Here are some initial thoughts on possible raw data formats as well as some specific issues with USB readout implementations in ID Lab firmware. This is really just brainstorming at the moment… hopefully something more structured will follow soon. Comments and suggestions are very welcome.

-Kurtis

**Possible generic waveform sampling ASIC raw data format:**

Each ASIC has a varying number of channels, sample cells per readout window, etc. However, the data out for each readout window is a set of ADC values. The data stream may also contain some diagnostic or housekeeping information, for example counters from feedback loops, temperatures, etc.

To inform the DAQ program of what to expect when it performs a readout, rather than needing hard-coded values, a possible format could be:

1. Header word – indicates N packets are coming (not including this header)
2. N packets, each containing the following:
   a. Packet header word – indicates M samples are coming (not including this header, but including the checksum at the end of the packet)
   b. M-1 samples
   c. Packet checksum

A packet might be one of the following:

1. Window of samples
2. Data derived from waveforms (charges, times)
3. Housekeeping / diagnostic information
4. Overall transfer checksum

This should allow the DAQ program to figure out how much it needs to read on-the-fly. However, it still needs to know the data format. Perhaps we could have some agreement on a command that is issued by the DAQ program in order to query the attached device and see how many channels it has (or which are being actively read out), how many windows per channel, how many housekeeping words, etc.

Open questions:

1. Should all these pieces of the raw data be recorded to file (including packet headers, checksums, etc.)?
2. The added burden in transferring headers and checksums is small compared to full waveform readouts, but if we move to only reading out derived data, it may be desirable to pare things down.
Specific USB readout issues (for existing ID Lab designs or those derived from them):

Current USB readout firmware interfaces with the Cypress FX2LP USB controller via a number of modules: USBr.ead.vhd, USBwrite.vhd, MESS.vhd, USB_IO.vhd. Based on these “legacy” firmware blocks, the basic readout procedure for a given board occurs as follows:

1. Two possible trigger scenarios:
   a. DAQ program issues a software trigger
   b. External trigger directly triggers the board controlling the ASIC
2. Firmware receives trigger
3. Firmware performs necessary ASIC control
4. Firmware negotiates with USB controller in preparation for readout
5. DAQ program attempts to read data through USB

Issues:

1. If the DAQ program attempts to read data and none is available, the program will at best hang briefly (a few seconds), or at worst will hang indefinitely or crash. This can occur, for example, if a readout is attempted in external trigger mode but no trigger has been issued.
   a. A current workaround causes the firmware to send a couple packets of short zero data if an external trigger has not been received in some arbitrary amount of time. This minimally works, but is not very robust.
2. Since the firmware and DAQ software never handshake, there’s no way to do a series of transfers for large data sizes. This has limited the amount of data that can be transferred to the size of the block RAM on the FPGA.

Possible solutions:

The firmware should never send data unless specifically requested by the DAQ software. When a readout is attempted, it should be initiated by sending an “I want data” command to the firmware. If the firmware has triggered, it can then send its data. If not, it can respond that it has none. This should allow enough flexibility to solve both issues raised above.