Final Project:
A compact, portable APRS Controller

Adam Swann
EE 4750-1 (Station 2)
Spring 2001

Introduction

Automatic Position Reporting System (APRS) was developed by Bob Bruninga (WB4APR) for the purpose of tracking remote, mobile amateur (ham) radio stations. The APRS protocol was first introduced in 1992. Since then, APRS usage has grown exponentially and APRS networks are online in most mid-sized American cities. However, the cost, bulk, and complexity of an APRS station are discouraging to would-be APRS users. The objective of this project is to provide a compact, affordable, and simple APRS solution.

Overview of APRS

An APRS station consists of one or more digipeaters (digital repeaters) and users whose packets are relayed by the digipeaters. Digipeaters are generally fixed stations that use high gain, power, and elevation transmission systems that cover a comparatively large geographical area. Mobile stations transmit their coordinates and the digipeater picks them up and relays the information to other fixed or mobile stations. Some digipeaters are even connected through the high frequency (HF) bands or the Internet to share the information over an even larger region. Home users can monitor the network and track positions as they are received. The information can also be pulled from the Internet using specialized client software.

A typical mobile station consists of a handheld global positioning system (GPS), a laptop computer, a terminal node controller (TNC), and a VHF transceiver. The GPS tracks the station's current position and relays it to the computer. The computer formats the data and sends it to the TNC at regular intervals (five minutes is typical) for transmission over the radio. Figure 1 shows the layout of a typical station.

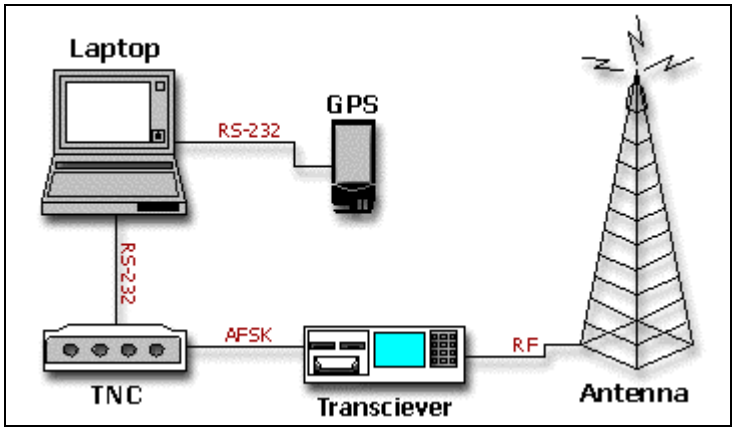

Figure 1: A typical mobile APRS station.

Theory

The goal of this project is to condense the laptop and TNC (shown in Figure 1) into a compact, affordable package that the electronics enthusiast can construct in a few hours. The laptop will be replaced with an Atml AVR microcontroller (AT90S8515). This 40-pin, 8-bit microcontroller has an internal UART, 32 digital configurable IOs, and 512

bytes of RAM. The modem will be replaced with a MAXCOM MX614 single-chip Bell 202 Compatible Modem.

Hearing the GPS: NMEA-0138

I used a Garmin GPS12XL 12-channel GPS. This particular model sells for around $250, although cheaper models with similar functionality are available. The only requirement for a GPS is that it has a data port that conforms to the NMEA-0138 standard. NMEA-0138 is the language the GPS uses to communicate with other devices. The standard provides an RS-232 compliant data interface that constantly transmits information about the station's location, speed, proximity to known landmarks, etc. There are about 15 different sentences that the GPS12XL supports. The only sentence used by this project is the "required minimum specific GPS/transit data" (RMC). Figure 2 shows the RMC sentence and its contents. All sentences begin with a '$' and end with a carriage return.

Sample RMC sentence:

```
$GPRMC,225446,A,4916.45,N,12311.12,W,000.5,054.7,191194,020.3,E*68
```

Sentence terms:

225446	Time of fix 22:54:46 UTC
A	Navigation receiver warning A = OK, V = warning
4916.45,N	Latitude 49 deg. 16.45 min North
12311.12,W	Longitude 123 deg. 11.12 min West
000.5	Speed over ground, Knots
054.7	Course Made Good, True
191194	Date of fix 19 November 1994
020.3,E	Magnetic variation 20.3 deg East
*68	mandatory checksum

Figure 2: Sample NMEA RMC sentence. (Taken from the NMEA FAQ v6.3)

The only information that this project requires is the time and the longitude and latitude. I do not verify the checksum the position is transmitted so often that errors will be on the network only as long as the transmit interval.

Audio-Frequency Shift Keying

Most regional APRS networks use 1200 baud audio-frequency shift keying (AFSK) to transmit the information over the air waves. At the lowest level, the AFSK signal consists of an NRZI data stream using tones at 1200- and 2200-Hz. A logical '0' is designated by a transition from low to high or high to low, and a '1' is designated by no change in the audio frequency. Because the protocol uses bit stuffing (described below), at most five ones will be sent consecutively in a packet allowing the receiving modem to use the audio signal as a clock.

Speaking the Right Language: AX.25

The amateur community developed AX.25, which is an adaptation of the X.25 High-level Data Link Control (HDLC) standard. Numerous packet types exist in the AX.25

protocol, but only the Unnumbered Information (UI) packets are used in APRS. UI packets are broadcast to any station listening on the network and are not acknowledged (i.e., even if there is an error in the packet or if the packet is never received, the packet is not resent.). The basic UI packet format is shown in Figure 3.

Data:	Flag	Dest.	Src.	Digi.	Control	PID	Info	FCS	Flag
Bytes:	1	7	7	0 – 56	1	1	N	2	1

Figure 3: The UI packet format

All characters except the Frame-Check Sequence (FCS) are transmitted LSB first. The FCS is transmitted MSB first.

A flag character is 01111110 (0x7E hex). The flag marks the beginning and end of a packet. Any number of flags may exist between packets. The modem must transmit either flags characters or a data packet at all times while the radio is transmitting. This requirement ensures that other stations can detect that the radio frequency is in use, even if no packets are being sent. It is useful to transmit ten or more flags before the packet to ensure that the transmitter and receiver have time to stabilize before the actual data is sent.

The address block can contain up to four addresses. At least two addresses must be included: the source callsign (e.g., KC5FRP) and the destination callsign. Optionally, two digipeater stations may be included. If a digipeater is included and the digipeater stations receives the packet, it will retransmit it (usually at a higher power and/or from a better location). The callsign consists of exactly six ASCII characters and a Secondary Station Identifier (SSID). If any of the six ASCII characters is not needed, a space character is used instead. The SSID is a 4-bit value between 0 and 15. The upper 4-bits of the SSID have a special meaning and are typically set high. The SSID allows up to sixteen stations with the same callsign to exist on the network (i.e., KC5FRP-0 through KC5FRP-15). Each character in the address is shifted left one bit before transmission, and the new bit is zero unless it is the last address character that will be transmitted. The destination is transmitted first, followed by the destination address and optionally any digipeater stations addresses. A sample destination address is shown in Figure 4. Note the final bit is set high to indicate that it is the last address bit. (In any packet, regardless of the number of digipeater stations, only the final bit of the final address is set high.)

Value	Binary Value
K	10110110
C	10000110
5	01101010
F	01011100
R	10100100
P	10100000
SSID = 0	11100001

Figure 4: A sample source address.

In a UI frame, the control byte is always set to 0x03 and the PID byte is always set to 0xF0.

The information section of the frame can hold up to 256 8-bit characters. No special manipulation is required.

The FCS is a 16-bit value calculated in accordance with the ISO 3309 standard. The FCS is generated by the sender and used by the receiver to ensure error-free transmissions.

The APRS Data Format

Everything from position, heading, and speed to temperature, wind direction, and even alphanumeric pages can be encoded into an APRS data packet. For now, we are only interested in transmitting the minimal information necessary to track a mobile station. The packet format is shown in Figure 5.

Sample APRS packet:

```
/225446z4916.45N/12311.12W-PHG2230/Hello from an AVR!
```

Packet terms:

225446	Time of fix 22:54:46 Zulu (UTC)
4916.45N	Latitude 49 deg. 16.45 min North
12311.12W	Longitude 123 deg. 11.12 min West
PHG2230	Power, height, and gain of the transmitting antenna.
	An optional message can be included in the packet.

Figure 5: Sample APRS Packet.

Putting it All Together

Microcontroller Configuration and Operation

PORTA of the microcontroller is used to control the MX614 modem chip. Pins 1, 2, 3, 6, and 7 should be configured as outputs, and the others should be configured as inputs. The pins are used as follows:

Pin	Function	Direction
0	RXD	Input
1	TXD	Output
2	M0	Output
3	M1	Output
4	unused	n/a
5	RDY	Input
6	CLK	Output
7	PTT	Output

The internal UART is used to receive the data from the GPS. Depending on the processor clock frequency and GPS data rate, the UART registers will need to be initialized to different values. My processor ran at 3.68 MHz with a GPS data rate of 2400 baud. I initialized the UART registers as follows:

Register	Value	Function
UBRR	0x5F	2400 bps @ 3.68 MHz
UCR	0x10	Receiver Enable

Modem Configuration/Operation

The modem has four operating modes (M0 and M1 are the mode control inputs on the chip):

Mode	M0	M1	Mode
0	0	0	Receive w/Backchannel
1	0	1	Transmit
2	1	0	Recieve
3	1	1	Power down.

Modes 1 and 2 are used in this project. When the modem is not being used, it is put into receive mode.

Timing for the 1200bps AFSK is handled by the modem's built-in retiming circuit. The uses three signals: Data (TXD), Ready (RDY), and Clock (CLK). The modem asserts its RDY signal when it is ready to transmit a bit. The software then must put the next bit to send on TXD and clock CLK. The modem has two internal shift registers that store the inputted values and transmit them with appropriate timing.

Software Description

The software begins by initializing the modem, LCD, serial port, etc. It displays a welcome message and then begins scanning the serial port for an RMC sentence. When it encounters the sentence, it begins buffering the data to RAM. Once all the useful information (lat, long, and time) have been buffered, it displays the coordinates on the LCD and then sends the formatted data to the modem.

The packet is assembled and sent byte-by-byte. A main control register determines the whether or not to calculate the CRC and bit-stuff. Each bit is rotated into the carry flag. Depending on whether the bit is a '1' or a '0', a different routine is called. The "TX1" routine does nothing but clock the modem (since a one is represented by no change in the AFSK). The "TX0" routine reads the current value being transmitted and toggles it (since a zero is represented by a transition in the AFSK).

Schematic Circuit Diagram

Because the microcontroller is mounted on the Atmel STK-500 development board, only the pin connections to the development board are shown. All connections are made to PORTA, although any port could be used with a minor modification to the program source code. The GPS is connected directly to the auxiliary serial port on the STK-500.

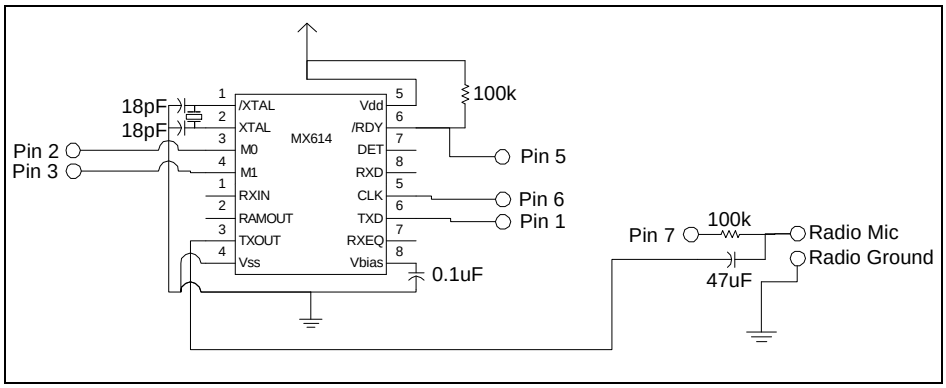

Figure 7: Project Schematic Diagram.

Program Flow Diagram

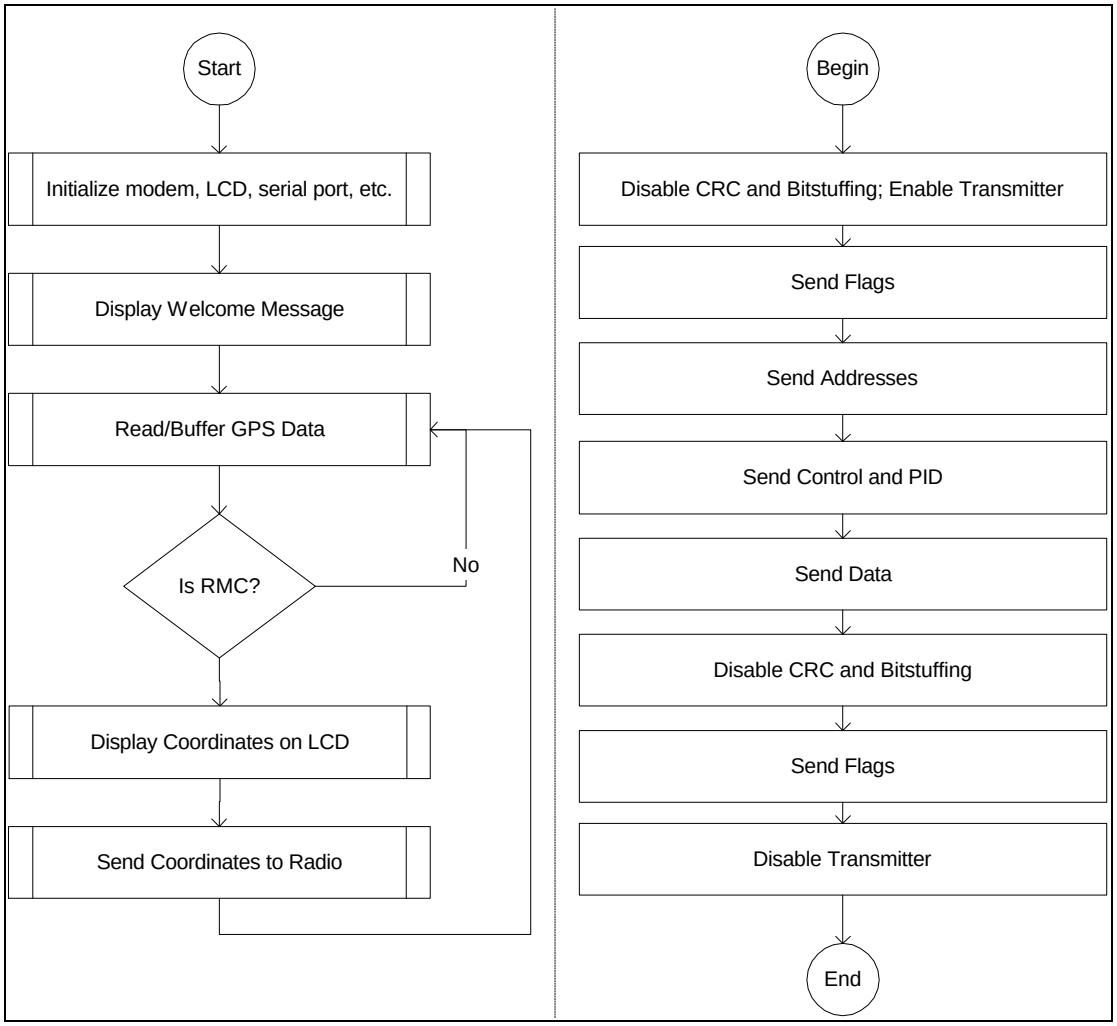

Figure 6: Main Program Flowchart (left) and Packet Transmission Routine.

Testing the System

The test system consisted of a "mobile station" – the circuit described above coupled with a portable radio and GPS – and a "base station," consisting of a high powered radio and

antenna, an external modem, and a computer. The Garmin GPS12XL features a simulator mode that allows the user to type coordinates in directly instead of using the satellite system. The system worked perfectly (once a bad connection in the radio cable was fixed). The most common APRS operating frequency is 144.490 MHz in the 2-meter band.

Conclusion

The hardware side of the project was surprisingly easy to implement – I did not deviate from the modem setup notes in the MX614 datasheet. Software was significantly more difficult because it required intimate knowledge of numerous protocols (NMEA, AX.25, and APRS). However, the final project worked exactly as I had hoped!

Pitfalls and Solutions

I encountered the following hurdles while developing the software.

- **AX.25 uses NRZI.**
  I had read this in the documentation, but made the false assumption that the Bell 202 protocol (which the MX614 is compatible with) includes NRZI. I spent many hours struggling before I realized my mistake. This can easily be checked by transmitting continuous zeros and monitoring the audio output – it should be a mixed 1200/2200 Hz tone, not a pure tone.
- **Consumer modems reject packets with invalid checksums.**
  By default, store-bought modems will reject any packets with invalid checksums. One is unlikely to get the design perfect on the first try, so it is useful, if using a TNC-2 compatible modem for testing, to enable the "PASSALL" mode, which ignores the checksum completely.
- **Use the serial port to debug the packet generator.**
  I inserted code to send a zero or one to the serial port when a zero or one is supposed to be transmitted by the modem. Of course, this will slow the packet down and make the AFSK unintelligible, but it's a good way to make sure that the flags are being assembled correctly (especially bitstuffing).

Future Consideration

In the future, I would like to make the following changes to the design:

- **Replace the microcontroller with a smaller, cheaper one.**
  The 8515 is overkill for this project (only 9 of the 32 IOs are used). The 4433 about half as much and is half the size (with half the IOs).
- **Eliminate the STK-500.**
  The development board is expensive, bulky, and was never intended to be used in a production design. Eliminating will be fairly simple. The modem already has a crystal that can be used to generate the main processor clock. A TTL to RS-232 level converter will need to be added, and also a power supply circuit.
- **Make the device serial-port programmable.**
  The callsign, repeater path, etc. must be burned into the flash memory. Since the AVR has built-in EEPROM and since a serial port is already used for receiving

GPS data, it would be worthwhile to make the callsign, path, etc. customizable through a simple serial port interface.

References

The following references were helpful in building this project.

?? **Atmel AT0S8515 Datasheet** (included as appendix)
http://64.71.159.17/atmel/acrobat/doc1195.pdf

?? **MX-COM MX614 Data bulletin** (included as appendix)
http://www.mxcom.com/data_bulletins/db614-4.pdf

?? **APRS Protocol Specification Version 1.0.1**
ftp://ftp.tapr.org/aprssig/aprsspec/spec/aprs101/APRS101.pdf

?? **AX.25 Amateur Packet-Radio Link-Layer Protocol, Ver 2.2, Nov 1997**
http://www.tapr.org/tapr/html/Fax25.html

?? **The NMEA FAQ, Version 6.3, April 2000**
http://vancouver-webpages.com/peter/nmeafaq.txt

?? **Carl Ott, N2RVQ**
Provided technical advice and FCS generation routines.

?? **Byon Garrabrant, N6BG**
Provided sample source for the PIC microcontroller.

Appendices

The following documents are attached as appendices:

?? **Project Source Code** (ModemTest.asm is the main program file)

?? **Atmel AT0S8515 Datasheet**

?? **MX-COM MX614 Data bulletin**