WISPR

Passive Acoustic Monitoring

File Formats

Hardware Version: WISPR V3.0

Embedded Ocean Systems

Manual Version: 1.0
Aug 2024

Contents

1. Configuration Files

System configuration is stored on each SD card in a text file called **wispr.txt**. At startup the system reads the configuration file to set the operating parameters. Operating parameters can be modified while the system is running using the command interface. For example, the ADC sampling rate can be changed using the ADC command. On controlled shutdown (using the EXI command) the system saves the current configuration to SD card to preserve the current settings for the next startup. If the system is powered down unexpectantly, the current configuration will be lost and the last saved configuration will be used on the next startup.

Config File Format:

The first line of the ascii configuration file must contain the line '`WISPR 3.0 configuration`', which indicates the version number and type of file. System variables follow as (`variable: value`) pairs on each line for the configuration file. An example of a typical configuration file is as follows:

```
WISPR 3.0 configuration
sensor_id: EOS
platform_id: OSU
adc_sample_size: 3
adc_sampling_rate: 200000
adc_gain: 0
adc_decimation: 4
adc_timestamp: 0
secs_per_file: 60
max_file_size: 102400
debug_level: 1
fft_size: 512
fft_overlap: 0
fft_window_type: 0
rtc_aging: 0
rtc_freq: 32767
msg_baud: 9600
msg_termination: 13
msg_handshake: 0
```

Configuration variables are:

Variable	Description	Value
sensor_id	User defined sensor ID	Max 16 chars
platform_id	User defined platform ID	Max 16 chars
adc_sample_size	ADC data sample size in bytes	2 or 3
adc_sampling_rate	ADC sampling rate in Hz	1000 to 300000
adc_gain	Preamp gain (0,6,12,18 dB)	0, 1, 2 or 3
adc_decimation	ADC decimation factor	4, 8, 16
adc_timestamp	ADC buffer timestamp size	0 or 6 bytes
secs_per_file	Duration of data file in secs	10 to 3600
max_file_size	Max file size in blocks	
debug_level	User defined debug print level	0, 1, 2, or 3
fft_size	PSD FFT size	128, 512, 1024
fft_overlap	PSD FFT Overlap	
fft_window_type	PSD FFT Window type	0, 1, or 2
rtc_aging	RTC clock aging coefficient	-128 to 128
rtc_freq	RTC clock frequency in Hz	32768
msg_baud	Command port baud rate	9600
msg_termination	Command termination char	10(NL) or 13(CR)

2. Data File Format

ADC data file is stored on the SD card in files of specified duration. SD cards must be formatted as ExFAT. The file name specifies the start date and time of the data in the file. For example, a file named *WISPR_210218_140500.dat* contains data starting at time 14:05 (Hour 14, minute 05, second 00) on Feb 21, 2021. Data files have a .dat file name extension.

Each data file contains a 512 byte ascii file header followed a fixed number of data blocks. Data blocks are vectors of binary integers stored in IEEE little-endian format.

The data file consisting of N data buffers has format:

Description	Size in bytes	File offset in bytes
File Header	512	0
Data Buffer 1	buffer_size	512
Data Buffer 2	buffer_size	buffer_size + 512
Data Buffer 3	buffer_size	2*buffer_size + 512
…	buffer_size	…
Data Buffer N	buffer_size	(N-1)*buffer size + 512

2.1 File Header:

The ascii file header defines the parameters required to read and interpret the data file. A typical file header will look like this:

```
% WISPR 3.0
sensor_id = 'EOS';
platform_id = 'EOS';
second = 1725485740.138920;
file_size = 70291;
buffer_size = 23040;
samples_per_buffer = 7678;
sample_size = 3;
sampling_rate = 200000;
adc_type = 'LTC2512';
adc_vref = 5.000000;
adc_df = 4;
gain = 0;
timestamp = 0;
```

The field 'time' is the data start time in seconds Linux epoch time. The fields 'platform_id' and 'sensor_id' are user defined fields taken from the configuration file.

The variable 'file_size' is the number of 512 byte data blocks in the file. The total data file size is always a multiple of 512 bytes blocks.

Each data buffer is of length 'buffer_size' bytes, which is always a multiple of 512 bytes blocks. The total number of data buffers in the file is:

```
N = file_size*512 / buffer_size;
```

Each data buffer contains a fixed number of samples defined by the header variable 'samples_per_buffer'. Each sample is of fixed size in bytes, defined by the variable 'sample_size'. The sample size can be 2 or 3, corresponding to 16 or 24 bit signed integers.

2.2 Data Buffer Format

A data buffer with `M = samples_per_buffer` samples has format:

Description	Size in bytes	File offset in bytes
Data Sample 1	sample_size	0
Data Sample 2	sample_size	sample_size
Data Sample 3	sample_size	2*sample_size
…	sample_size	…
Data Sample M	sample_size	(M-1)*sample_size
Buffer Timestamp	timestamp	M*sample_size

Samples are either 16 or 24 bit signed integers sampled by the ADC at the specified sampling frequency (sampling_rate) in Hz. At the end of each sequence of samples is an optional buffer timestamp, as described in the next section.

2.3 Buffer Timestamp

Each data buffer has an optional buffer timestamp, which specifies the time that the buffer finished and the start of the next buffer. There is no time gap between buffers. The data file header variable 'timestamp' defines the size in bytes of the timestamp value. If 'timestamp = 0', then there is no timestamp. If timestamp = 6, then a 6 bytes timestamp is appended to the end of each data buffer. If `timestamp = 8`, then an 8 bytes timestamp is appended to the end of each data buffer.

The 6 byte timestamp contains 2 leading bytes (a 16 bit integer) representing the seconds from the start of the data file and 4 bytes (a 32 bit integer) representing microsecond from the start of the second. The timestamp value is the time at the end of the data buffer, therefore the start time of the beginning of the next buffer.

```
buffer_duration = samples_per_buffer / sampling_rate;
seconds = fread(fp, 1, 'uint16' );
useconds = fread(fp, 1, 'uint32' );
buffer_start_time = t0 + (second + (0.000001 * useconds)) - buffer_du-
ration;
```

where `t0` is the epoch time for the start of the data file.

The 8 byte timestamp contains 4 leading bytes (a 32 bit unsigned integer) representing the epoch seconds and 4 bytes (a 32 bit unsigned integer) representing microsecond from the start of the second.

```
buffer_duration = samples_per_buffer / sampling_rate;
seconds = fread(fp, 1, 'uint32' );
useconds = fread(fp, 1, 'uint32' );
buffer_start_time = (second + (0.000001 * useconds)) - buffer_duration;
```

2.4 Matlab Script to Read Data Buffers

```
function [hdr, data, time, stamp] = read_wispr_file(name, first, last)
%
% matlab script to read wispr data from a dat file
%
% A data file consists of an ascii file header followed by binary data
% buffers. The ascii header is formatted as matlab expressions.
% The binary data words are formatted as signed 16 or 24 bit integers.
%
% The data file format is:
% - 512 byte ascii header.
% - adc buffer 1
% - adc buffer 2
% ...
% - adc buffer N
% where N is the number of adc buffers per file
%
% The number of adc buffers is defined as
% number_buffers = file_size*512 / buffer_size;
%
% The total data file size is always a multiple of 512 bytes blocks.
% The variable 'file_size' is the number of 512 blocks in the file.
%
% Each adc buffer is of length 'buffer_size' bytes.
% The adc buffer is always a multiple of 512 bytes blocks
% (32 blocks in most cases).
% Each adc buffer contains a fixed number of sample (samples_per_buffer).
% Each sample is of fixed size in bytes (sample_size).
% The sample size can be 2 or 3.
% If 3 byte samples are used, there will be extra bytes of padded
% at the end of each adc buffer.
% The number of bytes of padding is defined as:
% padding_per_buffer = buffer_size - (samples_per_buffer * sample_size);
%
% File header info is returned in hdr
% Data and time are returned as a matrix of
% size [samples_per_buffer, (last - first)]
%

fp = fopen( name, 'r', 'ieee-le' );

% back compatability
instrument_id = [];
location_id = [];
sensor_id = [];
platform_id = [];
timestamp = 0;
time = [];
second = 0;
usec = 0;
adc_type = 'LTC2512';
adc_df = [];
decimation = [];
channels = 1;

% read and eval the ascii header lines
for n = 1:18
    str = fgets(fp); % read full line
    % read ascii lines until a null is found,
    % so header buffer must be null terminated
    if( str(1) == 0 )
        break;
    end
    %fprintf('%s', str);
    eval(str);
end

if(sample_size == 2)
    q = adc_vref/32767.0;  % 16 bit scaling to volts
    fmt = 'int16';
elseif(sample_size == 3)
    q = adc_vref/8388608.0;  % 24 bit scaling to volts
    fmt = 'bit24';
elseif(sample_size == 4)
    q = 1.0;
    fmt = 'int32';
end

% The number of adc buffers in the file
number_buffers = file_size*512 / buffer_size;

% The number of bytes of padding after each adc buffer, if any
padding_per_buffer =
   buffer_size - timestamp - (samples_per_buffer * sample_size);

% buffer duration in secs
dt = 1.0 / sampling_rate;
duration = samples_per_buffer * dt;

% fill the header structure
hdr.time = time; % back compatability
hdr.second = second;
hdr.usec = usec;
hdr.sensor_id = sensor_id;
hdr.platform_id = platform_id;
hdr.instrument_id = instrument_id;
hdr.location_id = location_id;
hdr.file_size = file_size;
hdr.buffer_size = buffer_size;
hdr.samples_per_buffer = samples_per_buffer;
hdr.sample_size = sample_size;
hdr.sampling_rate = sampling_rate;
hdr.channels = channels;
hdr.gain = gain;
hdr.adc_type = adc_type;
hdr.adc_vref = adc_vref;
hdr.adc_df = adc_df;
hdr.decimation = decimation;
hdr.adc_timestamp = timestamp;
hdr.number_buffers = hdr.file_size * 512 / hdr.buffer_size;
hdr.buffer_duration = samples_per_buffer * dt;
hdr.file_duration = hdr.buffer_duration * hdr.number_buffers;

data = [];
time = [];
stamp = [];

% check to make sure first and last are valid
if( first >= number_buffers )
    return;
end

% if first == 0 then just read the header
if(first <= 0)
    return;
end

% read all buffers if last is zero
if(last == 0)
    last = number_buffers;
end

% past end of file
if( last > number_buffers )
    last = number_buffers;
end

% seek to the start of data
% header is always 512 bytes
fseek(fp, 512 + (hdr.buffer_size * (first-1)), -1);

% start time
t0 = (second + usec * 0.000001) + (first - 1)*duration;

% number of buffer to read and concatenate
num_bufs = last - first + 1;

for n = 1:num_bufs

    % read a data buffer
    raw = fread(fp, samples_per_buffer, fmt ); % data block
    if( length(raw) ~= samples_per_buffer )
        break;
    end

    % read buffer timestamp, which is time the buffer finished,
    % so remove the buffer duration
    if( timestamp == 8 )
        s = fread(fp, 1, 'uint32' ); % epoch second
        us = fread(fp, 1, 'uint32' );
        stamp(n) = (s + 0.000001 * us) - duration;
    elseif( timestamp == 6 )
        s = fread(fp, 1, 'uint16' ); % secs from start of file
        us = fread(fp, 1, 'uint32' );
        stamp(n) = ((second + s) + 0.000001 * us) - duration;
    else
        stamp(n) = t0 + (n-1) * duration;
    end

    % read padding, if any
    junk = fread(fp, padding_per_buffer, 'char');

    % add raw data buffer as a column to data matrix
    data(:,n) = double(raw)*q;

    % add a time column to the time matrix
    time(:,n) = stamp(n) + dt*(0:(samples_per_buffer-1));

end

fclose(fp);

return;
```