Bandwidth Unlimited
School of Engineering Science
Burnaby, BC
V5A 1S6
bandwidth-unlimited@sfu.ca

December 19, 2001

Dr. Andrew Rawicz
School of Engineering Science
Simon Fraser University
Burnaby, BC
V5A 1S6

Re: ENSC 340 Process Report for MRx Home Theatre Interface

Dear Dr. Rawicz:

Attached you will find the *MRx Home Theatre Interface Process Report* for Bandwidth Unlimited.

This document presents a brief summary of our project technical implementation, followed by a discussion of some of the issues we faced during the course of the semester. Each team member reflects on their personal ENSC 340 experience, and we discuss future improvements for our project.

Bandwidth Unlimited is a team comprised of five highly skilled and talented engineering science students: Mavis Chan, Brian Fraser, Manpreet Gakhal, Ben Lake, and Gabrielle Sheung. If you have any questions or concerns about our process report, please do not hesitate to email us at bandwidth-unlimited@sfu.ca, or to phone me at (604) 299-3199. Thank you for your time.

Sincerely,

Manpreet Kaur Gakhal
Bandwidth Unlimited

Enclosure: MRx Home Theatre Interface Process Report

Bandwidth Unlimited

MRx Home Theatre Interface Process Report

Project team:

Mavis Chan
Brian Fraser
Manpreet Gakhal
Ben Lake
Gabrielle Sheung

Contact Personnel:

Ben Lake – benl@sfu.ca

Submitted To:

Dr. Andrew Rawicz – ENSC 340
Steve Whitmore – ENSC 305
School of Engineering Science
Simon Fraser University

Issue Date:

December 19, 2001

Revision: *1.0*

Table of Contents

1. System Overview

The MRx system allows people to play MP3 sound files stored on the host computer on their own home theatre systems. Figure 1 below illustrates the basic structure of this system. The host computer contains the library of MP3 files, and will send a stream of MP3 encoded audio to the MRx device via an Ethernet connection. The MRx device will deliver the decoded sound to an adjacent home theatre.

A two-way infrared link between the Palm Pilot device and the MRx allows the Palm Pilot to be used as a remote control for the MRx. The user interfaces on the host computer and on the Palm Pilot device allow the user full control over the music selection and playback, and provide the user with updated status information from the MRx. The front panel of the MRx device also allows for limited control over the music playback and provides the user with graphical feedback via an LCD screen.

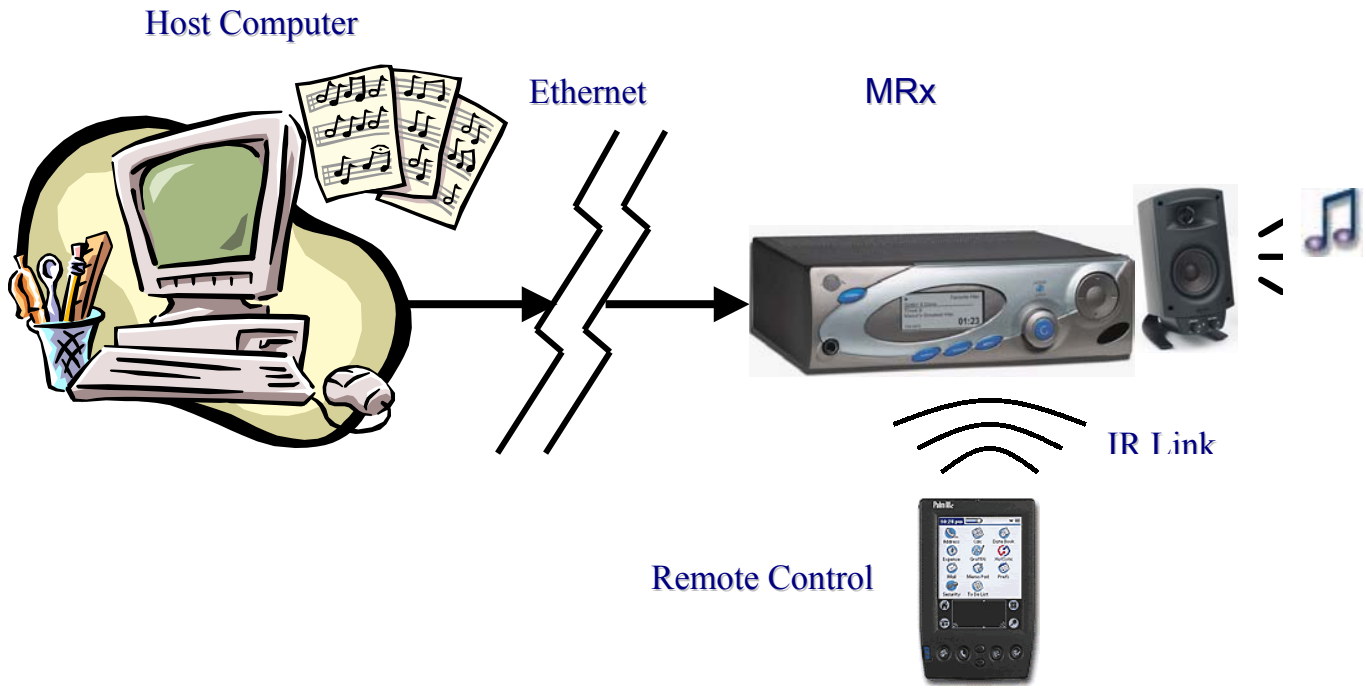


Figure 1: System Overview

2. Technical Implementation

The following sections briefly present the technical approach we took when implementing the various sections of our project.

2.1 Hardware Implementation

The MRx hardware is comprised of a high-speed CPU with sufficient memory resources and communication peripherals to satisfy the requirements laid out in the *MRx Functional Specification*. Only the subsystems within the MRx hardware that were subjected to rigourous research and development are discussed here. The rest of the hardware was implemented as intended in the *MRx Design Specification*. A block diagram of the hardware architecture is shown below in Figure 2.1.

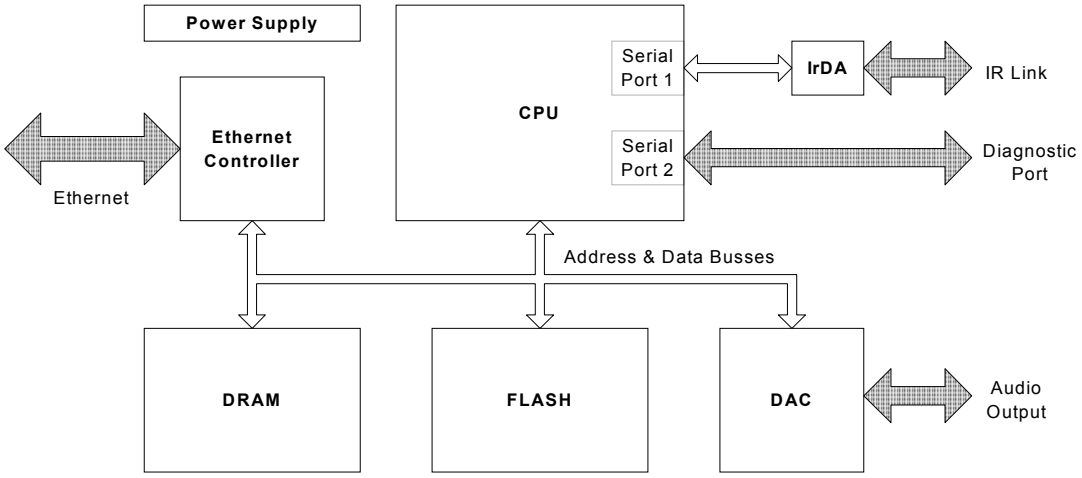


Figure 2.1 – MRx Hardware Architecture

CPU
The processor selected for the board is the Cirrus Logic EP7212. The CPU is affordable, resplendent with features suited to an embedded audio device such as ours, and features a readily available development board as well. The EP7212 offers RISC based processing in its ARM core, and offers integrated sub-systems that were similar to our project requirements. The processor datasheet specifies that the CPU will use 50-70% of its runtime to decode an MP3 file in real-time, allowing an acceptable amount of overhead for other data processing and message management.

Ethernet Interface
The Ethernet interface of the MRx is built around a Cirrus Logic CS8900A Ethernet Controller. The controller provides all required functionality for a 10 Base T Ethernet

interface and requires no additional logic to interface to the processor busses. No serial PROM is required to configure the controller, since it is not a requirement that the Ethernet connection be operational on power up. The CPU configures the Ethernet controller during the initialization phase of board start up. The Ethernet controller has operated flawlessly throughout the development phase of the product.

IR Interface
The design of the IR interface was chosen to use the IrDA standard for IR communication. There is no other comparable standard for IR data communication, and developing a new standard was estimated to be too much work. The IrDA standard specifies a maximum distance of 1m. Future hardware development for this product will include research and testing to extend the communication distance to at least 3m. In the design of the MRx hardware, initial research has shown that reducing the bit rate to 9600bps and increasing the numbers of IR transceivers on the MRx hardware should sufficiently increase the remote control range.

Printed Circuit Board
A schematic of the MRx motherboard was developed based on the block diagram shown above. A printed circuit board was then laid out, based on the estimated dimensions of the final product and the estimated location of the external connectors. The complete PCB schematic and layout are included in Appendices A and B, for reference.

2.2 Software Implementation

2.2.1 Overview

The software for our system consists of four separate executable programs: the Windows file server, the Windows GUI, the Palm GUI, and the Linux MRx code. All modules communicate directly with the MRx, and the MRx handles the flow of information through the system.

2.2.2 MRX Application Software

The MRx is both the center of communications, as well as the main processing center of the system. The MRx is written in C++, and compiled for Linux using the g++ General Public License (GPL) compiler and linker. Development of the system was done primarily under Turbo Linux running on a PC. For low level issues (such as sound, Ethernet and serial support) the Cirrus Logic EP7212 Evaluation Board was also used.

The software is an object oriented, multi-threaded based system. General processing in the system is done using more than ten threads within the single executable. All of these threads interact with each other using a custom designed message passing mechanism.

This new mechanism is necessary because Linux does not support inter-thread communication. This custom scheme uses only mutexes to implement a robust and efficient communication architecture. The system breaks down into the following general components:

Control Thread
The Control thread is the center point of the multi-threaded architecture. All of the major components exchange messages with other components through the control thread. This thread allows us to centralize control and easily segregate data production and data routing (i.e., "what data to send out" vs "who needs to get the data"). For example, when the MP3 thread generates status updates, the information is sent directly to the control thread. From the control thread, the information is redirected to the Palm and the Windows GUI's. If another client were added, the MP3 thread would never know the difference.

PC GUI Thread
This thread uses a class we created called CPort. The CPort class watches a UDP port from the Ethernet link and relays all incoming data to the control thread. Also, messages to the PC GUI are sent out through this thread.

PC File Server Thread
Similar to the PC GUI thread, the file server thread handles all activity between the file server and the MRx. Like the previous thread, it uses the CPort class.

IR Thread (to Palm Pilot)
The IR Thread is the third instance of the CPort class. It uses the identical structure as the previous two threads, except that it communicates over infrared or (as is currently used) the serial port.

MP3 Thread
The MP3 thread accepts the MP3 file fragments from the control thread and assembles them into a single MP3 file data structure. This thread spawns a new thread to play each song and controls which song is playing and all play controls.

2.2.3 MP3 Decoding

Because decoding MP3 is a complex process, we decided to use a free public library called MAD (MPEG audio decoder) instead of writing our own decoding algorithm. An MP3 decoding thread starts whenever we play a new song. This was chosen as opposed to a continuous thread decoding many songs because of performance. If only one thread is used and no music is playing, we would need to poll for a play command, which would take up a lot of CPU time. If one MP3 decoding thread is spawned for every song and

killed at the end of the song, this problem is eliminated. It was also easier to implement spawning a new thread each time a song is played.

The decoding thread takes an MP3 file pointer and extract blocks of data to decode. After the library has finished decoding, the decoded data is stored in an output buffer. The output buffer is then written a little at a time to the speaker. Because decoding takes much less time than writing to the audio port, stuttering of sound is not a concern. If the output buffer is ever near empty, the decoding algorithm could fill it up quickly.

2.2.4 File Server

The file server is a separate application that runs on the PC where all the playlist and MP3 files are stored. An Ethernet connection with the MRx device will allow the file server to communicate and transfer data with the MRx control software when requests are received. The separation of the file server functionality from the user interfaces, and the fact that the file server never interfaces directly with a user interface, allows the file server to support multiple user interfaces.

2.2.5 User Interfaces

PC User Interface

The PC Graphical User Interface is the front-end controller for the MRx device. We decided to use Visual Basic to implement the interface because it provided a rapid development environment and easy access to graphical components. The user interface provides easy access to the basic audio controls (play, stop, etc), playlist editing capabilities, and sufficient user feedback.

Palm User Interface

The Palm Graphical User Interface is the remote controller for the MRx device. Again, we used Visual Basic with an AppForge plug-in to allow us to write code for the Palm OS. The code for the PC user interface is written in such a way that it is portable to the Palm OS with only a few changes. Therefore, the Palm interface has the full functionality of the PC interface. We are only limited physically by the size of the palm screen.

3. Problems Encountered

As our project evolved over the course of the semester, we encountered various unexpected problems. The major issues are discussed in the following sections.

3.1.1 Technical Issues

Complexity vs. Cost

The MRx PCB we designed is extremely dense and there is very little spacing between traces around the CPU. Allowing more signal layers in the PCB would reduce congestion and signal crosstalk. The additional layers add considerable cost onto the PCB. Approximately $100-$200 extra per layer for prototype volume orders. With almost no money to construct the project, these board design costs halted all plans to build a prototype before December.

Evaluation Board

The evaluation board and software used for this project where loaned to our team for free. Normally, the hardware sells for $1,500US and the software for $3,000US. We found that the hardware was very well put together and useful. However, we are not so happy with the software. The software is BlueCat Linux by LynuxWorks. We received the package that explicitly supports our evaluation board and includes the operating system and some simple applications. There is very little useful documentation provided with the software about how to interface with the provided hardware on the board. For example, “how does one enable the infrared functionality?” and “how can one play sound?”. The majority of these problems were either resolved through sweat and tears, or begged support from LynuxWorks, and some of our questions remain unanswered. It is our opinion that a little extra sample code and documentation could go a long way into improving the software package.

Evaluation Board Memory Management Unit

The Memory Management Unit (MMU) of the EP7212 processor was a cause of considerable grief during the hardware development. It is suspected that some of the memory management problems observed during project development, could be traced back to the Linux kernel not correctly setting up the MMU to map one contiguous memory block for the program. The debugging tools and the time to isolate this problem were not available to our group.

Evaluation Board vs. Linux PC

We had two versions of our MRx software: one that was run on the evaluation board as intended, and one that was run on a Linux based PC that simulates the evaluation board. The two major limitations of the evaluation board version of the MRx compared to the PC version were stability and multitasking.

Stability is a problem for the evaluation board version because of a memory management issue. Since our system uses so many threads and processes so much data in real time, there are a substantial number of dynamic memory allocations within our code. We believe that our system leaks a lot of memory (i.e., it allocates blocks of memory and never releases them). This could be resolved given more time to debug and clean up the MRx code and a better understanding of the memory management on the evaluation board.

Multitasking is a problem for the board when it is required to perform two processor intensive tasks at once. For example, it cannot download and play an MP3 at the same time. This is expected to be a much more difficult issue to solve than simple memory management. We believe that the CPU on the board does not have sufficient processing power to process the two demanding tasks at once. Also, it may be an issue with the way the operating system (BlueCat Linux) processes interrupts.

MP3 Decoding and Sound Quality

The sound quality of the final output varies between the Linux computers we used to simulate the functionality of our evaluation board. The Linux PC provided by Patrick Leung has the best sound quality we observed. The sound plays correctly and is clear. Next, the sound on Brian's laptop (running Linux) has noticeable clicks and pops in it, which we believe is because the sound card support for this computer is lacking. We expect that as the support improves, so will the sound quality.

The sound on the evaluation board is limited to mono playback. If stereo music is played on the board, the music slows down with noticeable pauses; however, there were no problems with stereo files on the PC applications. We conclude that the likely cause of the stereo problems on the board lies in either the Linux sound driver support, or our usage of the driver. Perhaps the problem is the byte order for stereo vs mono signals for the ARM based system instead of the PC based system.

Also, we encountered some difficulties with the MP3 decoding library we used. The MPEG Audio Decoding (MAD) library is available under General Public License (GPL), and is only version 0.14.2b. Since it is still developmental (not even a 1.0 release), it is completely undocumented. Most of our knowledge about the library was derived from studying examples. We suspect that there are many functions of the library that we are

not currently using (for example, extracting the total time length of the song) that would improve our MP3 decoding ability. It is expected that when the library reaches maturity (and is documented), it could be integrated with our product to enhance our playback performance.

Programming in a Linux Environment

The first problem that arose while writing the multithreaded Linux application is that of "General Protection Faults". General protection faults occur whenever a program tries to perform an invalid operation on memory. For example, the most common occurrence of this is de-referencing a null pointer. When running a single threaded Linux application, when it faults, it creates a "core" file that tells the curious developer where the error occurred. However, with multiple threads, this file is useless because it only deals with the first thread the program created. To solve this issue, we were able to add some code that prints out which thread has performed the illegal operation. This way we are at least able to isolate which thread has caused our problem.

Another problem centered on the compiler not producing a functional program. Often, when the MRx was compiled, it would simply crash as soon as it was run. This was not a problem with our code because sometimes we would change a single print statement that is called after 10 minutes of operation, and it would begin crashing immediately. It was determined that changing the level of optimization of the compiler could (usually) fix this problem. This is expected to be a compiler problem, or perhaps an issue with our build process.

Programming on the Palm

Initially, the Palm interface was difficult to work with because we used CodeWarrior, which uses the C language, as our development environment. This required us to spend and extensive amount of time to set up the environment for development and learn how the use the interface objects. We later discovered we could used Appforge to develop in Visual Basic. This turned out to be a better solution because our carefully written code for the PC interface was easily portable to the Palm with minor changes. However, AppForge is not as perfect as was imagined: some functions did not exist under AppForge (such as screen management). Also, AppForge causes some serious problems when the program is run on the Palm Pilot. For example, the Palm Pilot often crashes when the program is loaded.

3.1.2 Financial Issues

The cost of developing the MRx hardware initially looked like it would be similar in cost to many other small two-layer boards constructed in the past. However, once the

schematic was entered and the components were designed, the initial attempts at routing the board layout clearly showed that a six or eight layer board is required to safely design the MRx motherboard. Purchasing of components for the board was halted once the cost of the PCB was known. In order to provide a complete picture for future investors, the cost of developing a prototype is tabulated in Table 1.

Table 1 – MRx Motherboard Prototype Budget

Part	Cost (CDN$)
Evaluation Board	$0 (on loan)
Main Processor	$0 (sample)
DRAM	$0 (sample)
DAC	$0 (sample)
FLASH	$90
Peripheral Components	$150
LCD	$160
Printed Circuit Board	$1500
Total	**$1900**

The MRx is a commercially viable product. Other than showing the market need for the device, the best way to show the 'marketability' of the product is to show the profit that it can generate when sold. To be competitive, the retail price of the device should be ~$299, corresponding to a wholesale price of ~$199. The estimated product cost is shown in Table 2.

Table 2 – MRx Motherboard Volume Production Cost

Part	Cost (CDN$)
PCB	$40
Main Processor	$20
DRAM	$10
DAC	$5
FLASH	$10
Peripheral Components	$15
LCD	$40
Chassis	$20
Assembly Overhead	$20
Total	**$180**

Additional pricing research is required to verify that these estimates are accurate. Increasing production volume and careful attention from manufacturing engineers could considerably streamline the cost structure of the MRx.

3.1.3 Scheduling Issues

Our expected schedule was relatively accurate compared to our actual schedule. Since our design was done in parallel with the project documentation deadlines, our documents were completed on time quite easily.

We took longer than expected with our design process, and we could have benefited from more implementation and debugging time. However, our longer design time was justified because we took the time to create solid protocol and thread definitions. This made our implementation of the different parts of the system easier in the long run as we ran into fewer unexpected problems.

Our communication threads and the main MRx architecture was completed on time, which made it easier to implement the other system components that were built on top of this base code. Also, our file server was completed early and made the implementation of the user interfaces much easier to test. We decided to devote more time to the PC user interface to make it more user-friendly and solid in functionality. We also had problems with the getting the sound playing which delayed our implementation of the MP3 decoding implementation a little. Changing our development software for the Palm interface during implementation delayed our expected finish date for the Palm interface a little. These delays in implementation cut into our debugging time, which we had planned to use for full testing of the system.

However, even with all the setbacks that we encountered we were still able to finish about 90% of our project goals by our target demonstration date.

3.1.4 Team Dynamics Issues

Our team worked really well together. This is mainly because the tasks were distributed evenly and were very well defined. Any interfaces between parts were also very well defined, so as long as the design criteria were followed, parts integrated smoothly together. Because of this, there were negligible conflicts between group members. Even though we had our own parts to do, we held weekly meetings to update each other on our progress, with more frequent informal meeting held during the implementation process. This way, we were accountable to other team members, which helped to keep our project progressing. Everyone did their part and finished tasks relatively on schedule, with no observable differences between the amount of work each member contributed. There was no defined leader in our group; everyone became the expert in their own portion of the project. We feel this lack of team hierarchy also helped eliminate any group dynamic issues we would have had otherwise.

4. What We Would Change

Timeline

We initially spent much time trying to choose between this project and a slightly different project. Had we made our final decision on our project earlier in the semester, we would have had more time for implementation of the project. Also, starting the hunt for hardware components earlier, and securing development tools sooner would have allowed more research time before the serious development began.

Project Scope

We feel that our initial project scope was very ambitious. Although we were successful in implementing most of our initial plans, we believe that if we had shortened our project scope slightly by having fewer features we would have been able to more solidly implement the core features of our project.

Hardware

The choice of a simplified hardware solution might have increased our chances of successfully constructing a PCB before December. A less complex processor driving a hardware MP3 codec does not require nearly the PCB density of our current hardware solution, and could have been easily implemented on a two or four layer board. The only reason that this solution was less attractive to our group initially was that the codec increases component cost, the design is less flexible to future changes, and one of the driving forces behind our project was to demonstrate the ability to decode MP3 files using only embedded software.

5. Personal Experiences

Brian Fraser

I learned how difficult it is to configure an evaluation board to run even a simple application. It took many hours, and many tries to have the board work the way we needed it to. From this, I learned how important it is to have a compatible PC environment to work with the embedded system. I expect that Linux is an obvious choice for my future projects for this reason.

I realized how important very strong code writing practices are in a large system. For example, one must always assert that a pointer is valid before de-referencing it. These types of bugs caused many problems throughout the semester. Also, I gained experience with how a strong software design can vastly reduce the amount of work required. This was especially true with the application's multithreaded architecture.

Finally, I believe that devotion of group members allowed this project to succeed. It is only now that I realize how few "team dynamics" problems we had, and how this allowed us to succeed. It is a wonderful feeling have team members finish all tasks well and on time. This is a relatively rare feeling for most projects; however, I frequently had this feeling this semester. Team members each took on responsibilities willingly, and then performed as required.

Ben Lake

My years in the electronics industry before coming to SFU have thrown me headfirst into many long-term hardware projects, so I anticipated taking Ensc340 with great zeal. My experience allowed me to expect many things during the project, although the compressed time schedule, and the lack of proper funding and equipment created many situations that were unexpected.

The hardware was more complicated than expected. It is very difficult to look at any design from a block diagram, and then figure out the total circuit complexity. Various bus interfaces on the board required a great deal more care an understanding than the 'high level system diagram' ever shows. One of the lessons that I learned was that you have to try to build every single component footprint and schematic symbol, draw the schematic, place the components, and route the board, before a full appreciation of the PCB CAD groups in most companies can be gained. These tasks are usually shared by multiple specialised people, not one undergrad hovering over his home PC for endless hours.

Also, during the first few project meetings, I realized that my knowledge of hardware was too specialized to lead an entire system-design project. Our group hierarchy ended up being completely flat, with each person specializing in a certain task.

Manpreet Gakhal

The experience I had this semester working on this project was certainly one that I will not forget. The constant pressure of trying to complete such a complex project within a very short amount of time, combined with the pressure of a normal full course load, certainly made the semester fly by. There were many lessons learned as a group and as individuals.

I feel that I have gained numerous technical skills and much experience from this project. This was the first time that I have worked on such a complex, multithreaded application, and I have certainly learned much about how to design, implement, and debug a project such as this. Writing the file server application allowed me to take a crash course in visual basic programming, and doing development work under a Linux environment has allowed me to improve the flexibility of my programming skills.

I believe that our success can be attributed to the positive attitude and work ethic that each member of this team approached this project with. I have learned how essential it is to be able to partition a large project into smaller tasks that one person can claim responsibility for. I recognise now more than ever before the importance of devoting lots of time to planning and design before moving to implementation. I feel that our team organisation and planning was invaluable in contributing to our success.

Mavis Chan

Since much of my work focussed on building the user interfaces and webpage, my experience during this project was a good follow up to my co-op work terms where I spent some time creating simple interfaces. I learned the importance of defining a good communications protocol between all the components of the system before implementation. At the time I didn't realize that the time we spent defining the protocol would make my coding much easier in the long run, but now I fully appreciate it.

Prior to this project, I had never created a graphical user interfaces with such extensive user interaction capability. It forced me think about how the user would perceive the finished user interfaces during the design phase. It would have obviously been easier and faster for me to implement the user interfaces using the bare minimum code. At the same time I had to realize the benefits of some automation for the user to easily understand how to use the finished product even if it meant more complicated code.

I developed better technical communication skills with my team members since we often had to discuss and mutually agree on a certain way of implementing something to make our project work. In retrospect, I must also add that the recipe for the success of our project was in large part due to our entire team's motivation to reach our goal and our accountability to each other to achieve this goal.

Gabrielle Sheung

ENSC 340 has been a tremendous learning experience for me. I developed both my technical skills as an engineer as well as my management skills to work as a team. On the technical side, I learned that MP3 encoding and decoding is much more fascinating than I previously thought. I also learned to read and decipher a poorly documented library enough to interface with it to decode our MP3's. This exercise taught me the value of comments and good documentation. Besides learning how to decode MP3, I also learned how to implement and use threads thoroughly. Although I had some experience with threads from ENSC 351, I never had to use them as extensively as I did for the MRx project.

Through our many hours of code development and debugging, I learned to be very patient and persistent. Although persistence pays off, we have learned to take breaks to refresh our eyes when looking at a problem. Staring at the same code for too long impairs our ability to see things clearly. Sometimes we leave a problem until the next morning, just to have the solution jump out at us right away. Another coding habit I found very important is a clear plan of what needs to be done. Without knowing exactly what needs to be done, there is no way of knowing when you are finished. Good planning resulted in less coding time. The design process also helped us plan out each function clearly. With well-defined functions, they could be reused easily within the code, thus making everything easier to code.

Working in a team pushed me to work more than ever. Since we were accountable to other group members, no one wants to seem like 'deadweight'. I found our organisation and coordination to be absolutely superb. Our weekly meetings were useful and kept everyone on track. I also learned that because there is always more than one way to approach a problem, keeping an open mind to others' ideas will result in the best solution for the team. This way we can compare and incorporate aspects of different ideas into one.

6. Future Improvements

Although we have accomplished much of what we had set out to achieve in this project, there are still a few things that we improve upon in the future:

MRx Interface
On our completed MRx box, we would like to have a simple LCD interface with several audio control buttons on the face of product to further enhance user interaction with the system.

IR Communication
At this point, our Palm interface can communicate to the MRx device by using our Palm Pilot on its cradle, which essentially sends the data over a serial cable to the MRx device. To fully realize our Palm as a remote control, we need to further investigate the infrared port communication on the MRx device. As discussed in the hardware section, the IR communication requires additional research and testing to verify that the range can indeed be extended as required.

Improved Sound Quality
At the moment, we are only able to play good mono MP3's. We need to investigate how to play good quality stereo MP3's to make this device more compatible with most available MP3's.

Custom Remote Control
A custom designed remote control would allow Bandwidth Unlimited to maintain more control over how the remote control behaves with respect to range, battery life, and ergonomics. Since we can not expect everyone who uses our product to have a Palm Pilot, we could build a custom remote control with a LCD display that would have the full functionality of the interface we have developed for the Palm. Custom designed devices can also be priced to allow for much higher profit margins, rather than just reselling other manufacturer’s equipment, such as the Palm.

Multiple MRx and PC GUI's
At the moment, we have only implemented our system to connect with one MRx device and one PC user interface. Ideally, the user should be able to access the system from different user interfaces on different computers or install multiple MRx devices in different rooms of their home.

Chassis Design
A folded sheet metal chassis with an elegantly molded plastic faceplate needs to be designed for the device so that the prototype can be shown to investors, and demonstrated to the public.

Build PCB Prototype
To develop our project, we have used an evaluation board. To fully implement the hardware portion of our project we will need to build and test the PCB prototype that has been designed for our project. However, this is an expensive task, so we will need sufficient funding.

Proper Business Plan
To attract attention to our project and get funding to fully implement all the features of this system, we will need to create a proper business plan. Accurate and concise prototype and volume hardware costs need to be calculated so that the feasibility of the business-side of the project can be properly demonstrated.

7. Acknowledgements

Bandwidth-Unlimited would like to thank a number of persons and companies for helping us get our project moving, and keeping it going over the past five months.

We would like to thank Andrew Rawicz, Steve Whitmore, Maria Trinh, Roch Ripley, Patrick Leung, and Fred Heep for their assistance and advice throughout the semester.

Dave Miller of DFM Technologies Inc. has been instrumental in acquiring hardware and software support for the main platform for the MRx.

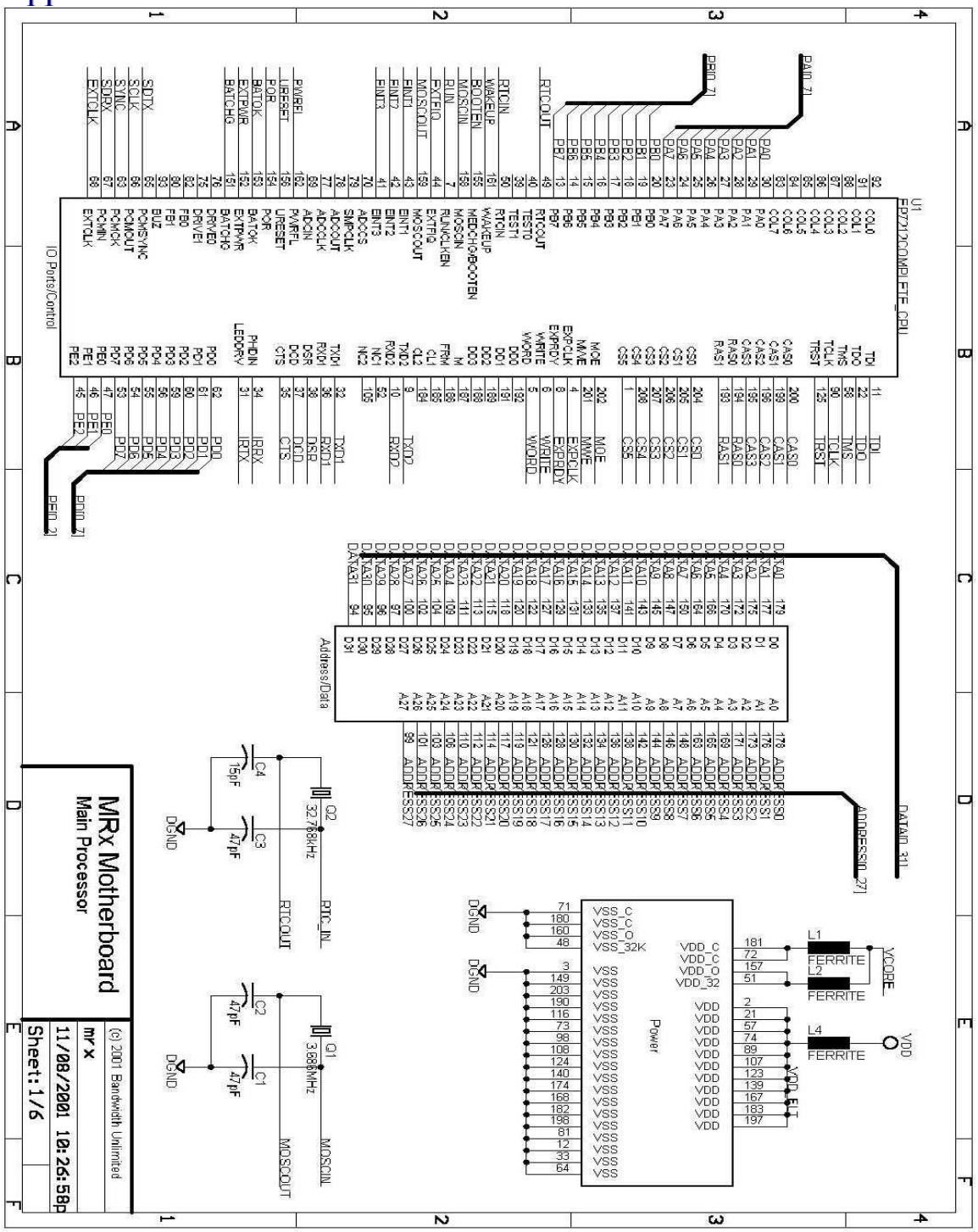
Mehdi Tamehi and Doug Stewart of Insight Components have worked with us to loan us the board and the software needed to run on it.

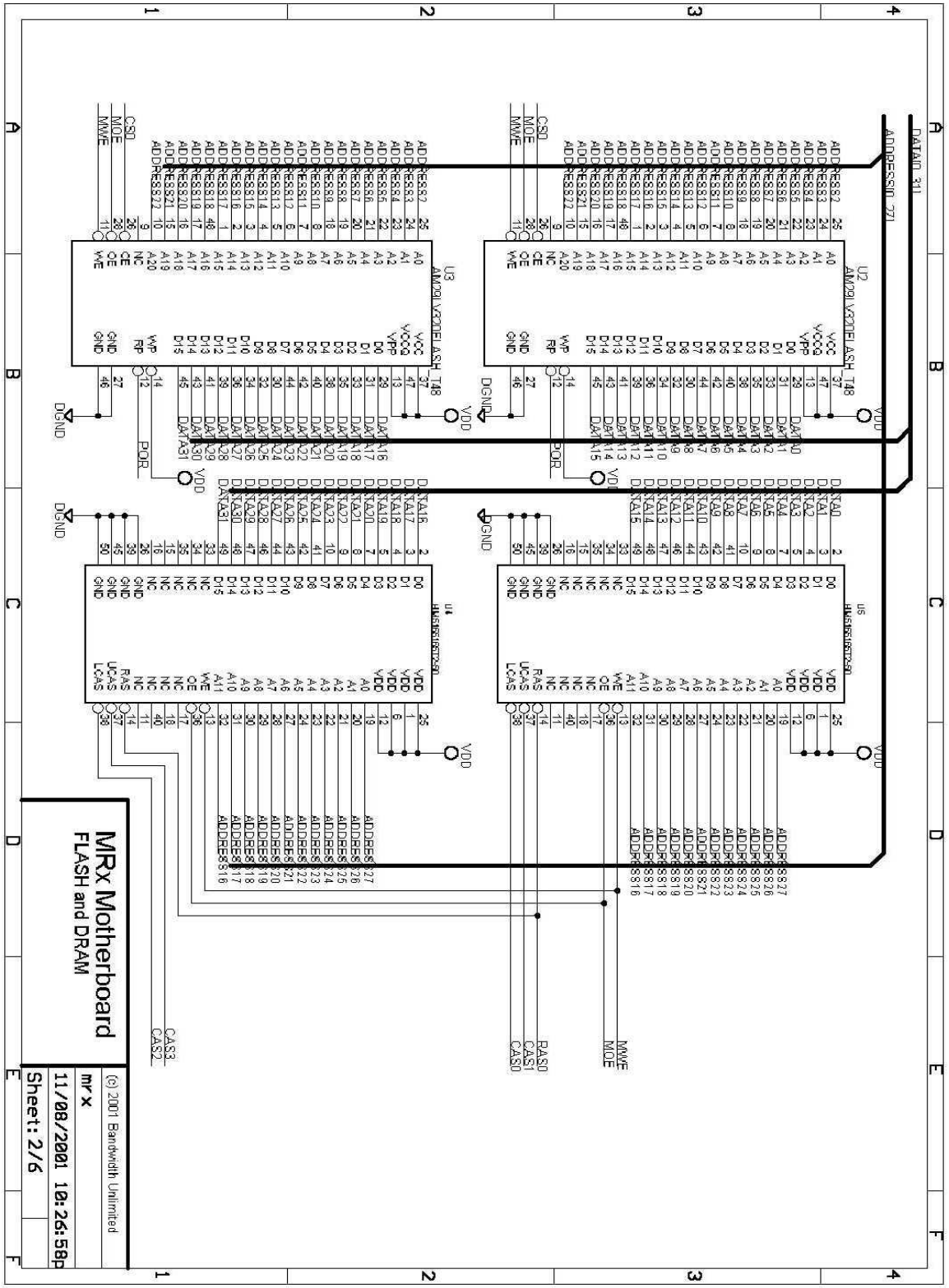
We would like to thank Ron Lockard of LynxWorks for the direct support he has given to us to configure and use BlueCat Linux. Also, we would like to thank Nels Esterby (local Cirrus Logic representative) of Micro-Electronics for providing assistance in borrowing the evaluation board from Insight Components.

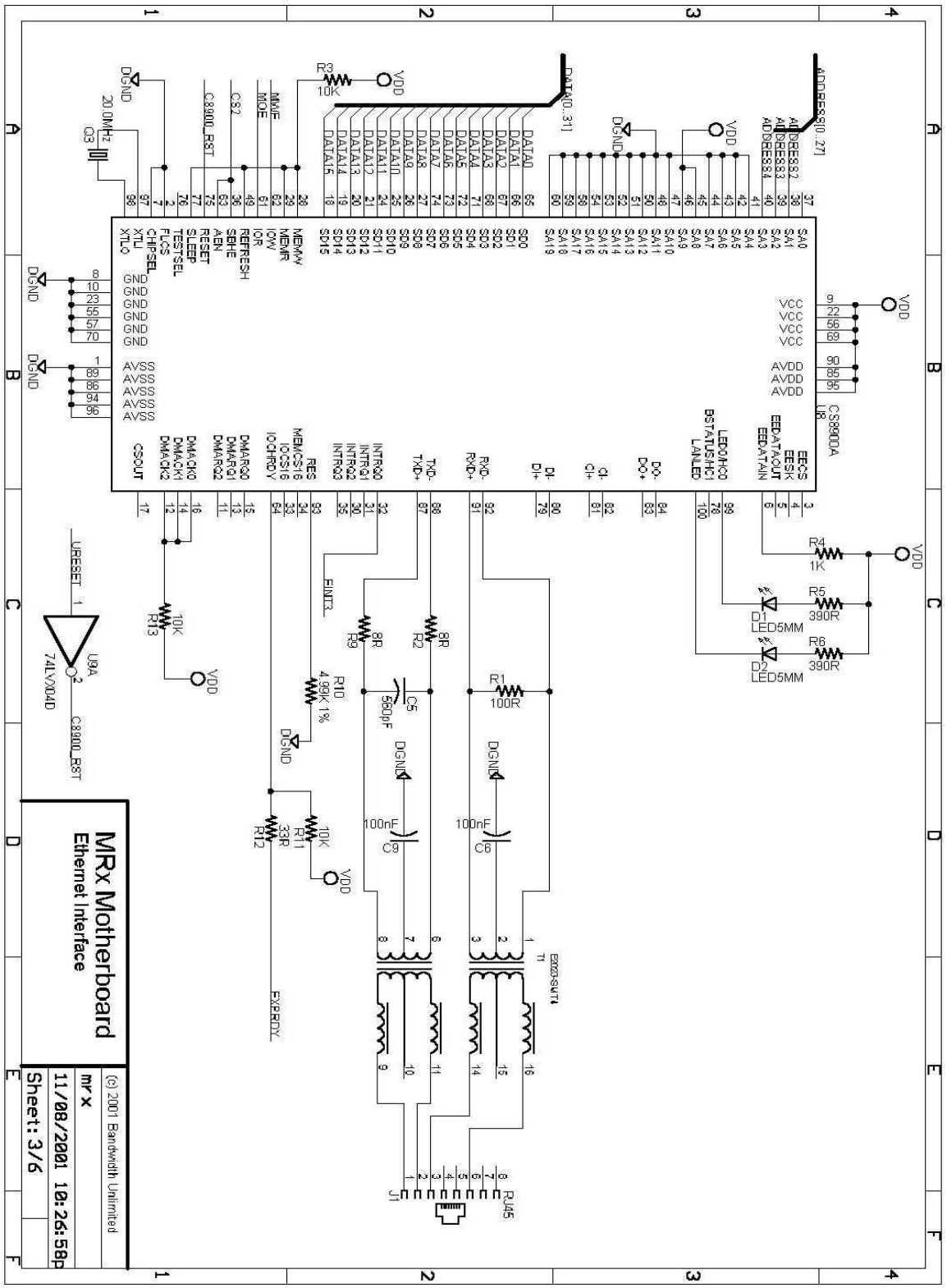
For hardware component support, we would like to thank Dave Easingwood of Insight Components Inc for supplying us with some of the crucial components required for the board and very helpful support. Finally, ON Semicoductor Inc. supplied us with several peripheral components for the board.

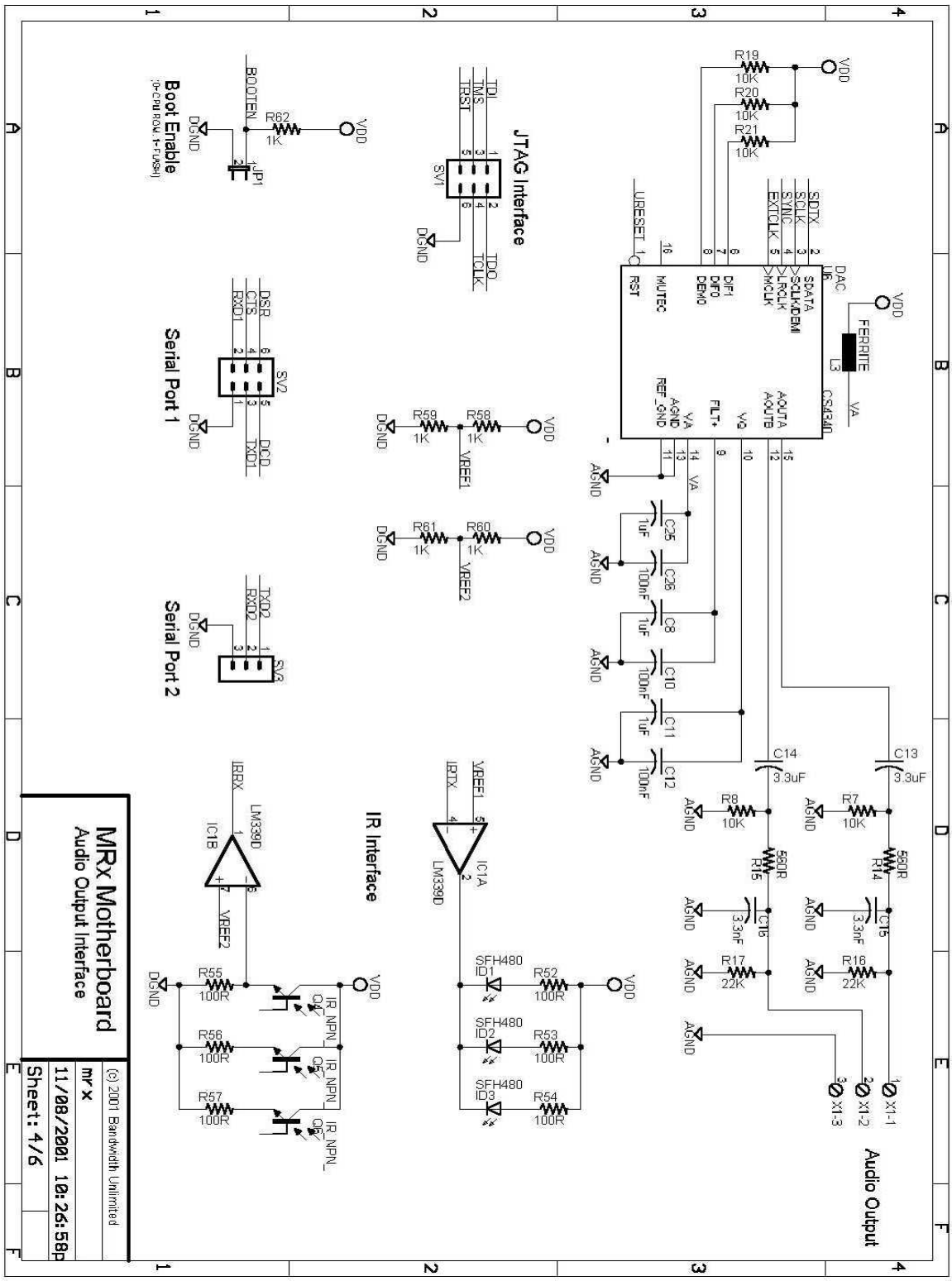
Our heart-felt thanks to everyone who contributed to making our project possible.

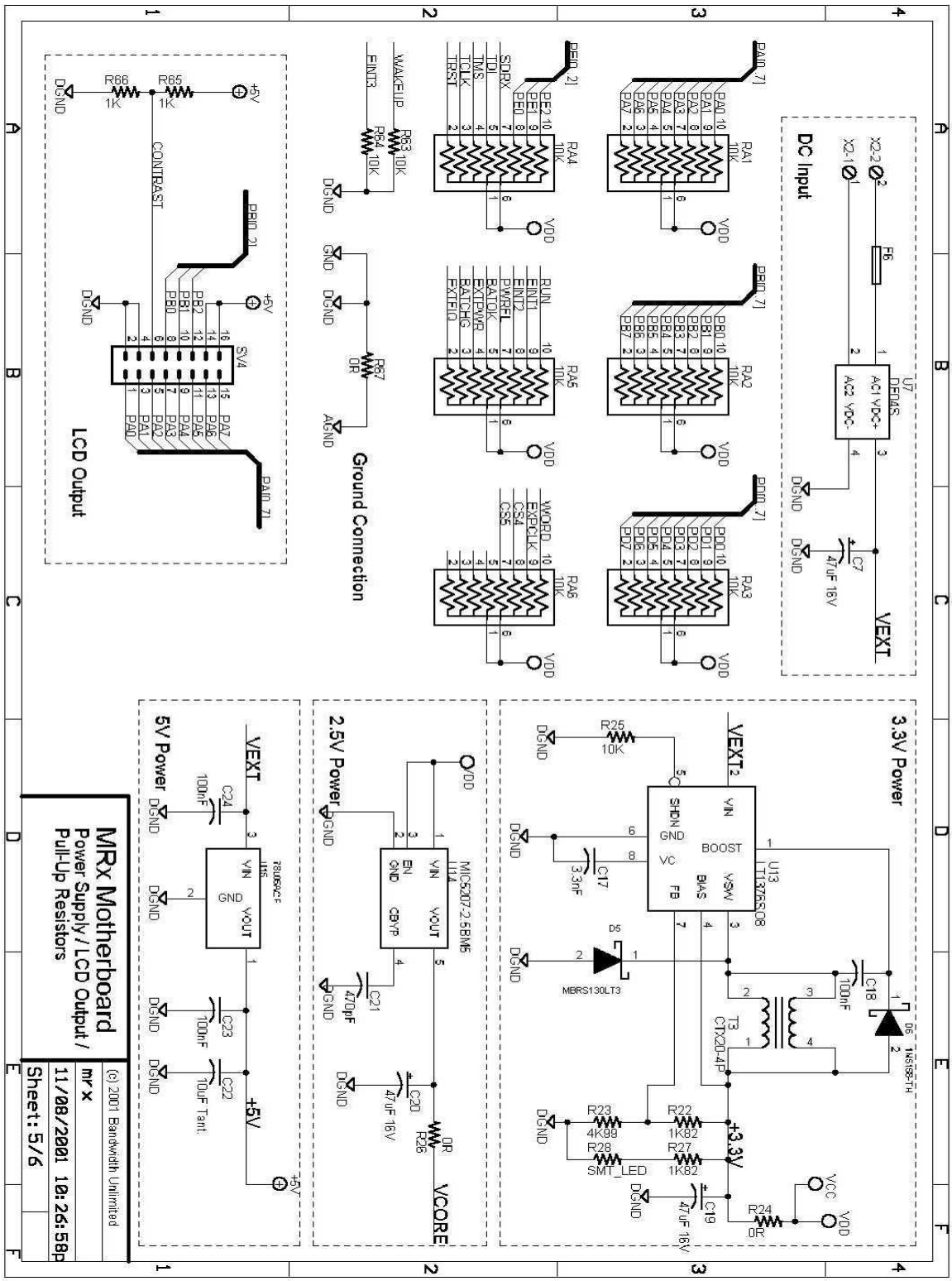
Appendix A – MRx Motherboard Schematic

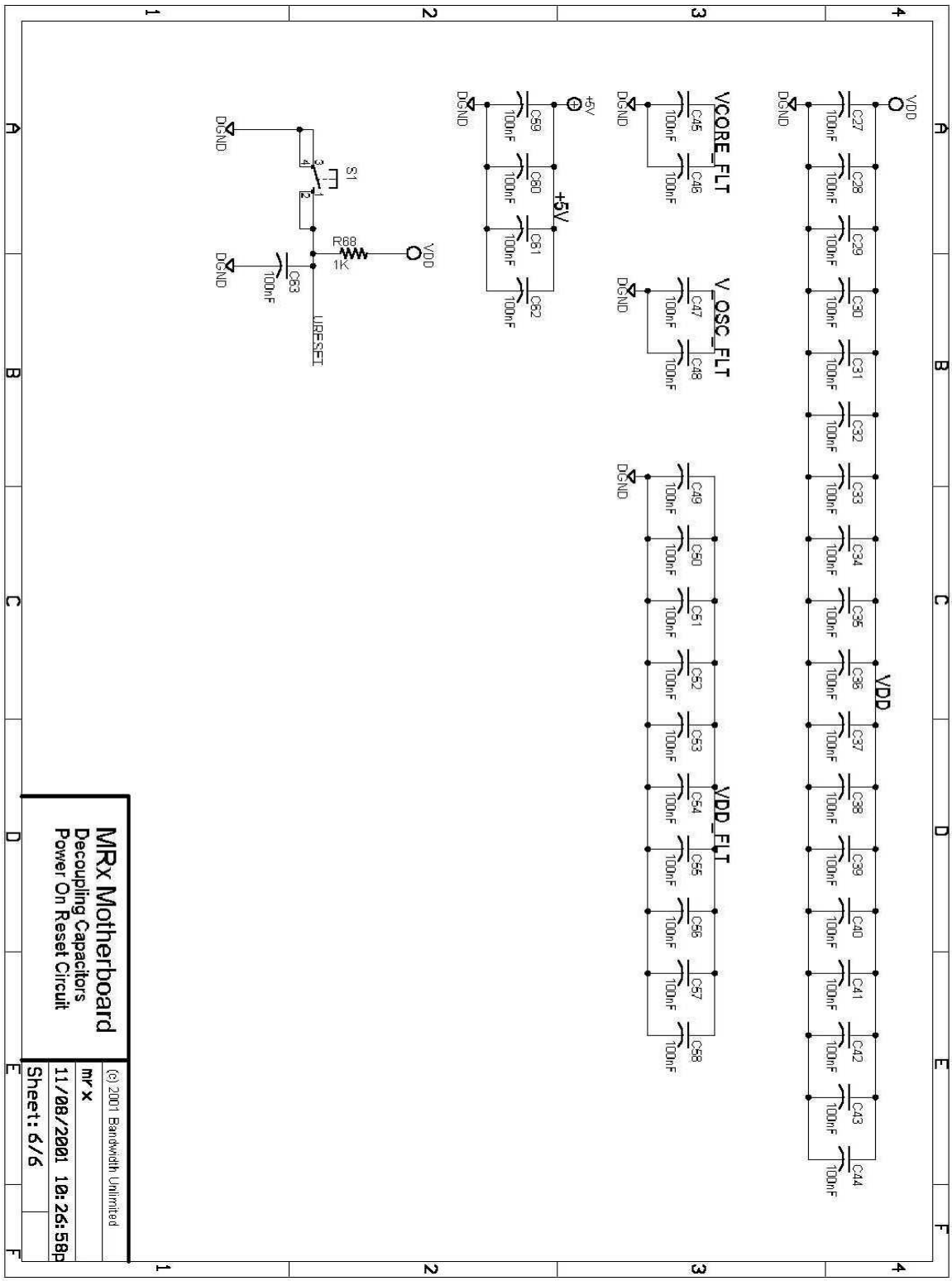



MRx Motherboard
FLASH and DRAM
(c) 2001 Bandwidth Unlimited
mrx
11/08/2001 10:26:58p
Sheet: 2/6

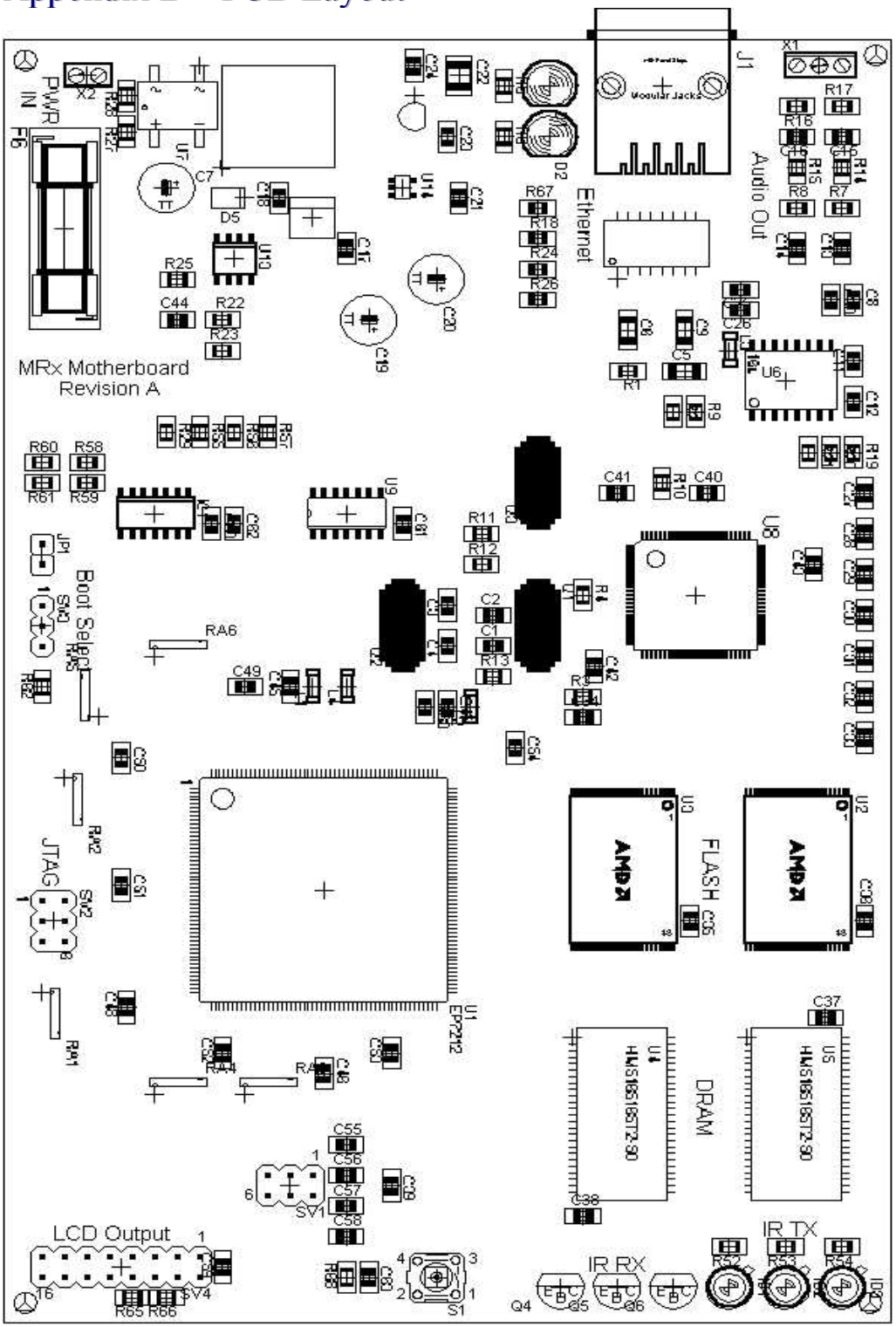

MRx Motherboard
Ethernet Interface
(c) 2001 Bandwidth Unlimited
mrx
11/08/2001 10:26:58p
Sheet: 3/6
CS8900A
U8
RJ45
J1
T1
Q3
20.0MHz
U9A
74LVX04D
D1
LED5MM
D2
LED5MM

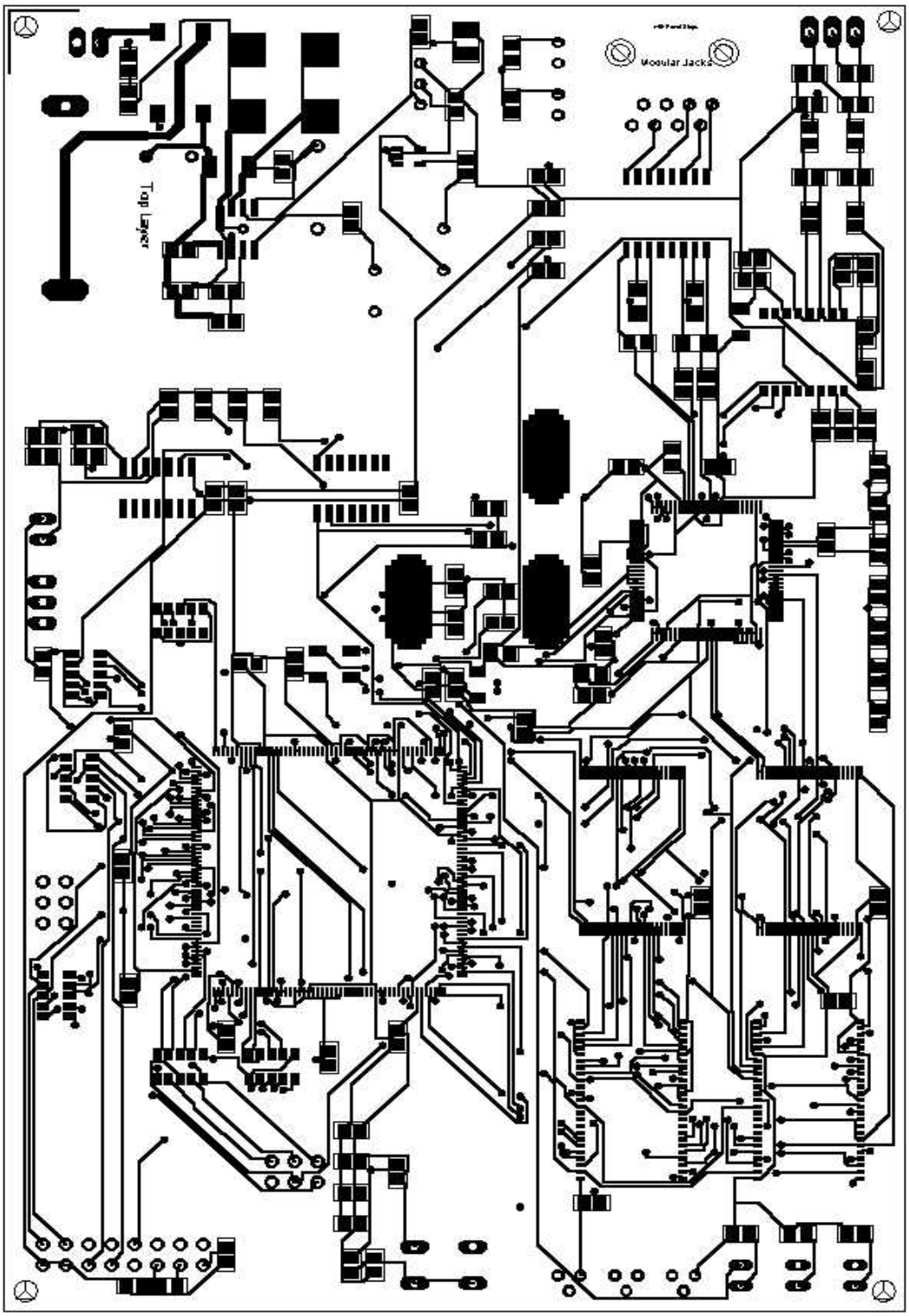

MRx Motherboard
Audio Output Interface
(c) 2001 Bandwidth Unlimited
mrx
11/08/2001 10:26:58p
Sheet: 4/6
Audio Output
JTAG Interface
Boot Enable
Serial Port 1
Serial Port 2
IR Interface
DAC U6
CS4340
FERRITE L3
LM339D
SFH480

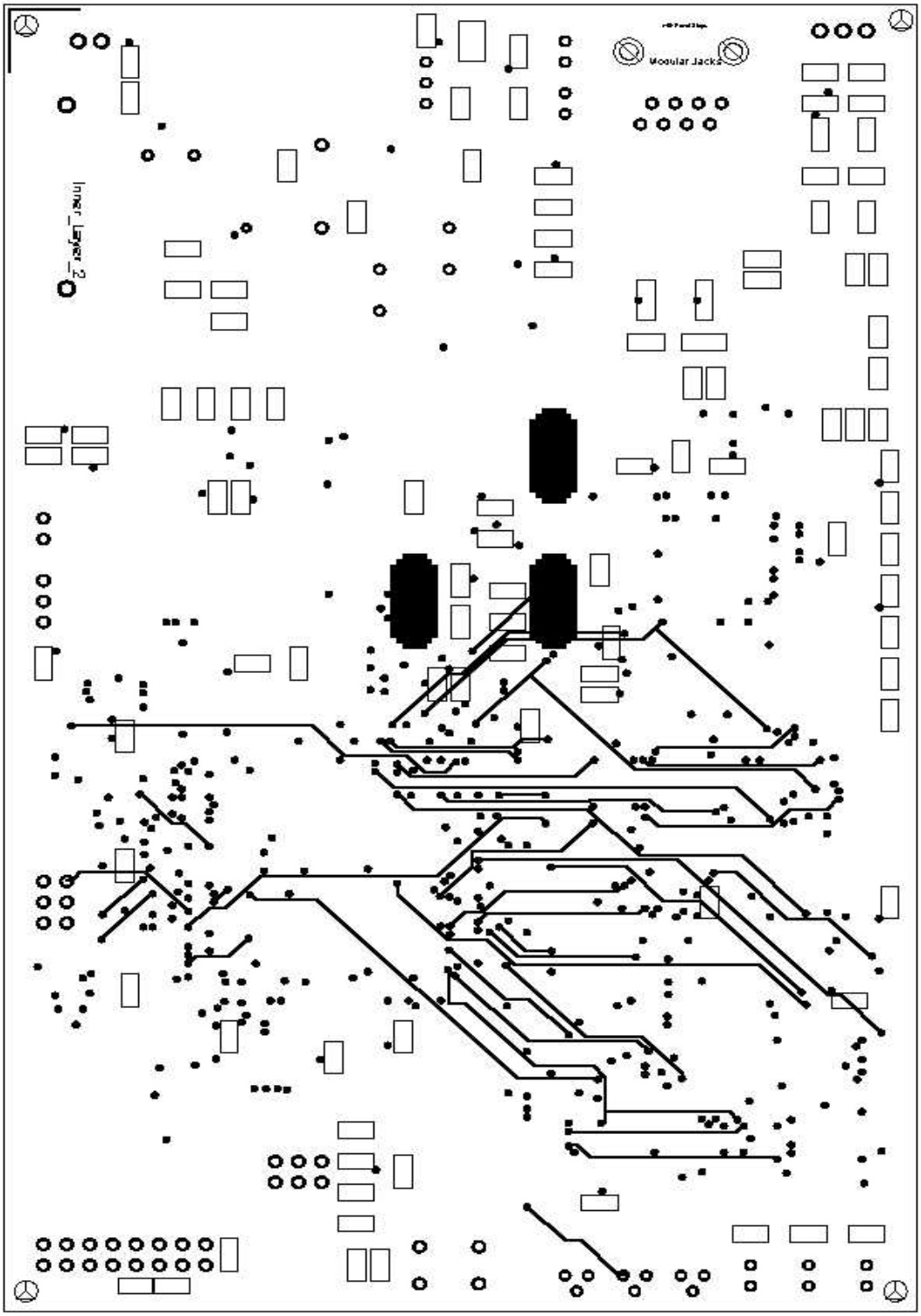

DC Input
VEXT
LCD Output
Ground Connection
3.3V Power
2.5V Power
5V Power
VCORE
+3.3V
+5V
MRx Motherboard
Power Supply / LCD Output /
Pull-Up Resistors
(c) 2001 Bandwidth Unlimited
mrx
11/08/2001 10:26:58p
Sheet: 5/6

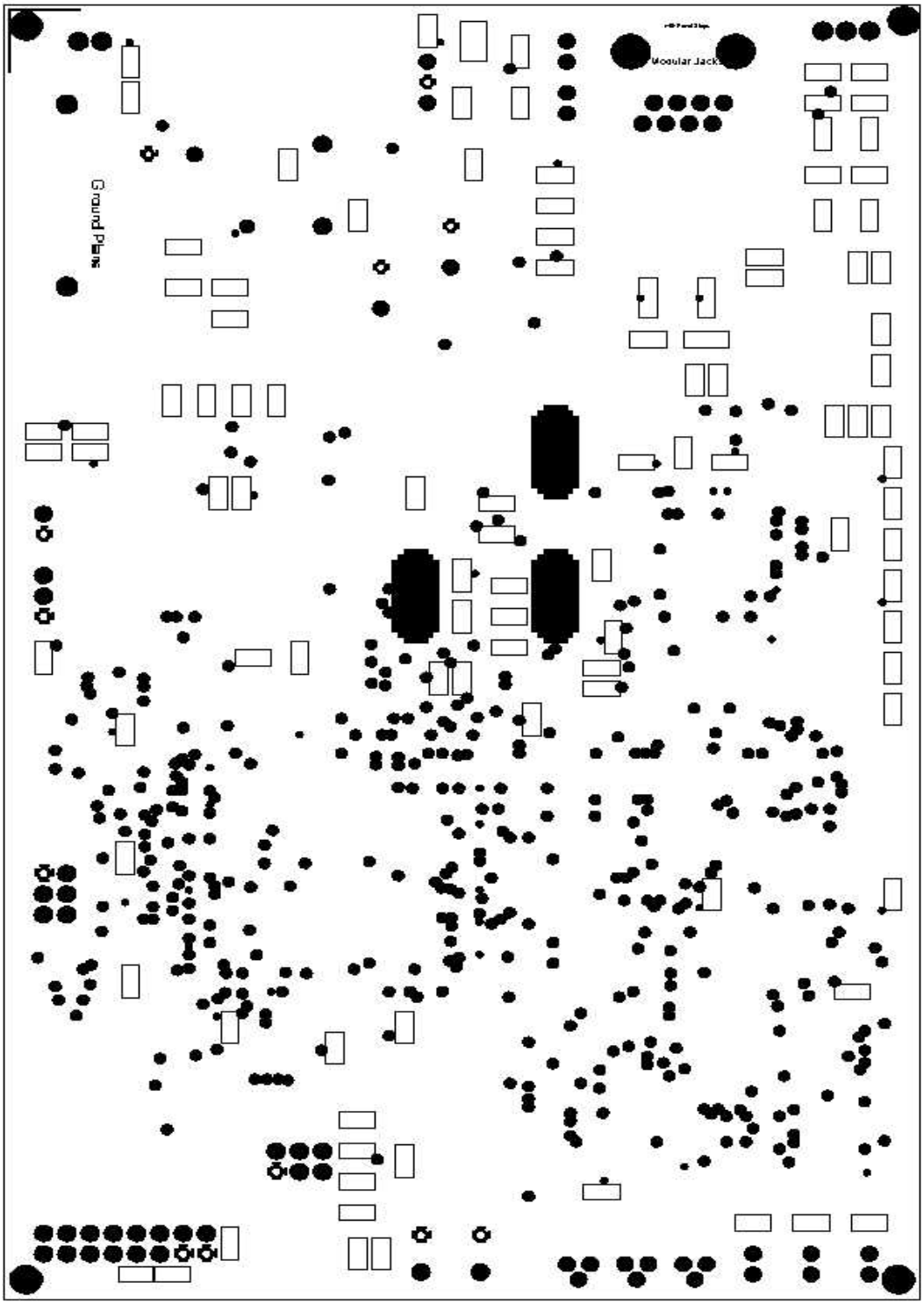

VDD
DGND
C27 100nF
C28 100nF
C29 100nF
C30 100nF
C31 100nF
C32 100nF
C33 100nF
C34 100nF
C35 100nF
C36 100nF
C37 100nF
C38 100nF
C39 100nF
C40 100nF
C41 100nF
C42 100nF
C43 100nF
C44 100nF
VDD
VCORE_FLT
C45 100nF
C46 100nF
V_OSC_FLT
C47 100nF
C48 100nF
C49 100nF
C50 100nF
C51 100nF
C52 100nF
C53 100nF
C54 100nF
C55 100nF
C56 100nF
C57 100nF
C58 100nF
VDD_FLT
+5V
C59 100nF
C60 100nF
C61 100nF
C62 100nF
S1
R68 1K
C63 100nF
URESET
MRx Motherboard
Decoupling Capacitors
Power On Reset Circuit
(c) 2001 Bandwidth Unlimited
mrx
11/08/2001 10:26:58p
Sheet: 6/6

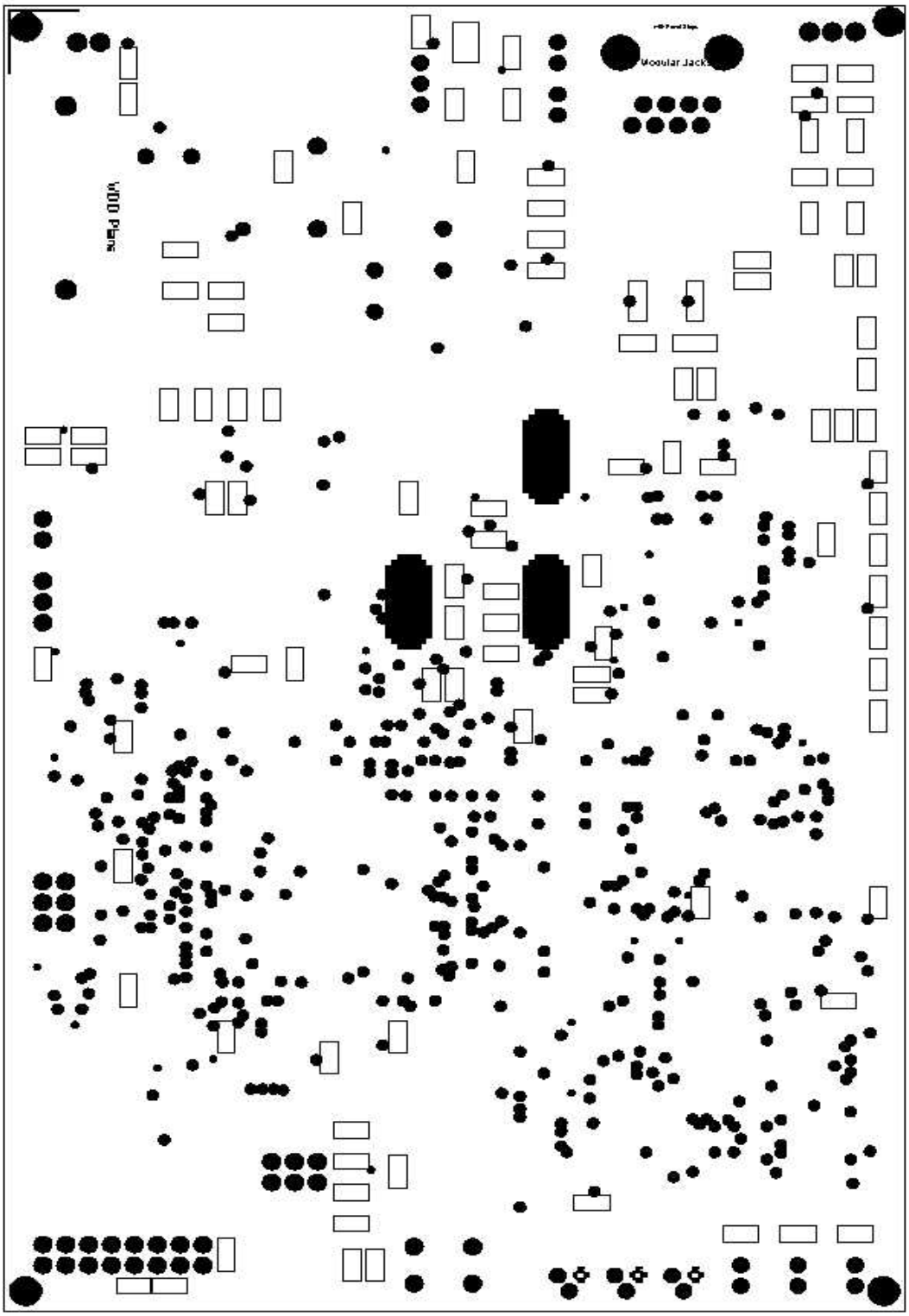
Appendix B – PCB Layout

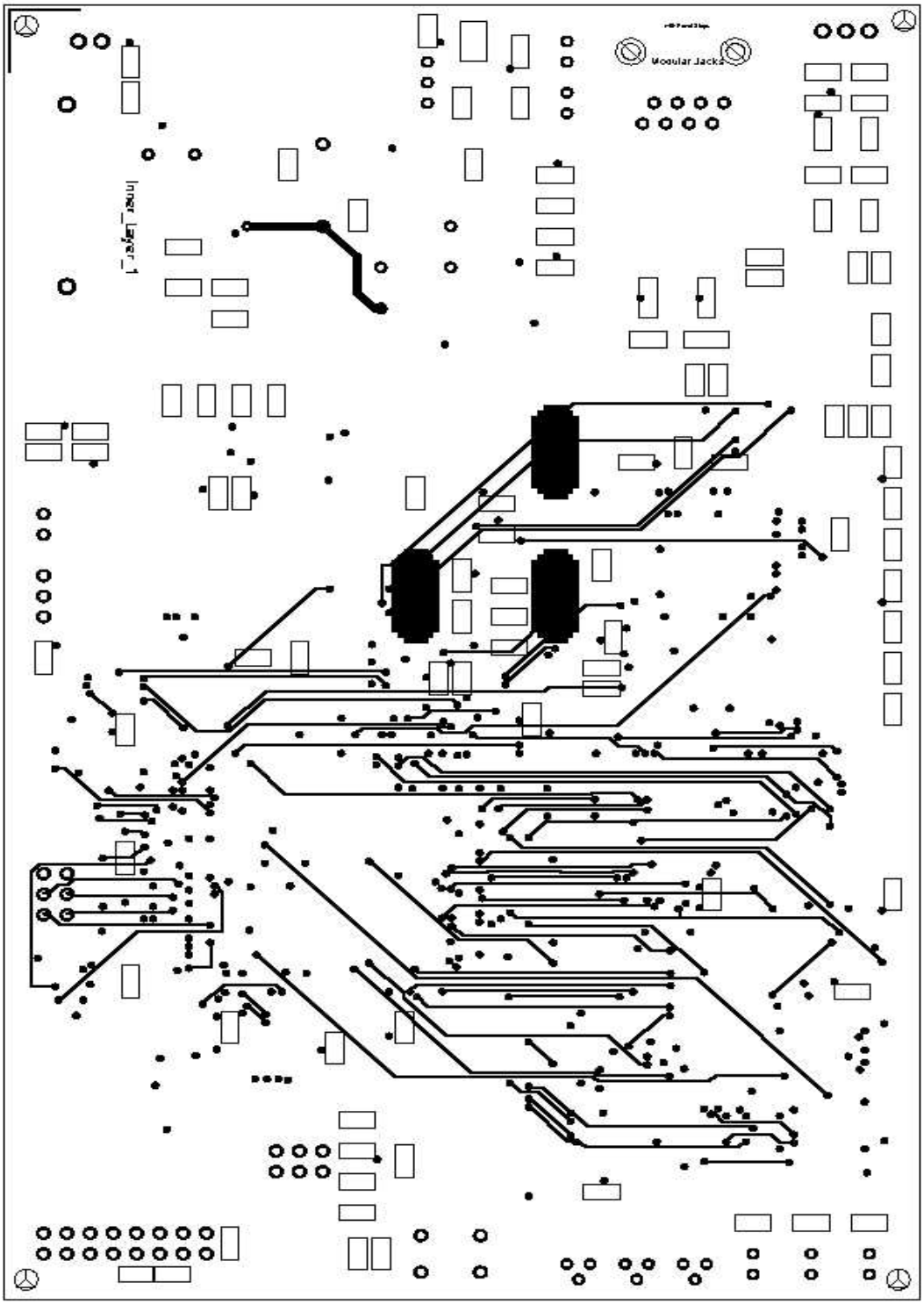


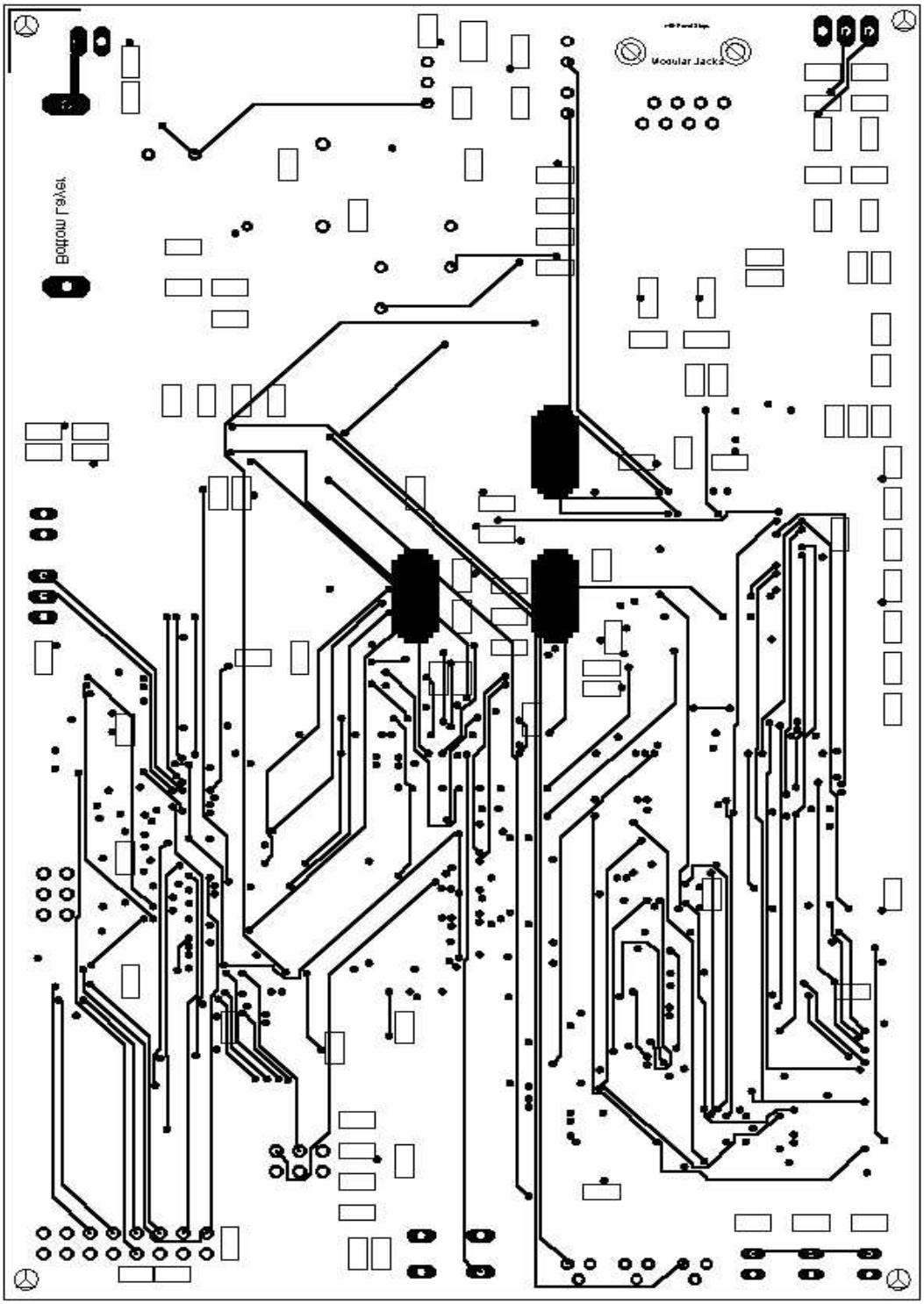

Modular Jacks
Top Layer


Modular Jacks
Inner_Layer_2


Modular Jacks
Ground Plane


Modular Jack
VDD Plane


Modular Jacks
Inner_Layer_1


Modular Jacks
Bottom Layer