Dire Wolf Developer Notes

WB2OSZ, Rough Draft 1a

Purpose

Some might wish to add their own enhancements. Others might want to adapt the code to run in some other environment, or just might be curious about how it works.

This is a collection of notes intended for anyone that wants to understand how the Dire Wolf application fits together, the rationale for a few decisions, and things to look out for.

Source Directory Structure

Directories:

- **cmake** - Stuff for cmake
- **conf** - Sample configuration files for direwolf and udev
- **data** - Codes for symbols and APxxxx vendor in AX.25 destination field
- **debian -** Debian packaging.
- **doc** – User Documentation.
- **external** - Third Party code.
    - **geotranz** – Conversions between latitude/longitude and other coordinate systems such as UTM. It is a subset of geotranz from https://github.com/smanders/geotranz These are compiled into **geotranz.a**.
    - **hidapi** - For Windows PTT using GPIO of CM108/CM119. We only need a small fraction of this. Trim it down or replace someday.
    - **misc** – Own copy of some string functions that are missing for some platforms. See README file for more details. These are compiled into **misc.a**.
    - **regex** – Regular expression library from http://gnuwin32.sourceforge.net/packages/regex.htm. Used only for the Windows version. For Linux, it is part of the standard C run time library. These are compiled into **regex.a**.
- **man** – Linux "man" pages.
- **rpm** - for RPM packaging.
- **scripts** - Sample scripts for automatic startup and interface to speech synthesizer.
    - **telemetry-toolkit** – Scripts and sample configurations for generating APRS Telemetry.
- **src** - Source code.
- systemd - Configuration to run as a service. Need a README explaining what to do.

Files:

- **CHANGES.md** - Revision history.
- **README.md -** Quick overview of project

Source Files

Most of the source files named *.c have a corresponding *.h which contains the interface to the functions.

Name	Purpose
aclients.c	Test program for comparing reception from multiple TNCs, side by side.
ais.c	Processing received AIS transmissions and converting to NMEA sentence representation.
aprs_tt.c	First half of APRStt gateway. Parse the tone sequences and extract meaning from them.
atest.c	Test fixture for using the demodulator with a file rather than live audio.
audio.c	Interface to the “sound card” audio devices for Linux.
audio_portaudio.c	Interface to the “sound card” audio devices for Mac OSX
audio_stats.c	Print audio input stream statistics for troubleshooting.
audio_win.c	Interface to the “sound card” audio devices for MS Windows.
ax25_link.c	AX.25 v2.2 Data Link State Machine for connected mode packet radio.
ax25_pad.c	Builds packet objects from HDLC frames or text representation. Extracts information, modifies them, prepares for printing and transmission.
ax25_pad2.c	The original ax25_pad.c was written with APRS in mind. It handles UI frames and transparency for a KISS TNC. Here we add new functions that can handle the more general cases of AX.25 frames.
beacon.c	Transmit information periodically, either at a fixed rate or based on movement.
cdigipeater.c	Digipeater for traditional connected mode packet. APRS digipeater is in digipeater.c
config.c	Parse the configuration file and put into a suitable form for later use.
decode_aprs.c	Decode information part of an APRS frame. Print in human readable format and point out problems.
dedupe.c	Avoid transmitting duplicate packets which are too close together.
demod.c	Common entry point for multiple types of demodulators. This calls functions in demod_9600.c, demod_afsk.c , or demod_psk.c, depending on the modem type selected.
demod_9600.c	Demodulator for scrambled baseband encoding used by early hardware designs by G3RUH and K9NG.
demod_afsk.c	Demodulator for Audio Frequency Shift Keying (AFSK).
demod_psk.c	Demodulator for Phase Shift Keying (PSK).
digipeater.c	Logic for APRS digipeater. UI frames only. See cdigipeater.c for traditional connected mode packet.
direwolf.c	Main program.
dlq.c	Received frame queue.
dsp.c	Generate the filters used by the demodulators.
dtime_now.c	Returns current time with more resolution than time() so we can measure short intervals of elapsed time.
dtmf.c	Decoder for DTMF, commonly known as "touch tones."
dwgps.c	Interface for obtaining location from GPS. This is independent of whether

	we are using gpsd or reading NMEA sentences from a serial port.
dwgpsd.c	Obtain GPS information using gpsd.
dwgpsnmea.c	Read directly from GPS over serial port and pars the NMEA sentences.
encode_aprs.c	Construct APRS packets (e.g. Position or Object Report) from various parameters.
fc s_calc.c	Calculate the FCS for an AX.25 frame.
fx25_encode.c	FX.25
fx25_extract.c	FX.25
fx25_init.c	FX.25
fx25_rec.c	FX.25
fx25_send.c	FX.25
gen_packets.c	Test program for generating AX.25 frames. Given messages are converted to audio and written to a .WAV type audio file.
gen_tone.c	Convert bits to AFSK for writing to .WAV sound file or a sound device.
hdlc_rec.c	Extract HDLC frames from a stream of bits.
hdlc_rec2.c	Extract HDLC frame from a block of bits after someone else has done the work of pulling it out from between the special "flag" sequences.
hdlc_send.c	Convert HDLC frames to a stream of bits.
igate.c	IGate client. Establish connection with a tier 2 IGate server and relay packets between RF and Internet.
Il2p_codec.c	IL2P
Il2p_header.c	IL2P
Il2p_init.c	IL2P
Il2p_payload.c	IL2P
Il2p_rec.c	IL2P
Il2p_scramble.c	IL2P
Il2p_send.c	IL2P
Il2p_test.c	IL2P
kiss.c	KISS TNC via pseudo terminal. Linux only.
kiss_frame.c	Common code used by Serial port and network versions of KISS protocol.
kissnet.c	KISS TNC over TCP.
Kissserial.c	KISS TNC over serial port.
latlong.c	Various functions for dealing with latitude and longitude.
ll2utm.c	Utility for Latitude / Longitude to UTM conversion.
log.c	Save received packets to a log file.
log2gpx.c	Separate application for converting log file to GPX format.
mheard.c	Maintain a list of all stations heard.
morse.c	Generate audio for Morse Code.
multi_modem.c	Use multiple modems in parallel to increase chances of decoding less than ideal signals. Pick the best one when there are duplicates.
nmea.c	Send NMEA waypoint sentences to GPS display or mapping application.
pfilter.c	Packet filtering for digipeating or IGate.
ptt.c	Activate the output control lines for push to talk (PTT) and other purposes.
rdq.c	Obsolete. Should be removed.
recv.c	Process audio input for receiving.
redecode.c	Obsolete. Should be removed.

rrbb.c	Raw Received Bit Buffer. An array of bits used to hold data out of the demodulator before feeding it into the HLDC decoding.
serial_port.c	Interface to serial port, hiding operating system differences.
server.c	Provide service to other applications via "AGW TCPIP Socket Interface".
symbols.c	Functions related to the APRS symbols.
telemetry.c	Decode APRS telemetry information. Point out where it violates the protocol spec and other applications might not interpret it properly.
textcolor.c	All text output, including color coding for different types of information.
tq.c	Transmit queue - hold packets for transmission until the channel is clear.
tt_text.c	Translate between text and touch tone representation.
tt_user.c	Second half of APRStt gateway. This maintains a list of recently heard APRStt users and prepares "object" format packets for transmission.
ttcalc.c	Simple Touch Tone to Speech calculator. Demonstration of how Dire Wolf can be used as a DTMF / Speech interface for ham radio applications.
utm2ll.c	Utility for UTM to Latitude / Longitude conversion.
walk96.c	Quick hack to read GPS location and send very frequent position reports frames to a KISS TNC.
waypoint.c	Send NMEA waypoint sentences to GPS display or mapping application.
xid.c	Encode and decode the info field of XID frames.
xmit.c	Transmit queued up packets when channel is clear.

Important Constants

Name	Location	Value	Description
MAX_ADEVS	direwolf.h	3	Maximum number of audio devices. I recommend using a separate audio interface for each radio to eliminate any interaction between them. For example, if you had 4 radios, this could be increased to 4 and run each in mono, rather than 2 devices in stereo.
MAX_CHANS	direwolf.h	MAX_ADEVS * 2	Maximum number of radio channels when all audio devices are operating in stereo.
AX25_MAX_ADDRS	ax25_pad.h	10	Maximum number of addresses in an AX.25 frame: destination, source, up to 8 digipeaters. Derived from AX.25 standard. **Do not change.**
AX25_MAX_INFO_LEN	ax25_pad.h	2048	Maximum length of the information part of an AX.25 frame. This was made large enough so it could hold an Ethernet frame. For APRS and, many other situations, you could reduce it to 256 if you are concerned about memory usage.
AX25_MIN_PACKET_LEN	ax25_pad.h	2 * 7 + 1	Minimum number of octets in an AX.25 frame, excluding the FCS. Two addresses and one control octet.
AX25_MAX_PACKET_LEN	ax25_pad.h	AX25_MAX_ADDRS * 7 + 2 + 3 + AX25_MAX_INFO_LEN	Maximum number of octets in an AX.25 frame, excluding the FCS. It is possible to have up to 2 control octets and 3 protocol octets.
To be continued…			

Major Data Structures

Packet Object (ax25_pad.c)

The details of the AX.25 frame format are pretty much all hidden here. Everyone else has an abstracted view. e.g. Get the 3rd digipeater address, decrement the remaining hop count, mark it as being used.

New instances can be constructed from received HDLC frames or from text in the standard monitoring format. Functions are provided to extract information and modify contents. Packets can also be prepared for transmission as HDLC frames or printed in standard monitoring format.

This representation is used everywhere as a packet is received, filtered, digipeated, transmitted, IGated, and printed.

Received Frame Queue (dlq.c)

When valid HDLC frames are received, they are converted to packet objects and placed here for later processing. Note that multiple threads can be feeding this when we have multiple audio devices.

Beacons can also be directed here to simulate reception.

Transmit Frame Queue (tq.c)

Packet objects are placed here while they wait for a clear channel so they can be transmitted.

This queue is fed by digipeating, IS to RF IGating, beacons, and of course attached applications.

Raw Received Bit Buffer (rrbb.c)

Rather than processing the HDLC stream a bit at a time, everything between two “flag” patterns is saved before processing. This what done for the “FIX_BITS” experiment.

Flow of Control

Dire Wolf makes use of "threads" for many of the different activities it performs. This allows them to proceed independently, and often at the same time on a multi-core system.

Main program

The general flow looks like this:

main (in direwolf.c):

- Read the configuration file.
- audio_init() -- opens audio devices.
- various other *_init() -- some create threads for processing.
- recv_init()
- recv_process() -- does not return

recv_init (recv.c)

- Start up a separate thread for each audio device.

    When a good frame is received it is appended to the receive queue. Received frames are removed from that queue and processed one at a time so we don't need to worry about later processing being reentrant.

recv_process (recv.c)

- This simply waits for something to show up in the receive packet queue (dlq.c) and calls app_process_rec_frame for each.

app_process_rec_frame (direwolf.c)

- Print station heard (direct or digipeater) and audio level information.
- Optional hex dump for debugging.
- Print frame in standard monitoring format.
- Decode APRS from information part of frame and print in human readable format.
- Write to log file.
- Send to any attached applications.
- If tones, send to APRStt processing.
- Send to IGate server if connected.
- Digipeat if appropriate.

Receive audio thread

There are 3 different audio interface implementations (audio.c, audio_win.c, audio_portaudio.c) depending on the platform. Each audio device has its own thread so they operate independently. On a multi-core system, each could be running simultaneously on different CPUs.

Received audio samples are passed to … describe … HDLC … fix up … queue …

Transmit thread

Loop (xmit.c) waits until there is something in the transmit queue and the corresponding channel is not busy . The frame is converted to audio (audio.c, audio_win.c, or audio_portaudio.c)

Push to Talk control (ptt.c) can use a serial ports, a parallel printer port, GPIO pins, or "hamlib" to activate the transmitter.

Beacon thread

Wakes up periodically (beacon.c), constructs Position or Object packets (encode_aprs.c), and appends them to the transmit queue.

Tracker Beacons use the GPS position (dwgps.c) and timing can be based on movement.

Client interface threads

Each client application connection, either KISS serial port, AGWPE socket (server.c), or KISS socket, has its own thread to listen for anything from the attached client application. In most cases this is a frame which is placed into the transmit queue.

IGate thread

xxxxxxxx

GPS read thread

This does a blocking read on data coming from either a serial port (dwgpsnema.c) or gpsd (dwgpsd.c). The current location is put in a memory location where it is read by dwgps_read (in dwgps.c) when needed.

Cmake files

TBD

Targets

- (default)

  Builds direwolf main application and various utilities.

- **tocalls-symbols**

  Get latest versions of

  - http://www.aprs.org/aprs11/tocalls.txt
  - http://www.aprs.org/symbols/symbols-new.txt
  - http://www.aprs.org/symbols/symbolsX.txt

    N.B. These are no longer being updated. Need new strategy.

- **test**

  Unit tests for regression errors.

- **install** (Linux only, must run as root)

  Install into /usr/local/…

- **clean**

  Remove results of compiling and linking.

- **dist-win** (Windows only)

  Bundle up Windows binary distribution as zip file.

Compiler options for code optimization

Most of the CPU time is consumed by the demodulator spinning around in little loops multiplying and adding arrays of numbers for each audio sample.

The Intel "SSE" instructions, introduced in 1999 with the Pentium III series, can speed this up considerably. SSE2 instructions, added in 2000, don't seem to offer any advantage.

When using x86, 32 bit, be sure we enable use of SSE instructions.

When building for a 64-bit target, the availability of SSE and other later vector instructions is implied.

Compiler options, Windows

I'm currently using **MinGW** version 7.4.0. There was a report of direwolf crashing due to a memory alignment error if using MinGW 4.8.1-4 with the –Ofast option. I think the problematic part has since been removed from the source code. There have been no further reports of the issue.

Most people will be using the pre-built version for Windows. Some are still using Windows XP and very old computers so I wanted to maximize compatibility with older systems.

By default, code is generated for the i686 processor.

A minimum of Windows XP is required due to some of the system features being used. XP requires a Pentium processor or later. So it should be safe to assume we have a Pentium or later.

The benchmark was running the "atest" application with Track 2 of the WA8LMF TNC Test CD. Here are the run times using different compiler options.

Seconds	Compiler options	Comments
119.8	-O2	Used for version 0.8
92.1	-O3	
88.7	-Ofast	Should be same as -O3 -ffastmath
87.5	-Ofast -march=pentium	
74.1	-Ofast -msse	
72.2	-Ofast -march=pentium -msse	
62.0	-Ofast -march=pentium3	pentium3 implies –msse so it is redundant.
61.9	-Ofast -march=pentium3 -msse	

For version 0.9, a Pentium 3 or equivalent is now the minimum required for the prebuilt Windows distribution. If you insist on using a computer from the previous century, you can compile this yourself with different options.

Compiler options, Linux x86, 32 bit

gcc 4.6.3 running on Ubuntu 12.04.05.

Intel(R) Celeron(R) CPU 2.53GHz. Appears to have only 32 bit instructions. Probably from around 2004 or 2005.

When gcc is generating code for a 32 bit x86 target, it assumes the ancient i386 processor. This is good for portability but bad for performance. The code can run considerably faster by taking advantage of the SSE instructions available in the Pentium 3 or later.

Seconds	Compiler options	Comments
524		No optimization.
183	-O2	
182	-O3	
183	-O3 -ffast-math	Should be same as -Ofast
184	-Ofast	
189	-O3 -ffast-math -march=pentium	
122	-O3 -ffast-math -msse	Big improvement with sse
122	-O3 -ffast-math -march=pentium -msse	
121	-O3 -ffast-math -march=pentium3	This implies –msse
120	-O3 -ffast-math -march=native	Note that "-march=native" is essentially the same as "-march=pentium3."

Compiler options, Linux x86, 64 bit

gcc 4.8.2 running on Ubuntu 14.04.1.

Intel Core 2 Duo from around 2007 or 2008.

64 bit target implies that we have SSE and probably even better vector instructions.

Seconds	Compiler options	Comments
245		
75	-O1	
72	-O2	
71	-O3	
73	-O3 -march=native	
42	-O3 -ffast-math	Big improvement with -ffast-math
42	-Ofast	See note below.
40	-O3 -ffast-math -march=native	

Note that "-Ofast" is a newer option roughly equivalent to "-O3 -ffast-math". I use the longer form because it is compatible with older compilers.

Why don't I don't have "-march=native?" Older compilers don't recognize "native" as one of the valid options. One article said it was added with gcc 4.2 but I haven't verified that.

The Makefile tests whether the compiler recognizes " –ffastmath" and adds it if so.

Compiler options, ARM, Raspberry Pi, models A, B

Raspberry Pi (before RPi model 2), ARM11 (ARMv6 + VFP2)

gcc (Debian 4.6.3-14+rpi1) 4.6.3

Seconds	Compiler options	Comments
892	-O3	
887	-O3 -ffast-math	
n/a	-O3 -ffast-math -march=native	error: bad value for -march switch
887	-O3 -ffast-math -mfpu=vfp	
890	-O3 -ffast-math -march=armv6zk -mcpu=arm1176jzf-s -mfloat-abi=hard -mfpu=vfp	

The compiler, supplied with Raspbian Wheezy, is configured with these options which are good for the pre version 2 models.

--with-arch=armv6 --with-fpu=vfp --with-float=hard

Run "gcc --help -v 2" and look near the end.

Compiler options, ARM, Raspberry Pi, model 2

Besides the higher clock speed (900 vs. 700 MHz), the Raspberry Pi 2 has the NEON instruction set which should make things considerably faster. Various forum discussions had different suggestions for the best options to use.

Also Raspbian Wheezy and same compiler version as above.

Seconds	Compiler options	Comments
426	-O3 -ffast-math	Already more than twice as fast
429	-O3 -mfpu=neon	
419	-O3 -mfpu=neon -funsafe-math-optimizations	
412	-O3 -ffast-math -mfpu=neon	Ended up using this.
413	-O3 -ffast-math -mfpu=neon-vfpv4	
430	-O3 -ffast-math -mfpu=neon-vfpv4 -march=armv7-a	
412	-O3 -ffast-math -mfpu=neon-vfpv4 -mtune=arm7	
410	-O3 -ffast-math -mfpu=neon-vfpv4 -funsafe-math-optimizations	

I expected the -mfpu=neon option to have a much larger impact.

Adding “-march=armv7-a,” as someone suggested, makes it slower!

If you compile with the RPi 2 specific options above and try to run it on the RPi model B (pre version 2), it will die with "illegal instruction."

Dire Wolf is known to work on similar ARM based single board computers such as BeagleBone, CubieBoard2, CHIP, etc. The best compiler options will depend on the specific type of processor and the compiler target defaults. “-O3 -ffast-math” would be a good starting point for further experimentation.

“-march=native” (if available) would fine tune the generated code for your particular processor type. This is fine for personal use but not so good if you are producing a .DEB package for use by others who might have a different CPU type.

Packaging

Several people have made Dire Wolf available for installation from RPM or DEB packages. It also ended up some ham radio software collections such as Andy's Ham Radio Linux.

Here are some things to keep in mind when building for use by someone else.

Compiling for portability

You would expect "-march=native" to produce the fastest code. Why don't I use it here?

1. In my benchmarks, above, it has a negligible impact if any at all.
2. Some older versions of gcc don't recognize "native" as a valid choice.
3. Results are less portable. Not a consideration if you are building only for your own use but very important for anyone redistributing a "binary" version.

If you are planning to distribute the binary version to other people (in some ham radio software collection, RPM, or DEB package), avoid fine tuning it for your particular computer. It could cause compatibility issues for those with older computers.

Requiring a minimum Pentium 3 for x86, 32 bit, is a reasonable tradeoff between performance and supporting older hardware.

Raspberry Pi

Dire Wolf compiled on an older RPi works fine on the model 2.

Dire Wolf compiled on the RPi model 2 takes advantage of the newer "neon" instructions. It will **not** run on the older models.

Keep this in mind if you want to redistribute a binary version.