Spring 2018

ECE 103 Engineering Programming

Chuck Faber: cfaber@pdx.edu

Abdulrahman Alamar: aalamar@pdx.edu

Ebtehal AlEnezi: ebte@pdx.edu

Portland State University

Hardware

The GPS module has been successfully tested to produce NMEA sentences to the serial console on the Arduino. It produces several NMEA protocol sentences from which we will read the latitude, longitude, altitude, and number of satellites connected to.

The radio module has been successfully tested and can produce high and low tones from the Arduino that we are able to hear using the software defined radio or from our handy talky radio.

Two sensors have been tested and found to be working. Most sensors communicate via I2C pins on the Arduino. Both are producing good data using code made by other sources.

Software

Research has been done on how to write code to read data sent from the GPS module from the serial console. The GPS module produces several NMEA sentences which will have to be scanned for particular prefix strings. The entire sentence string will have to be scanned for data and have that data stored in particular variables to be used by other parts of the program (i.e. the radio system for parsing and transmitting this data).

Likewise research is continuing on the radio system, in particular how to generate 1200Hz and 2200Hz sine waves using the Arduino. The data to be sent must be parsed into AX.25 data protocol format which is described well in the paper "PIC-et Radio: How to send AX.25 UI Frames using Inexpensive PIC Microprocessors" by John Hansen [W2FS]. This involves collecting and processing the data in some cases using bit shifting, as well as calculating a check flag. The data is sent bit by bit, with a transition between 1200Hz, and 2200Hz representing a zero, and a non-transition indicating a 1, as AX.25 packets use a HDLC NRZI modulation scheme. Producing the carrier wave using the Arduino which doesn't have digital to analog convertor hardware will require activating the FastPWM 16MHz clock pulse on the arduino and using a calculated table of values to produce the sine wave as well as some code to calculate which value to send via the PWM pin so we can generate a 2200 Hz signal when this isn't typically possible. This method is detailed in "Arduino Uno and FastPWM for AFSK1200" by Craig Chapman. Using this newly understood information, a more complete radio pseudocode can be written for sending APRS packets.

Preliminary pseudocode has been written for two of the sensors. A draft of initial code for the sensors has been tested and is performing correctly.

Further Work

For the GPS we need to begin writing code to read data from the serial console to read and parse the NMEA sentences to gather the data we need to be sent to the radio. This code will have to scan each line on the serial terminal, and use conditional statements to apply a different data scanning scheme to different NMEA strings.

During testing of the radio with basic code, the produced tones seemed to gradually increase in pitch. This may possibly be due to heating of the radio, which can't be good for the radio module. We need to understand better about how to keep the radio hardware in good functioning order. In addition, code needs to be written to take in the various pieces of data the program will use and parse them into AX.25 data arrays to be sent. Another function needs to be written to send the data by modulating the frequency appropriately using the PWM functions on the Arduino.

For the sensors, more code still needs to be written and tested to have the sensors communicate via the I2C protocol. If more time is available, we will add more sensors to the mix and collect more data.

Lastly, we need to start thinking about a front-end that a user will be able to interface with. This will likely have to use libraries allowing for the reading of data from a server, and producing the data which has been transmitted to the APRS network.

Discussion

Chuck states that trying to understand the AX.25 protocol and how to send AFSK1200 modulated signals via PWM was incredibly difficult, but also very satisfying when things started to click. Though things have started to click, he still feels that he still has a lot to understand about this kind of signal processing and transmission using Arduino and PWM.

AJ mentions that the Arduino labs offered by Dr. Wong and Nandita have helped by exposing us to using various sensors, offering him better understanding of how to implement the sensors in our project.

Ebtehal relates that it was difficult to actually set up the GPS for sending signals to the Arduino and then being able to read those signal, in particular because the Arduino has only one serial port which was already being taken up by the GPS module. A new library had to be installed for debugging purposes called SoftwareSerial to create a virtual serial port from which the signals could be read. She states that she's learned a lot so far from this project, but she is beginning to understand how everything works and some things that seemed difficult early on now seem simple.

Milestones

For a function breakdown, see Appendix A.
For front end C code, see Appendix B.
For more pseudocode see Appendix C.

Appendix A: Function Breakdown

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 altitude 1500ft 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).

Front End Display Function

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

Appendix B: Front End Code

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 g/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);
```

Appendix C: Other Pseudocode

GPS Software Debug Pseudocode:

```
PROGRAM
        INCLUDE software serial
        INITIALIZE softwareserial GPS(TX pin, RX pin)
        ASSIGN data to space

        FUNCTION setup
                ASSIGN serial begin to 115200 speed
                ASSIGN GPS begin to 9600 speed
        END FUNCTION

        FUNCTION loop
                IF GPS() available
                ASSIGN data to read GPS
                PRINT serial(data)
                END IF
        END FUNCTION
END PROGRAM
```

Temperature Sensor Pseudocode

Making use of a library Adafruit_MCP9808.h

```
        SET the data rate in bits per second(Baud) to the desired speed

        IF (sensor is not found)
                { PRINT sensor not found
                LOOP until sensor is found}

        SET celsius_var to READ the temperature from the sensor
        SET fahrenheit_var  to celsius_var *9 / 5.0 + 32

        DISPLAY celsius_var
        DISPLAY fahrenheit_var

        SET  the DELAY to 1000
```

APRS Pseudocode:

1. Collect all Data
    a. DEVID: “APRS”
    b. Call Sign “K5TRL”
    c. SSID “11”
    d. WIDE Path Configuration (cond: on altitude)
    e. Latitude (read from GPS script-serial)
    f. Longitude (read from GPS)
    g. Altitude (read from NMEA GPS)
    h. # of GPS satellites (read from GPS)
    i. Sensor Telemetry
        i. Accelerometer (Analog)
        ii. Temperature (I2C - “)
        iii. Humidity and Temperature (I2C)
        iv. Pressure, Altitude, and Temperature (I2C - “Slave - 0x60” “8-bit Read - 0xC1” “8-bit Write 0xC0”)
2. Convert Data to AX.25
    a. Flag
        i. 7E (01111110) - - send at least once before message
    b. Address
        i. Calculate SSID value for
            1. Callsign
                a. K5TRL (SSID = 11)
            2. Destination
                a. APRS or GPS (SSID = 0)
            3. Via
                a. WIDE1-1 or WIDE2-1 (SSID = 1 for both)
        ii. SSID should end in 0 if there are more call signs, and in 1 if there are no more call signs
    c. Control
        i. 3F
    d. Protocol ID
        i. FO
    e. Information
        i. Latitude
        ii. Longitude
        iii. Altitude
        iv. No. of Satellites
        v. Sensor Data
    f. Frame Check Sequence
        i. Dunno, hard:
        ii. https://barrgroup.com/Embedded-Systems/How-To/CRC-Calculation-C-Code
        iii. http://practicingelectronics.com/articles/article-100003/CRC_CCITT_Generator.m

iv. https://en.wikipedia.org/wiki/Computation_of_cyclic_redundancy_checks
v. https://github.com/tcort/va2epr-tnc/blob/master/firmware/aprs.c

g. Flag
   i. 7E -- send at least once after message

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}

4. Modulate Data to AFSK1200
   a. Activate timer for encoding and timer for sinewave generation
   b. Set interrupt values for both timers
      i. Sinewave timer ISR
         1. Generate Sine Wave Data table for PWM comparator values
      ii. Modulation timer ISR
         1. Flipping from 1200 to 2200 is a zero, staying the same is a one. (NRZI)

a. Collect all data
b. Calculate SSID values
c. Put data into sentence array with address and SSIDs and data to send
d. Process array by bit, calculating the FCS for each byte (but not if they are flag or fcs bytes)
   i. Do the bit stuffing procedure if there are more than 5 1s in a row