Ettus Research USRP-1 + gnuradio software as a radar digitizing system

The GNUradio project is an open-source "software-defined radio" project. The codebase supports various hardware, including the "USRP" (Universial Software Radio Peripheral) series of high-speed devices produced by Ettus Research. Although not intended for this purpose, the USRP-1 and the LFRX (Low-Frequency Receiver) daughterboard can be adapted (with minor hardware and significant firmware/software changes) to act as a high-quality marine (pulsed, scanning) radar digitizing card attaching to a host PC via USB (or ethernet, for other USRP models). This page describes how we did it. Source code, FPGA (field-programmable gate array) build, and schematics are attached to the page. This digitzing card works with radR under both Linux and Windows, using the usrp plugin.

Note: To use this device under Windows, you must first install the driver. There are some known issues with the USRP1+Windows on some computers.

Here are two views of the prototype:
usrp_front_panel_connections.jpg

usrp_inside_with_lfrx_and_cabling.jpg

6 May, 2011: These photos show the version 2.0 interface scheme, in which ARP (heading) and ACP (azimuth) connections have been moved from a 2nd LFRX daughterboard to auxilliary digital inputs on the first LFRX daughterboard. This saves power, reduces cost, and allow this basic scheme to be used in the 14-bit family of USRPs, which can only host a single receiver daughterboard.

13 May 2011: The plugin has now been built under windows. This screenshot shows indoor testing with a synthetic pulse source triggered by the Furuno (whose own magnetron is disabled):

radR_usrp_plugin_windows_screenshot.png

The customized USRP-1

Our FPGA build and host-side code has the following features:

USB packets

There are 256 16-bit words in each USB packet. The upper 4 bits (or eventually upper 2 bits, for the 14-bit USRP model) are used to hold up to 1024 (or up to 512 for the 14-bit USRP model) bits of metadata - see below.

Digitizing modes

One of these is active at a time. The mode is selected by sending a command to the USRP.

1. **normal**: USB packets are sent each time a trigger pulse is detected; samples are unsigned 12-bit integers encoded as bits 11:0 of a 16-bit word. The number of packets sent for each pulse, the sampling rate, and the trigger delay are all set by commands sent to the USRP.
2. **raw video**: USB packets are sent continuously, with 12-bit samples from the video line
3. **raw trigger**:USB packets are sent continuously, with 12-bit samples from the trigger line (note: we are using a 12-bit ADC for trigger)

4. **raw ACP/azimuth**:USB packets are sent continuously, with 12-bit (or 1-bit, using Dave's front end) samples from the ACP line
5. **raw ARP/heading**:USB packets are sent continuously, with 12-bit (or 1-bit, using Dave's front end) samples from the ARP line
6. **multiplex**: USB packets are sent continuously; 12-bit samples are interleaved from all 4 signal lines. This allows 4 MSPS simultaneously on each channel. (32 MB/s USB bandwidth = 4 channels * 4 MSPS * 2 bytes per sample). This mode is intended for an "intelligent setup" client, which will more or less automatically determine appropriate gain, delay, and threshold parameters for all channels.

Metadata

Every USB packet is stamped with a serial number (32 bits) encoded in the first 32 available bits of metadata (i.e. top 4 bits of bytes 0, 2, 4, ..., 14 for the 12-bit unit; top 2 bits of bytes 0, 2, 4, ..., 30 of 14-bit unit).
In normal digitizing mode, additional metadata are sent with the first USB packet from each digitized pulse. The metadata in the 12-bit unit are as follows (taken from file gnuradio/usrp/host/include/ usrp_bbprx.h

```
// NB: order of fields in metadata must be reverse of instance "stage_4

      unsigned int USB_serial_no;               // serial number of USB
      unsigned int n_ACPs;                      // number of ACPs since
      unsigned int n_ARPs;                      // number of ARPs since
      unsigned int n_trigs;                     // number of triggers si
      unsigned int ACP_interval_last_ARP;       // ticks in ACP interval
      unsigned int ACP_age_last_ARP;            // ticks since last ACP
      unsigned int ACP_count_last_ARP;          // ACP counter at last AR
      unsigned int ticks_since_last_ACP;        // clock ticks since las
      unsigned int ticks_since_last_ARP;        // clock ticks since las
      unsigned int ACP_interval;                // clock ticks between m
      unsigned int ARP_interval;                // clock ticks between m
      unsigned int trig_interval;               // clock ticks between m
      unsigned long long clock_ticks;           // number of (64MHz) clo
```

Note:

- the metadata use up 448 of the 1024 metadata bits, leaving room for additional metadata such as inclinometer readings or weather station data
- no attempt has been made to optimize the size of counters (we don't really need 32-bits to count heading pulses!), but this can be done to make more room for other metadata
- the apparently overlapping meanings of these fields allow us enough information to very precisely estimate the azimuth angle for each digitized pulse
- so far, I haven't incorporated Dave's "window" scheme for determining whether the azimuth has returned to heading based on the heading line, or whether the ACP count should be used
- the USB packet serial numbers allow us to detect various buffer overruns, and to throw away data from incomplete pulses
- only the first USB packet for each pulse contains the radar metdata; these metadata (except for the USB serial no) are zero in the subsequent packets for a given pulse. This lets us determine which packets are the first packet of a pulse by checking whether the clock_ticks field is non-zero.
- we keep track of the true radar PRF independently of how we are digitizing, by counting trigger pulses. For example, even if the listen period is longer than the inter-pulse interval, so that we are dropping pulses, we can detect this by looking at the n_trigs field. It is guaranteed not to miss valid pulses because this counter is updated independently of any video digitizing or USB data transfers

Sampling rate

The following diagram illustrates the buffers data pass through along the way from the ADC to the USB bus. It also shows sustained rates possible for transfer between buffers, in both megabytes per second (MB / s) and 512-byte packets per second (P/s):

```
USB bus <-------  Cypress FX/2 TX buffer (2 KB) <------ FPGA TX FIFO (8 K
         32 MB/s                                   96 MB/s
         64 KP/s                                  192 KP/s
```

Over the long run, because we only transmit data when a trigger pulse is detected, and only for a preset listening interval, we can sustain digitizing 4096 samples per pulse at PRF=2100, because this is a total data rate of 4096 * 2100 * 2= 17.2 MB/s, well within the

USB bus bandwidth. Moreover, because the 8KB FPGA TX FIFO can be filled at full speed, we are guaranteed to be able to capture 4096 consecutive samples at 64 MSPS before filling it. The rest of the pipeline then empties the FIFO at a lower rate, but in sufficient time before the next incoming trigger pulse. Theoretically, we should be able to digitize pulses at 4 K samples per pulse up to a PRF of about 32 MB / s / (8 KB / pulse) = 4 K pulse / s; i.e. a PRF of 4096 Hz. And if we are willing to reduce the number of samples per pulse to 2K, then we should be capable of digitizing up to PRF = 8192 Hz. In tests, the radR software has been able to sustain incoming data gated to 4096 pulses per sweep at 25 RPM, 4096 samples per pulse, and 64 MSPS, for a total bandwidth of 14.0 MB/s entering radR (with 17.2 MB/s entering the USRP host-side code from the USB bus) - see the sample blipmovie attached below.

Proposal for Hardware front-end using LFRX card (May 2012: Note; upcoming changes to the USRP front-end do not necessarily follow this scheme.)

To allow control of and acquisition of data from serial devices in a time-pinned way (so that we know which radar data correspond to what states of the devices), and to reduce the number of ADC channels required for the radar from 4 (current build) to 2 (next version), here is a proposal for interfacing the USRP to the standard quad of radar signal lines via a single LFRX card.

lfrx_as_radar_digitizing_interface.png

Exising (version 2.0) approach:
DANGER: this circuit may cause problems with your radar; it requires high impedance outputs on radar Video, Azimuth, Heading, and Trigger lines. This is not true for all radars, not even for all Furuno radars, not even for all 1954/1964-style scanners! We will shortly be updating this page with a Schmidt-trigger circuit on Azimuth, Heading, and Trigger lines; subsequent improvements planned include buffering, biasing, and additional gain/ attenuation on the video line.

For this version, I used 2 ADC channels to capture the radar video and trigger signals, but used auxilliary digitial I/O lines for ACP and ARP signals. I used simple resistor ladders to drop voltages into acceptable ranges for each line. This is no doubt a noisy and fragile setup, and yet I was able to use it to record blipmovies at 4K samples per pulse, 4K pulses per sweep, 64 MSPS (range cell size 2.34 metres) with radR using a Furuno 1954C/BB on the roof of the Huggins building. Samples were effectively 11-bits, because the voltage range of the video signal was more like [0V, +1V] than the full [-1V, +1V] domain of

the ADC. Here is the schematic: (note that jumpers are included for ACP and ARP because I've found the ARP signal, at least, in both 5VDC and 12VDC on the same models of Furuno radars.)

lfrx_usrp1_as_radar_digitizer_version2.png

Sample blipmovie

Attached below are a sample blipmovie obtained from radR using the USRP plugin, digitizing at a resolution of 4096 gated pulses per sweep, 4096 samples per pulse, with sampling rate 64 MSPS for a range cell size of 2.34m. This is the only hardware so far from which radR can obtain such high resolution data. However, as shown in the accompanying histogram, the video voltage range is not mapped to the full ADC input range of [-1.0V, +1.0V], and so samples are distributed mainly above 2047, for an effective resolution of approximately 11 bits per sample.

sample_blipmovie_4kx4k_sample_histo.png

The front end proposed above will remedy this. The blipmovie is not very interesting, unfortunately. Here is a screenshot which shows the sample range coverage and dimensions of the scan matrix for one sweep:

sample_blipmovie_4kx4k_screenshot.png