

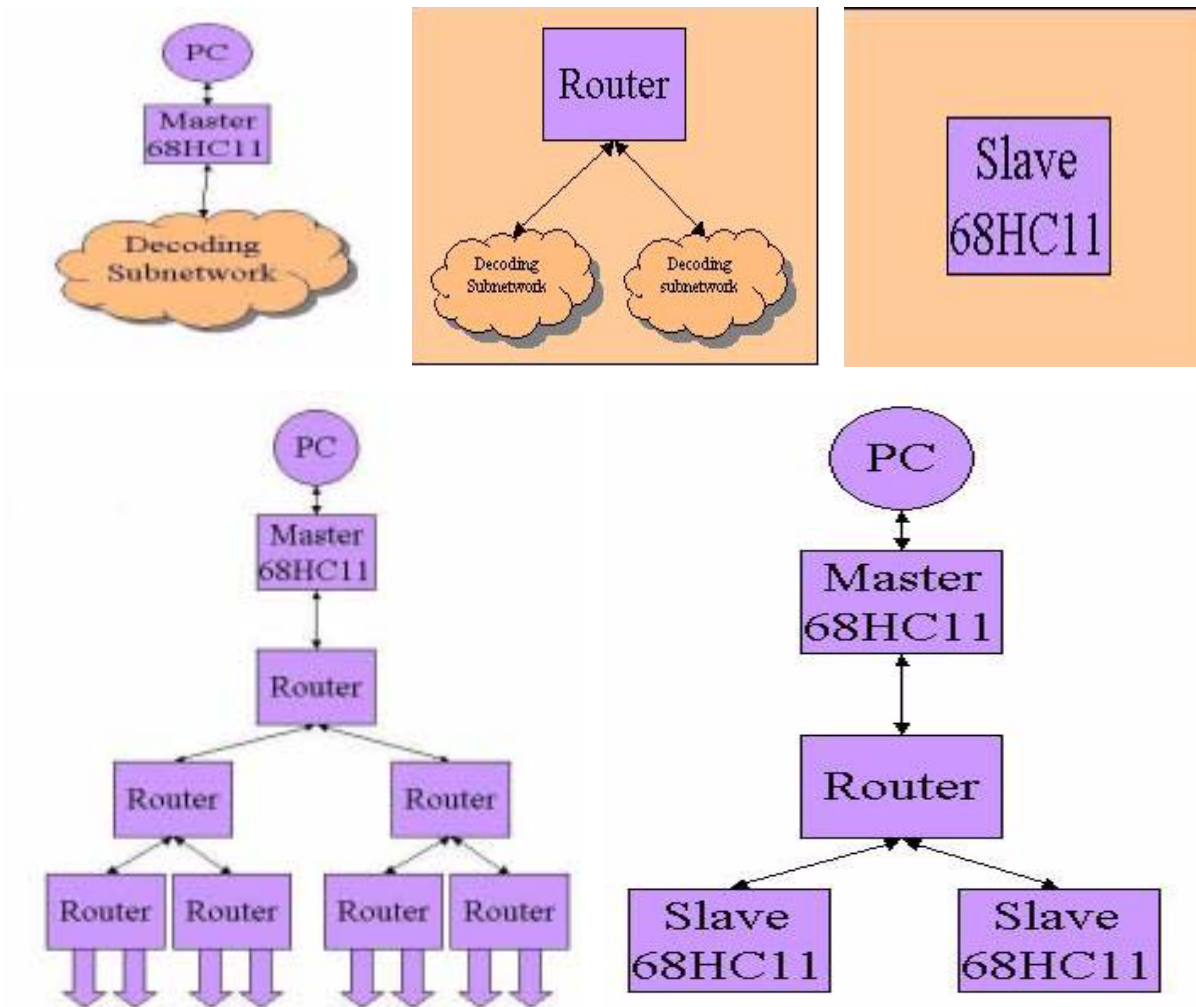
18-545 Advanced Digital Design Project

Final Report

0xf1d0

Project Description

Our primary design goal was to create a scalable, modular MP3 decoding network.



We will specify and design the pieces necessary for this network and put together a proof of concept to demonstrate the functionality of this design.

The network is designed so that each node (except for the head node) can either be a “leaf” node that does the actual MP3 decoding, or it can be a “router” node that has two child subnetworks beneath it in the tree as shown below. The head node is a special node that is responsible for getting the mp3 file from the pc interface and writing the wav file back to the pc. It is also responsible for dispatching the mp3 to the correct leaf node for decoding.

Our actual implementation of the network amounted to one head processor and two leaf nodes connected by a single router implemented on the FPGA. Also intended to be implemented on the FPGA were some of the more intensive operations such as subband synthesis and IMDCT.

Group story

After forming our group, the first thing we did was decide who was to work in primarily what area. Our division of labor was approximately as follows:

- Rod: Low-level software (embedded-style coding)
- Eric: High-level hardware (hardware system architecture)
- Ming: Low- and mid-level hardware (the board; Verilog implementation)
- Jim: High-level software

We ended up sticking loosely to these original specifications. What actually happened was that each person’s primary focus (ignoring other side projects and delegations) was as follows:

- Rod: MP3 decoding
- Eric: Hardware for communications and computation assist
- Ming: The board; software enhancement; hardware implementation
- Jim: Software for communications

The next step was to decide on an architecture. We enumerated several non-competitive design goals that we might want to shoot for, and eventually decided on multiprocessing. We enumerated a handful of possible multiprocessing architectures and eventually narrowed it down to two, which we now call fine-grained and coarse-grained. The coarse-grained approach refers to decoding multiple MP3s in parallel, and the fine-grained approach would be one in which a single MP3 could be decoded by multiple processors. Both were specified as modular, scalable message-passing processor networks. We thought that the fine-grained approach would be more interesting, although the coarse-grained approach would be more realistic. We had worries that fine-grained approach would be fundamentally impossible, due to dependencies within the decoding process, but after a cursory examination we concluded (incorrectly) that this apparently was not the case. We ended up choosing to go with the more challenging fine-grained decoding architecture. Our main design goal was modularity: We wanted the system to be trivially expandable to add more processors, increasing decoding power.

Our work got off to a good start, when around mid-semester we came across a static buffer holding information from the previous frame. This meant that we could not split frames arbitrarily among processors, as we had aimed to do; we had the same fundamental problem with the architecture that we earlier feared but judged not to be an issue. Interestingly, due to the lack of mention of this issue in documentation and the obscurity of the code implementing it, it seems that we could not have found this limitation except by finding it the hard way, which is what we did. Realizing that our architecture was not workable as originally conceived, we came up with several options, most notably splitting up MP3s differently; however, these options would have rendered our architecture neither modular nor adaptable nor particularly interesting. Instead, we chose to revert to an architecture which we earlier rejected, namely the coarse-grained parallelism model. We chose this architecture because it best

exemplified our original design goal of modularity, used some of the earlier work we had done on communications, and did not make it necessary to change the physical layout of the board.

This change, despite efforts to reuse as much as possible, was a significant setback, particularly for software: Basically all the work Rod had done on software needed to be scrapped; Eric's and Jim's work also needed substantial changes. In many ways, this unforeseen tiny bit of code caused us to entirely lose half a semester's work. Indeed, it was the MP3 decoding software, the area in which the most work was lost, that was the main thing keeping us from completing the project in time. Given the amount of time we were set back, and the amount of time other groups needed to finish after getting as far as we did, we feel that we would most likely would have finished, were it not for our unfortunate choice of architecture.

As the semester came to a close, we began to realize that we could not expect to finish in time. The board worked, and the communications architecture was in place in both hardware and software. After considerable extra effort from all group members, the MP3 software was modified, hacked, and hardware-implemented to such a point that it was able to fit within the 64 kilobytes of RAM available. However, there were still unknown numbers of bugs in the software. We really have no idea what the scope of these bugs might be; we know that the decoder ran and appeared to return data, but that it was invalid, and we observed that after about 20 minutes decoding, it would return an error.

Discussion of group story

Why we did not meet our design goals

When it came time for demo3, our project did not yet meet our original design goals. Specifically, our board could not decode an MP3 from beginning to end on the board. There are several major reasons for this failure.

At the beginning of the semester, as we were spending all of our time scribbling on whiteboards and discussing the advantages and disadvantages of various architectures for the project, we failed to do enough research into the code, and thus missed a point that turned out to make our chosen architecture infeasible. In retrospect, however, it would have been difficult to have known that frames were not independent without spending the time to go through the entire code, which would have taken far more time than we had available. During our original whiteboard sessions, we were somewhat overconfident and overambitious about our design. One of our intentions was to choose an architecture that was interesting to us. We wanted an architecture that would be interesting enough to keep our interest, and therefore, our focus, on the project for the entire semester.

There were several things that we did not research as well as we should have and therefore did not understand some of the repercussions of our original choices. For example, we did not realize that our code would not fit in the allotted space until there were only a couple of weeks to go in the semester. We should have started off by doing some modifications to the mp3 decoding code from the very beginning to make it fit in the 64K of allotted space on SRAMs.

It should be noted that the problems we ran into were not problems in adapting MP3 to our architecture. The most significant work required was replacing standard library calls with calls to our communications library. This was even a trivial task, since our library functions shared the same prototypes as the C standard library functions.

Group dynamics

We faced two main problems as a group: How to get everyone to work, and how to ensure that all the work being done is productive. A perfect division of labor is useless if no one does anything; and having everyone spend sixty hours per week in the lab is useless if people are duplicating each other's work or, worse, performing work that never gets used.

To achieve the first objective, we employed a few different techniques: We made it a policy to have everyone Cc: everyone on weekly status reports; we had three meetings every week in which everyone was supposed to update the rest of the group on what progress they had made since the last meeting; we assigned specific due dates for intermediate tasks; and group members were regularly grilled when they failed to meet these objectives.

Unfortunately, despite all these tactics, there was still a significant amount of failure to meet deadlines. This was partially due to the group leader's failure to emphasize the importance of these deadlines and the reasons why it's bad to miss them. Additionally, we suspect that one of the reasons for failure to keep everyone working was something known as the bystander effect, which can be summarized as follows: The more people there are to do something, the less likely it is that it will get done, mainly because everyone will assume that someone else will do the work. Another problem was simple procrastination, with everyone putting off work until the last minute, and then being unable to complete it. We attempted to avoid the bystander effect and procrastination by dividing up tasks specifically among group members and keeping a tight schedule: The assignment of tasks to specific individuals kept the bystander effect from happening, and the short turnaround times on smaller tasks kept procrastination in check.

The second issue, that of ensuring that all work being done is productive, is perhaps a trickier one than the first. Code written depends to a large degree on other code -- coding becomes largely a serial process. Dividing up this work among four people is therefore difficult. The best strategy to use against this serialization is therefore good planning. Specifically, we decided early on what we were going to do, and set up interfaces such that each of us could make assumptions about what would later be available. The first thing we did was clearly defining exactly what would need to be physically connected together, so that Ming could get to work on wiring up the board. Next, Eric and Jim defined the communications pro-

protocol, so that Eric was free to work on hardware to support the protocol while Jim worked on software to interface with it. The API included functions with the same prototypes as the C file input/output functions, so that Rod could work on the MP3 software, getting it to work on UNIX, and then with only minor changes adapt it for use on the board. In actuality, our separation of work was fairly successful. Only near the end of the project, when smaller tasks needed to get done fairly quickly, did problems arise with what to hand off to whom and how to avoid duplication of effort.

One of the most successful tactics for eliminating both of the problems mentioned above was the idea of work sessions: As many group members as possible would come to work at the same time in the same place. This had the obvious advantage that people would be reluctant to procrastinate when they could be seen doing so by their team members. Work sessions also stopped duplication of effort by encouraging team members to ask each other questions, both questions about how to do things and questions about the project itself. Frequently, people will go and spend several hours working on something instead of contacting people to see if something is already done.

Work sessions also help with something which is key to the timely completion of any project: delegation. In order to ensure that everyone has something to do, and that no one is left with too much work, it is absolutely essential that group members be willing to hand tasks off to others. We actively encouraged delegation, and we found that group members were actually willing to entrust their work to others. This sometimes resulted in people getting involved in areas outside their stated expertise (for example, Ming got assigned the task of removing floating point code from the MP3 software despite his stated interest in doing hardware) but in general the result was overwhelmingly positive.

Self Dynamics, or how to say no to Slashdot

A strong personal work ethic is necessary in any professional or academic environment, but the need is multiplied when working in groups. We found out that it is very necessary (and difficult) to accurately estimate the time tasks will take. It is also not intuitively obvious to include outside distractions into the estimation. On top of that, it is easy to procrastinate which makes it even more difficult to complete tasks in the allotted time. Unfortunately this will propagate in a group environment because of the interdependencies. It is also impossible to keep your motivation and intensity up indefinitely so it is necessary to designate some time to unwind. In our group it didn't always work like this, but in general it works well if you pick blocks of time to be devoted work periods and you try to ensure there are no distractions during those times. If you can manage to separate work and play into distinct time blocks, you will be more productive and still have time to relax. As mentioned before, we attempted to accomplish this by scheduling work sessions in the lab. That way, people feel more compelled to work because they can be actively monitored by everyone else in the group. The only problem we faced with this, if it can be considered a problem, is that over the semester we began to talk socially much more as we figured out we had a lot in common. Although this distracted us, it made the semester much more enjoyable.

One thing that really helped us out was providing positive feedback when people were making progress. Without feedback, people lose sight of their goals, and their motivation suffers as a result.

Further insights

What we had going for us

Despite our shortcomings, as a group we had some strong points in our favor. For one thing, we communicated rather well. Despite not sending daily status reports to each other, we

were open and honest with each other about what we had done, what we expected of others, and what needed to be done.

We also clearly defined our goals and commitments. In any significant project, it is necessary to clearly define what needs to be accomplished, or else: people will fall short on their commitments, redefining what was expected when the due date comes; and there will be problems with duplication of effort and work which isn't useful anywhere.

Additionally, we feel that our architecture was an interesting one. It is a simple architecture, easy to understand and not too difficult to implement. We also feel that it is applicable to other similar problems: Anywhere there is a need for a streaming transformation of input split among several processors, the architecture should be applicable to it with minimal porting effort as simply by replacing calls to the standard file functions with calls to the communications library functions. In short, the architecture should be easily reusable in other similar designs.

Lifelong learning

Through the process of working on this project, we learned a lot about something that will be very useful for us in the future: the importance of adaptivity. Sure, we've known that being adaptive is a good thing, but as with many things, nothing REALLY hits home unless you experience it first hand at full force. The fact that we were forced to change our entire architecture half-way through the semester definitely gave us a crash course in how important it is to be able to adapt.

It can be argued that we learned as much, if not more, from the failures that we suffered in working on our project than from the successes that we enjoyed. Because we were not able to fully adapt ourselves quickly enough to handle all the necessary modifications and complications in the process of taking the MP3 decoding algorithm and running it on the

68HC11s, we were unable to finish our project in time. In a rather twisted way, this allowed us to experience what it is like to fail to adapt quickly enough in the real world. If one does not adapt oneself in time in the real world, it is all over. You become outdated. You become unproductive. You become unemployed! Fortunately, in this case, our very careers were not necessarily on the line. This still does give us a lot to think about. The fact of the matter is that we all have some work to do in the area of becoming more flexible and adaptive to the circumstances that present themselves before us. Sometimes you just need to step back and try to look beyond the few feet before you and see what's farther ahead, because seeing a dead end ahead of time saves a lot time and trouble. If we had done enough research ahead of time, we could possibly have figured out that it was not possible to split MP3 frames up like we were planning to do, and we would not have tried taking a route that would ultimately lead us to a dead end in the middle of the semester. In fact, if we had kept on looking ahead, we could have possibly caught our mistakes earlier even after we started down the wrong path. Although it would have still led to the necessary change in architecture half-way through, we would have at least found out earlier about it, giving us more time to work on the new architecture later on in the semester.

Thus, as we learned from our architecture disaster, adaptivity is helped a lot by being well-informed. If you can keep yourself well-informed, you can, at the very least, make a well-informed guess at what is the best way to undertake a task and have some sort of early warning when something goes wrong so that you have a little extra time to fix your mistakes. If you lack some of this information and have to base your decision on unsupported assumptions, it is a whole lot easier to get burned in the process. This was perfectly exemplified by our hasty assumption that we could split mp3 encoding up and do it in parallel on two different processors. Thus, if we plan to have any shot at lasting longer in our employment than our current skills will support us, we will have to keep learning and keeping ourselves well-informed about the newest information and techniques relevant to our fields. This will allow

us to make an effective attempt at adapting ourselves to the inevitable changes that will occur within our fields and keep ourselves useful and employed. As seen in our project experience, failure to adapt oneself in time can cause result in the failure of our projects to achieve their objects. If this were to happen frequently in our careers, the consequences would be grim indeed.

Adaptivity is not only important for us in the workplace, but it is also important for us as people. As people, we need to be resilient and adaptive in order to live happy lives. If we are too rigid in our thinking and in our dealings with other people, we will never be able to get along with anyone or be able to live happily, because we will be forever at conflict with our environment and the other people in it. We learned in our group interactions how important it is to get along well with the people you live and work with. If we had not been able to be understanding and flexible in our discussions and working together, we would never have been able to get as far as we have despite all the difficulties that we have suffered. The ability get along and adapt to the behavior of other people around us is enormously important to the success of any social or team work environment. Without the ability to adapt and accept other people, we would be limited to our own abilities, which would never be able to weather a project of a large magnitude under tight time constraints.

In addition to adaptivity, we also learned that in order to effectively adapt, sometimes you just have to drop what you having been working on and move on. If something starts becoming a hopeless cause (like our first architecture), one needs to take a step back and make the difficult choice of calling it a sunk cost. It is extremely easy to grow attached to something that one has been working on for a long period of time, because there is an overwhelming tendency to think that all it will take is some more time, and all the problems you've been having can eventually be fixed. After all, are we not competent enough to MAKE it work if we put out minds to it? One more week, and everything will be fixed, we tell ourselves. We KNOW

that we're right. We KNOW that one week is sufficient. Our pride demands it. This is what happened for a good week or two after we found out that decoding MP3's in parallel may not be possible. We kept thinking that there has GOT to be a way around this, that there must be a way for us to overcome this obstacle without doing the unattractive task of scrapping our efforts and starting anew. Unfortunately, in some instances, something that you've sweat and bled for can become useless and ultimately counterproductive if you put any more time into it. This is what happened during those two weeks that we spent stubbornly trying to fix something that defied our every effort. In the end, though, we realized after unnecessarily wasting much time that there was no other way around our problem and a rehaul of our architecture was. It takes a lot of strength to realize that sometimes, you just need to let go of your stubborn pride and admit that your were wrong from the very start. Although this may be initially difficult and discouraging, it is far more preferable to spending that additional week...perhaps SEVERAL weeks...fighting a problem that will never go away. We did not do this in time, and as a result, lost almost two weeks of our precious time fighting a doomed fight simply because we had grown overly attached to our incorrect initial plans. Because we let ourselves become blinded, we were not able to adapt ourselves quickly enough to this problem. Pride and stubbornness can be good when it pushes us to achieve new heights, but in excess, it can ultimately lead to failure.

Individual Narratives

Rod Green

In engineering, the goal is always to solve a problem. The most important thing I learned from this project is that fully understanding the domain of the problem is just as important, and perhaps equally as difficult, as actually solving it. We failed to complete the project at least in part due to my initial misunderstanding of the MP3 specification, and that is why I believe this was the most important lesson I learned.

How it actually happened:

While planning out our architecture we considered many possibilities, all involving multiple processors. We considered pipelining the decoding since the software seemed suitable for that, but we feared we would end up with an uneven distribution with one or two stages taking a large percentage of the decoding time. Instead, we decided parallelism was the way to go. After a bit of research, including Christopher's MP3 lecture, I was convinced parallel frame decoding was possible. Everything I read said that frames were not independent, but they all said it was due to the bit reservoir. I figured that was easy enough to fix, and I drew overlapping frames on the whiteboard and worked out pseudocode to rearrange the data into separate chunks. It all seemed simple. We committed ourselves to building it and got to work.

On March 18th, I discovered a very serious problem. I ran the first multithreaded version of the code using a single decoding thread and it worked great. With two decoding threads the sound was a bit garbled. To my dismay, the output continued to deteriorate as I added more threads. Since this was two months into the semester, we had already devoted an immense amount of time to design and much to implementation. Ugh oh!

How it should have happened:

In retrospect, building a less accurate simulation much sooner in the semester would have saved us weeks of work. I spent a lot of time carefully arranging the code into functional blocks so we would have an accurate software model, but it ended up being completely worthless. Since I ran into the problem two weeks before spring break and it took us a week to decide to change architectures, I didn't get geared back up until weeks later. All of this should have been figured out in the early weeks of the semester so we wouldn't have spent so much time walking in the wrong direction. In the future I will try to approach problems in a much

more hierarchical manner to try to shake out serious misconceptions while there is still time to correct for them.

If this happens again:

Whether I like it or not, I can imagine myself running into similar situations in the working world. Time constraints are very real, and mistakes happen. In 545, I lost a lot of my motivation when all of my work to date had to be thrown out, especially since we switched to an architecture that I found to be much less interesting. I honestly do not know how I can correct for this in the future, but past experience will definitely help.

Eric Haas

During the course of this project, I was reminded just how difficult it is to predict the types of problems that will hurt your progress. In the past I have worked on many projects where it is very important to try and predict what problems you will run into, and to correctly estimate the time required to solve each problem. If your prediction was off by a substantial percentage, it could cause major problems between my company and the client in question.

On this project we ran into several problems that could have potentially been avoided with some extra planning in the beginning. It is good to try and figure out what problems you could run into when starting any substantial project, however, you must be careful not to spend too much time planning in the beginning. While some planning at the beginning of a project can foresee some pitfalls that will save you time later, too much planning will only waste time that could have more effectively have been spent working on the actual project.

In our case, I believe that we spent about as much time in the planning stages as we could have without becoming counter-productive. There were several problems that we ran

into that could have been solved by more, or more productive planning, but had we spent the amount of time planning that I believe would have been required to learn these lessons, we would not have had enough time to finish the project even with our new-found knowledge.

Even those groups that most carefully plan their project can run into unexpected problems that will set back your schedule. How you deal with these problems and attempt to remedy them is an important part in how your group works together, which will significantly affect the final quality of your project or product as well as your ability to complete at all.

Ming Luo

I believe that the most important thing that I learned from this class was the importance of properly adapting myself to the different problems that present themselves in hardware and software design. Through the duration of this project, I have worked on parts of it that were on completely different sides of the hardware/software spectrum. When I first started off, I was working exclusively on wiring up the board. Since I had no experience working with hardware bugs like this in the past, this made for a lot of ineffective debugging, because trying to fix random things on the board based on hunches is not very useful if you do not know exactly what the root of the problem is. After a while, though, I saw the error of my ways and started using the logic probe and checking to see exactly where I had my problems. This, of course, was much more effective than making random stabs in the dark.

After I finished wiring up the board and testing to see if it worked, I switched to converting functions from the MP3 to hardware. Due to my lack of previous experience in tasks like this, instead of trying to fully understand the problem first, I decided that I would dive right into the problem and start by using the original `im_dct` function as my framework to work with. Unfortunately, I did not stop to think things over as I should have. If I had just looked at what the code was doing more carefully, I would have seen that over half the original `im_dct` code was dedicated to initializing a table using constants. In other words, it was a

simple matter to just pre-generate the entire table and simply use this table later on. Instead, I spent a good week or two hacking at something that just seemed to get more and more complicated by the minute.

By now, it became apparent to us that before we could do anything, we needed to take floating point out of the code as soon as possible, so I volunteered to drop my work on `im_dct` and do so. This time, I had learned from my previous experiences in adaptivity. Instead of diving into the problem, I took a lot of time thinking about the problem and asking people for advice about so of the confusions I had. Also, when I finally started to attack the problem, I used a completely step-by-step method of implementation, starting from the very end of the code and slowly converting the code to fixed point, leaving the code before it as floating point. Although it still took me quite a bit of work to get floating point fully ripped out of the code from here, my previous experiences had taught me enough so that I was at least applying my efforts in an effective manner to expedite my progress as much as I could. Unfortunately, by the time I got the floating point ripped out, it was already too late for us. Even though the deletion of the floating point code cut the code down enough to fit it on the board, we had simply decided to do attack the floating point problem too late in the game.

I believe that learning the importance of using different approaches to adapt to different problems is the single most important lesson that I took from this class. I have never taken a class that has thrown me into the complete different ends of the hardware/software pools, so I have never seen for myself how different design and debugging for these two ends of the spectrum are. This class has allowed me to gain a lot of hands-on experience in implementing a system while working on both sides of this spectrum. In retrospect, I can see how the many issues that I faced could have been fixed or alleviated so much more easily if I had taken a more effective approach to designing and or debugging it. I feel rather sheepish for not doing things the right way the first time, but I believe that I have learned from my experiences in 545

and that I will be much more likely to take time to rethink my problems in the future and do a better job of finding a better method of adapting to it.

Jim Shuma

The most important thing I learned was that anytime I assume any sort of leadership position, I should take a more active role in executing it. I found that the key areas in which I could have improved were in project planning (specifically, encouraging more structured analysis of problems and intermediate tasks) and the actual execution of the project (ensuring that work is high-quality and done on time).

It has become painfully clear to me just how important it is to conduct in-depth planning. This made itself clear not only in our failure to notice that our architecture was fundamentally unworkable, but also in the somewhat unrealistic timing estimates we made. Although in this particular case it would most likely not have been possible to find the part of the code which caused the limitation, this is not necessarily true of other projects. Problems of this type can often be avoided through careful planning, saving untold hours of work.

Planning also is helpful in analyzing how to break up tasks into intermediate goals. There were a few problems within our group with making sure everyone had neither too little nor too much work assigned. However, one of our major problems was that we didn't do enough research into the finer details of the project. Rather, we chose some larger tasks which seemed to fit together fairly obviously, but without actually looking into them more deeply we didn't know how realistic our timing estimates would be or how applicable the subgoals were. For instance, we were significantly set back by delaying our decision to remove floating point code; we saw the value of this removal to be primarily in speeding up the code, and speed was one of our lower-priority goals, so we avoided it. More detailed research would have made it clear much earlier on that removal of floating point code was necessary, and we would have had more than two weeks to accomplish that task.

Regarding execution, I found that I was not nearly assertive enough in enforcing group deadlines. Some of the group members have given me feedback to this effect. The importance of making and meeting deadlines cannot be overemphasized. Only as May 3 came critically close did I make the following clear to the group: It is not an acceptable excuse that work could not be completed due to its taking unexpectedly long, because technical work always takes longer than expected, and this should have been accounted for. I found that taking this rather inflexible approach to deadlines had a positive effect on completion, and the project would most likely have gone better if I started using this approach earlier.

Interestingly, I find that these leadership lessons are also applicable to my own work: When I work on a large project (or a large piece of an even larger project) I should do my homework and analyze exactly what I need to do. This will have a profound effect on my performance and ability to complete the project, just as it does in a group setting. Similarly, I should refuse to let myself miss deadlines. Admittedly, this is a tough one to enforce, but once I realize that missed deadlines cannot be blamed on lack of foresight, I should be on schedule much more often.