Report Drafting

Introduction

This project began as a conceived innovation to the MIT Coffee Can Radar [1], hereafter referred to as the reference design. The reference design demonstrated three different radar modes - Continuous-Wave (CW) Doppler, Frequency Modulated CW, and crude Synthetica Aperture Radar (SAR) - with minor hardware adjustments and using different processing algorithms. The system was interfaced with a laptop in order to acquire a block of data and then process the data offline and show results in Matlab. A block diagram is shown for reference.

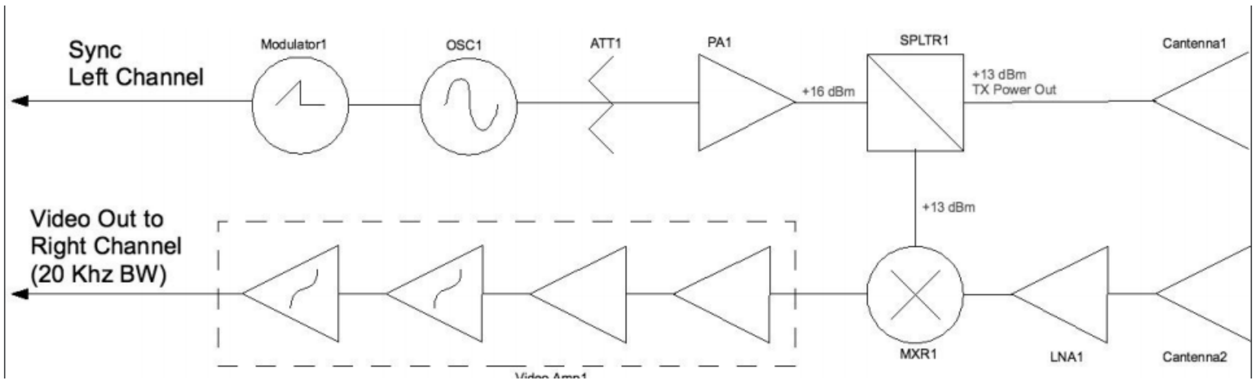

Figure 1: MIT Coffee Can Radar Block Diagram

Our goal was to implement two of these modes using streaming processing and human-in-the-loop mode switching, thus demonstrating a Software-Defined Radar (SDR) system. This project could be extended to interface the control software with waveform tuning hardware and demonstrate Cognitive Radar (CR).

Our initial innovation to the reference design was to design hardware improvements and add a real-time control loop with network offloading of results. The control portion was chosen to be implemented on a Raspberry Pi 2 running Arch Linux for ARM whereas the reference design interfaced with a laptop and used offline batch-mode processing. The modified high-level design is shown below for reference.

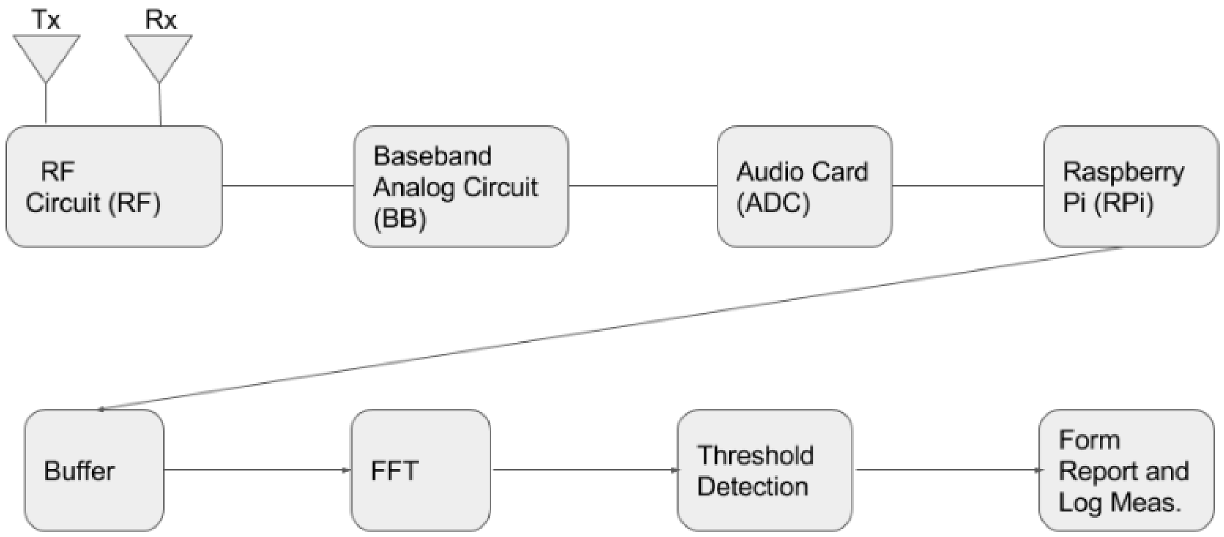

Figure 2: SDR Block Diagram

Discussion of Project

Design Procedure

The SDR design can be summarized into two components: 1. The design improvements to the reference hardware system and 2. The software design.

Hardware Improvements

Software Design

The reference design provided Matlab scripts for ingesting a block of audio data and then processing to show results. There was no detection or tracking software included in this reference and neither did we attempt to develop sophisticated detection or tracking software. Rather, the data can be transformed and displayed as an image, which then allows the eye to perform the job of a target detector/tracker system.

Initial design began with experimenting and rewriting the reference Matlab code to handle processing the data as a stream and displaying the results in an animated waterfall image. This process was repeated for each of the two radar modes - Doppler and FMCW. These Matlab prototypes then became the reference point for developing Python on the Pi.

This process naturally led to the development of two tools which became essential throughout the development - software audio oscilliscopes that plugged into various source data formats. Eventually this settled into one program used to pull various results from the Pi to the development laptop and display the data in real-time in Matlab figures. The second program is similar except that is fetches data from either the disk or the sound card and performs some processing and then displays those results in a streaming fashion.

Finally, algorithms were migrated to Python on the Linux system and development proceeded over remote shell only. During this phase, publishing results to sockets that could be read from my laptop using the tool mentioned above was critical for testing and integration.

The design procedure relied on having the reference design as a starting point, signal processing knowledge, an understanding of the mathematics and physics involved, and the ability to iterate in rapid-prototyping environments like Matlab and Python until results agreed with expectations. We feel that this approach is suitable for research projects intending to demonstrate capability. While it is acknowledged that a more formal top-down methodology ensures success in a production environment, working in research and development, we believe that creativity and persistence produce demonstration results quicker and cheaper than the process-oriented nature of production.

System Description

Specification of the public interface

Inputs

Outputs

The Raspberry Pi does have the capability of running a desktop environment with graphics in order to display results, yet the overhead is significant and so the decision was made leave the Pi running in a headless manner and offload the resulting data over a local network to the development laptop for display. For test and debugging purposes, it was desirable to have the ability to visualize the data as it progressed throught the processing chain. The system output interface is visualized below.

As seen, the program `netscope.m` takes a message name string for which a ZeroMQ socket is created which subscribes to that message type. The PUBLISH/SUBSCRIBE topology enables the Pi to artifically put all the data on the wire, yet

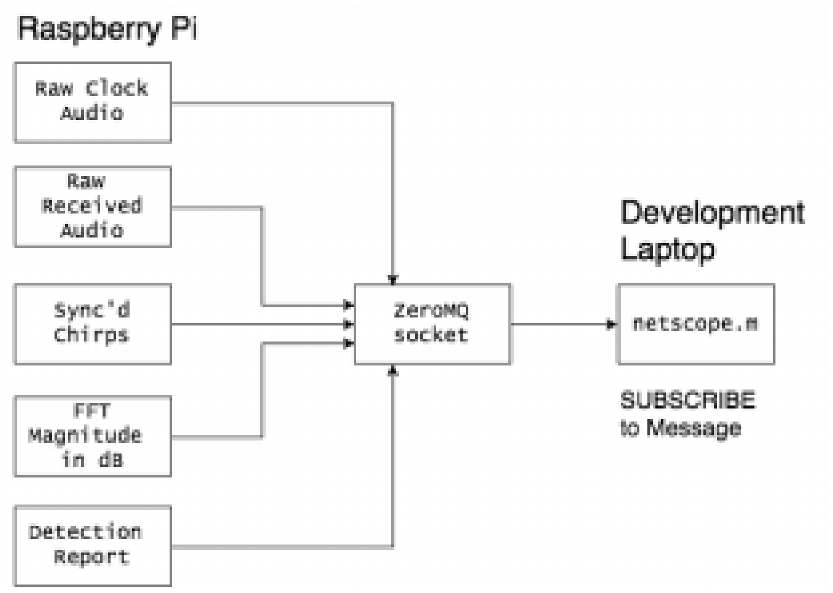

Figure 3: SDR to Development Laptop Interface

only transfer the data that is requested. Matlab does not actually have a ZeroMQ implementation, yet it does expose a Python interface, through which ZeroMQ can be reached seamlessly.

In addition to the above the system also prints out status messages to the Raspberry Pi console.

Were the system to progress into a more productized instantiation, offloading reports could be achieved over a ZigBee link. Of course, this would limit the data rate to discrete detection measurements as opposed to the current toolset which allows viewing the data at any stage in the processing chain.

Algorithm Descriptions

Doppler CW

A Doppler CW system is able to measure the instantaneous radial velocity of a moving object. When an electromagnetic wave reflects off of a moving object, say a car, the wave is shifted in frequency by an amount proportional to the wavelength of the RF signal and the projection of the car's velocity onto the line from whence the wave originated. Doppler CW systems use a continuous sinusoid shifted to the carrier frequency. When the received signal is mixed with the transmitted signal, the difference is output. By performing a Fourier Transform on the received data, over some period of coherency, the radar is able to measure the magnitude response of the data at various frequencies. These are then related to speed using the wavelength.

Frequency Modulated CW (FMCW)

In FMCW, a triangle wave is generated rather than a sinusoid. When the triangle is passed through a voltage-controled oscillator (VCO) the ramp produces a linear frequency modulation (LFM) known as a chirp. This enables measurement of object distance from transmitter over the period of the ramp. As with Doppler, a frequency difference is measured and related to range by the speed of light and the bandwidth of the frequency ramp. However, for FMCW the coherent period is constrained to the period of the triangle ramp. Therefore, FMCW signals must be synchronized to the reference clock signal. This enables knowing the time the ramp was transmitted which can be differenced with the peak return in frequency for a measurement of distance.

Another difference is the dominance of stationary objects on the response spectrum. These are collectively referred to as clutter and reside at zero Hz (DC), though the energy bleeds into the nearby frequencies as well. If a radar is primarily interested in objects that move, clutter can be mitigated by taking a slow-time derivative of the data. Slow-time in that the time interval is measured in ramps instead of samples. By subtracting the previous respnse from the current response, much of the DC energy is removed and moving objects are left in the response. This clutter-mitigation strategy is known as a two-pulse canceller.

Timing constraints

For a sensor of any type, the time scale is ultimately driven by the kinematics of the objects of interest in the environment. At some point a simplification must be made and a period extablished over which it is assumed that the environment is stationary. For the case of our SDR, the objects of interest are of the human-walking and car-driving variety. Each of these is slow relative to airborne jets and so our time scale has some margin compared to typical radars.

If we assume the upper bound of stationarity to be a car moving at 35 m.p.h that translates to about 1 foot in 20 milliseconds and 5 feet in 100 milliseconds. Somewhere between 20 and 100 milliseconds we can reasonably assume our sensed environment is static. Our chirp ramps are tuned to last for 20 milliseconds, and so this becomes the fundamental unit of time for processing. Additionally, we can average over 1 to 5 of these pulses to smooth the output results and still have some confidence of the scene remaining roughly stable.

The other time constraint in this case is the ability of processing resources to handle the throughput requirements. Fortunately, the relatively slow speed of the objects of interest coupled with the fact that our system does not have the power to see beyond 1 km for a 10 square-meter target, means the information of interest is contained within a very narrow region of spectrum near DC. In fact, it is within the bandwidth of human auditory sensing. This pleasant coincidence results in an abundance of analog-to-digital converters with the requisite sample rates. This also means that our incoming data rate of 48000 samples/second is manageable for modern processor chips. Ultimately, required processing throughput depends on the data rate and the Raspberry Pi can handle a few audio signal processing operations within the required time frame.

Error handling

We can categorize the classes of errors as those which are induced by unexpected inputs, those induced by uncaught exceptions within the primary Python routine, and those induced by kernel scheduling, causing lag or loss of flow.

The first occur when the routine is unable to synchronize to the reference clock signal. This exception is caught by wrapping the sync block in a try/catch structure. If we are unable to sync, we want the routine to keep trying without falling apart. This reflects the software's inability to control external events, such as low battery power, fried circuits, or some other failure of the signal chain. Similarly, the routines downstream of sync need to predicate their execution on the indication from sync that everything is working. This is achieved with control flags once sync is achieved. Additionally, each pulse iteration checks the period of the sync interval to validate stability and throws a syncLost flag if sync is lost. Therefore, acquireSync is the initial state to which the routine returns if exceptions are encountered.

The second, exceptions within the main Python routine, are due to software bugs and unexpected corner cases. For our prototyping system, these are addressed using the built-in Linux kernel control wrapper called `systemd`. We stress that when the input is operating correctly and the kernel is able to keep up with the data processing and throughput requirements, there have as yet not been uncaught errors within the main loop.

However, in order to handle the unexpected, three `systemd` unit modules were written to indicate what actions should be taken when one of the main programs crashes unexpectedly. The modules can be enabled as services which allow them be started automatically after a reboot once the kernel boot reaches the point where the device drivers have been intialized. Additionally, event actions can be specified, such as OnFailure or OnWatchdog. Precedence may be set such that program two waits until program one has successfully intitialized. In this way, we are leveraging the existing tool set within the wider Unix community to handle simple control flow and error handling. This is opposed to writing a custom routine to interact with the kernel scheduler. For a rapid-prototyped demonstrator, this reduces risk and cost. A produciton system might require more extensive effort to guarantee proper exception protocol and real-time scheduling priority.

This framework also enables handling the third failure mode where the kernel scheduling causes the audio server to fall behind resulting in audio discontinuities. In this case, the audio server, which is implemented using the third-party Jack library with ALSA as the device backend, is controlled by a custom `systemd` service which stops and restarts the server in the case where errors occurs due to overruns.

Hardware Implementation

Software Implementation

For brevity, we will focus on the Python real-time code here and not on the Matlab tools or prototyping routines. A flow chart is presented below for reference.

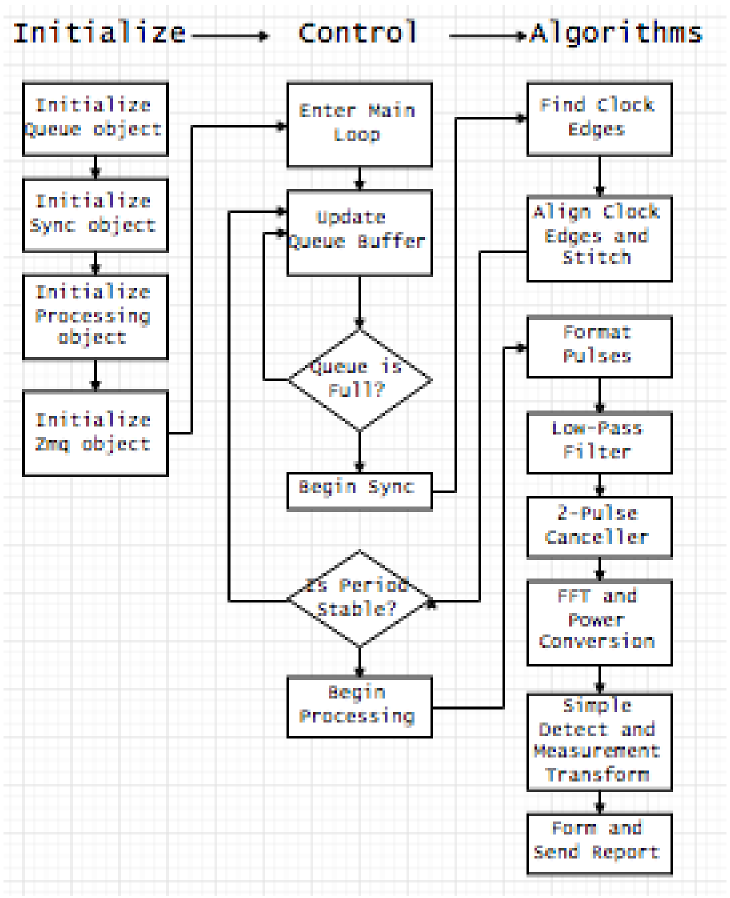

Figure 4: Software Flow Chart

Test Plan

Presentation, Discussion, and Analysis of Results

Analysis of Any Errors

Analysis of why the Project may not have worked and Efforts made to Identify root cause issues

Summary and Conclusion

Code Listings

Matlab Visualization Tools

```
% filename: netscope.m
% author: Paul Adams

function netscope(varargin)
    % parse input args
    args = containers.Map(varargin(1:2:end), varargin(2:2:end));
    %socket_ip =char(py.socket.gethostbyname('thebes'));
    socket_ip = get_arg(args, 'socket_ip', 'thebes');
    sub_topic = get_arg(args, 'sub_topic', 'raw');
    v2db = @(x) 10*log10(x);
    dev = init_data_device(socket_ip, sub_topic);
    not_init = true;
    figure(1); clf; figure(2); clf;
    i_frame = 0;

    % main loop
    while true
        % update data
        data = dev.recv();
        idx = strfind(char(data), ';;;');
        header = data(1:idx);
        data = data(idx+3:end); % remove header
        shape = regexp(char(header), '.*n_row: (\d+).*', 'tokens');
        n_row = str2double(shape{1});
        v = cell2mat(cell(py.list(py.numpy.fromstring(data))));
        if n_row > 1 && n_row < 100
            v = reshape(v, length(v)/n_row, n_row);
            v = mean(v, 2);
        end
        if strcmp(sub_topic, 'filt') || strcmp(sub_topic, 'avg')
            v = v2db(v);
        end

        if not_init
            not_init = false;
            [x, wf, pl, img] = init_figures(v.');
        end

        % update waterfall
        i_frame = i_frame + 1;
        n_frame = size(wf, 1);
        if i_frame <=  n_frame
            idx = i_frame;
        else
            idx = n_frame;
            wf = circshift(wf, [-1, 0]);
        end
        wf(idx, 1:length(v)) = v.';
        x(1:length(v)) = v;
        %update plots
        %caxis([6, 30] + median(wf(:)));
        img.CData = wf;
        if length(pl.YData) < length(x)
            [x, wf, pl, img] = init_figures(v.');
        end
        pl.YData = x;
        drawnow;
    end
end

function val = get_arg(args, key, default)
    if isKey(args, key)
        val = args(key);
    else
        val = default;
    end
end

function dev = init_data_device(socket_ip, sub_topic)
    tcp = sprintf('tcp://%s:5556', socket_ip);
    ctx = py.zmq.Context();
    dev = ctx.socket(py.zmq.SUB);
    dev.connect(tcp);
    dev.setsockopt(py.zmq.SUBSCRIBE, py.str(sub_topic));
end

function [pulse_idx, n_samp_pulse] = pulse_sync(x, sync_chan)
    % perform pulse sync
    a = x(:, sync_chan) > 0; % square wave
    b = diff([0; a]) > 0.5; % true on rising edges
    c = diff([0; a]) < - 0.5; % true on falling edges

    rise_idx = find(b);
    fall_idx = find(c);
    n = min(length(rise_idx), length(fall_idx));
    ramp_idx = fall_idx(1:n) - rise_idx(1:n);

    % guaranteed shortest ramp time so we don't accidentally integrate over boundaries
    n_samp_pulse = min(ramp_idx);
    pulse_idx = rise_idx;
end

function [x, wf, pl, img] = init_figures(v)
    n_frame = 100;

    % init figures
    f1 = figure(1);
    f1.WindowStyle = 'docked';
    t = (0:length(v) - 1)/48000;
    x = zeros(length(t), 1);
    pl = plot(t*1000, v, '.-');
    grid on;
    xlabel('time [ms]')
    f2 = figure(2);
    f2.WindowStyle = 'docked';
    wf = zeros(n_frame, length(v));
    wf(1, :) = v;
    %img = imagr(wf, 'fignum', 5);
    img = imagesc(wf);
    colorbar; colormap hot
    xlabel('sample')
    ylabel('time')
    %caxis([-20, 20])
    ui_init();
end

function cb_clim(src, evnt)
    obj = src.Parent.Children(end - 1);
    cur_value = obj.Limits;
    dval = 1;
    if strcmp(src.Tag, 'nf_up')
        new_val = cur_value + dval;
    elseif strcmp(src.Tag, 'nf_dn')
        new_val = cur_value - dval;
    elseif strcmp(src.Tag, 'cal')
        new_val = [3, 30] + median(src.Parent.Children(end).Children(1).CData(:));
    elseif strcmp(src.Tag, 'dr_up')
        new_val(1) = cur_value(1) - dval;
        new_val(2) = cur_value(2) + dval;
    elseif strcmp(src.Tag, 'dr_dn')
        new_val(1) = cur_value(1) + dval;
        new_val(2) = cur_value(2) - dval;
    elseif strcmp(src.Tag, 'dr')
        new_val = cur_value*src.Value;
    elseif strcmp(src.Tag, 'nf')
        new_val = cur_value + src.Value;
    end
    obj.Limits = new_val;
    caxis(new_val);
end

function ui_init()
    cb = colorbar();

    x0 = cb.Position(1);
    dx = cb.Position(3);
    y0 = cb.Position(2);
    dy = cb.Position(4);

    uicontrol('Style', 'text', 'string', 'NF', ...
        'units', 'normalized',...
        'position', [x0 + 1.4*dx, y0, dx, dx], ...
        'Fontweight', 'bold', 'fontsize', 14);

    uicontrol('Style', 'pushbutton', 'string', 'cal',...
        'units', 'normalized',...
        'position', [x0 + 1.2*dx, y0 + 5*dx, dx, dx], ...
        'callback', @cb_clim, 'tag', 'cal');

    uicontrol('Style', 'pushbutton', 'string', '+', ...
        'units', 'normalized',...
        'position', [x0 + 1.2*dx, y0 + dx, dx, dx], ...
        'callback', @cb_clim, 'tag', 'nf_up');

    uicontrol('Style', 'pushbutton', 'string', '-', ...
        'units', 'normalized',...
        'position', [x0 + 2.4*dx, y0 + dx, dx, dx], ...
        'callback', @cb_clim, 'tag', 'nf_dn');

    uicontrol('Style', 'text', 'string', 'DR', ...
        'units', 'normalized',...
        'position', [x0 + 1.4*dx, y0 + dy - dx, dx, dx], ...
        'Fontweight', 'bold', 'fontsize', 14);

    uicontrol('Style', 'pushbutton', 'string', '+', ...
        'units', 'normalized',...
        'position', [x0 + 1.2*dx, y0 + dy - 2*dx, dx, dx], ...
        'callback', @cb_clim, 'tag', 'dr_up');

    uicontrol('Style', 'pushbutton', 'string', '-',...
        'units', 'normalized',...
        'position', [x0 + 2.4*dx, y0 + dy - 2*dx, dx, dx], ...
        'callback', @cb_clim, 'tag', 'dr_dn');
end

% filename: audioscope.m
% author: Paul Adams
%
function scope(varargin)
    % playback params
    fs = 48000;
    n_chan = 2;
    n_bit = 16;

    % parse input args
    args = containers.Map(varargin(1:2:end), varargin(2:2:end));
    oversample = get_arg(args, 'oversample', 2);
    socket_ip = get_arg(args, 'socket_ip', '192.168.0.122');
    display_on = get_arg(args, 'display_on', true);
    record_on = get_arg(args, 'record_on', false);
    dwell_ms = get_arg(args, 'dwell_ms', 100);
    window_func = get_arg(args, 'window_func', @rectwin);
    source = get_arg(args, 'source', 'zmq');
    record_duration = get_arg(args, 'record_duration', 10);

    T = dwell_ms/1000;
    dB_lim = [-0, 60];
    i_samp = 0;
    n_period = 1024;
    n_samp_frame = T*fs;
```

```
% update playback params
nframe = floor(n_samp_frame/n_period);
n_samp_frame = 4096;
T = n_samp_frame/fs;
dontstopcantstopwontstop = true;

% init source
device = init_data_device(source, args, fs, n_bit, n_chan, T);

w = repmat(window(window_func, n_samp_frame), 1, n_chan);

% init figures
if display_on
    v2db = @(x) 20*log10(abs(x));
    dt = T/n_samp_frame;
    t = 0:dt:T - dt;
    n_fft = oversample*2^nextpow2(n_samp_frame);
    df = fs/n_fft;
    f = 0:df:fs/2 - df;
    n_frame_wf_buffer = 500;
    i_frame = 0;
    t_slow = 0:T:n_frame_wf_buffer*T - T;
    wf = zeros(n_frame_wf_buffer, n_fft/2);
    f_ = figure(1);
    f_.WindowStyle = 'docked';
    f_.UserData.i_chan = 1;
    uicontrol('Style', 'pushbutton', 'string',...
        'channel toggle', 'position', [10, 10, 80, 30], 'callback', @cb_chan);
    subplot(211); l1 = plot(t*1e3, zeros(n_samp_frame, n_chan), '.-');
    xlabel('time [ms]'); ylabel('dB'); title('time view');
    ylim([-1.1, 1.1]);
    grid on;
    subplot(212); l2 = plot(f/1e3, zeros(n_fft/2, n_chan), '.-');
    xlabel('freq [kHz]'); ylabel('dB'); title('frequency view');
    ylim(dB_lim);
    grid on;

    f2 = figure(2);
    f2.WindowStyle = 'docked';
    img = imagesc(f/1e3, t_slow, wf);
    colorbar;
    caxis(dB_lim);
    ylabel('time [s]'); xlabel('freq [kHz]');
end

% main loop
while dontstopcantstopwontstop
    % update data
    if strcmp(source, 'audio_card')
        if record_on
            device.recordblocking(record_duration);
            dontstopcantstopwontstop = false;
            fprintf('writing data to file...\n');
            fname = sprintf('%d_%s', round(now*10000), 'radar_data_cache.wav');
```

```
                audiowrite(fname, device.getaudiodata(), fs);
            else
                device.record(T);
                while device.get('TotalSamples') < n_samp_frame; end
                v = device.getaudiodata();
            end

        elseif strcmp(source, 'wav_file')
            v = device((1:n_samp_frame) + i_samp, :);
            end_samp = size(device, 1);
            i_samp = i_samp + n_samp_frame;
            if i_samp >= end_samp - n_samp_frame
                dontstopcantstopwontstop = false;
            end
        end

        if display_on
            % update spectrum
            x = v;
            %x(:, 1) = x(:, 1)/max(x(:, 1));
            %x(:, 2) = x(:, 2)/max(x(:, 2));
            %sound(v(:, 1));
            V = fft(x, n_fft);
            V = v2db(V(1:n_fft/2, :));

            % update waterfall
            i_frame = i_frame + 1;
            if i_frame <=  n_frame_wf_buffer
                idx = i_frame;
            else
                idx = n_frame_wf_buffer;
                wf = circshift(wf, [-1, 0]);
            end

            wf(idx, :) = V(:, f_.UserData.i_chan).';

            %update plots
            for i = 1:size(x, 2); l1(i).YData = x(:, i);    end
            for i = 1:size(V, 2); l2(i).YData = V(:, i);    end
            % img.CData = wf;
            drawnow;
        end
    end
    if record_on
    end
end

function cb_chan(src, data)
    fig = src.Parent;
    if fig.UserData.i_chan == 1
        fig.UserData.i_chan = 2;
    else
        fig.UserData.i_chan = 1;
    end
```

```
end

function val = get_arg(args, key, default)
    if isKey(args, key)
        val = args(key);
    else
        val = default;
    end
end

function dev = init_data_device(source, args, fs, n_bit, n_chan, T)
    if strcmp(source, 'audio_card')
        device_id = 2;
        dev = audiorecorder(fs, n_bit, n_chan, device_id);
    elseif strcmp(source, 'wav_file')
        dev = audioread('radar_data_cache.wav');
    else
        error('Unrecognized audio source')
    end
end
```

Matlab Algorithm Protyping

```
% filename: batch_fmcw_detection.m
% author: Paul Adams
%
function x = batch_fmcw_detection(varargin)
    % init and params
    dbv = @(x) 20*log10(abs(x));
    fname = dir('*radar_data*');
    [~, idx] = max([fname.datenum]);
    fname = fname(idx).name;

    % parse input args
    args = containers.Map(varargin(1:2:end), varargin(2:2:end));
    oversample = get_arg(args, 'oversample', 5);
    window_func = get_arg(args, 'window_func', @rectwin);
    fname = get_arg(args, 'fname', fname);
    detrend_flag = get_arg(args, 'detrend_flag', false);

    sync_chan = 1;
    signal_chan = 2;

    % fetch data
    [x, fs] = audioread(fname);

    % sync pulses
    [pulse_idx, n_samp_pulse] = pulse_sync(x, sync_chan);

    % derived params
    n_pulse = length(pulse_idx) - 1;
    N = size(x, 1);
    t = (0:N - 1)/fs;
    n_fft = 2^nextpow2(n_samp_pulse)*8;
    f = (0:n_fft/2-1)*fs/n_fft;
    taper = repmat(window(window_func, n_samp_pulse)', n_pulse, 1);
    pulse_proto = 1 - abs(linspace(-1, 1, n_samp_pulse));

    %radar parameters
    c = 3E8; %(m/s) speed of light
    fstart = 2260E6; %(Hz) LFM start frequency for example
    fstop = 2590E6; %(Hz) LFM stop frequency for example
    BW = fstop-fstart; %(Hz) transmti bandwidth

    %range resolution
    rr = c/(2*BW);
    max_range = rr*n_samp_pulse/2;
    r = linspace(0, max_range, n_fft);

    % perform pulse compression
    % x(:, signal_chan) = filter(pulse_proto-0.5, 1, x(:, signal_chan));

    % form pulse matrix (optionally detrend)
    y = zeros(n_pulse, n_samp_pulse);
    for i = 1:n_pulse
```

```
        pulse = x((1:n_samp_pulse) + pulse_idx(i), signal_chan)';
        y(i, :) = pulse - detrend_flag*mean(pulse);
    end

    % clutter suppression
    dy = [zeros(1, n_samp_pulse); y(1:n_pulse - 1, :)];
    dyy = [zeros(2, n_samp_pulse); y(1:n_pulse - 2, :)];

    z2 = y - dy; % 2-pulse canceller
    z3 = y - 2*dy + dyy; % 3-pulse canceller

    if 1
        % Taper, FFT, Power
        Z1 = fft(y.*taper, n_fft, 2);
        Z1= dbv(Z1(:, 1:n_fft/2));

        im = imagr(Z1, 'x', r, 'y', t, 'fignum', 101, 'colormap', 'hot');
        xlabel('range bin');
        ylabel('time');
        %%xlim([0, 50]);
        title('Range Time Indicator - No clutter suppresion');

        % Taper, FFT, Power
        Z2 = fft(z2.*taper, n_fft, 2);
        Z2= dbv(Z2(:, 1:n_fft/2));

        im = imagr(Z2, 'x', r, 'y', t, 'fignum', 102, 'colormap', 'hot');
        xlabel('range bin');
        ylabel('time');
        %xlim([0, 50]);
        title('Range Time Indicator - 2-Pulse Canceller');
    end

    % Taper, FFT, Power
    Z3 = fft(z3.*taper, n_fft, 2);
    Z3= dbv(Z3(:, 1:n_fft/2));
    %Z3 = dbv(Z3);

    im = imagr(Z3, 'x', r, 'y', t, 'fignum', 103, 'colormap', 'hot');
    xlabel('range bin');
    ylabel('time');
    %xlim([0, 50]);
    title('Range Time Indicator - 3-Pulse Canceller');

    [~, range_gates] = max(Z3, [], 2);
    figure(3)
    plot(r(range_gates), t(pulse_idx(1:end-1)), 'o')
    xlabel('range [m]'); ylabel('time')
    %xlim([0, 50]);
    grid on;
    ax = gca;
    ax.YAxis.Direction = 'reverse';
    title(fname);
end
```

```
function [pulse_idx, n_samp_pulse] = pulse_sync(x, sync_chan)
    % perform pulse sync
    a = x(:, sync_chan) > 0; % square wave
    b = diff([0; a]) > 0.5; % true on rising edges
    c = diff([0; a]) < - 0.5; % true on falling edges

    rise_idx = find(b);
    fall_idx = find(c);
    n = min(length(rise_idx), length(fall_idx));
    ramp_idx = fall_idx(1:n) - rise_idx(1:n);

    % guaranteed shortest ramp time so we don't accidentally integrate over boundaries
    n_samp_pulse = min(ramp_idx);
    pulse_idx = rise_idx;
end

function val = get_arg(args, key, default)
    if isKey(args, key)
        val = args(key);
    else
        val = default;
    end
end

% filename: DopplerConfig.m
% author: Paul Adams

classdef DopplerConfig < handle
    properties
        ax
        state
        wav
        is_file
        n_total
        n_samp
        Tp
        oversample
        n_dwell_detect
        taper
        i_samp
        i_chan
        i_dwell
        n_filt
        v_mph
        funcs
        max_filt
        debug_level
        noise_est
        alpha_n
        beta_n
    end

    properties(Constant)
```

```
        MPH_CF = 2.23694;
        FC = 2590E6; %(Hz) Center frequency (connected VCO Vtune to +5 for example)
        C_LIGHT = 3e8; %(m/s) speed of light
        AUDIO_DEVICE_ID = 2;
        FS = 44100;
        % Thresholds
        pass_thresh = 0.5; % instantaneous noise level indicating vehicle crossing
        actv_thresh = 28; % peak must be X dB above noise est
        v_max_mph = 4000;% suppress speeds above 50 mph
    end

    methods
        function me = DopplerConfig(args)
            taper_func = eval(sprintf('@%swin', args('windowing_function')));
            me.debug_level = args('debug_level');

            me.Tp = args('dwell_period_ms')/1000;
            me.oversample = args('oversample_factor');
            me.n_dwell_detect = args('n_detection_dwell');
            me.i_samp = 0;
            me.i_dwell = 0;
            me.wav = args('wav_file');
            me.i_chan = args('channel_vector');
            if ~isempty(me.wav)
                I = audioinfo(me.wav);
                me.n_total = I.TotalSamples;
                me.FS = I.SampleRate;
                me.is_file = true;
            else
                me.n_total = inf;
                me.is_file = false;
                me.wav = audiorecorder(me.FS, 16, (me.i_chan), me.AUDIO_DEVICE_ID);
                me.wav.record(2*me.Tp);
            end
            me.noise_est = [];
            me.n_samp = round(me.Tp*me.FS);
            me.Tp = me.n_samp/me.FS;
            me.taper = window(taper_func, me.n_samp).';
            me.n_filt = me.oversample*2^nextpow2(me.n_samp);

            % State
            me.state.real_time = args('real_time');
            me.state.Active = false; % indicates there is moving object in frame,
            % assuming peak value is x dB above noise
            me.state.Passing = false; % indicates an object is passing the radar,
            % assumes instantaneous noise level is x dB above historical value
            me.state.VehicleCount = 0;

            % calculate velocity
            lambda = me.C_LIGHT/me.FC;
            df = me.FS/me.n_filt;
            doppler = 0:df:me.FS/2 - df; % doppler spectrum
            me.funcs.dop2mph = @(dop) me.MPH_CF*dop*lambda/2;
            me.funcs.mph2dop = @(mph) 2*mph/(me.MPH_CF*lambda);
            me.v_mph = me.MPH_CF*doppler*lambda/2;
            max_doppler = 2*me.v_max_mph/(me.MPH_CF*lambda);
            [~, me.max_filt] = min(abs(doppler - max_doppler));

            % iir filter coeffs
            % noise filter
            me.alpha_n = 0.90; % historical value weighting
            me.beta_n = 1 - me.alpha_n; % current measurement weighting

            plot(me.v_mph, me.v_mph, '.-'); hold on;
            plot(me.v_mph(1), 1, 'r*'); hold off;
            xlim([0, 200])
            ylim([-50, 50])
            xlabel('velocity [mph]'); ylabel('dB'); title('Audio Spectrum');
            me.ax = gca;
            grid on;
        end
    end
end

% filename: batch_doppler_example.m
% author: Paul Adams
%
function batch_doppler_example(varargin)
    % config
    keys = varargin(1:2:end);
    vals = varargin(2:2:end);
    opts = containers.Map(keys, vals);

    fname = fullfile('..', 'reference', 'charvat_doppler', 'Off of Newton Exit 17.wav');
    fname = 'test.wav';
    os_factor = 8; % oversample factor
    i_chan = 1; % the channel to process

    Tp = opts('Tp'); %(s) integration period
    do_animation = opts('animate');
    taper_func = opts('window');

    % Process/Plot
    tic;
    [v, x] = fetch_and_format(fname, Tp, i_chan); toc
    X = process_doppler(x, os_factor, taper_func);  toc

    if do_animation
        animate(X, Tp, length(v), opts);
    else
        plots(v, x, X); toc
    end
end

function [v, x] = fetch_and_format(fname, Tp, i_chan)
    fprintf('Fetch and Format...');
    [Y, fs] = audioread(fname);
    % dependant parameters
    n_samp = round(Tp*fs); %# of samples per pulse
    n_total = size(Y, 1);
    n_chan = size(Y, 2);
    n_total = floor(n_total/n_samp)*n_samp;
    n_dwell = n_total/n_samp;

    v = Y(1:n_total, i_chan); % Truncate to be N*L/N
    x = reshape(v', n_samp, n_dwell).'; % let each row have n_samp samples
end

function X = process_doppler(x, os_factor, taper_func)
    fprintf('Begin Doppler Process... \n');
    [n_dwell, n_samp] = size(x);
    n_filt = os_factor*2^nextpow2(size(x, 2));
    taper = window(taper_func, n_samp).';
    x_w = bsxfun(@times, x, taper);
    X = fft(x_w, n_filt, 2);
    X = X(:, 1:n_filt/2); % keep only first image
end

function plots(v, x, X)
    fprintf('Begin Plots... \n');
    set_clim = @(x) caxis([median(x(:)) + 30, max(x(:)) - 3]);
    volt2dB = @(x) 20*log10(abs(x));
    n_total = length(v);
    [n_dwell, n_samp] = size(x);
    n_filt = size(X, 2);

    % constants
    MPH_CF = 2.23694;
    FC = 2590E6; %(Hz) Center frequency (connected VCO Vtune to +5 for example)
    C_LIGHT = 3e8; %(m/s) speed of light
    fs = 44.1e3;

    df = fs/n_filt;
    doppler = 0:df:fs/2 - df; % doppler spectrum

    df = fs/n_total;
    f = -fs/2:df:fs/2 - df; % sampled spectrum

    dt = 1/fs;
    t = 0:dt:n_total*dt - dt;

    %calculate velocity
    lambda = C_LIGHT/FC;
    v_mph = MPH_CF*doppler*lambda/2;

    %calculate slow time
    Tp = n_samp*dt;
    slow_time = 0:Tp:n_total*dt - dt;

    X_db = volt2dB(X);
```

```
% Time/Freq view
if 1
    figure(2); clf;
    subplot(211);
    plot(t, v);
    xlabel('time');
    title('Time/Freq view of Doppler Data')
    grid on;

    subplot(212);
    plot(f, volt2dB(fftshift(fft(v))));
    xlabel('frequency Hz')
    xlim([0, 2e3])
    grid on;
end

% complete image
if 0
    figure(3); clf;
    imagesc((1:n_samp)*dt, 1:n_dwell, volt2dB(x));
    colorbar;
    xlabel('time'); ylabel('dwell');
    title(sprintf('Slow Time Image %d %d-sample dwells', n_dwell, n_samp))
end

% slow image
if 1
    figure(4); clf;
    imagesc(X_db);
    colorbar;
    try; set_clim(X_db); catch; end
    title('Full Doppler Image')
end

% plot charvat reference image
if 0
    fname = fullfile('..', 'reference', 'charvat_doppler', 'charvat_results');
    load(fname);
    figure(5); clf
    imagesc(velocity*MPH_CF,time,v-mmax,[-50, 0]);
    colorbar;
    xlim([0 100]); %limit velocity axis
    xlabel('Velocity (mph)');
    ylabel('time (sec)');
    title('Charvat Reference Image')
end

% truncated image
if 1
    figure(10); clf;
    imagesc(v_mph, slow_time, X_db);
    colorbar;
    try; set_clim(X_db); catch; end
    xlim([0, 50])
```

```
        xlabel('mph');
        ylabel('time (sec)');
        title(sprintf('Doppler Image to 100 mph - %d dwells at %0.3f sec. dwell time', ...
            n_dwell, n_samp*dt));
    end
end

function animate(X, Tp, n_total, opts)
    MPH_CF = 2.23694;
    FC = 2590E6; %(Hz) Center frequency (connected VCO Vtune to +5 for example)
    C_LIGHT = 3e8; %(m/s) speed of light

    set_clim = @(x) caxis([median(x(:)) + 40, max(x(:)) - 3]);
    volt2dB = @(x) 20*log10(abs(x));
    v = X.';
    v = ifft(v(:));

    [n_dwell, n_filt] = size(X);
    fs = 44100;
    dt = 1/fs;

    %calculate velocity
    df = 0.5*fs/n_filt;
    doppler = 0:df:fs/2 - df; % doppler spectrum
    lambda = C_LIGHT/FC;
    v_mph = MPH_CF*doppler*lambda/2;

    %calculate slow time
    slow_time = 0:Tp:n_total*dt - dt;

    X_db = volt2dB(X);

    f = figure(1); clf;
    if opts('image')
        imagesc(v_mph, slow_time, X_db);
        colorbar; colormap hot;
        try; set_clim(X_db); catch; end
        xlim([0, 45]);
        ylim([0, 5]);
        ax = gca;
        xlabel('mph');
        ylabel('time (sec)');
        title(sprintf('Doppler Image to 100 mph - %d dwells at %0.3f sec. dwell time', ...
            n_dwell, dt));
        for i = 1:n_dwell - 11
            ax.YLim = ax.YLim + Tp;
            pause(0.1);
        end
    else
        idx = 1:opts('n_dwell');
        ax(1) = subplot(211);
        plot(v_mph, mean(X_db(idx, :)), '.');
        ax(1).NextPlot = 'add';
        plot(0, 0, 'r*');
        xlim([0, 100]);
        ylim([-50, 30]);
        grid on;
        xlabel('mph');
        ylabel('dB');
        ax(1).Title.String = sprintf('Time: %d sec. Speed: %d mph', ...
                    floor(opts('Tp')), 0);
        ax(1).Title.FontSize = 16;

        ax(2) = subplot(212);
        imagesc(v_mph, (1:5)*opts('Tp'), X_db(idx, :));
        colormap hot;
        caxis([-10, 30]);
        xlim([0, 100]);
        xlabel('mph');
        ylabel('sec');
        ax(2).Title.String = sprintf('Window: %s. Dwell Period: %d ms. Detection Period: %d ms', ...
                char(opts('window')), 1e3*(opts('Tp')), 1e3*5*opts('Tp'));

        if 0
            vid = VideoWriter('doppler_playback.avi');
            vid.FrameRate = round(1/opts('Tp'));
            vid.Quality = 90;
            open(vid);
        end

        for i = 2:n_dwell - 5
            tic;
            ydata = mean(X_db(idx + i, :));
            [v_peak, i_peak] = max(ydata);
            ax(1).Children(2).YData = ydata;
            ax(1).Title.String = sprintf('Time: %d seconds. Speed: %d mph', ...
                floor(i*opts('Tp')), round(v_mph(i_peak)));
            ax(1).Children(1).XData = v_mph(i_peak);
            ax(1).Children(1).YData = v_peak;

            ax(2).Children(1).CData = X_db(idx + i, :);
            ydata = opts('Tp')*(idx + i);
            ax(2).Children(1).YData = ydata;
            ax(2).YLim = ([min(ydata), max(ydata)]);

            if 0
                frame = getframe(f);
                writeVideo(vid, frame);
            end
            pause(opts('Tp') - toc);
        end
        if 0; close(vid); end
    end
end

% filename: run_event_doppler.m
% author: Paul Adams
%
```

```
% Script to setup and run event_doppler

clear DopplerConfig;
%fullfile('..', 'reference', 'doppler_files', 'Off of Newton Exit 17.wav'), ...
dc = event_doppler('wav_file',...
        [], ...
    'debug_level', 1, ...
    'oversample_factor', 4, ...
    'dwell_period_ms', 200, ...
    'n_detection_dwell', 2, ...
    'windowing_function', 'taylor', ...
    'real_time', false, ...
    'channel_vector', 1);
```

```
% filename: event_doppler.m
% author: Paul Adams
%
function dc = event_doppler(varargin)
    % config
    args = containers.Map(varargin(1:2:end), varargin(2:2:end));
    dc = DopplerConfig(args);
    dc.wav.record(dc.n_dwell_detect*dc.Tp);

    % Loop until EOF
    eof = false;
    while ~eof
        tic;
        x = fetch_and_format(dc);
        [v, i, n0] = reduce_doppler(x, dc);
        update_state(v, i, n0, dc);

        % Check for EOF and enforce real-time
        eof = dc.i_samp >= dc.n_total - dc.n_dwell_detect*dc.n_samp; % within one dwell of eof

        show_state(dc, v, n0);

        margin = dc.Tp*dc.n_dwell_detect - toc;
        if args('real_time')
            pause(margin);
        else
            drawnow
        end
    end
end

function x = fetch_and_format(dc)
    if dc.is_file
        start_samp = dc.i_samp + 1;
        dc.i_samp = start_samp + dc.n_samp*dc.n_dwell_detect - 1;
        if dc.debug_level >= 2; fprintf('Fetch and Format... samples: %d to %d...', start_samp, dc.i_sa
        [v, ~] = audioread(dc.wav, [start_samp, dc.i_samp]);
    else
        while dc.wav.get('TotalSamples') < dc.n_samp*dc.n_dwell_detect; end
        v = dc.wav.getaudiodata();
        dc.wav.record(dc.n_dwell_detect*dc.Tp);
    end
    x = reshape(v(:, dc.i_chan)', dc.n_samp, dc.n_dwell_detect).'; % let each row have n_samp samples
end

function [v, i, n0] = reduce_doppler(x, dc)
    if dc.debug_level >= 2; fprintf('Doppler Process... \n'); end
    x_w = bsxfun(@times, x, dc.taper); % apply taper to each row
    X = fft(x_w, dc.n_filt, 2); % fft along rows
    %X = 20*log10(abs(X(:, 1:dc.max_filt)));
    X = 20*log10(abs(X(:, 1:dc.n_filt/2)));
    n0 = mean(X(:));
    [v, i] = max(mean(X, 1)); % take mean over rows, then select the max filter bin
    dc.ax.Children(2).YData = mean(X, 1);
    dc.ax.Children(1).XData = dc.v_mph(i);
    dc.ax.Children(1).YData = v;
end

function update_state(v, i, n0, dc)
    if isempty(dc.noise_est)
        dc.noise_est = n0; % initial value
    else
        % current noise est. is weight last est. + weighted current measurement
        dc.noise_est = dc.alpha_n*dc.noise_est + dc.beta_n*n0;
    end

    dc.state.SpeedEst = nan;
    if n0 > dc.noise_est + dc.pass_thresh
        if ~dc.state.Passing % if changing state, inc. counter
            dc.state.VehicleCount = dc.state.VehicleCount + 1;
        end
        dc.state.Passing = true;
    elseif v > dc.noise_est + dc.actv_thresh
        dc.state.Passing = false; % turn off Passing
        dc.state.Active = true;
        dc.state.SpeedEst = dc.v_mph(i);
    end
    if v <= dc.noise_est + dc.actv_thresh; dc.state.Active = false; end

    % update debug variables
    if dc.debug_level >= 1
        dc.i_dwell = dc.i_dwell + 1;
        dc.state.n_i(dc.i_dwell) = n0;
        dc.state.n_a(dc.i_dwell) = dc.noise_est;
        dc.state.v_mph(dc.i_dwell) = dc.state.SpeedEst;
    end
end

function show_state(dc, v, n0)
    fprintf('Time: %.2f\tnoise: %.2f\tpower: %.2f\tPeak: %.2f\tPassing: %d\tActive: %d\tSpeed: %0.1f\tVe
        dc.i_samp/dc.FS, dc.noise_est, n0, v, dc.state.Passing, dc.state.Active, round(dc.state.SpeedEst
end
```

Python Main Programs

```
#!/usr/bin/python2
# -*- coding: utf-8 -*-
# __file__ serv-alsa.py
# __author__ Paul Adams

import socket
import pyaudio
import zmq
import time

N_SAMP_BUFF = 8*1082 # samples in callback buffer
N_CHAN = 2
FS = 48000
pa = pyaudio.PyAudio()

# setup zmq
ctx = zmq.Context()
pub = ctx.socket(zmq.PUB)
tcp = "tcp://%s:5555" % socket.gethostbyname('thebes')
pub.bind(tcp)

def alsa_callback(data, frames, time, status):
    pub.send('%s %s;;;%s' % ('pcm_raw', 'time:%f' % (time['current_time']), data))
    return (data, pyaudio.paContinue)

class Alsa():
    def __init__(self):
        self.stream = pa.open(format=pyaudio.paInt32,
                rate=FS, input=True, channels=N_CHAN,
                frames_per_buffer=N_SAMP_BUFF,
                stream_callback=alsa_callback)

    def loop(self):
        while self.stream.is_active():
            time.sleep(0.1)

        # stop stream
        self.stream.stop_stream()
        self.stream.close()
        pa.terminate()

def main():
    a = Alsa()
    a.loop()

if __name__ == '__main__':
    main()
```

```
#!/usr/bin/python2
# -*- coding: utf-8 -*-
# __file__ serv-fmcw.py
# __author__ Paul Adams
# imports
import numpy as np
from scipy.signal import lfilter, butter
import socket
import pyaudio
import zmq
import time
import json

# constants
M2FT = 3.28084
C_LIGHT = 3e8
BW = 300e6
SYNC_CHAN = 0
SGNL_CHAN = 1
N_CHAN = 2
N_SAMP_BUFF_APPROX = 1000 # samples in callback buffer
N_FFT = 4096
FS = 48000
SUB_PORT = 5555
PUB_PORT = 5556
MAX_PERIOD_DELTA = 50

pa = pyaudio.PyAudio()

class Zmq():
    def __init__(self):
        # setup zmq
        self.ctx = zmq.Context()
        self.sub = self.ctx.socket(zmq.SUB)
        self.sub.setsockopt(zmq.SUBSCRIBE, 'pcm_raw')
        self.ip = socket.gethostbyname('thebes')
        tcp = "tcp://%s:%s" % (self.ip, SUB_PORT)
        self.sub.connect(tcp)
        print 'Listening on %s' % tcp

        self.pub = self.ctx.socket(zmq.PUB)
        tcp = "tcp://%s:%s" % (self.ip, PUB_PORT)
        self.pub.bind(tcp)
        print 'Publishing on %s' % tcp

class Queue():
    def __init__(self):
        self.buff_idx = 0
        self.ref = []
        self.sig = []
        self.raw = []
        self.n_buff =  1
        self.time = []

    def re_init(self):
        self.buff_idx = 0
        self.ref = []
        self.sig = []

    def fetch_format(self):
        # fetch format data
        data = z.sub.recv()
        idx = data[0:100].find(';;;')
        header = data[0:idx]
        self.time = float(header[header.find(':')+1:header.find(';;;')])
        #import pdb; pdb.set_trace()
        x = (np.fromstring(data[idx+3::], np.int32)).astype(np.float)/2**31
        self.sig = x[SGNL_CHAN::2]
        self.ref = x[SYNC_CHAN::2]
        debug_hook(self.ref, 'clock')
        debug_hook(self.sig, 'signal')

    def update_buff(self):
        self.buff_idx += 1
        self.fetch_format()

    def is_full(self):
        return self.buff_idx == self.n_buff

class Sync():
    def __init__(self):
        self.have_period = False
        self.period = []
        self.edges = {}
        self.T = []
        self.pulses ={}
        self.tail = np.array([0])
        self.head = []
        self.stable_period_count = 0

    def get_edges(self, q):
        dref = np.diff((q.ref > 0).astype(np.float))
        # find indices of rising edges
        self.edges['rise'] = np.where(dref == 1)[0]

        # find indices of falling edges
        self.edges['fall'] = np.where(dref == -1)[0]

    def align_edges(self, q):
        # make sure fall follows rise, save head
        head_idx = np.argmax(self.edges['fall'] > self.edges['rise'][0])
        self.edges['fall'] = self.edges['fall'][head_idx:-1]
        head_idx = self.edges['rise'][0] - 1

        # make sure each vector is equi-length
        if len(self.edges['rise']) > len(self.edges['fall']):
            self.edges['rise'] = self.edges['rise'][0:len(self.edges['fall'])]
        else:
            self.edges['fall'] = self.edges['fall'][0:len(self.edges['rise'])]

        # try stitch previous tail to current head
        self.head = q.ref[0:head_idx]
        self.stitch(q)
        self.tail = q.ref[self.edges['fall'][-1] + 1:-1]

    def check_period(self):
        if self.period:
            prev_period = self.period
        else:
            prev_period = 0

        self.period = np.floor(np.mean(self.edges['fall'] - self.edges['rise']))
        rez = np.abs(self.period - prev_period)

        if rez < MAX_PERIOD_DELTA:
            self.stable_period_count += 1
            if not self.have_period:
                print 'pulse period acquired --> %d samples' % (self.period)
                self.have_period = True
                self.T = self.period*FS

        elif self.have_period:
            self.stable_period_count = 0
            self.have_period = False
            self.period = []
            print 'pulse period lost. residual --> %d samples' % (rez)

    def stitch(self, q):
        if self.tail.any():
            x = np.hstack((self.tail, self.head))

            # sync clock signal
            dx = np.diff((x > 0).astype(np.float))

            # find indices of rising edges
            rise = np.where(dx == 1)[0].tolist()

            while rise:
                r = rise.pop(0)
                if self.period.__class__ is list:
                    pass
                else:
                    ts = float(r)/self.period*(q.time - 0.180)
                    self.pulses[ts] = q.sig[r:r+self.period]

    # given a buffer of audio frames, find the pulses within the clock signal and extract received chir
    def extract_pulses(self, sig):
        rises = self.edges['rise'].tolist()
        while rises:
            idx = rises.pop(0)
            ts = float(idx)/self.period*q.time
            self.pulses[ts] = sig[idx:idx + self.period]

class Processor():
    def __init__(self):
        self.do_cancel = True
        self.n_fft = N_FFT
        self.x = 0
        self.cfar_filt = [1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1]
        self.alpha = 0.6 # cfar scalar
        self.detects = []
        self.range_rez = C_LIGHT/(2*BW)
        self.n_samp = 0
        self.report_dict = {}
        self.range_lu = np.zeros((1, 1))
        self.ranges = []
        self.lpf_b = np.fromfile('/home/paul/ee542/lpf.npy')
        self.prior = []
        self.string = ''
        self.t0 = 0

    def lowpass(self):
        self.x = lfilter(self.lpf_b, 1, self.x, axis=1)

    def format(self, pulses):
        #import pdb; pdb.set_trace()
        self.n_samp = min([len(p) for p in pulses.values()])
        self.x = np.zeros((len(pulses) , self.n_samp))
        k = pulses.keys()
        k.sort()
        for i, j in enumerate(k):
            self.x[i, :] = pulses[j][0:self.n_samp]

        debug_hook(self.x, 'raw')

    def averager(self):
        self.x = np.mean(self.x, axis=0)
        debug_hook(self.x, 'avg')

    def canceller(self):
        if len(self.prior) > 0:
            nsamp = min([len(self.prior), self.x.shape[1]])
        else:
            nsamp = self.x.shape[1]
            self.prior = np.zeros(nsamp)

        self.x = self.x[:, 0:nsamp]
        y = self.x.copy()
        self.x[0, :] -= self.prior[0:nsamp]
        for i in range(1, self.x.shape[0]):
            self.x[i, :] -= y[i-1, :]

        self.prior = self.x[i, :]

    def filter(self):
        self.x = np.abs(np.fft.ifft(self.x, n=self.n_fft)[:, 0:self.n_fft/2])**2
        debug_hook(self.x, 'filt')

    def detect(self):
        #cfar = signal.lfilter(self.cfar_filt, 1, self.x)
        self.detects = [np.argmax(self.x[50:-1]) + 50]

    def transform(self):
        self.ranges = []
        if not self.range_lu.any():
            max_range = self.range_rez*self.n_samp/2
            self.range_lu = np.linspace(0, max_range, self.n_fft/2)

        if len(self.detects) > 0:
            for d in self.detects:
                self.ranges.append(self.range_lu[d])

    def report(self):
        self.report_dict = {}
        for i in range(len(self.detects)):
            ts = time.time()
            self.report_dict[ts] = {}
            self.report_dict[ts]['gate'] = self.detects[i]
            self.report_dict[ts]['range'] = self.ranges[i]

        self.report_str = json.dumps(self.report_dict)

    def print_debug(self, doit):
        dt = q.time - self.t0
        self.t0 = q.time
        self.string += 'time: %.3f -> dt: %.3f ms -> pulses: %d -> samp: %d stable -> %d -> detect: %d r
        if doit:
            print self.string,
            self.string = ''

    def process_pulses(self, pulses):
        self.reshape(pulses)
        #self.lowpass()
        #self.canceller()
        self.filter()
        self.averager()
        self.detect()
        self.transform()
        self.report()

# init objects
q = Queue()
s = Sync()
p = Processor()
z = Zmq()

def debug_hook(data, topic):
    n_row = data.shape[0]
    z.pub.send('%s n_row: %s;;;%s' % (topic, str(n_row), data.tostring()))

def main():
    print 'Queue and Sync initzd... Entering loop now... '
    i = 0
    while True:
        i += 1
        q.update_buff()

        if q.is_full():
            s.get_edges(q)
            s.align_edges(q)
            s.check_period()

            if s.have_period:
                s.extract_pulses(q.sig)
                p.process_pulses(s.pulses)
                z.pub.send('%s %s' % ('report', p.report_str))
                p.print_debug(i % 5 == 0)
                s.pulses = {} # reset pulses
            else:
                pass

            q.re_init()


if __name__ == '__main__':
    main()
```

Shell Scripts and systemd Modules

```
#!/bin/bash
# file: eth0-startup.sh
# author: Paul Adams

carrier_state=$(</sys/class/net/eth0/carrier)
not_up=1

while [ "$not_up" -eq 1 ]; do
    if [ "$carrier_state" -eq 1 ]; then
        ip addr add 192.168.2.108/255.255.255.0 dev eth0
        ip link set eth0 up
        not_up=0
        echo "success"
    else
        echo "ethernet not connected foo"
        sleep 10
    fi
done

echo "exit program"
```

```
#!/bin/bash
# file: kill-all.sh
# author: Paul Adams

pgrep python2 | xargs kill -9
pgrep jackd | xargs kill -9
```

```
#!/bin/bash
# file: start-all.sh
# author: Paul Adams

/usr/bin/jackd -P70 -p16 -t2000 -d alsa -d hw:0 -p 128 -n 3 -r 48000 -s&
echo 'started jackd...'
sleep 1
/usr/bin/python2 /home/paul/ee542/serv-alsa.py&
echo 'started serv-alsa.py...'
sleep 1
/usr/bin/python2 /home/paul/ee542/serv-fmcw.py
```

```
[Unit]
Description=my jackd management wrapper
Before=serv-alsa.service serv-fmcw.service

[Service]
ExecStart=/usr/bin/jackd -P70 -p16 -t2000 -d alsa -d hw:0 -p 128 -n 3 -r 48000 -s
Type=simple
Restart=always
RestartSec=15s

[Install]
WantedBy=multi-user.target
```

```
[Unit]
```

```
Description=my fmcw server wrapper
After=serv-alsa.service jackd.service
Requires=jackd.service serv-alsa.service

[Service]
Environment=PYTHONBUFFERED=true
ExecStart=/usr/bin/python2 /home/paul/ee542/serv-fmcw.py
Type=simple
Restart=always
RestartSec=15s

[Install]
WantedBy=multi-user.target
```

```
[Unit]
Description=my alsa server wrapper
After=jackd.service
Before=serv-fmcw.service
Requires=jackd.service

[Service]
Environment=PYTHONBUFFERED=true
ExecStart=/usr/bin/python2 /home/paul/ee542/serv-alsa.py
Type=simple
Restart=always
RestartSec=15s

[Install]
WantedBy=multi-user.target
```