overall code size. I have no experience with the compiler, so I have no idea the quality of the code that it will generate. You might find it beneficial to rewrite portions of the code is assembly by hand to further optimize it. Also, you will need to write an interrupt handling routine (probably mostly in assembly) to handle the interrupts from the DACs. Our machine will not support the context switching of the ADSP-2181, so you will have to save any important register data to memory in the beginning of the interrupt handling routine and restore this data at the end.

The FlashMedia interface will look like a peripheral device and you will need to poll it (checking a memory-mapped status word) when reading and writing it.

Your work items are:

- Remove the Layer-1 and Layer-2 support from the code. (Approximate check-point date: 2/14)

- Do a first-pass removal of all the floating-point operations from the code. (Approximate check-point date: 2/25)

- Complete the first compile for the ADSP-2181 and determine an initial estimate of the overall code size (if > 8K, worry and optimize!). (Approximate check-point date: 3/1)

- Get the code running on the ADSP-2181 evaluation board. Do an initial performance evaluation to make sure that we can service the DACs fast enough at the target cycle time. (Approximate check-point date: 3/22)

- Begin co-simulation with the hardware. (Approximate check-point date: 4/10)
Columbia University
Department of Electrical Engineering
EE E4332, Spring, 2000
Software group work items

General comments: Your group is responsible for delivering the software required to turn our signal processing hardware into an MP3 player. Since this is an embedded application, your main concern is code size. You will have an on-chip memory of 8k 24-bit instruction words. If the code does not fit in this on-chip memory, we will have to go off-chip to fetch instructions, which will slow down everything since there will be a multicycle latency to get data from off-chip SRAMs. If you do a good job in your efforts, there will almost assuredly be commercial interest in your code.

You will start from the existing code in /u2/ee4332/mp3, which is functioning. I recommend that you do all the software development on the Suns, using gcc. We also have “purify” to help in your development efforts. (The ADSP-2181 compiler that we are using is also gcc based, so presumably this will eliminate any code compatibility problems.) The first order of business will be to remove all the Layer 1 and Layer 2 support from the code, since we are only supporting Layer 3. Next, you should remove all the floating point usage from the code. Floating-point is not necessary since a fixed range should be adequate; furthermore, we are not supporting any of the “floating-point” assist instructions in our implementation. Floating-point first gets introduced in the code in the III_dequantize_sample routine. I would recommend removing it routine-by-routine and verifying that the code still works in two ways: compare the PCM samples (they might differ in the last few significant bits, but that should be it); and listen to the resulting music from some known good encodings (“believe.mp3” is a reasonably good encoding, for example). While you are getting a feel for the range of values you are getting in a certain region of the code, you might want to add some “printf” statements to print out the values as an example is being decoded. The other thing is that we are not supporting the “divide” instructions so there should be no divisions in the code (shifts are, of course, Okay, just no general fixed-point divides).

After you have rendered the code “fixed-point”, you should try to compile it for the ADSP-2181 and evaluate the quality of the generated code and the