Point-to-point Data Link, Collaborative Whiteboard, and Voice Conference

MIT 6.111 Project Proposal

Michael F. Robbins — *Email:* mrobbins@mit.edu
Scott B. Ostler — *Email:* sostler@mit.edu

November 3, 2006

Table of contents

1 Overview

We intend to construct a digital data link using LEDs and photodiodes to implement a communications channel in free space. A packet engine will interface between the serial stream of data coming in over the channel to packets that are useful to specific applications. We will construct two simple applications to run over this protocol, namely a collaborative whiteboard with shared drawing space betwen the two stations, and a simultaneous voice conference.

A major part of the effort in this project is aimed at making a reliable digital datalink even in the face of contamination from many noise sources, such as ambient sunlight, indoor lighting, and then simply the fact that there is an unknown channel gain because the stations may be close together or far apart. However, most noise sources are narrowband in nature, for example from line level lighting at 120Hz, or some fluorescent ballasts near 26kHz. For this reason, we have chosen to use Orthogonal Frequency Division Multiplexing as our digital encoding scheme. While this may sound complicated, it is in reality just using a many on-off modulated carriers at once. A bit being sent over the chanenl may corresond to a bin in the frequency domain, and using the inverse FFT, are able to construct a time domain signal to send. The narrowband noise will contaminate some of these bits, but using forward error correction techniques such as Hamming codes, we can tolerate a few bands of interference. The receiving end will take the FFT of the time-domain signal, and will be able to extract the various bits that were sent. When compared to simply sending the raw bitstream, this technique provides resistance to narrowband noise and also to impulse ("popcorn") noise. This is a good use of the muscle of the FPGA and should be interesting to watch in action.

This kind of frequency-domain encoding requires special timing synchronization because of the random phase offset between stations. We have devised a simple strategy with a frequency-domain "guard sequence" which will alert the receiver to the correct synchronization phase, and by transmitting each FFT length twice, we are guaranteed to have a signal that is properly phased on alternating groups of samples. (A simliar technique is done in the time domain over any asynchronous serial link, where the signal is oversampled to wait for a start bit.) While this technique cuts the channel capacity in half, it allows us to make a fairly simple way of guaranteeing synchronicity.

2 Modes of Operation

Although our goal is for two independent stations to interact over our communications channel, we will be developing our project using only one labkit. Therefore, to facilitate

testing of our module's functionality during development, we have specified three modes of operation, with only one mode requiring both labkits.

The three modes are as follows:

2.1 Local Mode

In local mode, there is only one labkit, and the application modules do not interact with the communications channel. For example, the Voice module immediately plays back what it records. The goal of this mode is to test the application functionality independent of the communications module.

2.2 Loopback Mode

In loopback mode, there is only one labkit, and the application modules transmit their data over the communications channel, back to themselves. For example, the Voice module records voice samples, transmits them over the communications channel, then plays them back. The goal of this mode is to test the interaction of the communications channel and the application modules.

2.3 Link Mode

In link mode, there are two labkits, and the application modules in each communicate as peers. For example, the Voice modules in each labkit transmit recorded voice samples over the communications channel into the other, for playback. This mode is the desired functionality of the project, but will only be attempted when the two other modes are working correctly.

3 Implementation

A draft block diagram is shown below in Figure 1:

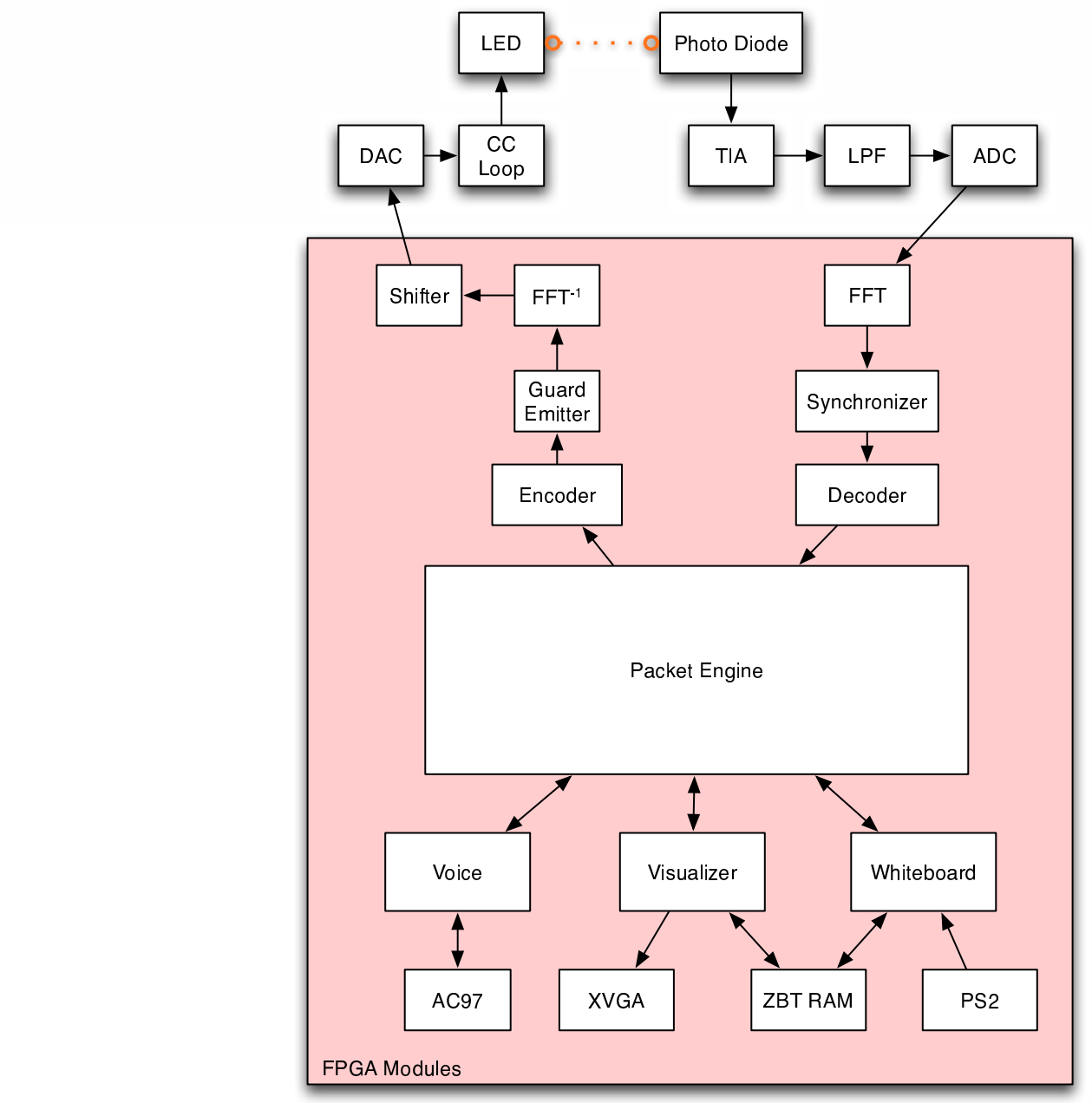

Figure 1. Draft block diagram.

3.1 Optical Components

Our communications scheme is generally applicable to transmitting digital data over any noisy analog channel. While most communications systems today lie in the radio and microwave regions of the RF spectrum, we have chosen to use visible light as our medium of choice. This provides several advantages for this project, including easy development of analog components, relatively easy debugging, and easy ways to "distort" the channel and introduce noise.

For the transmitting end, we are using a simple red Light Emitting Diode (LED), a device which has a remarkably linear relationship between operating current and light intensity. It is also possible to modulate an LED quickly. A current control loop will be constructed to take the voltage output of the DAC (described later) and adjust the current going through the LED. A generic red LED, easily obtained in the 6.002 lab or elsewhere, will be the used.

On the receiving end, a photodiode and transimpedance amplifier will measure the light received. A photodiode, when properly biased, will allow a reverse current to flow proportional to the intensity of light striking its sensor area. The transimpedance amplifier will convert this current into a voltage. After going through a simple RC highpass filter to attenuate ambient light (DC) and line voltage lighting (120Hz), this voltage will proceed into the analog front end. Several options are currently being considered for the photodiode and transimpedance amplifier. One is to use a Texas Instruments OPT101, a chip available in DIP package that contains both the sensor and the amplifier. This makes it very easy to use in our system, but its frequency response characteristics are not ideal, and discrete and separate sensor and amplifier systems are also being investigated.

3.2 Analog Front-End

As almost all of our signal processing will be done digitally, we need a way to convert the analog waveforms to and from digital representations. Currently, we intend to use a 1MS/s rate for both ADC and DAC functions. While we have not yet determined the necessary resolution, this resolution is much more critical for the ADC than for the DAC, because there will be an unknown, time-varying channel gain that must be handled while still providing a useful level of dynamic range.

For the DAC, we are currently considering using an 8-bit DAC. This is fairly easy to obtain (or even build with an R-2R network), and we are currently considering the Analog Devices AD7528 for this function. It is available in a DIP package for easy use on the labkit breadboard, and is relatively inexpensive, and has free samples available from the manufacturer.

For the ADC, we have two options: we can either use a very high resolution ADC so that as the transmitter and receiver move apart, there are still enough bits of dynamic range to provide useful data, or alternatively, we can implement some sort of automatic gain control (AGC), which would allow us to maximize our dynamic range regardless of the attenuation. We choose to implement the latter, using the second half of the AD7528 DAC as a multiplying gain stage, and then using an inexpensive successive approximation ADC for the input. We are currently considering the AD7821, a high speed 8-bit ADC which is available in a DIP package and has free samples available from the manufacturer.

It should also be noted that the analog front end is independent of the actual physical channel, and if significant difficulty emerges in constructing the optical LED/photodiode circuitry, we can easily instead convey the signal over a pair of wires. Although this would make for a less impressive demonstration, the signals would still be carried in an analog way that makes them more resistant to narrowband noise.

3.3 Encoder

The fundamental frame of our system is a 9-bit word, which will be processed by the packet engine and routed appropriately. The signal redundancy is layered on top, and the encoder module is only aware that it needs to protect these 9 bits.

As we are currently considering using FFTs with a length of 64 time samples, we can add considerable redundancy with the (64-9)=55 additional frequency bins. (Not all of these are available; the low frequencies should be avoided due to 1/f noise, and some frequencies must be reserved for the guard signal.) We will use a Hamming code to allow for the correction of several of the nine bits in the word.

The input of this module will be the 9-bit word, and the output will be a 64-bit long signal corresponding to which frequencies to excite.

3.4 Guard Emitter

As described in the overview, we must avoid phase synchronization issues by sometimes inserting a special guard frame, which will be recognized by the synchronizer. This must happen more frequently if there is a large clock offset between stations. The guard emitter module will either propogate the 64 bits from the encoder, or will insert its guard signal and delay the encoder.

3.5 Shifter

Because the output from Xilinx IP Core FFT modules is not in time sequence order, the shifter will buffer the output of an inverse FFT until all time sequences are ready. It will then play the entire sequence to the DAC. (It will actually play the entire sequence twice, so that there is a guarantee that the receiver end will have one phase-synchronous FFT frame in which to get the data.)

3.6 Synchronizer

The synchronizer takes the output of the FFT and performs three important tasks:

- Uses a threshhold to determine whether each bin is "on" or "off".
- Determines whether we have received a guard frame, and if so, resets its internal counter
- Reads alternating FFT frames (those that are synchronous with the guard frame) and passes the 64-bit on/off threshhold status to the decoder.

3.7 Decoder

The decoder module takes the 64-bit long map of which frequency bins have been detected and applies the same Hamming code as used in the Encoder module. By correcting the appropriate bits, the original 9-bit word is determined and sent to the packet engine.

The decoder also provides useful statistics on the number of bit errors to the visualization module.

3.8 Packet Engine

The Packet Engine transmits packetized units of information (called packets) from the application modules to the communications channel. It also dispatches the packets that it receives to the appropriate application module. We describe each function in more detail.

Application modules give packets to the Packet Module by serially saving a bit-stream into a packet queue unique to that module; when the stream has been completely saved into the queue, it is ready to be sent over the channel. The Packet Engine alternatives between queues, sending one packet from each. Packets are converted into a series of frames, and then each frame is sent to the Encoder.

Frames consist of eight bits of information as well as a control bit. A packet is represented by a start frame, a length frame, an appropriate number of data frames, and finally a checksum frame. The start frame's control bit is set high, and the start frame's payload identifies the packet's intended application. The length frame's control bit is set low, and its payload is the number of data frames in the packet. Each byte of the packet is put into its own data frame, with the control bit of the data frames set to zero. The final frame for a given packet is a checksum frame, with low control bit, and payload containing an 8bit checksum for the packet.

The Packet Engine receives frames from the Decoder module, and assembles those frames into packets, which are then dispatched to the appropriate Application module. When a packet is assembled, the Packet Engine verifies the packet's checksum before dispatching it to appropriate Application module. If a packet becomes corrupted, due to a missing frame or invalid checksum, that packet is discarded by the Packet Engine, and the Engine simply waits for the next start frame.

3.9 Whiteboard

The Whiteboard module conducts an interactive drawing session between two stations. A user on one station can draw lines and shapes on their own station's display, using their PS/2 mouse, and their drawings will be transmitted across the communications link and also replicated on the other station's display. In this way, two users can collaboratively draw a single picture.

The Whiteboard module recognizes drawing commands entered via the mouse, and computes the corresponding picture to display on the station's video monitor. The mouse acts as a paintbrush tool, that is moved by physically moving the mouse. A pointer is drawn on the screen to indicate the mouse's current position. When the mouse's left button is held down, the area under the pointer is considered to have been drawn upon. At that point, an instruction corresponding to that action is formulated, and sent to one or both of the local station and the remote station. When a Whiteboard module receives a drawing instruction in its drawing path, it modifies its palette according to the instruction.

The path of a given drawing instruction depends on the station's operating mode:

- In Local Mode, the Whiteboard module directs all drawing instructions into its drawing path. This means simply painting the palette with the user's mouse-strokes, for the purpose of testing the drawing mechanism.
- In Loopback Mode, the Whiteboard module directs all drawing instructions o the Packet Engine. The Packet Engine then transmits the instruction over the communications channel. When the Packet Engine receives the drawing instruction as input, it dispatches that instruction to the Whiteboard module's drawing path.
- In Link Mode, the Whiteboard module combines the functionality of the other two operating modes; that is, it internally draws its own drawing instructions, while also sending them through the Packet Engine.

The Whiteboard module has an output to the Visualizer, to draw its palette on the screen.

3.10 Visualizer

The Whiteboard and Visualizer share a framebuffer in the ZBT memory. The visualizer displays statistics about detected signal and noise levels, bit errors, etc. This frame is then sent to a VGA output for display.

3.11 Voice

The Voice module conducts an audio link between two stations. A user on one station can speak into their microphone, and a digitally sampled recording of their voice will be transmitted across the communication link, and played back on the other station's headphones. In this way, two users can talk together.

The Voice module records and plays back digitized audio samples. Similar to Lab 3, the Voice module periodically samples the AC97's microphone output to record audio, and on playback outputs those recorded samples to the AC97's input. The module uses a near-ideal low-pass filter to remove harmonics from the output. The Voice Module connects to the Packet Engine, so that it can both send and receive audio packets over the station's link layer.

The path of a given audio sample depends on the station's operating mode:

- In Local Mode, the Voice module directs all recorded samples into its playback path. This means simply replaying the recorded samples, for the purpose of testing the recording and playback mechanisms.
- In both Loopback Mode and Link Mode, the Voice module directs all recorded samples through the Packet Engine. The Packet Engine then transmits the audio sample over the communication channel. When the Packet Engine receives an audio sample as input, the Engine gives it to the Voice module as a sample to be played. The Voice module then directs that sample to the playback path. Note that the only difference between these two modes is whether the Packet Engine gives the Voice module the samples that it recorded, or the samples that another Voice module recorded; this is not an important distinction to the Voice module.

The Voice module has an output to the Visualizer, to graphically display diagnostic information.

4 Testing

Because we have designed the system with the multiple modes of operation in mind, we will be able to separately test the applications from the data link. We will also be able to seperately test the physical channel from the link encoding algorithms. This modularity in testing is important because of the complexity of this task.

5 Work Assignment

Mike Robbins will work on the analog frontend, and Scott Ostler will work on the application modules. The Packet Engine, and associated Encoder/Decoder modules, will be mutually designed. We have experience working together, and are comfortable sharing the design of the analog/digital interface.