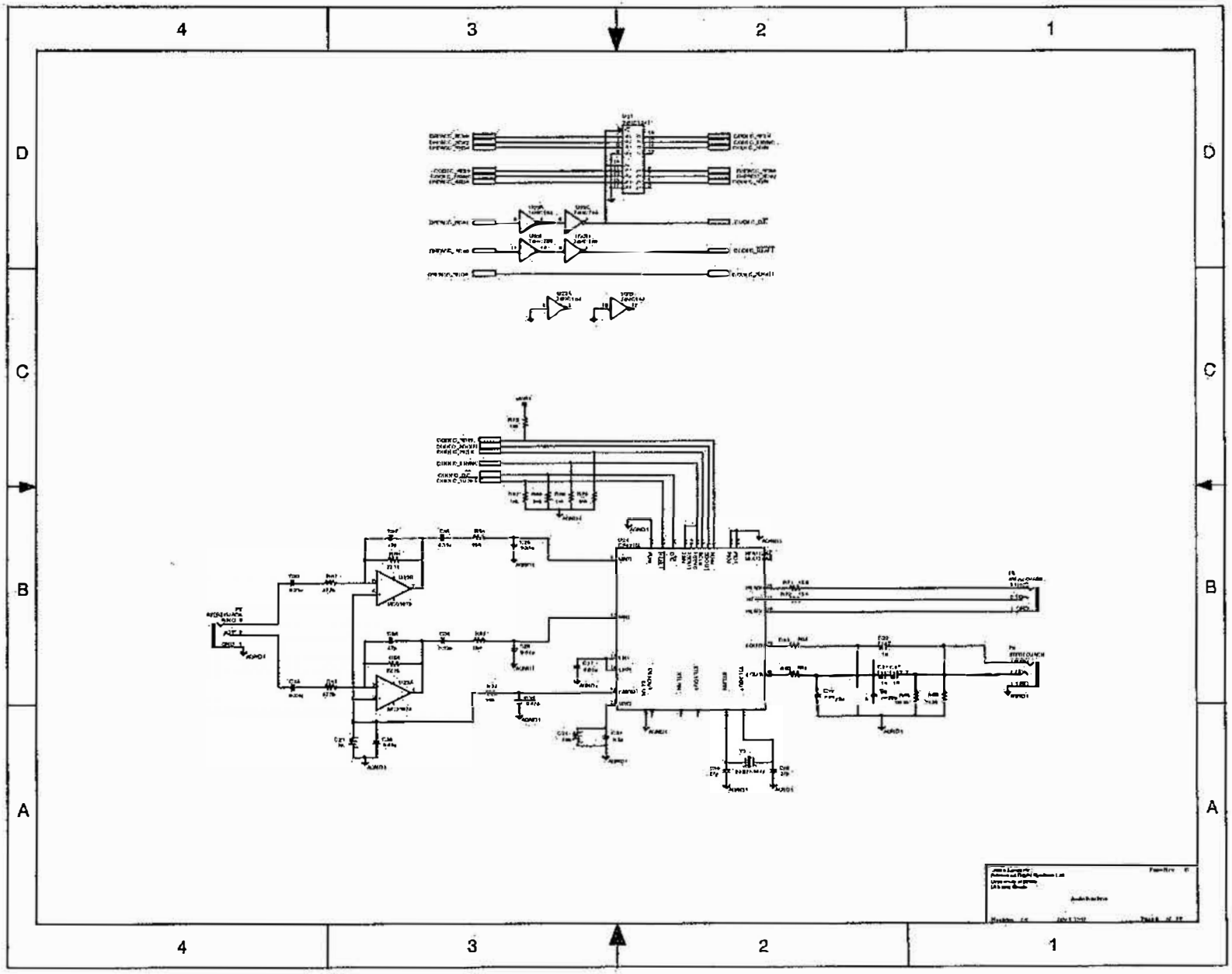

Figure D.6 Audio Interface.

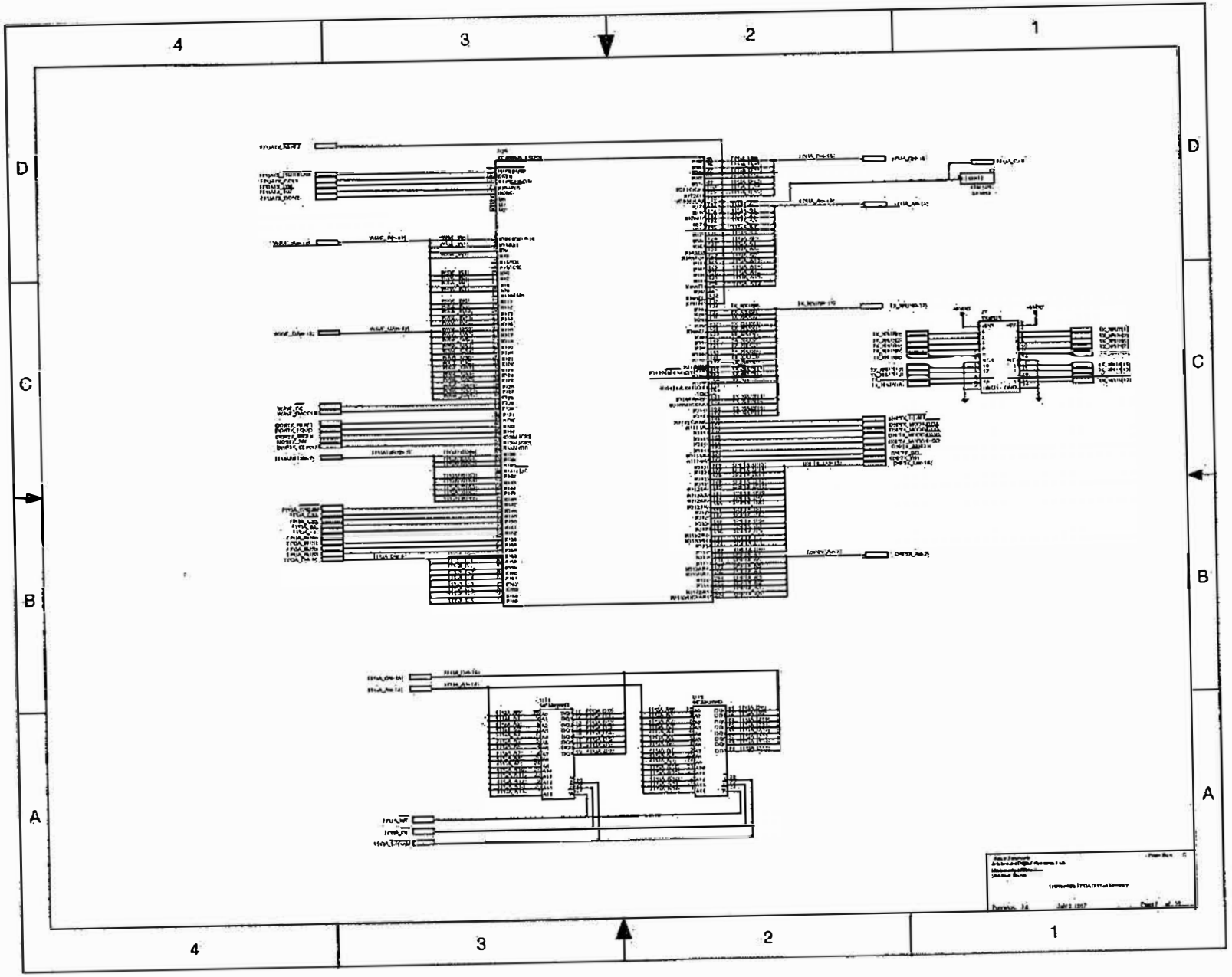

Figure D.7 Transmitter FPGA and FPGA Memory.

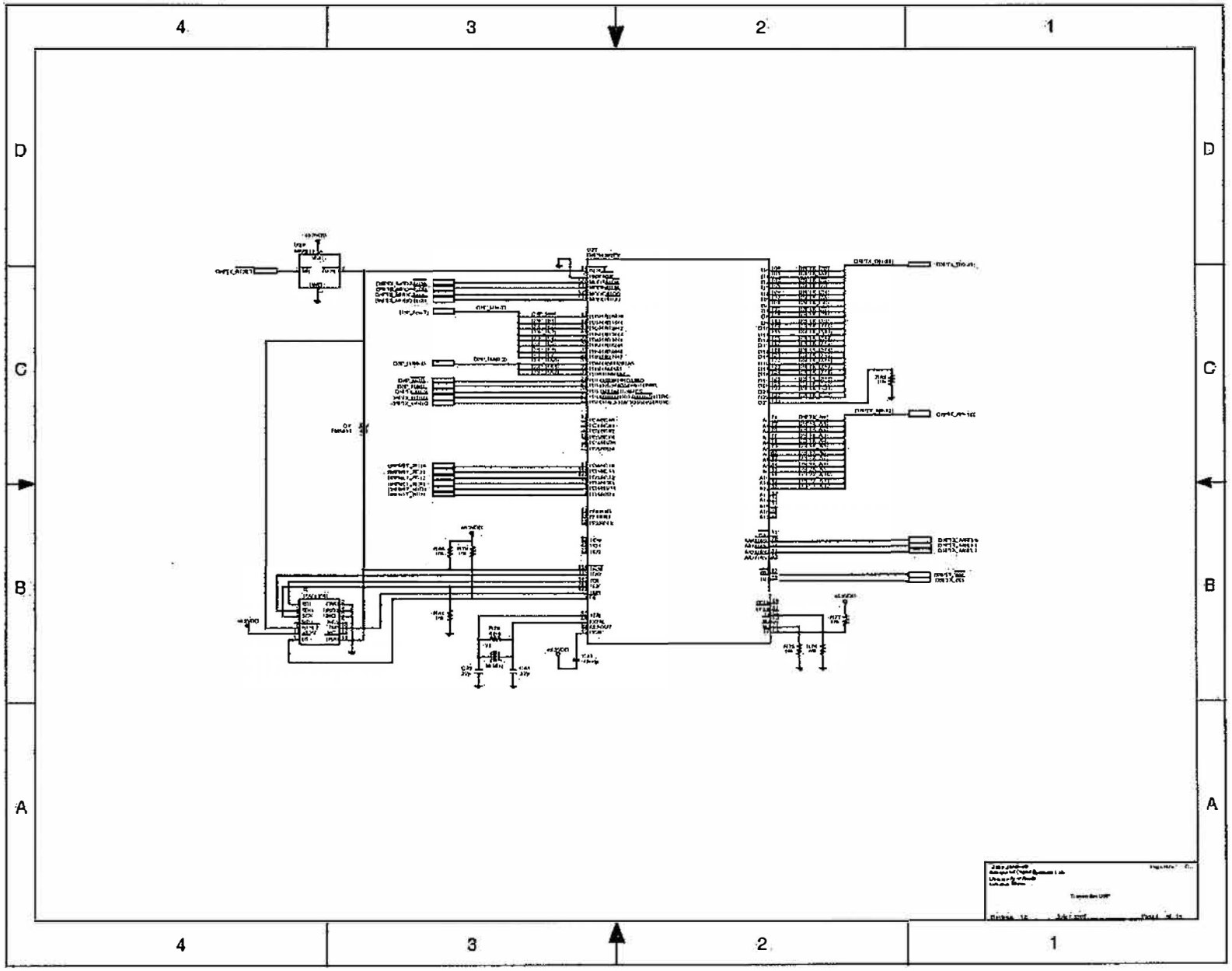

Figure D.8 Transmitter DSP.

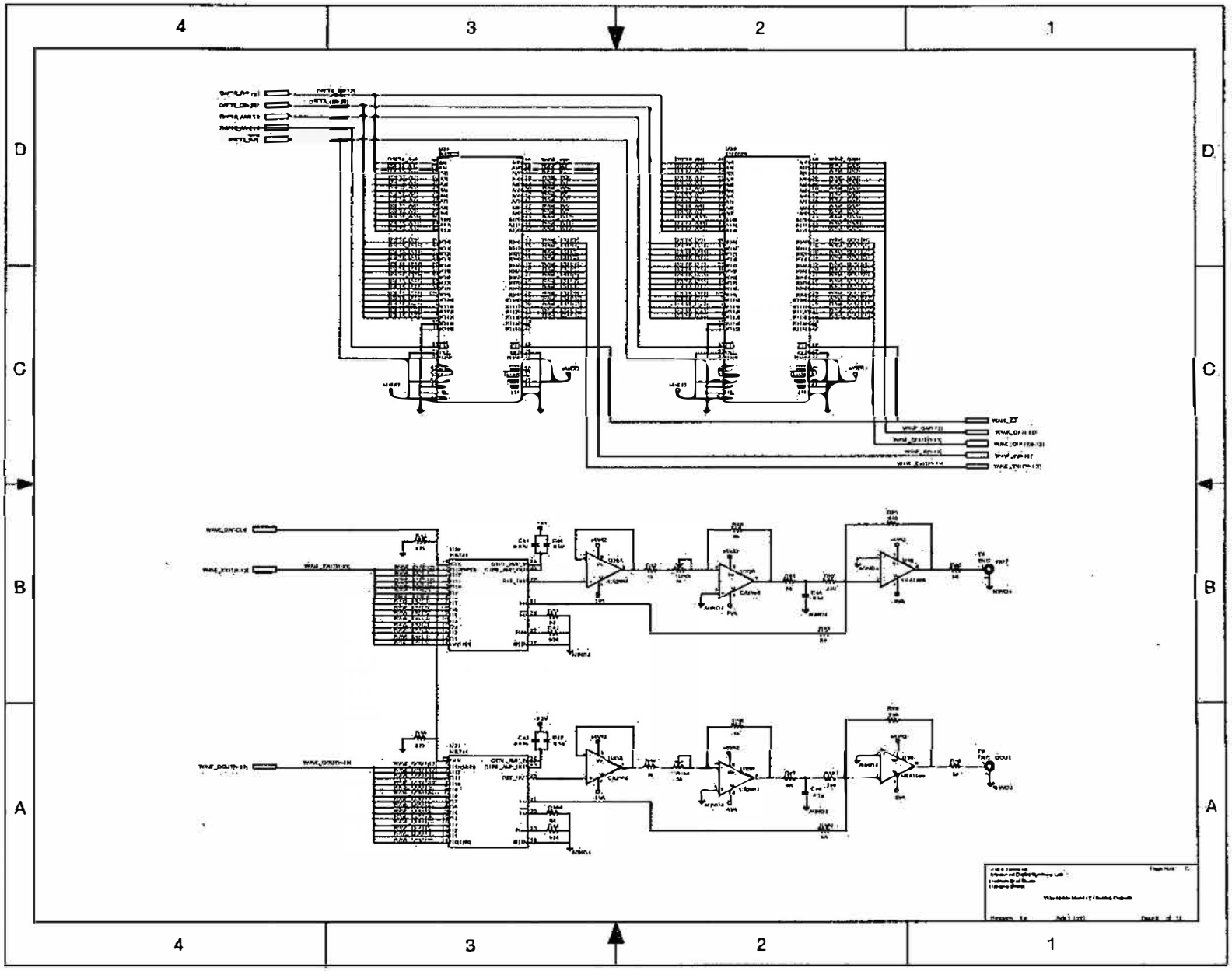

Figure D.9 Wavetable Memory and Analog Outputs.

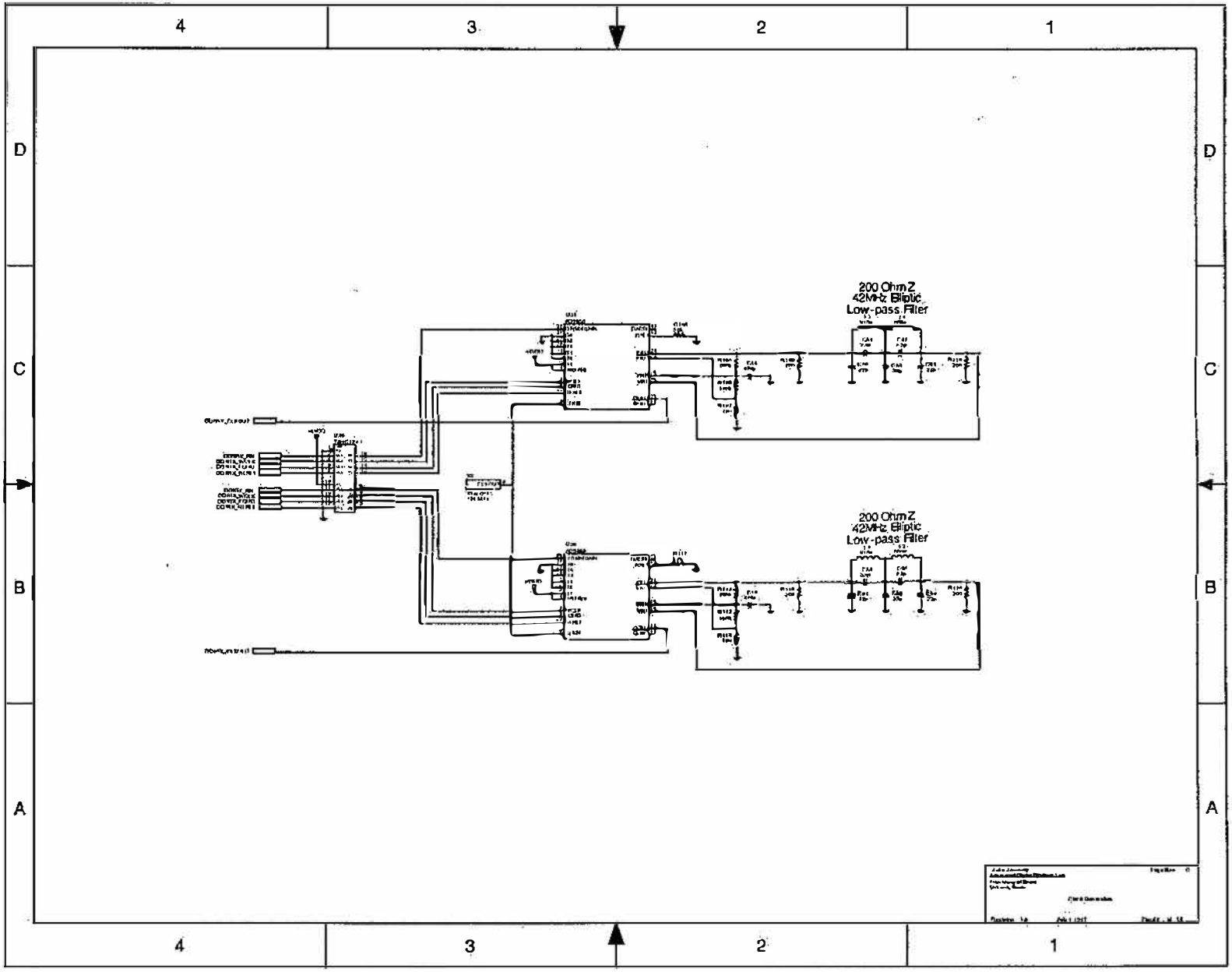

Figure D.10 Clock Generation.

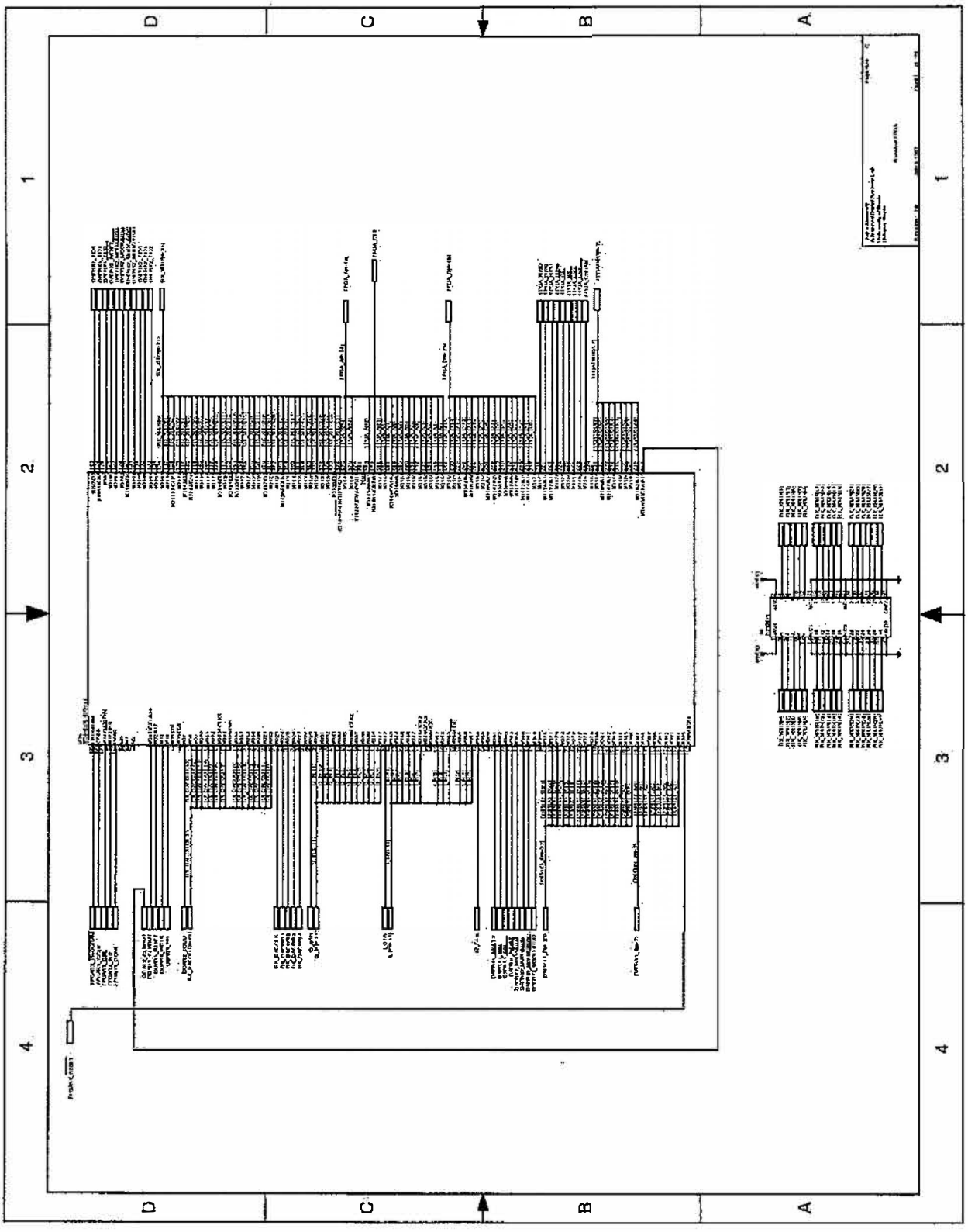

Figure D.11 Receiver FPGA.

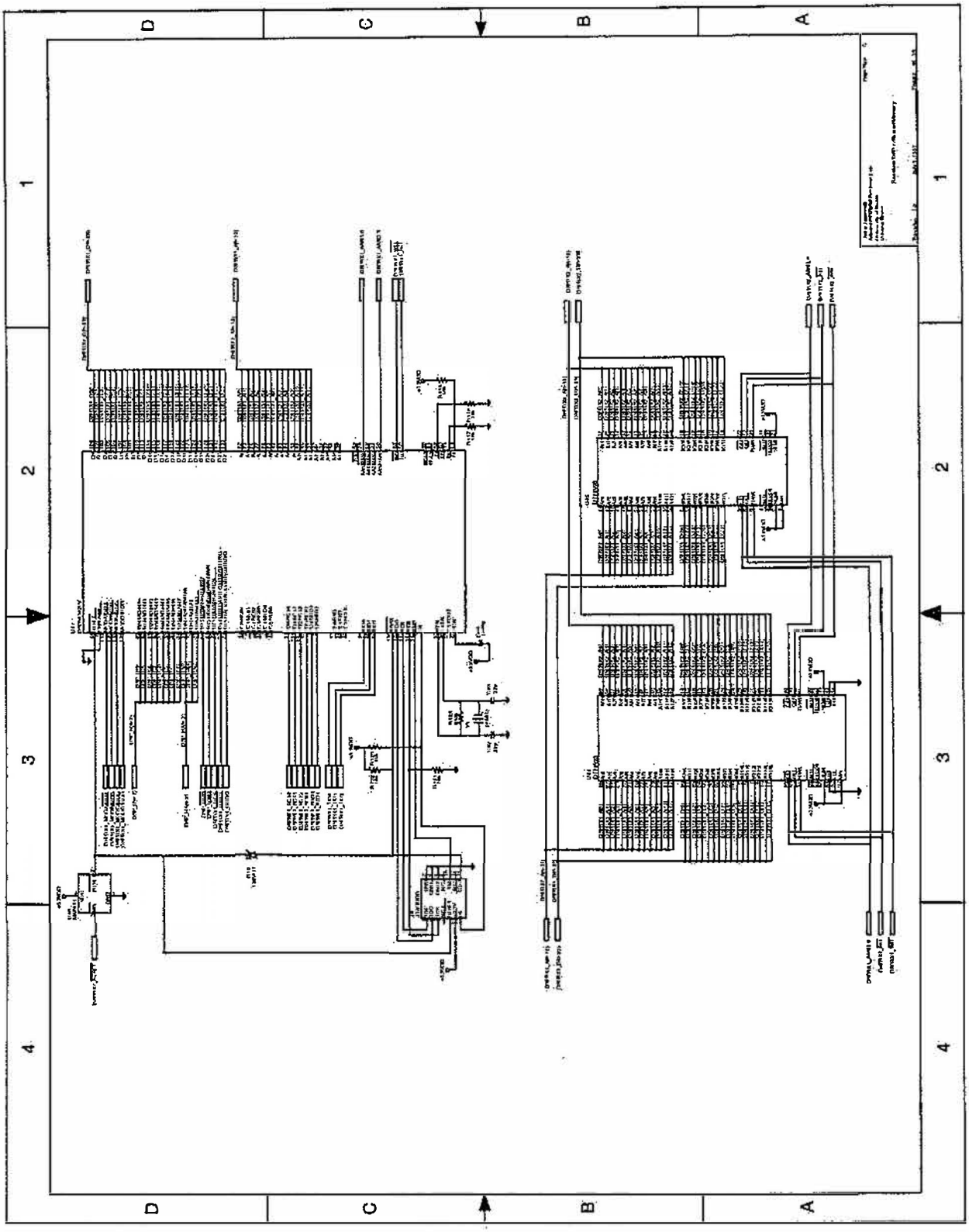

Figure D.12 Receiver DSP 1 and Shared Memory.

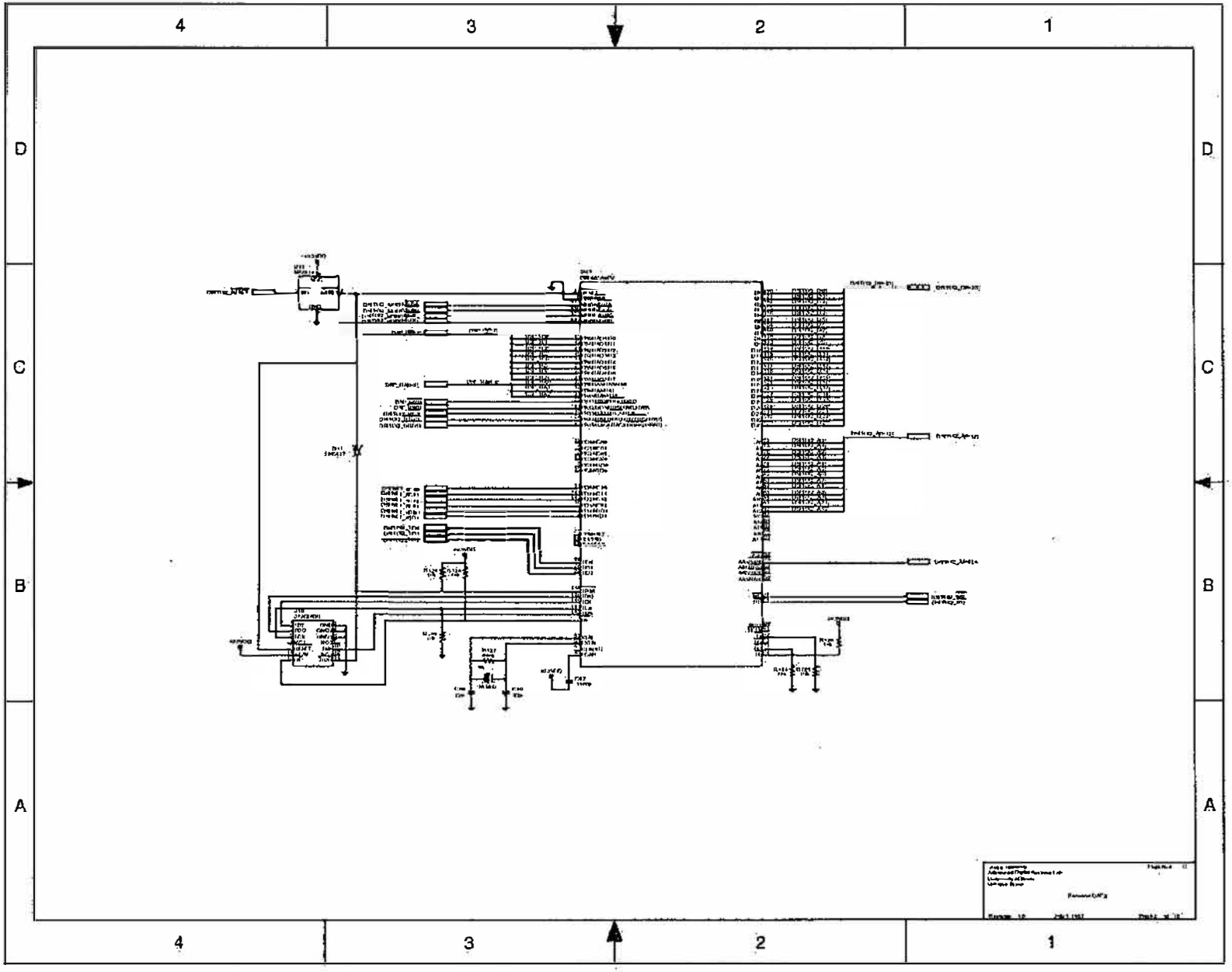

Figure D.13 Receiver DSP 2.

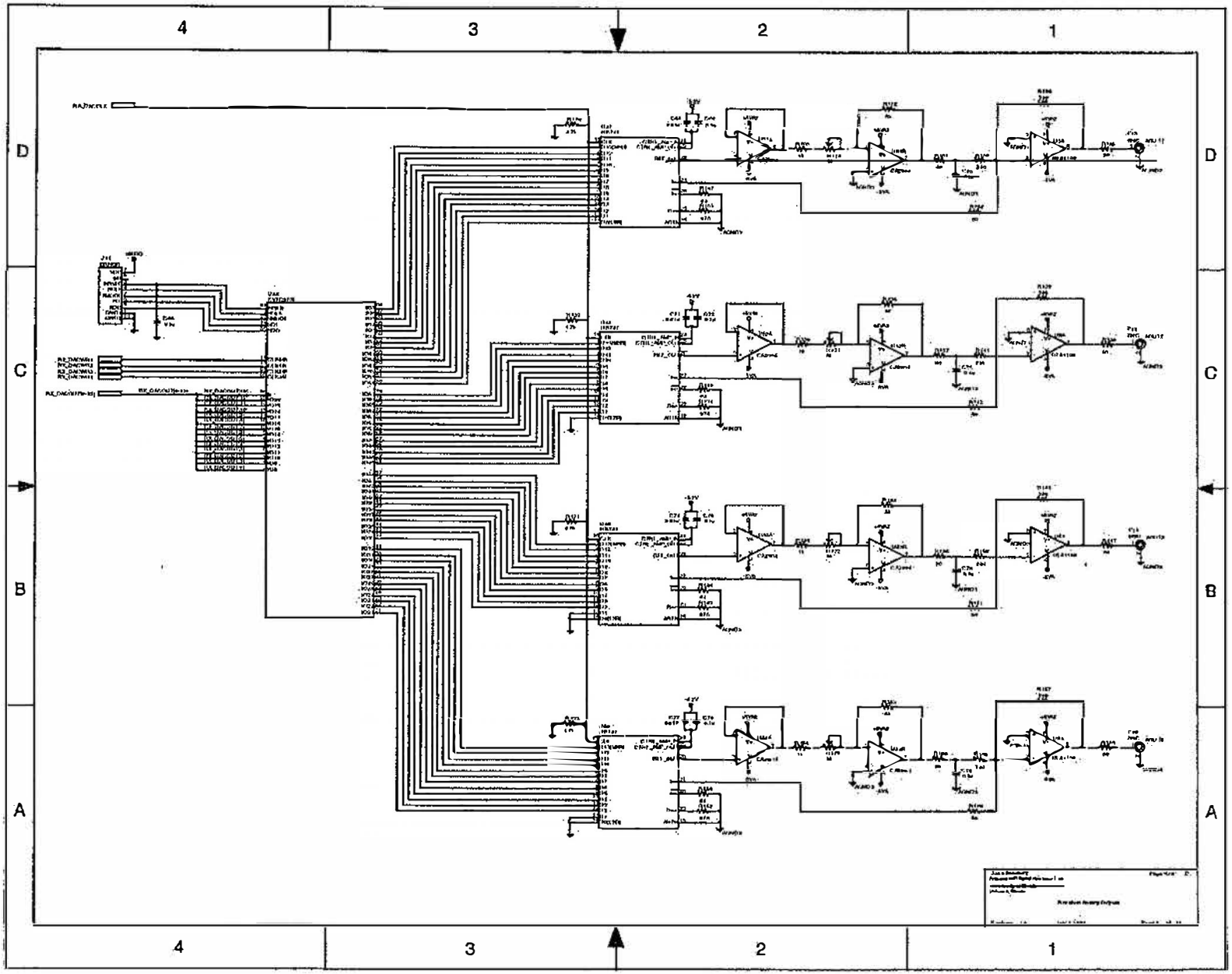

Figure D.14 Receiver Analog Outputs.

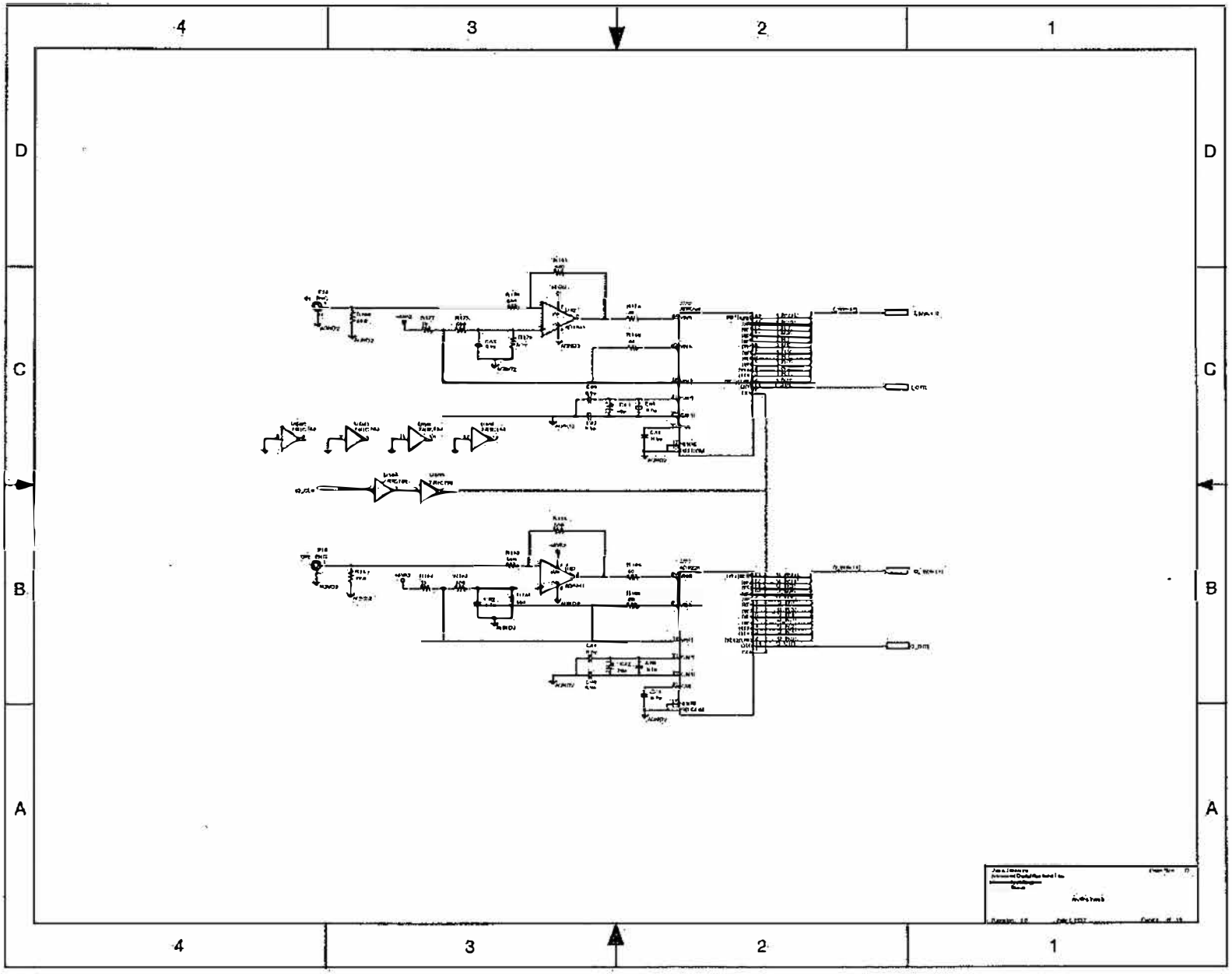

Figure D.15 Analog Inputs.

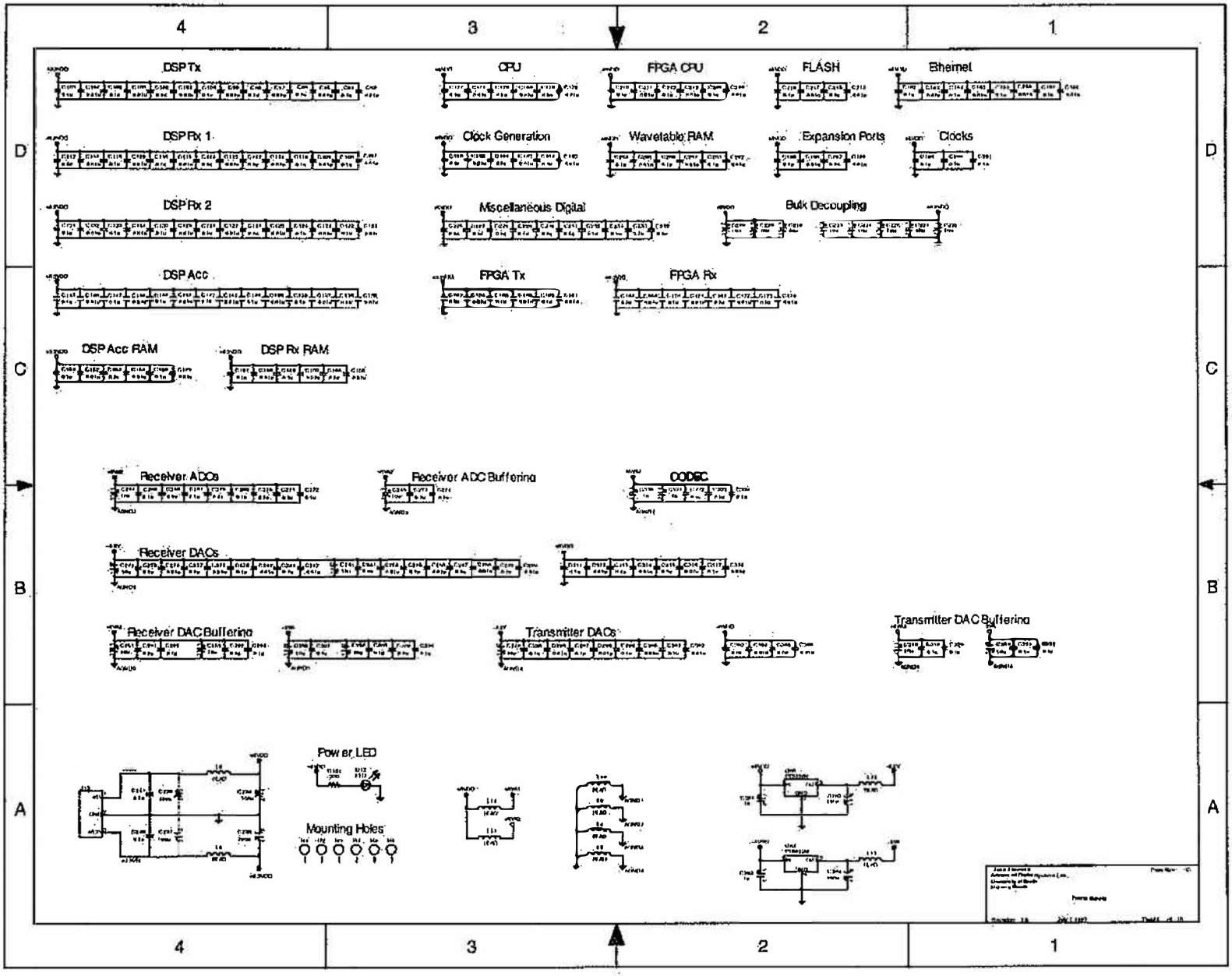

Figure D.16 Power Supply.

APPENDIX E

BILL OF MATERIALS

Table E.1 contains the bill of materials for ERITHACUS. The reference designators correspond to the part designators given in the schematics in Appendix D. Pricing is given as approximate for unit quantities except for resistors and capacitors.

Table E.1: Bill of Materials.

RefDes	Component Description	Manufacturer Part Number	Unit Price
U5, U6	1 M x 8 bit FLASH memory	AMD Am29F080-90SC	$25.00
U4	72-pin SIMM socket	AMP 822031-4	$2.50
U62, U63	160-MHz rail-to-rail op-amp	Analog Devices AD8041AN	$3.00
U60, U61	12-bit, 10-MSPS monolithic A/D converter	Analog Devices AD9220AR	$25.00
U37, U38	125-MHz complete DDS	Analog Devices AD9850BRS	$25.00
R103, R104, R171, R172, R173, R174	1-kΩ Trimmer potentiometer	Bourns 3296W (1 kΩ)	$2.34
X3	125-MHz crystal oscillator	Champion Technologies K1300C	$35.00
L1, L3	910-nH 5% Inductor (1008 SMD)	Coilcraft 1008CS-911XJBC	$1.00
L2, L4	680-nH 5% Inductor (1008 SMD)	Coilcraft 1008CS-681XJBC	$1.00
P4	PCB-mount RJ-45 connector	Corcom RJ45-8L-B	$3.23
U24	16-bit audio CODEC	Crystal Semiconductor CS4215-KL	$35.00

Table E.1: (Continued)

RefDes	Component Description	Manufacturer Part Number	Unit Price
U46	Flash CPLD	Cypress Semiconductor CY7C373i-66AC	$20.00
U32, U33, U51, U52, U53, U54	Voltage feedback op-amp	Harris CA2904M	$1.00
U34, U35, U55, U56, U57, U58	Current feedback op-amp	Harris HFA1100IP	$5.00
U30, U31, U47, U48, U49, U50	14-bit, 100-MSPS D/A converter	Harris HI5741BIB	$35.00
U28, U29	8 k x 16 bit, 15-ns, 5-V dual-port SRAM	IDT IDT7025S15PF	$40.00
U45	8 k x 8 bit, 25-ns, 3.3-V dual-port SRAM	IDT IDT70V05S25PF	$30.00
U44	8 k x 16bit, 25-ns, 3.3-V dual-port SRAM	IDT IDT70V25S25PF	$45.00
U3	Multichannel RS-232 driver/receiver	Maxim MAX232ACSE	$4.00
U16, U26, U40, U42	Microprocessor voltage monitor (2.93-V threshold)	Maxim MAX811SEUS-T	$1.00
U1	Microprocessor voltage monitor (4.63-V threshold)	Maxim MAX811LEUS-T	$1.00
U23	Audio op-amp	Motorola MC33078D	$1.50
U13	Bus Transceiver	Motorola MC74HCT245ADW	$1.00
U21, U36	Buffer	Motorola MC74HCT241ADW	$2.00
U22, U59	Hex inverter	Motorola MC74HCT04AD	$1.00
U2	33 MHz ethernet-enabled QUICC processor	Motorola MC68EN360FE33C	$75.00
U12	Enhanced ethernet transceiver	Motorola MC68160FB	$20.00
U14, U15	32 k x 8 bit, 5-V, 15-ns SRAM	Motorola MCM6206BAEJ15	$4.00
U18, U19, U20	32 k x 8 bit, 3.3-V, 15-ns SRAM	Motorola MCM6306DJ15	$5.00
U4	8 M x 36 bit, 70-ns DRAM SIMM	Motorola MCM36800-70	$25.00
U17, U27, U41, U43	80-MHz digital signal processor	Motorola DSP56302PV80	$50.00
U10	Coaxial transceiver interface	National Semiconductor DP8392CN	$5.00

Table E.1: (Continued)

RefDes	Component Description	Manufacturer Part Number	Unit Price
U64	+5 V to -5.0 V dc-dc converter	Power Trends PT5022N	$16.00
U65	+5 V to -5.2 V dc-dc converter	Power Trends PT5026S	$16.00
D1	Diode	1N916	$1.00
D8, D9, D10, D11	Schottky diode	Rectron FM5817	$1.00
P5, P6, P7	3.5-mm PCB-mount stereo jack	Switchcraft 35RAPC4BHN2	$1.13
U11	Isolated ethernet DC/DC converter (5 V to 9 V)	Valor PM6125	$10.00
U9	Ethernet AUI transformer	Valor LT6033	$3.00
U8	Ethernet 10BASE-T filter module	Valor FL1012	$5.00
U25	Field-programmable gate array	Xilinx XC4028XL-3HQ240C	$190.00
U39	Field-programmable gate array	Xilinx XC4010XL-3PQ208C	$65.00
U7	Field-programmable gate array	Xilinx XC4008E-4PQ208C	$80.00
X2	50-MHz crystal oscillator		$5.00
X1	25-MHz crystal oscillator		$1.00
Y3	24.576-MHz crystal		$1.00
Y1	20-MHz crystal		$1.00
Y2, Y4, Y5, Y6	16-MHz crystal		$1.00
2	33-pF capacitor (1206 SMD)	IMS	$0.00
2	3.3-pF capacitor (1206 SMD)	IMS	$0.00
12	0.01-μF capacitor (1206 SMD)	IMS	$0.00
2	47-pF capacitor (1206 SMD)	IMS	$0.00
2	0.0022-μF capacitor (1206 SMD)	IMS	$0.00
1	3900-pF capacitor (1206 SMD)	IMS	$0.00
2	470-pF capacitor (1206 SMD)	IMS	$0.00
4	0.33-μF capacitor (1206 SMD)	IMS	$0.00
31	0.1-μF capacitor (1206 SMD)	IMS	$0.00
1	0.47-μF capacitor (1206 SMD)	IMS	$0.00
2	27-pF capacitor (1206 SMD)	IMS	$0.00
4	1800-pF capacitor (1206 SMD)	IMS	$0.00
2	8.2-pF capacitor (1206 SMD)	IMS	$0.00
12	22-pF capacitor (1206 SMD)	IMS	$0.00
1	400-pF capacitor (1206 SMD)	IMS	$0.00

Table E.1: (Continued)

RefDes	Component Description	Manufacturer Part Number	Unit Price
2	20-pF capacitor (1206 SMD)	IMS	$0.00
1	0.039-μF capacitor (1206 SMD)	IMS	$0.00
See note [1]	0.01-μF capacitor (0805 SMD)		$0.00
See note [2]	0.1-μF capacitor (0805 SMD)		$0.00
See note [3]	10-μF 16-V tantalum capacitor (6032 SMD)	Kemet T491C106K016AS	$0.60
3	1-μF bipolar capacitor (1206 SMD)		$0.00
2	1-μF bipolar capacitor		$0.00
6	100-μF 10-V tantalum capacitor	Panasonic ECS-F1AE107	$4.25
1	1-kΩ resistor		$0.00
2	15.8-Ω resistor (1206 SMD)	IMS	$0.00
6	39-Ω resistor (1206 SMD)	IMS	$0.00
4	40-Ω resistor (1206 SMD)	IMS	$0.00
2	49.9-Ω resistor (1206 SMD)	IMS	$0.00
12	50-Ω resistor (1206 SMD)	IMS	$0.00
6	60-Ω resistor (1206 SMD)	IMS	$0.00
6	64-Ω resistor (1206 SMD)	IMS	$0.00
3	100-Ω resistor (1206 SMD)	IMS	$0.00
2	150-Ω resistor (1206 SMD)	IMS	$0.00
4	200-Ω resistor (1206 SMD)	IMS	$0.00
12	240-Ω resistor (1206 SMD)	IMS	$0.00
1	300-Ω resistor (1206 SMD)	IMS	$0.00
??	330-Ω resistor (1206 SMD)	IMS	$0.00
12	500-Ω resistor (1206 SMD)	IMS	$0.00
7	604-Ω resistor (1206 SMD)	IMS	$0.00
6	976-Ω resistor (1206 SMD)	IMS	$0.00
8	1-kΩ resistor (1206 SMD)	IMS	$0.00
2	3.9-kΩ resistor (1206 SMD)	IMS	$0.00

[1] C93, C95, C97, C99, C101, C104, C106, C107, C109, C111, C113, C115, C118, C120, C121, C123, C125, C127, C129, C132, C134, C135, C127, C139, C141, C143, C146, C148, C149, C152, C154, C155, C158, C160, C161, C164, C166, C169, C171, C172, C174, C175, C178, C180, C183, C185, C186, C188, C190, C192, C193, C196, C198, C202, C205, C207, C208, C212, C213, C214, C217, C276, C278, C280, C282, C284, C286, C288, C290, C296, C298, C300, C302 C304, C206 C312, C314, C316, C318

[2] C94, C96, C98, C100, C102, C105, C103, C108, C110, C112, C114, C116, C117, C119, C122, C124, C126, C128, C130, C131, C133, C136, C138, C140, C142, C144, C145, C147, C150, C151, C153, C156, C157, C159, C162, C163, C165, C167, C168, C170, C173, C176, C177, C179, C181, C182, C184, C187, C189, C191, C194, C195, C197, C199, C200, C201, C203, C204, C206, C209, C210, C212, C215, C216, C226, C227, C228, C229, C230, C231, C232, C233, C234, C235, C240, C241, C265, C266, C267, C268, C269, C270, C271, C272, C273, C274, C275, C277, C279, C281, C283, C285, C287, C289, C291, C292, C293, C295, C297, C299, C301, C303, C305 C307, C308, C309, C310 C313, C315, C317, C319, C320, C321, C322, C332, C333, C334

[3] C24, C81, C82, C218, C219, C220, C221, C222, C223, C224, C225, C242, C243, C244, C245, C248, C249, C251, C252, C255, C256, C259

Table E.1: (Continued)

RefDes	Component Description	Manufacturer Part Number	Unit Price
See note [4]	4.7-kΩ resistor (1206 SMD)	IMS	$0.00
6	5-kΩ resistor (1206 SMD)	IMS	$0.00
See note [5]	10-kΩ resistor (1206 SMD)	IMS	$0.00
2	22.1-kΩ resistor (1206 SMD)	IMS	$0.00
2	39.2-kΩ resistor (1206 SMD)	IMS	$0.00
2	47.5-kΩ resistor (1206 SMD)	IMS	$0.00
4	100-kΩ resistor (1206 SMD)	IMS	$0.00
4	680-kΩ resistor (1206 SMD)	IMS	$0.00
P1	Female PCB-mount DB-9 connector		$0.00
P2	Male PCB-mount DB-9 connector		$0.00
P3, P8, P9, P10, P11, P12, P13, P14, P15	PCB-mount BNC connector		$0.00
L7, L8, L9, L10, L11, L12, L13, L14	Ferrite bead		$0.00
N/A	Printed circuit board	Paramount Circuits N/A	$600.00

[4] R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25, R26, R27, R28, R29, R84, R85, R131, R132, R133, R170

[5] R31, R50, R51, R52, R54, R55, R56, R67, R68, R69, R70, R73, R75, R76, R77, R79, R80, R81, R82, R117, R118, R119, R120, R121, R122, R124, R125, R126, R128, R129, R130

APPENDIX F

REVISION 1.0 ERRATA

Date: September 30, 1997

RefDes: P1, P2

Device: 9-pin DIN connectors for RS-232 interface to 68360.

Error: Footprint placed in incorrect orientation.

Fix: Will have to jumper wires to that connector when it is used. No part can be mounted in the current configuration.

Date: October 8, 1997

RefDes: U4

Device: 72-pin DRAM SIMM socket.

Error: Footprint mounting holes are incorrectly placed.

Fix: Must modify DRAM SIMM socket to accomodate the board. Simply remove one of the plastic alignment pegs so that it fits. The part can still be used with this modification.

Date: October 12, 1997

RefDes: U1

Device: MAX811 microprocessor voltage monitor.

Error: Device causes timing problems when in development mode.

Fix: This device causes problems when the board is in development mode (debugging through background debugger). Therefore, it is best to remove the part. When the part is not on the board, short pins 2 and 3.

Date: February 12, 1998

RefDes: N/A

Device: AD9220 VREF decoupling capacitor.

Error: Device is missing from schematic.

Fix: A 0.1-μF decoupling capacitor must be added to the board between VREF and AGND2. The best place for this capacitor is on the bottom side of the board right next to the via closest to VREF. A small amount of the solder mask can be scraped off with a knife to provide a connection to AGND2.

Date: April 24, 1998

RefDes: U60, U61

Device: AD9220 input range.

Error: Design change from +/- 2.5 V input range to +/- 1 V.

Fix: Disconnect SENSE (pin 17) from GND and reconnect to VREF (pin 18). The change is easiest done by desoldering the SENSE pin and pulling the pin off the surface. Then connect (using fine wire) to VREF.

Date: April 24, 1998

RefDes: R177, R184

Device: AD9220 VREF pullup resistor.

Error: Value change from 1 kΩ to 7.5 kΩ.

Fix: Value change may help offset error. Preliminary tests show that it does not.

REFERENCES

[1] B. V. Herzen, "Signal processing at 250 MHz using high-performance FPGA's," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, pp. 238–246, June 1998.

[2] A. Tsutsui, T. Miyazaki, K. Yamada, and N. Ohta, "Special purpose FPGA for high-speed digital telecommunication systems," in *Proceesings of the IEEE International Conference on Computer Design: VLSI in Computers and Processing*, 1995, pp. 486–491.

[3] S. Hauck, "The roles of FPGA's in reprogrammable systems," *Proceedings of the IEEE*, vol. 86, pp. 615–638, Apr. 1998.

[4] J. McCloskey, "Application of VHDL to software radio technology," in *Proceedings of the 1998 International Verilog HDL Conference and VHDL International Users Forum*, 1998, pp. 90–95.

[5] M. Vaupel, T. Grötker, and H. Meyr, "ComBox: Library-based generation of VHDL modules," in *Workshop on VLSI Signal Processing, IX*, 1996, pp. 293–302.

[6] M. Dörfel and R. Hofmann, "A prototyping system for high performance communication systems," in *Proceedings of the Ninth International Workshop on Rapid Systems Prototyping*, 1998, pp. 84–85.

[7] C. Deltoso, C. Joanblanq, M. Cand, and P. Senn, "Fast prototyping based on generic and synthesizable VHDL models a case study: Punctured Viterbi decoders," in *Proceedings of the Seventh International Workshop on Rapid Systems Prototyping*, 1996, pp. 158–163.

[8] E. P. Mariatos, M. K. Birbas, and A. N. Birbas, "A reconfigurable DSP board based on CORDIC elements," in *Proceedings of the Fifth International Workshop on Rapid Systems Prototyping: Shortening the Path from Specification to Prototype*, 1994, pp. 22–25.

[9] S. Swanchara, S. Harper, and P. Athanas, "A stream-based configurable computing radio testbed," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, 1998, pp. 40–47.

[10] B. Kamali, "Development of an undergraduate structured laboratory to support classical and new base technology experiments in communications," *IEEE Transactions on Education*, vol. 2, pp. 97–100, Feb. 1994.

[11] B. Overstreet and J. Austen, "A DSP based laboratory system for exploring digital communication concepts," in *IEEE Proceedings of Southeastcom '95: Visualize the Future*, 1995, pp. 278–281.

[12] J. O. Hamblen, H. L. Owen, S. Yalamanchili, and B. Dao, "An undergraduate computer engineering rapid systems prototyping design laboratory," *IEEE Transactions on Education*, vol. 42, pp. 8–14, Feb. 1999.

[13] Electrical and Computer Engineering Department, University of Illinois at Urbana-Champaign, "Microcomputer laboratory," http://www.ece.uiuc.edu/ece311/index.html, 1999.

[14] Xilinx, Inc., *The Programmable Logic Databook*, 1996.

[15] Xilinx, Inc., *CORE Solutions Databook*, 1998.

[16] P. J. Ashenden, *The Designer's Guide to VHDL*. San Francisco: Morgan Kaufmann Publishers, Inc., 1996.

[17] OAR Corp., *RTEMS C User's Guide*, 1999.

[18] J. G. Proakis and M. Salehi, *Communications Systems Engineering*. Englewood Cliffs: Prentice-Hall, Inc., 1994.

[19] Analog Devices, Inc., *Complete 12-Bit 1.5/3.0/10.0 MSPS Monolithic A/D Converters*, 1999.

[20] J. P. Costas, "Means for Obtaining Character Time in a Radio Communication System Receiver," U.S. Patent 3,047,660, 1962.

[21] E. A. Lee and D. G. Messerschmitt, *Digital Communication*. Boston: Kluwer Academic Publishers, 1988.

[22] M. S. Palac, "A first generation test bed for the analysis of joint source-channel coding algorithms, with a discussion of their application to the broadcast scenario," M.S. thesis, University of Illinois at Urbana-Champaign, 1998.

[23] Analog Devices, Inc., *AD9850 Datasheet*, 1997.

[24] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters - parts I, II," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, pp. 1099–1134, July 1989.

[25] N. R. Shanbhag, "Algorithm transformation techniques for low-power wireless VLSI systems design," *International Journal of Wireless Information Networks*, pp. 23–40, 1998.

[26] Motorola, Inc., *Motorola DSP 56300 Family Manual*, 1997.

[27] J. Janovetz, "Erithacus digital communications system," http://www.uiuc.edu/ph/www/janovetz/erithacus.html, 1999.