HAB Project Pseudocode

**main function:**

parameters - arguments: -debug flag (will be used to enter debug mode)

Will have the main configuration for APRS address, the sensors to enable or disable, and the pin configuration for each item.

Will call a timer that determines the transmission time.

**RADIO**

**AX.25 packet function**

Parameters: Lat, Long, Temp, Pressure, etc. pointer to string to transmit

Returns Pointer to prepared transmit buffer string.

This function will take the APRS configuration and the data and create an appropriate array of bits/bytes to be sent by the transmit function. Appends all of the data after it’s been processed by the following sub functions: i.e.: address (calc SSID), Data, CRC, bitstuffing, then append flags.

***Sub Functions***

**Path/Via function**

Parameters: Altitude, string pointer

Returns: correct path string pointer?

Uses the altitude config. If below 1500m of altitude it uses WIDE1-1, and WIDE2-1. If above, it just ses WIDE2-1. Calls the call sign function with the correct SSIDs to format as correct destination format. Returns the pointer to the string that will be added to the AX.25 packet

**Callsign Function**

Parameters: Callsign, SSID, String Pointer

Returns: String Pointer

Takes in a call sign. If it is less than 6 characters it appends spaces. This calls the SSID function to calculate the correct SSID to append as the 7th character. Returns the string pointer to this newly formatted callsign

**SSID function**

Parameters: SSID number, last or not

This function will take in the SSID and format it correctly for sending in the AX.25 frame

**CRC function**

Parameters: character, current crc

This function will take the current CRC and process the next character in the string if it is not a flag or CRC byte. [CRC Calculated before bit stuffing]

**Bitstuffing Function**

parameters: completed string minus flags

Takes the completed string (minus flags), and does a running count of consecutive 1s. If there are 5, it stuffs in a zero and shifts the rest of the string to the right.

**AFSK1200 Modulation Function** (Read more about Arduino PWM Cheat Sheet)

Parameters: array of bits to send,

Sets fast PWM. Uses timers 2 and 3. 3 used for encoding data, and 2 used for waveform generation using sine table. Uses an interrupt function (ISR) send data at appropriate frequency.

***Subfunctions***

**Configuration (not a function)**

Uses appropriate settings to calculate the correct values for the Arduino PWM timers and Interrupts to produce 1200 and 2200 Hz frequencies. Configures timer 2 and timer 3 to the correct clock values, and such.

**Sine Table (not a function)**

Stored in program memory. Iteratively indexed by the transmit ISR to generate the sine wave. When using a faster frequency, the table is indexed less often to produce a faster wave.

**Generate Sinewave ISR Function**

Parameters: Timer 2 Vector

Runs at a time rate determined by timer 2. It produces a sinewave by indexing the sine table. By changing the OCR2A value, we can change the frequency at which this ISR runs. (may have something to do with an R-2R ladder?)

**Encode AFSK ISR Function**

Parameters: Timer 3 Vector

Runs the encoding process at the rate of timer 3. What it does is keep the freq (on timer 2 sinewave) the same if the value is a 1 and toggles the frequency on timer 2 if it is a zero. It also keeps track of the number of 1 bits, and if there are 5 in a row, it inserts a zero. This also takes care of sending the AX.25 flag when first getting a byte from the string, and after the buffer is empty. This interrupt function runs faster than the sinewave function I think.

**GPS**

**Start GPS to Serial Function**

Parameters: none

Returns: Writes NMEA strings to serial console

Sends high to the GPS enable pin, starts a serial port on 9600. Read strings into a circular buffer. If the string starts with [whatever NMEA code], parse the string for location data. Save this information into several variables (longtitude, latitude, altitude).

If the string starts with [some other NMEA code] parse the string for no. of satellites data. Save this in a different variable. This information will be used by the AX.25 function.

***Optional***: Only read codes if the “Valid” identifier is in the string.

SoftwareSerial only required for debugging.

**SENSORS**

**Read Sensors Function**

Parameters: I2C addresses (possibly in an array), pointer to array of read values

This function should initiate a read request to the correct address device. Then while there is data to read from the device it will store each byte, in the proper string variable depending on the device. These variables will be read by the AX.25 function.

**FRONT END**

**Get Data from APRS-IS Function**

Parameters: Call Sign

This function connects to the APRS-IS server using the socket library and collects packet strings that start with our call sign (K5TRL).

user K5TRL pass 8325 vers ece103group3 1.0 filter t/p

Connect to: **rotate.aprs.net:14580**

User Login:

**user K5TRL-11 pass 8325 vers ece103group3 1.0 filter b/K5TRL\***

*8325 is my passcode*

**Front End Display Function**

This function provides a front end using the nCurses library that displays the packets taken from the APRS-IS server.

The following is incomplete written C code for the Front End

# INCLUDE <sys/socket.h>

# INCLUDE <stdio.h>

if ((sock = socket(DOMAIN, TYPE, PROTOCOL)) < 0)

{

printf("\n Socket creation error \n");

return -1;

}

if (connect(something something something) < 0)

{

printf("\nConnection Failed \n");

return -1;

}

// Following request will be sent to APRS-IS:

**user K5TRL-11 pass -1 vers ece103group3 1.0 filter b/K5TRL\***

FILE \*fp;

if (fp = fdopen(stream, “r”); == NULL)

{

printf("\nStream Open Failure \n");

return -1;

}

// Packets will be in the following format:

// K5TRL-11>APRS,WIDE2-1: LAT, LONG, GPS-ALT, NUMSATS, TEMP, PRESSURE, BARO-ALT, HUMIDITY\n

fscanf(fp, “K5TRL-11>APRS,WIDE2-1: %s, %s, %s, %d, %lf, %lf, %lf, %lf\n”, lat, long, gpsalt, &temp, &press, &baroalt, &humid);

printf(“Latitude: %s\nLongitude: %s\nAltitude (from GPS): %s ft\nNo. of Connected Satellites: %d\nTemperature: %.3lf deg C\nPressure: %.3lf atm\nAltitude (from barometer): %.3lf ft\nHumidity: %lf %%\n”, lat, long, gpsalt, temp, press, baroalt, humid);

More Pseudocode:

APRS Pseudocode:

1. Collect all Data
   1. DEVID: “APRS”
   2. Call Sign “K5TRL”
   3. SSID “11”
   4. WIDE Path Configuration (cond: on altitude)
   5. Latitude (read from GPS script-serial)
   6. Longitude (read from GPS)
   7. Altitude (read from NMEA GPS)
   8. # of GPS satellites (read from GPS)
   9. Sensor Telemetry
      1. Accelerometer (Analog)
      2. Temperature (I2C - “)
      3. Humidity and Temperature (I2C)
      4. Pressure, Altitude, and Temperature (I2C - “Slave - 0x60” “8-bit Read - 0xC1” “8-bit Write 0xC0”)
2. Convert Data to AX.25
   1. Flag
      1. 7E (01111110) - - send at least once before message
   2. Address
      1. Calculate SSID value for
         1. Callsign
            1. K5TRL (SSID = 11)
         2. Destination
            1. APRS or GPS (SSID = 0)
         3. Via
            1. WIDE1-1 or WIDE2-1 (SSID = 1 for both)
      2. SSID should end in 0 if there are more call signs, and in 1 if there are no more call signs
   3. Control
      1. 3F
   4. Protocol ID
      1. FO
   5. Information
      1. Latitude
      2. Longitude
      3. Altitude
      4. No. of Satellites
      5. Sensor Data
   6. Frame Check Sequence
      1. Dunno, hard:
      2. <https://barrgroup.com/Embedded-Systems/How-To/CRC-Calculation-C-Code>
      3. <http://practicingelectronics.com/articles/article-100003/CRC_CCITT_Generator.m>
      4. <https://en.wikipedia.org/wiki/Computation_of_cyclic_redundancy_checks>
      5. <https://github.com/tcort/va2epr-tnc/blob/master/firmware/aprs.c>
   7. Flag
      1. 7E -- send at least once
3. Form Sentence

***EXAMPLE of how the packet will be read by APRS-IS:***

K5TRL-11>APRS,WIDE2-1: packet data

***EXAMPLE of the data to send bit by bit:***

{7E, ‘A’, ‘P’, ‘R’, ‘S’, ‘ ’, ‘ ’, 011 0000 0, ‘K’, ‘5’, ‘T’, ‘R’, ‘L’, ‘ ’, 011 1011 0, ‘W’, ‘I’, ‘D’, ‘E’, ‘2’, ‘ ’, 011 0001 1, 3F, F0, ‘@’, ‘4’, ‘5’, ‘.’, ‘5’, ‘1’, ‘1’, ‘2’, ‘N’, ‘-’, ‘1’, ‘2’, ‘2’, ‘.’, ‘3’, ‘1’, ‘4’, ‘8’, … etc… , fcsHI, fcsLO, 7E}

1. Modulate Data to AFSK1200
   1. Flipping from 1200 to 2200 is a zero, staying the same is a one. (NRZI)
   2. Generate Sine Wave Data table for PWM comparator values
   3. Turn on FastPWM
2. Transmit
   1. Check Transmission Time
   2. Transmit (Interrupt Sequence if Time)
3. Collect all data
4. Calculate SSID values
5. Put data into sentence array with address and SSIDs and data to send
6. Process array by bit, calculating the FCS for each byte (but not if they are flag or fcs bytes)
   1. Do the bit stuffing procedure if there are more than 5 1s in a row