1) AOSLO Emulator. We will want an AOSLO emulator, so we can develop and test our system without needing to take over the actual instrument.

* The emulator doesn’t have to be perfect, I don’t think. But the signals should come at approximately the same rate as with the real system and be analog with the same voltage levels.
* We can record movies of real signals in multiple channels, so that we have a digital version of the raw acquired pixels. The acquisition only digitizes some of the time – not during the return scan, front porch, and back porch. So our acquired movies don’t fully represent the real time signals. We do, however, know the real timing that goes with the acquired pixels, so we could process acquired movies to put the signals back into a larger movie that played as a regular movie at some frame rate.
* The simplest idea would be to construct such a movie and then play it out a VGA adaptor. That would give us R, G, and B analog signals so we could emulate up to three channels.
* We’d also have the hsync and vsync signals on the VGA. But, these would be for the playback movie and wouldn’t correspond to the hsync and vsync signals provided by the SLO. So we need to think about how to emulate the SLO hsync and vsync.
* A system like this could just run on a dedicated laptop and would allow us to test things out.

2) AOM Emulator. The output of our system will be the signals we send to 1 or more acousto optic modulations (AOMs). We’d like to know what these are doing on exactly the same timebase as the incoming imaging signals.

* An attraction of the frame grabber board that Alf has identified is its 16 channels. This would allow us to simply pump the AOM control signals back into the frame grabber, and get images that are exactly aligned with the incoming image data. Even if we program our own frame grabber, we may want to set up multiple channels to allow this monitoring.
* There is presumably some latency in the AOM response, between when its control signal modulates and when the light modulates. We should look into how fast this is, and hope it is negligible. If not, we could compensate for it and use a signal from a photodiode that picks off part of the beam before it goes into the eye to really know when we were modulating. This assumes no latency in the PMT and amplifier, but I assume that’s a speed of light thing and can be safely ignored.

3) Frame Grabber.

* The frame grabber needs to grab and store the incoming data. We want this to make it to disk so we have it later. Ideally that will then be a record of both the imaging data and the stimulus modulation in all the channels that we are using.
  + It is possible that we might not need to store the image data, since it will be stored in parallel on the separate acquisition computer. If we really trusted the two devices in the two computes to be perfectly synced, we could just rely on the acquisition. But, I don’t think we can trust without checking.
* We could either buy a frame grabber or implement our own as part of an FPGA or similar solution.
  + Key is the latency with which we can get the data. One line buffering seems fine, but not more than that.
  + Another important consideration is how much of the CPU bandwidth (if any) the frame grabber data transfer will use up. Ideally, some sort of DMA system would just move the data where we want it without slowing down the CPU or bus. Or, with the FPGA solution we’ll still want to get the data over to disk, but the real time processing could
  + We should decide an upper bound on the number of channels we need.
  + An advantage of using the frame grabber Alf identifies is that we may be able to just use his code for this, and it will probably be easier to keep the two systems grabbing in sync, since apparently you can slave one of these boards to the other. Also, anytime we can spend money and save time is a win.

4) Processing steps. Here is my take on the steps that need to happen as we conduct an experiment.

* Acquire a reference frame for the retinal area being studied. This doesn’t need to happen in real time and can be done by the CPU if desired.
* Take a raw movie. The raw data should be saved
* Desinusoid each line to produce a desinusoided image. For this purpose this could happen slowly, but since we will need it to happen fast for the real time processing (see below), we will probably want to do this the fast way.
* Process the movie to obtain a good reference frame. Currently I believe this is just selecting a good looking frame by eye. As we work on this problem we can incorporate whatever our current best practice is.
* The reference frame needs to be clocked into whatever hardware (GPU, FPGA) that is going to do the real time tracking.
* Desinusoiding. Because the horizontal scanner has a sinusoidal velocity profile, we need to resample the raw pixel data to make the pixels correspond to a regular spatial grid. This is called desinusoiding. This needs to happen to all image data before aligning, because once eye movements have occurred the sinusoidal effects will screw up the image alignment.
  + There are (at least) two approaches to desinusoiding. One is to acquire an image of a rectangular grid and process this to determine the appropriate transform of the raw data. This is our current method. The other is that in the future we may have digital access to a positional signal from the horizontal scanner, which could then be combined with the incoming image data to do the resampling.
  + The processing of the grid image to determine the transformation can be pretty slow and done in advance, at the start of the imaging session. But the processing of individual lines needs to be very fast since it needs to be done before we can align incoming data with the reference frame.
  + So, the first step as data comes off the A
* Measure transverse chromatic aberration (TCA) and store offset between positions in the stimulation channel (one wavelength) and imaging channel at a different wavelength. We can do this calculation now and it just needs to be done at the beginning and end of a session. The offset gets incorporated when we decide when to clock the AOMs.
* We also clock in the stimulus pattern we want to present. This is determined by the user relative to the reference image, and needs to be converted to a time varying signal for the AOMs for some reference eye position. Note that the pattern is a spatio-temporal one, so that there could be several frames worth of pattern, each different, waiting to go.
* Real time eye tracking. This is the core part.
  + We have the reference frame stored and now need to figure out where they are and where they are going to be.
  + Clock in N lines of the incoming image, with N something like 16. Align this with the reference frame. If the acquisition of the N lines and were instantaneous, this would give us an estimate of the eye position at the acquisition time. In reality, it took some time between when the first and last of the N lines were acquired, so our estimate reflects some average time. I guess a simple guess is where the eyes were when the N/2 line was acquired.
  + As new lines are clocked in, we can repeat the alignment process with new blocks of N lines. These can overlap (or not) with the first block, but basically what we get is an estimate of the eye position relative to the reference frame at a series of times since the start of the frame. This gives us eye position and velocity estimates, and we can use these to predict where the eye will be in the future, probably with some simple linear predictor.
  + As the time to start modulating the AOMs approaches, we need to convert the nominal time varying pattern (specified with respect to eye position when the reference frame was acquired) to the pattern appropriate for the current eye position. Or more precisely, the predicted eye position at the time when the modulation is supposed to occur.
  + Finally, as go time arrives we clock out the appropriate pattern.
  + This gets repeated over frames.
* This is a quote from Yang et al. (2010) on the reasons that they switched to the integrated FPGA solution. “Second, because the imaging and stimulus interfaces could only be synchronized at the beginning of each scan line (i.e. via the H-sync signal), the timing for the acquisition of a given pixel on the imaging side and the delivery of the stimulus that should correspond to that same pixel was only implicit, and depended on the similarity of the dynamics of two independent phase-locked loops (PLLs) on separate boards, made by two different manufacturers.” I think what they mean here is that when they drive the AOM, they need to synchronize the clocking device to the AOM to the time base of acquisition. If the pixel clock is on the same board that is generating the AOM control signals, then there is only one clock. But if not, there is more work and perhaps less precision. On this point, we need to find out a bit more about the timing signals we can get from the frame grabber that Alf is going to be using.
* Also in Yang et al (2010) they discuss other issues they had with the multiboard solution. The main ones are latency on buffering by the A/D board used to get the data and an even larger latency (1-3 msec) on the D/A board that controlled the AOM. I would think we’d want and could get microsecond latencies on these aspects of input and output, but perhaps I’m optimistic.

5) User Interface. Haven’t really thought this through at all.

* We need to think through the user interface to the system. This involves both what happens when we run experiments and what needs to be done to set up an experiment.
  + Ideally, we’d do as little in hardware/firmware as necessary and have an API to pass key information to the hardware/firmware.
  + We want it to be easy to run the experiment.
  + We want as much flexibility as possible in what the experiment looks like, with this being something we can set up easily.
* There needs to be some interface to user response, staircase on stimulus intensity/trial type, etc.