PiWxRx
A Pi-based weather Alerting System

Theory of Operation

Written by:
Martin Alcock, M. Sc, VE6VH

For software revision: 4.3

Praebius Communications Inc,
43 Citadel Bay NW,
Calgary, AB T3G 3Y4
www.praebius.com

Table of Contents

List of Figures

List of Tables

Document Revision History

Date	Rev	Description
November 2022	4.3	New document

Table 1 Revision History

Distribution

Rev	Distribution
All	Not for distribution

Table 2 Distribution

Reference Documents

[1] gnu.org, "General Public Licence," [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html. [Accessed 25th February 2018].

[2] Network Working Group, "RFC 2543: SIP: Session Initiation Protocol," 2002.

[3] National Weather Service, "Instruction 10-1712," 10 April 2022. [Online]. Available: https://www.nws.noaa.gov/directives/sym/pd01017012curr.pdf. [Accessed 25 Nov 2022].

[4] National Oceanographic and Atmospheric Administration, "NOAA Weather Radio Alerts," [Online]. Available: https://www.weather.gov/grb/nwr_same. [Accessed 24 May 2020].

Glossary of Terms

PBX	Private Branch Exchange. A node in a telephone network that provides connectivity for a series of local extensions to a set of trunks.
VOIP	Voice over Internet Protocol. A system where telephone calls are placed, and audio is exchanged using the Internet Protocol.
NOAA	National Oceanographic and Atmospheric Administration. Originator of weather broadcasts in the US.
SAME	Selective Area message Encoding.
POE	Power over ethernet.
JSON	Javascript object notation.

Intellectual Property Notice

The hardware components and all intellectual property described herein is the exclusive property of Praebius Communications Inc ("the owner"), all rights are reserved.

The owner grants licence to any Amateur for personal or club use, on an as is and where is basis under the condition that its use is for non-commercial activities only, all other usages are strictly prohibited. Terms and conditions are governed by the GNU public licence [1].

No warranty, either express or implied or transfer of rights is granted in this licence and the owner is not liable for any outcome whatsoever arising from such usage.

Disclaimer

This document is a preliminary release for a product still in development and may be subject to change in future revisions. The software may be subject to unpredictable behaviour without notice. You are advised to keep a can of RAID™ Ant, Roach and Program Bug killer handy. Spray liberally on the affected area when needed.

Overview

PiWxRx is a software based weather broadcast receiver that runs on a raspberry pi, with the aid of an external audio source. It implements an auto-answer SIP client for a PBX, as well as a data receiver, interpreter and message forwarder. Figure 1 illustrates a block diagram of the software structure.

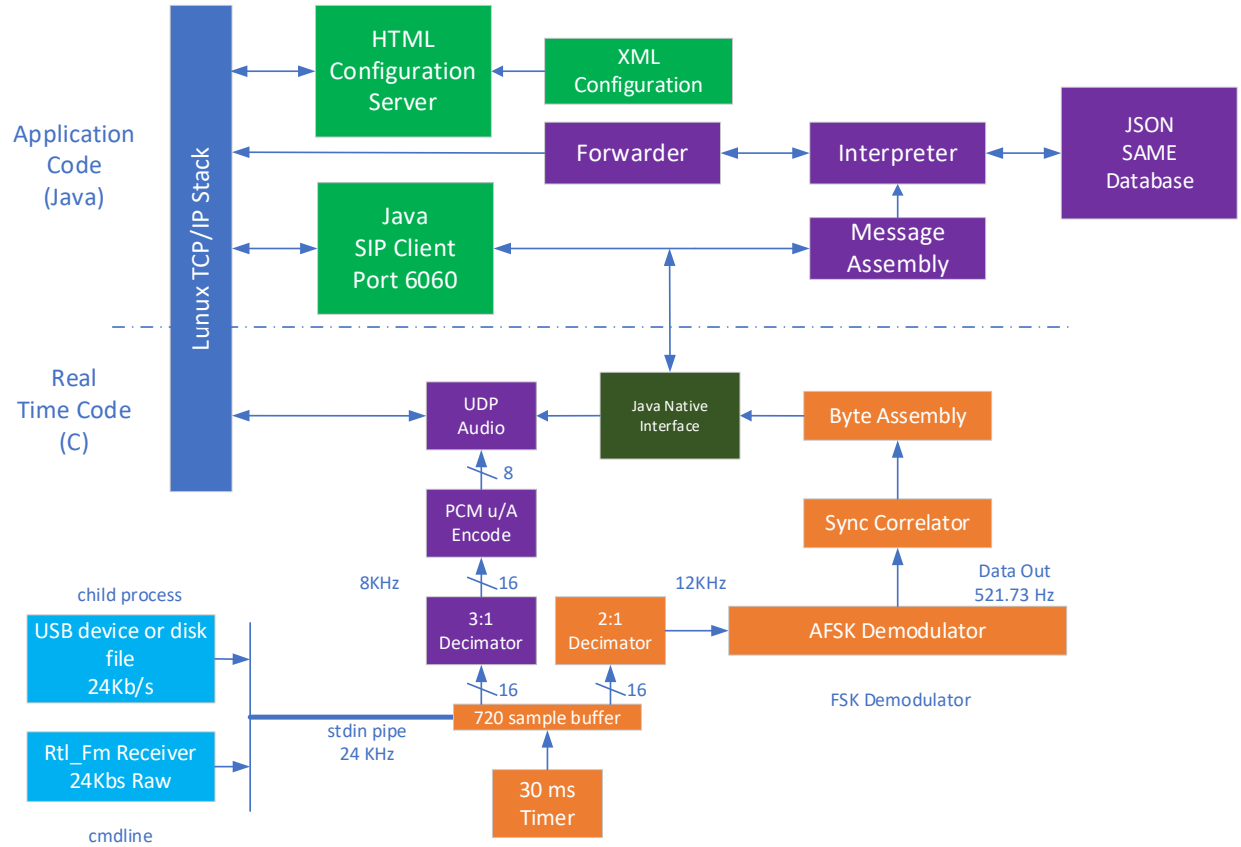

Figure 1 PiWxRx software Block Diagram

There are seven components to the software:

1. Audio generation process.
2. UDP packet generator.
3. AFSK Demodulator and byte assembly
4. Message decoder and interpreter
5. Forwarder
6. SIP Client
7. HTML server

The software is written in two different languages; the lower layers that are time-critical are written in the C language, and the less critical in the Java language. The interface between the two is implemented using the Java Native Interface (JNI). The first three components are contained in a loadable library, the remainder are in Java. For performance purposes on a raspberry Pi, the java code is not encapsulated in a Java archive (jar) file, instead the individual class files are stored.

Audio generation process

The audio generation process is an autonomous process that generates a 16 bit PCM digital audio stream at 24 Ks/s, written to stdout. The process is specific to the technology, which can be a software defined radio dongle, or a USB sound card connected to a radio receiver. Virtually any source can be utilized.

UDP Packet Generator

The packet generator decimates the 24Ks/s audio by a factor of three to 8Ks/s, then compresses the 16-bit PCM to 8 bits with either µLaw or A-Law compression, then encapsulates it into UDP packets conforming to the real time protocol (RTP), for transmission to the PBX. The destination address and port are supplied by the SIP client, transmission is only active during an inbound call.

AFSK Demodulator and message assembly

The AFSK demodulator decimates the audio source to a sample rate of 12Ks/s to decode the data messages from the broadcast station. The demodulator is described in a later section. The data message is assembled into bytes and processed by the decoder.

Message Decoder and interpreter

The message decoder takes a completed message and interprets the fields which are mostly numeric into more meaningful data using a JSON database, which is kept locally. The result is a fully interpreted message.

Forwarder

The forward forwards the message to a recipient either as a console message, an e-mail, or by posting it to a website using the http or SOAP protocols. When posting, the SAME codes are not interpreted, instead sent as numeric fields.

SIP Client

The SIP [2] client implements an answer-only extension to an Asterisk PBX, and supplies destination IP and port information to the UDP packet generator.

HTML Server

The HTML server shows up-to-the-minute status of critical system variables and enables on-the-fly changes of some parameters. Changes are not permanent, and only persist while the code is running.

Message Decoder

SAME Message Format

The following is an extract taken from the NOAA specification for SAME messages [3].

Physical

The following definitions of a bit are based on a bit period equaling 1920 microseconds (± one microsecond).

1) The data rate is 520.83 bits per second.
2) Logic zero is 1562.5 Hz.
3) Logic one is 2083.3 Hz.
4) Mark and space bit periods are equal at 1.92 milliseconds (3 cycles on 1562.5, 4 of 2083.3).

Data Message

A SAME transmitted data message consists of six (6) possible elements in the following sequence:

1) Preamble
2) Data message
3) Warning Alarm Tone/Attention Signal
4) Voice Message
5) Preamble
6) End Of Message (EOM)

Preamble

The first 16 bytes (prior to the header code and EOM) of the data transmission constitute a preamble, with each byte having the same value of hexadecimal AB (8- bit byte [10101011]). For all bytes, the least significant bit (LSB) is sent first. The bytes following the preamble constitute the actual message data transmission.

Data message

Each header and EOM data transmission consists of a series string of eight (8) bit bytes similar to standard asynchronous serial communications. There are no start, stop, or parity bits. Bit and byte synchronization is attained by a preamble code at the beginning of each header code or EOM data transmission. Data transmissions are phase continuous at the bit boundary. The message data (header) code is transmitted using American Standard Code for Information Interchange (ASCII) characters, as defined in ANSI International Committee for Information Technology Standards (INCITS) 4-1986 (R2002) for 7-bit ASCII, with the eighth (8th) bit always set to zero (0). Each separate header code data transmission should not exceed a total of 268 bytes if the maximum allowable geographical locations (31) are included.

Warning Alert Tone

The Warning Alert Tone, if transmitted, is sent within one (1) to three (3) seconds following the third header code burst. The frequency of the WAT is 1050 Hz (± 0.3%) for 8 to 10 seconds, at no less than 80% modulation (± 4.0 kHz deviation minimum, ± 5.0 kHz deviation maximum).

Voice message

If transmitted, the actual voiced message begins within three (3) to five (5) seconds following the last SAME code burst or WAT, whichever is last. The voice audio ranges between 20% modulation (± 1 kHz deviation) and 90% modulation (± 4.5 kHz deviation) with occasional lulls near zero and peaks as high as, but not exceeding, 100% modulation (± 5.0 kHz deviation). The total length of the voice message should not exceed two (2) minutes.

Preamble

Same as the preamble at the head of the message.

End Of Message

EOM is identified by the use of "NNNN".

Demodulator Theory

The two discrete mark and space tones previously mentioned can also be expressed as a centre frequency, of 1822.9Hz (ωc) with an offset of ±260.4Hz (Δf), to the mark and space frequencies. Ignoring amplitude, this can be expressed as a trigonometric function as:

$$\cos(\omega c \pm \Delta f)$$

By trigonometric identity:

$$\cos(\omega c) \cdot \cos(\Delta f) \mp \sin(\omega c) \cdot \sin(\Delta f)$$

Multiplication of this carrier by a complex exponential, $e^{-j\omega c}$ at the centre frequency, yields the following terms:

$$\begin{aligned} &\cos(\omega c)\,[\cos(\omega c) \cdot \cos(\Delta f) \mp \sin(\omega c) \cdot \sin(\Delta f)] \\ -j \cdot &\sin(\omega c) \cdot [\cos(\omega c) \cdot \cos(\Delta f) \mp \sin(\omega c) \cdot \sin(\Delta f)] \end{aligned}$$

Expanding the real term:

$$\cos^2(\omega c) \cdot \cos(\Delta f) \mp \cos(\omega c) \cdot \sin(\omega c) \cdot \sin(\Delta f)$$

Or, by the double angle identity:

$$\frac{1}{2}[1 - \cos(2\omega c)] \cdot \cos(\Delta f) \mp \frac{1}{2} \sin(2\omega c) \cdot \sin(\Delta f)$$

And expanding the inner terms yields:

$$\frac{\cos(\Delta f) - \cos(2\omega c \pm \Delta f)}{2} \qquad (1)$$

For the imaginary term, a similar expansion applies:

$$-j \cdot \sin(\omega c) \cdot \cos(\omega c) \cdot \cos(\Delta f) \mp \sin^2(\omega c) \cdot \sin(\Delta f)$$

Then:

$$\frac{1}{-2j}\sin(2\omega c) \cdot \cos(\Delta f) \mp \frac{1}{-2j}[1 - \cos^2(\omega c)] \cdot \sin(\Delta f)$$

Again, by the double angle identity:

$$\frac{\pm \sin(\Delta f) + \cos(2\omega c \mp \Delta f)}{-2j} \quad (2)$$

After low pass filtering of (1) and (2), the high frequency components are removed and the following complex terms remain:

$$Re = \frac{\cos(\Delta f)}{2} \qquad Im = \frac{\pm \sin(\Delta f)}{-2j}$$

Combined together they form the complex exponential:

$$\frac{e^{\mp j\Delta f}}{2} \quad (3)$$

The data can be recovered by multiplying the result in (3) by the complex conjugate of its counterpart delayed by one bit period. The phase angle can be found on the sign of the imaginary component.

$$D(t) = sgn\{Im[\frac{e^{\mp j\Delta f t}}{2} \times \frac{e^{\pm \Delta f(t-d)*}}{2}]\}$$

Where t is the current sample in time, and d is the number of samples in a bit period.

Demodulator Implementation

Figure 2 illustrates the implementation of the demodulator. Starting in the upper right, the samples from the audio source are stored in quanta of 30ms, then decimated by 2 to become 360 samples.

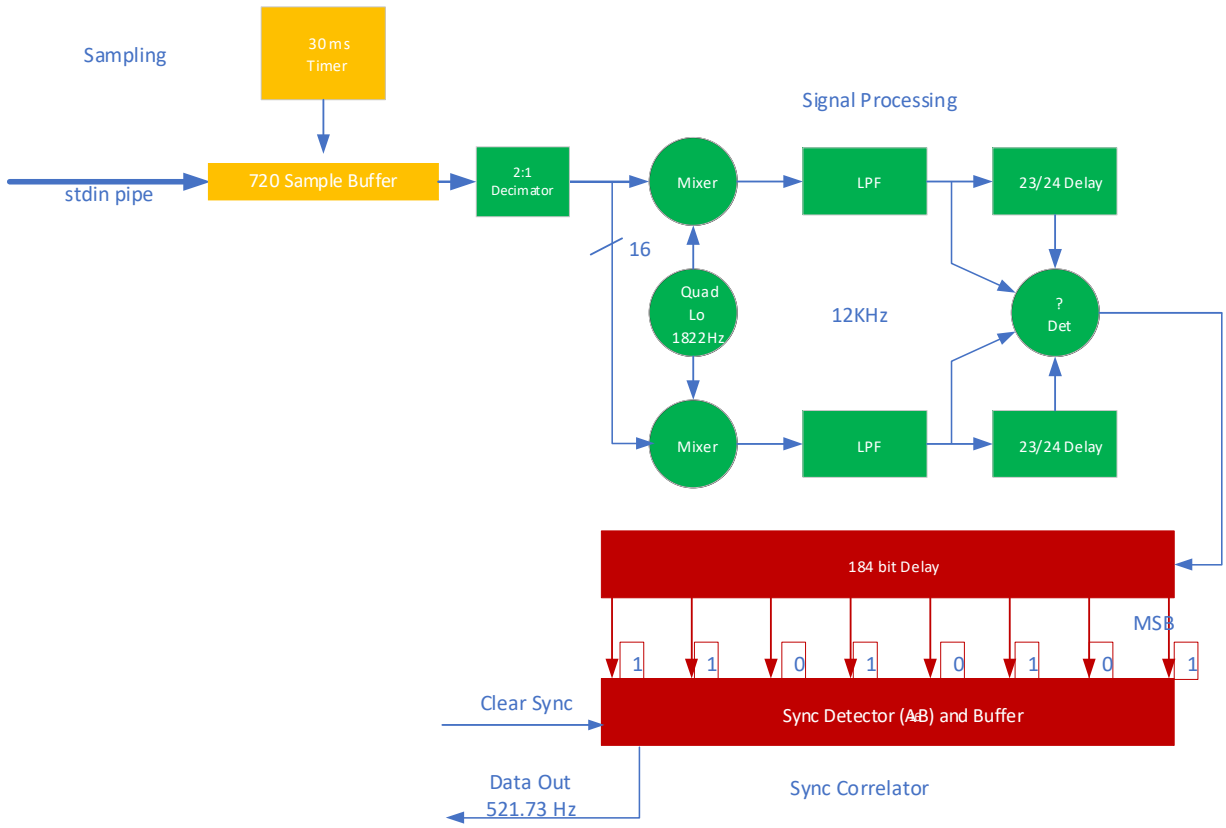

Figure 2 FSK Demodulator Signal Processing

The quadrature oscillator generates the real and imaginary component of the mixer, $e^{-j\omega c}$ to generate the I (in phase) and Q quadrature components of carrier shifted down in frequency. As the mixing is performed on a real signal with no complex components, the resultant products are shown in Table 3.

Tone	Cos(2ωc±Δf)	±sin(Δf)	Equation (2)	ωc
Mark 2083.3 Hz	3906.2Hz	260.4Hz	3645.8	1822.9
Space 1562.5 Hz	3385.4Hz	-260.4Hz		

Table 3 Demodulator mixing products

The difference of the high and low products prove that the post-demodulator equation (2) is correct and contains both high and low frequency terms. The unwanted term is then removed by low pass filtering. The filter specification it to be flat to 260Hz, then -20dB down at the lower of the high products, 3385.4Hz.

Preamble Correlator

The bit demodulator uses a divide by 23/24 swallow counter to generate the bit clock at 520.83 bits/second. The counter nominally divides by 23 until the 24th count when it 'swallows' a count, resulting in a fractional divider of 23.04, which is close to the data rate, and will not drift over the period of one message.

The decoded data is then delayed by a 184-bit correlator that is tapped every 23 bits. As the message is transmitted LSB first, the taps represent the data pattern 11010101_2, which is the bit reversed form of the preamble pattern AB_{16}. When the pattern is detected, the correlator determines that is in sync, and a divide by 23 counter is started. At the zero count, the 8 bits are then taken from the correlator taps, bit-reversed and presented to the upper layer as a data byte. As the preamble pattern is repeated, it will also be found at the start of each message.

Message Assembly

The general form of a message, from the specification [3], is as follows:
(Preamble) ZCZC-ORG-EEE-PSSCCC-PSSCCC+TTTT-JJJHHMM-LLLLLLLL-

The message is assembled by a state machine Figure 3 illustrates the states.

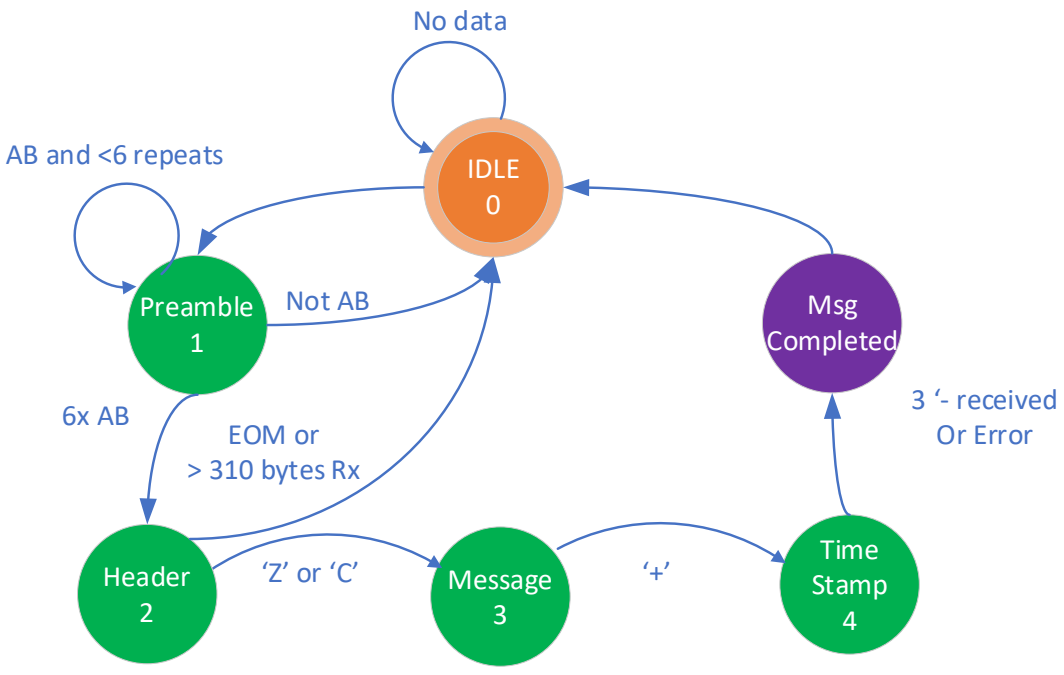

Figure 3 Message Assembly state machine

The machine exists in one of 6 states:

1. Idle. The machine remains here until a preamble pattern is found by the correlator.
2. Preamble. At least 6 preamble characters (AB) must be received to be considered a valid message. The NOAA specification calls for 16 to be sent.
3. Header. This state looks for the 'Z' or 'C' at the beginning of the message.
4. Message. Bytes are collected until the '+' character, signalling the timestamp on the message, is received.
5. Time Stamp. This state collects the three fields in the timestamp and declares a completed message when all are received. A message that is longer than 310 bytes, contains a null (00) character, or a non-ascii character (bit 7 set) is also considered complete.
6. Message Completed. The message is passed on for further processing. The machine then returns to the IDLE state, and sends a 'Clear Sync' message to the correlator to reset it.

Interpreter

Each message contains six major fields, as show in Table 4.

Field Designator	Contents
ORG	Originator of the message
EEE	Type of event
PSSCCC	Geographical area block
TTTT	Purge time, ie length of validity of the message, in 15 minute increments
JJJHHMM	Julian calendar date and time when the message was disseminated in UTC
LLLLLLL	Originator or rebroadcaster of the message

Table 4 SAME message Fields

JSON database

The message fields are interpreted locally by a JSON database which contains the

Name	Value
Agencies	An array of two strings containing the agency designator and name
Bulletins	An array of two strings, interpreting the EEE field in the message
Dateoffset	A positive integer added to the Julian date before interpretation
Houroffset	A positive integer subtracted from the UTC for local time
Locations	An array of twos strings interpreting the PSSCCC field in the message

Table 5 JSON database fields