Station Automation Symposium Code Handout
Mid-Atlantic States VHF Conference
Friday October 6, 2017

Roger Rehr, W3SZ
August 23, 2017

Arduino VHFLog / RoverLog Bandswitch Code

```
/*
Program to interface between VHFLOG contest logger and the Kenwood TS2000.

Developed by WA3DRC starting 1/3/2017.  Adapted for bandswitching by W3SZ 7/2017.

This program is meant to intercept commands from VHFLOG and provide transverter
    bandswitching. VHFLOG thinks it is talking
to a CANAKIT board to switch bands. https://www.canakit.com/Media/Manuals/UK1104.pdf

W3KM sends the first two digits of the new band with each band change in VHFLOG:

 */

//include string handling library
#include <string.h>

//define variables
String commandInputString = "";         // input buffer string to hold incoming data
boolean commandStringComplete = false;  // true when the input string is complete
String command = "";  // incoming data string for parsing

boolean hwCR = false;  // true if '\r' has been received

//define constant pin aliases
const int Pin50 = 2; //number of 50 MHz pin
const int Pin144 = 3; //number of 144 MHz pin
const int Pin222 = 4; //number of 222 MHz pin
const int Pin432 = 5; //number of 432 MHz pin
const int Pin902 = 6; //number of 902 MHz pin
const int Pin1296 = 8; //number of 1296 MHz pin
const int Pin2304 = A5; //number of 2304 MHz pin
const int Pin3G = A4; //number of 3GHz pin
const int Pin5G = A3; //number of 5GHz pin
const int Pin10G = A2; //number of 10GHz pin
const int Pin24G = A1; //number of 24GHz pin
const int Pin47G = A0; //number of 47GHz pin
const int Pin76G = 7; //number of 76GHz pin

void setup() {

// define GPIO pins as output pins
pinMode(Pin50,OUTPUT);
pinMode(Pin144,OUTPUT);
pinMode(Pin222,OUTPUT);
pinMode(Pin432,OUTPUT);
pinMode(Pin902,OUTPUT);
pinMode(Pin1296,OUTPUT);
pinMode(Pin2304,OUTPUT);
pinMode(Pin3G,OUTPUT);
pinMode(Pin5G,OUTPUT);
pinMode(Pin10G,OUTPUT);
pinMode(Pin24G,OUTPUT);
pinMode(Pin47G,OUTPUT);
pinMode(Pin76G,OUTPUT);

//initialize all GPIO pin values to low
digitalWrite(Pin50,LOW);
digitalWrite(Pin144,LOW);
digitalWrite(Pin222,LOW);
digitalWrite(Pin432,LOW);
digitalWrite(Pin902,LOW);
digitalWrite(Pin1296,LOW);
digitalWrite(Pin2304,LOW);
digitalWrite(Pin3G,LOW);
digitalWrite(Pin5G,LOW);
digitalWrite(Pin10G,LOW);
```

```
 66: digitalWrite(Pin24G,LOW);
 67: digitalWrite(Pin47G,LOW);
 68: digitalWrite(Pin76G,LOW);
 69:
 70:  // define, start, flush serial port Serial 0
 71:  // VHF log will send commands to this port
 72:   Serial.begin(9600, SERIAL_8N1); // 9600/8/N/1
 73:   Serial.println("VHFLog/RoverLog Bandswitch");
 74:   Serial.println("By W3SZ");
 75:   Serial.println("Uses USB-Serial Port");
 76:   Serial.println("50 MHz thru 76 GHz");
 77:   delay(100);
 78:
 79:   Serial.flush(); // clear buffers
 80: }
 81:
 82: void loop() { //MAIN
 83:
 84: ///////////////// Get the Command ////////////////////////////////////////////////
 85:   // get VHFLOG command from serial0
 86:   if (commandStringComplete) {
 87:    command = commandInputString;
 88:     // save this new command then clear the input buffer
 89:     // clear the string:
 90:     commandInputString = "";
 91:     //set string complete flag to false in preparation for next VHFLOG command;
 92:     commandStringComplete = false;
 93:   }
 94: ///////////////// End Command ////////////////////////////////////////////////////
 95: // now process the VHFLOG command
 96:   if (command.length() > 0){
 97: /////////////// Commands ////////////////////////////////////////////////////////
 98:
 99:     if (command.startsWith("50")) { // set band to 6m
100:           //set Pin50 high, all other pins low
101: digitalWrite(Pin50,HIGH);
102: digitalWrite(Pin144,LOW);
103: digitalWrite(Pin222,LOW);
104: digitalWrite(Pin432,LOW);
105: digitalWrite(Pin902,LOW);
106: digitalWrite(Pin1296,LOW);
107: digitalWrite(Pin2304,LOW);
108: digitalWrite(Pin3G,LOW);
109: digitalWrite(Pin5G,LOW);
110: digitalWrite(Pin10G,LOW);
111: digitalWrite(Pin24G,LOW);
112: digitalWrite(Pin47G,LOW);
113: digitalWrite(Pin76G,LOW);
114:     }
115:
116:    else if (command.startsWith("14")) { // set band to 2m
117:           //set Pin144 high, all other pins low
118: digitalWrite(Pin50,LOW);
119: digitalWrite(Pin144,HIGH);
120: digitalWrite(Pin222,LOW);
121: digitalWrite(Pin432,LOW);
122: digitalWrite(Pin902,LOW);
123: digitalWrite(Pin1296,LOW);
124: digitalWrite(Pin2304,LOW);
125: digitalWrite(Pin3G,LOW);
126: digitalWrite(Pin5G,LOW);
127: digitalWrite(Pin10G,LOW);
128: digitalWrite(Pin24G,LOW);
129: digitalWrite(Pin47G,LOW);
130: digitalWrite(Pin76G,LOW);
131:     }
```

```
132:
133:     else if (command.startsWith("22")) { // set band to 222
134:           //set Pin222 high, all other pins low
135: digitalWrite(Pin50,LOW);
136: digitalWrite(Pin144,LOW);
137: digitalWrite(Pin222,HIGH);
138: digitalWrite(Pin432,LOW);
139: digitalWrite(Pin902,LOW);
140: digitalWrite(Pin1296,LOW);
141: digitalWrite(Pin2304,LOW);
142: digitalWrite(Pin3G,LOW);
143: digitalWrite(Pin5G,LOW);
144: digitalWrite(Pin10G,LOW);
145: digitalWrite(Pin24G,LOW);
146: digitalWrite(Pin47G,LOW);
147: digitalWrite(Pin76G,LOW);
148:     }
149:
150:     else if (command.startsWith("43")) { // set band to 432
151:           //set Pin432 high, all other pins low
152: digitalWrite(Pin50,LOW);
153: digitalWrite(Pin144,LOW);
154: digitalWrite(Pin222,LOW);
155: digitalWrite(Pin432,HIGH);
156: digitalWrite(Pin902,LOW);
157: digitalWrite(Pin1296,LOW);
158: digitalWrite(Pin2304,LOW);
159: digitalWrite(Pin3G,LOW);
160: digitalWrite(Pin5G,LOW);
161: digitalWrite(Pin10G,LOW);
162: digitalWrite(Pin24G,LOW);
163: digitalWrite(Pin47G,LOW);
164: digitalWrite(Pin76G,LOW);
165:     }
166:
167:     else if (command.startsWith("90")) { // set band to 903
168:           //set Pin902 high, all other pins low
169: digitalWrite(Pin50,LOW);
170: digitalWrite(Pin144,LOW);
171: digitalWrite(Pin222,LOW);
172: digitalWrite(Pin432,LOW);
173: digitalWrite(Pin902,HIGH);
174: digitalWrite(Pin1296,LOW);
175: digitalWrite(Pin2304,LOW);
176: digitalWrite(Pin3G,LOW);
177: digitalWrite(Pin5G,LOW);
178: digitalWrite(Pin10G,LOW);
179: digitalWrite(Pin24G,LOW);
180: digitalWrite(Pin47G,LOW);
181: digitalWrite(Pin76G,LOW);
182:     }
183:
184:     else if (command.startsWith("12")) { // set band to 1296
185:           //set Pin1296 high, all other pins low
186: digitalWrite(Pin50,LOW);
187: digitalWrite(Pin144,LOW);
188: digitalWrite(Pin222,LOW);
189: digitalWrite(Pin432,LOW);
190: digitalWrite(Pin902,LOW);
191: digitalWrite(Pin1296,HIGH);
192: digitalWrite(Pin2304,LOW);
193: digitalWrite(Pin3G,LOW);
194: digitalWrite(Pin5G,LOW);
195: digitalWrite(Pin10G,LOW);
196: digitalWrite(Pin24G,LOW);
197: digitalWrite(Pin47G,LOW);
```

```
198: digitalWrite(Pin76G,LOW);
199:     }
200:
201:     else if (command.startsWith("23")) { // set band to 2304
202:           //set Pin2304 high, all other pins low
203: digitalWrite(Pin50,LOW);
204: digitalWrite(Pin144,LOW);
205: digitalWrite(Pin222,LOW);
206: digitalWrite(Pin432,LOW);
207: digitalWrite(Pin902,LOW);
208: digitalWrite(Pin1296,LOW);
209: digitalWrite(Pin2304,HIGH);
210: digitalWrite(Pin3G,LOW);
211: digitalWrite(Pin5G,LOW);
212: digitalWrite(Pin10G,LOW);
213: digitalWrite(Pin24G,LOW);
214: digitalWrite(Pin47G,LOW);
215: digitalWrite(Pin76G,LOW);
216:     }
217:
218:     else if (command.startsWith("34")) { // set band to 3456
219:           //set Pin3G high, all other pins low
220: digitalWrite(Pin50,LOW);
221: digitalWrite(Pin144,LOW);
222: digitalWrite(Pin222,LOW);
223: digitalWrite(Pin432,LOW);
224: digitalWrite(Pin902,LOW);
225: digitalWrite(Pin1296,LOW);
226: digitalWrite(Pin2304,LOW);
227: digitalWrite(Pin3G,HIGH);
228: digitalWrite(Pin5G,LOW);
229: digitalWrite(Pin10G,LOW);
230: digitalWrite(Pin24G,LOW);
231: digitalWrite(Pin47G,LOW);
232: digitalWrite(Pin76G,LOW);
233:     }
234:
235:     else if (command.startsWith("57")) { // set band to 5760
236:           //set Pin5G high, all other pins low
237: digitalWrite(Pin50,LOW);
238: digitalWrite(Pin144,LOW);
239: digitalWrite(Pin222,LOW);
240: digitalWrite(Pin432,LOW);
241: digitalWrite(Pin902,LOW);
242: digitalWrite(Pin1296,LOW);
243: digitalWrite(Pin2304,LOW);
244: digitalWrite(Pin3G,LOW);
245: digitalWrite(Pin5G,HIGH);
246: digitalWrite(Pin10G,LOW);
247: digitalWrite(Pin24G,LOW);
248: digitalWrite(Pin47G,LOW);
249: digitalWrite(Pin76G,LOW);
250:     }
251:
252:     else if (command.startsWith("10")) { // set band to 10368
253:           //set Pin10G high, all other pins low
254: digitalWrite(Pin50,LOW);
255: digitalWrite(Pin144,LOW);
256: digitalWrite(Pin222,LOW);
257: digitalWrite(Pin432,LOW);
258: digitalWrite(Pin902,LOW);
259: digitalWrite(Pin1296,LOW);
260: digitalWrite(Pin2304,LOW);
261: digitalWrite(Pin3G,LOW);
262: digitalWrite(Pin5G,LOW);
263: digitalWrite(Pin10G,HIGH);
```

```
digitalWrite(Pin24G,LOW);
digitalWrite(Pin47G,LOW);
digitalWrite(Pin76G,LOW);
    }

    else if (command.startsWith("24")) { // set band to 24 GHz
          //set Pin24G high, all other pins low
digitalWrite(Pin50,LOW);
digitalWrite(Pin144,LOW);
digitalWrite(Pin222,LOW);
digitalWrite(Pin432,LOW);
digitalWrite(Pin902,LOW);
digitalWrite(Pin1296,LOW);
digitalWrite(Pin2304,LOW);
digitalWrite(Pin3G,LOW);
digitalWrite(Pin5G,LOW);
digitalWrite(Pin10G,LOW);
digitalWrite(Pin24G,HIGH);
digitalWrite(Pin47G,LOW);
digitalWrite(Pin76G,LOW);
    }

    else if (command.startsWith("47")) { // set band 47 GHz
          //set Pin47G high, all other pins low
digitalWrite(Pin50,LOW);
digitalWrite(Pin144,LOW);
digitalWrite(Pin222,LOW);
digitalWrite(Pin432,LOW);
digitalWrite(Pin902,LOW);
digitalWrite(Pin1296,LOW);
digitalWrite(Pin2304,LOW);
digitalWrite(Pin3G,LOW);
digitalWrite(Pin5G,LOW);
digitalWrite(Pin10G,LOW);
digitalWrite(Pin24G,LOW);
digitalWrite(Pin47G,HIGH);
digitalWrite(Pin76G,LOW);
    }

    else if (command.startsWith("76")) { // set band to 76 GHz
          //set Pin76G high, all other pins low
digitalWrite(Pin50,LOW);
digitalWrite(Pin144,LOW);
digitalWrite(Pin222,LOW);
digitalWrite(Pin432,LOW);
digitalWrite(Pin902,LOW);
digitalWrite(Pin1296,LOW);
digitalWrite(Pin2304,LOW);
digitalWrite(Pin3G,LOW);
digitalWrite(Pin5G,LOW);
digitalWrite(Pin10G,LOW);
digitalWrite(Pin24G,LOW);
digitalWrite(Pin47G,LOW);
digitalWrite(Pin76G,HIGH);
    }
    // cleanup
    command = ""; // clear the VHFLOG command
  }
/////////////// END COMMANDS ////////////////////////////////////////////////////


  delay(25); //  long enough for the radio to return its frequency

} //END MAIN


```

```
/*
  SerialEvent occurs whenever a new data comes in the
 hardware serial RX.  This routine is run between each
 time loop() runs, so using  inside loop can
 response.  Multiple bytes of data may be available.
 */
void serialEvent() {

  char commandInChar;

  while (Serial.available()) { // interrupt generated by hardware serial port
     // get the new byte:
     commandInChar = (char)Serial.read();

   // add it to the commandInputString:
   commandInputString += commandInChar;  // append

   // look for a carriage return, then a line feed; set a flag
   // so the main loop can do something about it:
   if (commandInChar == '\r') { // the commands all end with a CR and then a LF (13 10
       )
     hwCR = true;
   }
   if ( commandInChar == '\n') {
     if ( hwCR ) {
       hwCR = false; // cleanup
       commandStringComplete = true;
     }
   }
  }
 }
```

BeagleBone Black N1MM Ethernet Bandswitch Code

```
  1: # import libraries
  2: import socket
  3: import Adafruit_BBIO.GPIO as GPIO
  4:
  5: #define variables
  6: oldband = " "
  7:
  8: # define GPIO pin aliases
  9:
 10: # define GPIO pin aliases
 11: PIN50 = "P9_12"
 12: PIN144 = "P9_18"
 13: PIN222 = "P9_24"
 14: PIN432 = "P9_30"
 15: PIN902 = "P9_31"
 16: PIN1296 = "P9_42"
 17: PIN2304 = "P8_9"
 18: PIN3456 = "P8_15"
 19: PIN5760 = "P8_18"
 20: PIN10G = "P8_27"
 21: PIN24G = "P8_33"
 22: PIN47G = "P8_39"
 23:
 24: #setup and initialize GPIO pins
 25: GPIO.setup(PIN50, GPIO.OUT)
 26: GPIO.output(PIN50, GPIO.LOW)
 27: GPIO.setup(PIN144, GPIO.OUT)
 28: GPIO.output(PIN144, GPIO.LOW)
 29: GPIO.setup(PIN222, GPIO.OUT)
 30: GPIO.output(PIN222, GPIO.LOW)
 31: GPIO.setup(PIN432, GPIO.OUT)
 32: GPIO.output(PIN432, GPIO.LOW)
 33: GPIO.setup(PIN902, GPIO.OUT)
 34: GPIO.output(PIN902, GPIO.LOW)
 35: GPIO.setup(PIN1296, GPIO.OUT)
 36: GPIO.output(PIN1296, GPIO.LOW)
 37: GPIO.setup(PIN2304, GPIO.OUT)
 38: GPIO.output(PIN2304, GPIO.LOW)
 39: GPIO.setup(PIN3456, GPIO.OUT)
 40: GPIO.output(PIN3456, GPIO.LOW)
 41: GPIO.setup(PIN5760, GPIO.OUT)
 42: GPIO.output(PIN5760, GPIO.LOW)
 43: GPIO.setup(PIN10G, GPIO.OUT)
 44: GPIO.output(PIN10G, GPIO.LOW)
 45: GPIO.setup(PIN24G, GPIO.OUT)
 46: GPIO.output(PIN24G, GPIO.LOW)
 47: GPIO.setup(PIN47G, GPIO.OUT)
 48: GPIO.output(PIN47G, GPIO.LOW)
 49:
 50: # setup ethernet UPD socket and start UDP server
 51: port = 13063
 52: s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
 53: s.bind(("",port))
 54: print "waiting on port:", port
 55:
 56: #start loop to receive UDP packets from N1MM
 57: while 1:
 58:     data, addr =  s.recvfrom(1024)
 59:
 60:     #parse incoming data
 61:     str1 = "<RadioNr>1</RadioNr>"
 62:     str2 = "<Freq>"
 63:     pos1 = data.find(str1)
 64:     pos2 = data.find(str2)
 65:
 66:     #if have valid XML data for N1MM Radio 1 then set band
```

```
 67:     if pos1 >= 0:
 68:         band = data[pos2+6:pos2+8]
 69:
 70:         #start band switch code
 71:         if band == "50" and band != oldband:
 72:             print "found 50"
 73:             oldband = "50"
 74:             GPIO.output(PIN50, GPIO.HIGH)
 75:             GPIO.output(PIN144, GPIO.LOW)
 76:             GPIO.output(PIN222, GPIO.LOW)
 77:             GPIO.output(PIN432, GPIO.LOW)
 78:             GPIO.output(PIN902, GPIO.LOW)
 79:             GPIO.output(PIN1296, GPIO.LOW)
 80:             GPIO.output(PIN2304, GPIO.LOW)
 81:             GPIO.output(PIN3456, GPIO.LOW)
 82:             GPIO.output(PIN5760, GPIO.LOW)
 83:             GPIO.output(PIN10G, GPIO.LOW)
 84:             GPIO.output(PIN24G, GPIO.LOW)
 85:             GPIO.output(PIN47G, GPIO.LOW)
 86:         elif band =="14" and band != oldband:
 87:             print "found 144"
 88:             oldband = "14"
 89:             GPIO.output(PIN50, GPIO.LOW)
 90:             GPIO.output(PIN144, GPIO.HIGH)
 91:             GPIO.output(PIN222, GPIO.LOW)
 92:             GPIO.output(PIN432, GPIO.LOW)
 93:             GPIO.output(PIN902, GPIO.LOW)
 94:             GPIO.output(PIN1296, GPIO.LOW)
 95:             GPIO.output(PIN2304, GPIO.LOW)
 96:             GPIO.output(PIN3456, GPIO.LOW)
 97:             GPIO.output(PIN5760, GPIO.LOW)
 98:             GPIO.output(PIN10G, GPIO.LOW)
 99:             GPIO.output(PIN24G, GPIO.LOW)
100:             GPIO.output(PIN47G, GPIO.LOW)
101:         elif band =="22" and band != oldband:
102:             print "found 222"
103:             oldband = "22"
104:             GPIO.output(PIN50, GPIO.LOW)
105:             GPIO.output(PIN144, GPIO.LOW)
106:             GPIO.output(PIN222, GPIO.HIGH)
107:             GPIO.output(PIN432, GPIO.LOW)
108:             GPIO.output(PIN902, GPIO.LOW)
109:             GPIO.output(PIN1296, GPIO.LOW)
110:             GPIO.output(PIN2304, GPIO.LOW)
111:             GPIO.output(PIN3456, GPIO.LOW)
112:             GPIO.output(PIN5760, GPIO.LOW)
113:             GPIO.output(PIN10G, GPIO.LOW)
114:             GPIO.output(PIN24G, GPIO.LOW)
115:             GPIO.output(PIN47G, GPIO.LOW)
116:         elif band =="43" and band != oldband:
117:             print "found 432"
118:             oldband = "43"
119:             GPIO.output(PIN50, GPIO.LOW)
120:             GPIO.output(PIN144, GPIO.LOW)
121:             GPIO.output(PIN222, GPIO.LOW)
122:             GPIO.output(PIN432, GPIO.HIGH)
123:             GPIO.output(PIN902, GPIO.LOW)
124:             GPIO.output(PIN1296, GPIO.LOW)
125:             GPIO.output(PIN2304, GPIO.LOW)
126:             GPIO.output(PIN3456, GPIO.LOW)
127:             GPIO.output(PIN5760, GPIO.LOW)
128:             GPIO.output(PIN10G, GPIO.LOW)
129:             GPIO.output(PIN24G, GPIO.LOW)
130:             GPIO.output(PIN47G, GPIO.LOW)
131:         elif band =="90" and band != oldband:
132:             print "found 902"
```

```
133:             oldband = "90"
134:             GPIO.output(PIN50, GPIO.LOW)
135:             GPIO.output(PIN144, GPIO.LOW)
136:             GPIO.output(PIN222, GPIO.LOW)
137:             GPIO.output(PIN432, GPIO.LOW)
138:             GPIO.output(PIN902, GPIO.HIGH)
139:             GPIO.output(PIN1296, GPIO.LOW)
140:             GPIO.output(PIN2304, GPIO.LOW)
141:             GPIO.output(PIN3456, GPIO.LOW)
142:             GPIO.output(PIN5760, GPIO.LOW)
143:             GPIO.output(PIN10G, GPIO.LOW)
144:             GPIO.output(PIN24G, GPIO.LOW)
145:             GPIO.output(PIN47G, GPIO.LOW)
146:         elif band =="12" and band != oldband:
147:             print "found 1296"
148:             oldband = "12"
149:             GPIO.output(PIN50, GPIO.LOW)
150:             GPIO.output(PIN144, GPIO.LOW)
151:             GPIO.output(PIN222, GPIO.LOW)
152:             GPIO.output(PIN432, GPIO.LOW)
153:             GPIO.output(PIN902, GPIO.LOW)
154:             GPIO.output(PIN1296, GPIO.HIGH)
155:             GPIO.output(PIN2304, GPIO.LOW)
156:             GPIO.output(PIN3456, GPIO.LOW)
157:             GPIO.output(PIN5760, GPIO.LOW)
158:             GPIO.output(PIN10G, GPIO.LOW)
159:             GPIO.output(PIN24G, GPIO.LOW)
160:             GPIO.output(PIN47G, GPIO.LOW)
161:         elif band =="23" and band != oldband:
162:             print "found 2304"
163:             oldband = "23"
164:             GPIO.output(PIN50, GPIO.LOW)
165:             GPIO.output(PIN144, GPIO.LOW)
166:             GPIO.output(PIN222, GPIO.LOW)
167:             GPIO.output(PIN432, GPIO.LOW)
168:             GPIO.output(PIN902, GPIO.LOW)
169:             GPIO.output(PIN1296, GPIO.LOW)
170:             GPIO.output(PIN2304, GPIO.HIGH)
171:             GPIO.output(PIN3456, GPIO.LOW)
172:             GPIO.output(PIN5760, GPIO.LOW)
173:             GPIO.output(PIN10G, GPIO.LOW)
174:             GPIO.output(PIN24G, GPIO.LOW)
175:             GPIO.output(PIN47G, GPIO.LOW)
176:         elif band =="34" and band != oldband:
177:             print "found 3456"
178:             oldband = "34"
179:             GPIO.output(PIN50, GPIO.LOW)
180:             GPIO.output(PIN144, GPIO.LOW)
181:             GPIO.output(PIN222, GPIO.LOW)
182:             GPIO.output(PIN432, GPIO.LOW)
183:             GPIO.output(PIN902, GPIO.LOW)
184:             GPIO.output(PIN1296, GPIO.LOW)
185:             GPIO.output(PIN2304, GPIO.LOW)
186:             GPIO.output(PIN3456, GPIO.HIGH)
187:             GPIO.output(PIN5760, GPIO.LOW)
188:             GPIO.output(PIN10G, GPIO.LOW)
189:             GPIO.output(PIN24G, GPIO.LOW)
190:             GPIO.output(PIN47G, GPIO.LOW)
191:         elif band =="57" and band != oldband:
192:             print "found 5760"
193:             oldband = "57"
194:             GPIO.output(PIN50, GPIO.LOW)
195:             GPIO.output(PIN144, GPIO.LOW)
196:             GPIO.output(PIN222, GPIO.LOW)
197:             GPIO.output(PIN432, GPIO.LOW)
198:             GPIO.output(PIN902, GPIO.LOW)
```

```
199:             GPIO.output(PIN1296, GPIO.LOW)
200:             GPIO.output(PIN2304, GPIO.LOW)
201:             GPIO.output(PIN3456, GPIO.LOW)
202:             GPIO.output(PIN5760, GPIO.HIGH)
203:             GPIO.output(PIN10G, GPIO.LOW)
204:             GPIO.output(PIN24G, GPIO.LOW)
205:             GPIO.output(PIN47G, GPIO.LOW)
206:         elif band =="10" and band != oldband:
207:             print "found 10 GHz"
208:             oldband = "10"
209:             GPIO.output(PIN50, GPIO.LOW)
210:             GPIO.output(PIN144, GPIO.LOW)
211:             GPIO.output(PIN222, GPIO.LOW)
212:             GPIO.output(PIN432, GPIO.LOW)
213:             GPIO.output(PIN902, GPIO.LOW)
214:             GPIO.output(PIN1296, GPIO.LOW)
215:             GPIO.output(PIN2304, GPIO.LOW)
216:             GPIO.output(PIN3456, GPIO.LOW)
217:             GPIO.output(PIN5760, GPIO.LOW)
218:             GPIO.output(PIN10G, GPIO.HIGH)
219:             GPIO.output(PIN24G, GPIO.LOW)
220:             GPIO.output(PIN47G, GPIO.LOW)
221:         elif band =="24" and band != oldband:
222:             print "found 24 GHz"
223:             oldband = "24"
224:             GPIO.output(PIN50, GPIO.LOW)
225:             GPIO.output(PIN144, GPIO.LOW)
226:             GPIO.output(PIN222, GPIO.LOW)
227:             GPIO.output(PIN432, GPIO.LOW)
228:             GPIO.output(PIN902, GPIO.LOW)
229:             GPIO.output(PIN1296, GPIO.LOW)
230:             GPIO.output(PIN2304, GPIO.LOW)
231:             GPIO.output(PIN3456, GPIO.LOW)
232:             GPIO.output(PIN5760, GPIO.LOW)
233:             GPIO.output(PIN10G, GPIO.LOW)
234:             GPIO.output(PIN24G, GPIO.HIGH)
235:             GPIO.output(PIN47G, GPIO.LOW)
236:         elif band =="47" and band != oldband:
237:             print "found 47 GHz"
238:             oldband = "47"
239:             GPIO.output(PIN50, GPIO.LOW)
240:             GPIO.output(PIN144, GPIO.LOW)
241:             GPIO.output(PIN222, GPIO.LOW)
242:             GPIO.output(PIN432, GPIO.LOW)
243:             GPIO.output(PIN902, GPIO.LOW)
244:             GPIO.output(PIN1296, GPIO.LOW)
245:             GPIO.output(PIN2304, GPIO.LOW)
246:             GPIO.output(PIN3456, GPIO.LOW)
247:             GPIO.output(PIN5760, GPIO.LOW)
248:             GPIO.output(PIN10G, GPIO.LOW)
249:             GPIO.output(PIN24G, GPIO.LOW)
250:             GPIO.output(PIN47G, GPIO.HIGH)
```

Propeller Radio Device Bandswitching Code

```
  1: ''RadioManager.spin
  2: '' This is the top object for the Propeller Radio Controller interface.
  3: '' It is meant to run over a USB port, getting its instructions from the Apple Mac Pro
  4: ''
  5:
  6: CON
  7:     _clkmode = xtal1 + pll16x
  8:     _xinfreq = 5_000_000
  9:
 10: VAR
 11: BYTE myStr
 12: byte LF
 13: byte RF
 14: byte MICR
 15: byte KEYR
 16: byte STRT
 17: byte stay1
 18:
 19: OBJ
 20:     Debu : "LED_DEMO_Extended_FDSerial"
 21:
 22:
 23: PUB LedTEST
 24:     DIRA[00..29]~~
 25:     OUTA[00..29] := 0
 26:
 27:     Debu.start(31, 30, 0, 19200)
 28:     waitcnt(clkfreq*2 + cnt)
 29:
 30:     Debu.rxflush
 31:
 32: repeat
 33:   Debu.rxflush
 34:   Debu.str(string("Start Data Acquisition",10,13))
 35:     repeat until STRT == 49
 36:          Debu.str(string("Enter STRT",10,13))
 37:          STRT := Debu.rx
 38: '        OUTA[23] := 1
 39:          Debu.tx(STRT)
 40:          Debu.str(string("  equals STRT",10,13,10,13))
 41:          if STRT == 49
 42: '              OUTA[23] := 1
 43:          else
 44:               Debu.str(string("Wrong initialization constant",10,13,10,13))
 45:
 46:     STRT := 0
 47:     Debu.str(string("Enter LF",10,13))
 48:     OUTA[00..06] := 0
 49:     repeat
 50:           LF := Debu.rx
 51:     while LF <49 or LF >55
 52:     if LF == 49
 53:           OUTA[00] := 1
 54: '         OUTA[23] := 1
 55:     elseif LF ==50
 56:           OUTA[01] := 1
 57:     elseif LF ==51
 58:           OUTA[02] := 1
 59:     elseif LF ==52
 60:           OUTA[03] := 1
 61:     elseif LF ==53
 62:           OUTA[04] := 1
 63:     elseif LF ==54
 64:           OUTA[05] := 1
 65:     elseif LF ==55
 66:           OUTA[06] := 1
```

```
  67:           Debu.tx(LF)
  68:           Debu.str(string("   equals LF ",10,13,10,13))
  69:     elseif LF == 255
  70:           Debu.str(string(" No data received for LF ",10,13,10,13))
  71:     Debu.tx(LF)
  72:     Debu.str(string("   equals LF ",10,13,10,13))
  73: '       OUTA[23] := 0
  74:
  75:     Debu.str(string("Enter RF",10,13))
  76: '   RF := Debu.rxDecTime(2000)
  77:     repeat
  78:           RF := Debu.rx
  79:     while RF < 49 or RF > 55
  80:     OUTA[07..13] := 0
  81:     if RF == 49
  82:           OUTA[07] := 1
  83: '             OUTA[23] := 1
  84:     elseif RF ==50
  85:           OUTA[08] := 1
  86:     elseif RF ==51
  87:           OUTA[09] := 1
  88:     elseif RF ==52
  89:           OUTA[10] := 1
  90:     elseif RF ==53
  91:           OUTA[11] := 1
  92:     elseif RF ==54
  93:           OUTA[12] := 1
  94:     elseif RF ==55
  95:           OUTA[13] := 1
  96:           Debu.dec(RF)
  97:           Debu.str(string("   equals RF ",10,13,10,13))
  98:     elseif RF == 255
  99:           Debu.str(string(" No data received for RF ",10,13,10,13))
 100:
 101:     Debu.tx(RF)
 102:     Debu.str(string("   equals RF ",10,13,10,13))
 103: '          OUTA[23] := 0
 104:
 105:     Debu.str(string("Enter MICR",10,13))
 106: '    MICR := Debu.rxDecTime(2000)
 107:     repeat
 108:           MICR := Debu.rx
 109:     while MICR < 49 or MICR > 55
 110:     OUTA[14..20] := 0
 111:     if MICR == 49
 112:           OUTA[14] := 1
 113:     elseif MICR ==50
 114:           OUTA[15] := 1
 115:     elseif MICR ==51
 116:           OUTA[16] := 1
 117:     elseif MICR ==52
 118:           OUTA[17] := 1
 119:     elseif MICR ==53
 120:           OUTA[18] := 1
 121:     elseif MICR ==54
 122:           OUTA[19] := 1
 123:     elseif MICR ==55
 124:           OUTA[20] := 1
 125:     elseif MICR == 255
 126:        Debu.str(string(" No data received for MICR ",10,13,10,13))
 127:
 128:     Debu.tx(MICR)
 129:     Debu.str(string("   equals MICR ",10,13,10,13))
 130: '       OUTA[23] := 0
 131:
 132:     Debu.str(string("Enter KEYR",10,13))
```

```
133: '    KEYR := Debu.rxDecTime(2000)
134:     repeat
135:           KEYR := Debu.rx
136:     while KEYR < 49 or KEYR > 55
137:     OUTA[21..27] := 0
138:     if KEYR == 49
139:           OUTA[21] := 1
140: '             OUTA[23] := 1
141:     elseif KEYR ==50
142:           OUTA[22] := 1
143:     elseif KEYR ==51
144:           OUTA[23] := 1
145:     elseif KEYR ==52
146:           OUTA[24] := 1
147:     elseif KEYR ==53
148:           OUTA[25] := 1
149:     elseif KEYR ==54
150:           OUTA[26] := 1
151:     elseif KEYR ==55
152:           OUTA[27] := 1
153:           Debu.dec(KEYR)
154:           Debu.str(string("   equals KEYR ",10,13,10,13))
155:     elseif KEYR == 255
156:        Debu.str(string(" No data received for KEYR ",10,13,10,13))
157:
158:     Debu.tx(KEYR)
159:     Debu.str(string("   equals KEYR ",10,13,10,13))
160:     Debu.str(string("End Data Acquisition",10,13,10,13))
161:     Debu.rxflush
162: '        OUTA[23] := 0
163:
164:
```

Propeller Audio Device Bandswitching Code

```
''AudioController.spin
'' This is the top object for the Propeller Audio Controller interface.
'' It is meant to run over a USB port, getting its instructions from the Apple Mac Pro
''

CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000

VAR
BYTE myStr
byte LF
byte RF
byte MICR
byte KEYR
byte STRT
byte stay1

OBJ
    Debu : "LED_DEMO_Extended_FDSerial"


PUB LedTEST
    DIRA[00..29]~~
    OUTA[00..29] := 0

    Debu.start(31, 30, 0, 19200)
    waitcnt(clkfreq*2 + cnt)

    Debu.rxflush

repeat
  Debu.rxflush
  Debu.str(string("Start Data Acquisition",10,13))
    repeat until STRT == 48
         Debu.str(string("Enter STRT",10,13))
         STRT := Debu.rx
'        OUTA[23] := 1
         Debu.tx(STRT)
         Debu.str(string("  equals STRT",10,13,10,13))
         if STRT == 48
'              OUTA[23] := 1
         else
              Debu.str(string("Wrong initialization constant",10,13,10,13))

    STRT := 0
    Debu.str(string("Enter LF",10,13))
    OUTA[00..06] := 0
    repeat
          LF := Debu.rx
    while LF <48 or LF >56
    if LF == 49
          OUTA[00] := 1
'         OUTA[23] := 1
    elseif LF ==50
          OUTA[01] := 1
    elseif LF ==51
          OUTA[02] := 1
    elseif LF ==52
          OUTA[03] := 1
    elseif LF ==53
          OUTA[04] := 1
    elseif LF ==54
          OUTA[05] := 1
    elseif LF ==55
          OUTA[06] := 1
```

```
 67:     elseif LF ==56
 68:           OUTA[00..06] := 0
 69:           Debu.tx(LF)
 70:           Debu.str(string("   equals LF ",10,13,10,13))
 71:     elseif LF == 255
 72:           Debu.str(string(" No data received for LF ",10,13,10,13))
 73:     Debu.tx(LF)
 74:     Debu.str(string("   equals LF ",10,13,10,13))
 75: '       OUTA[23] := 0
 76:
 77:     Debu.str(string("Enter RF",10,13))
 78: '   RF := Debu.rxDecTime(2000)
 79:     repeat
 80:           RF := Debu.rx
 81:     while RF < 48 or RF > 56
 82:     OUTA[07..13] := 0
 83:     if RF == 49
 84:           OUTA[07] := 1
 85: '              OUTA[23] := 1
 86:     elseif RF ==50
 87:           OUTA[08] := 1
 88:     elseif RF ==51
 89:           OUTA[09] := 1
 90:     elseif RF ==52
 91:           OUTA[10] := 1
 92:     elseif RF ==53
 93:           OUTA[11] := 1
 94:     elseif RF ==54
 95:           OUTA[12] := 1
 96:     elseif RF ==55
 97:           OUTA[13] := 1
 98:     elseif LF ==56
 99:           OUTA[08..13] := 0
100:           Debu.dec(RF)
101:           Debu.str(string("   equals RF ",10,13,10,13))
102:     elseif RF == 255
103:           Debu.str(string(" No data received for RF ",10,13,10,13))
104:
105:     Debu.tx(RF)
106:     Debu.str(string("   equals RF ",10,13,10,13))
107: '          OUTA[23] := 0
108:
109:     Debu.str(string("Enter MICR",10,13))
110: '    MICR := Debu.rxDecTime(2000)
111:     repeat
112:           MICR := Debu.rx
113:     while MICR < 48 or MICR > 56
114:     OUTA[14..20] := 0
115:     if MICR == 49
116:           OUTA[14] := 1
117:     elseif MICR ==50
118:           OUTA[15] := 1
119:     elseif MICR ==51
120:           OUTA[16] := 1
121:     elseif MICR ==52
122:           OUTA[17] := 1
123:     elseif MICR ==53
124:           OUTA[18] := 1
125:     elseif MICR ==54
126:           OUTA[19] := 1
127:     elseif MICR ==55
128:           OUTA[20] := 1
129:     elseif LF ==56
130:           OUTA[14..20] := 0
131:     elseif MICR == 255
132:        Debu.str(string(" No data received for MICR ",10,13,10,13))
```

```
133:
134:     Debu.tx(MICR)
135:     Debu.str(string("   equals MICR ",10,13,10,13))
136: '       OUTA[23] := 0
137:
138:     Debu.str(string("End Data Acquisition",10,13,10,13))
139:     Debu.rxflush
140: '       OUTA[23] := 0
141:
142:
```

Arduino Ethernet Device Control Code

```
/*
  ETHERNET SWITCH
    BY ROGER REHR w3sz

  Ethernet shield connected to pins 10, 11, 12, 13
  This requires a MEGA as it used 3196 bytes of Dynamic Memory
*/

#include <Ethernet.h>  //for ethernet port
#include <string.h> // for string handling

String commandInputString = "";
String serIn;
String serOut1;
String serOut2;
String serOut3;
String serOut4;
String serOut5;
String serOut6;
String serOut7;
String serOut8;
String serOut9;
String serOut10;
String serOut11;
String serOut12;
String serOut13;
String serOut14;
String serOut15;
String serOut16;

// Enter MAC address and IP address for Arduino below.
// The IP address is dependent on your local network:
byte mac[] = { 0x90, 0xAA, 0xBB, 0xCC, 0xDA, 0x02 };
IPAddress ip(192, 168, 10, 176); //<< ENTER YOUR IP ADDRESS HERE <<

// Initialize the Ethernet server library
// We'll use port 80 for HTTP):
EthernetServer server(80);
EthernetClient client;

const int PinR1 = 2; //number of Relay 1 pin
const int PinR2 = 3; //number of Relay 2 pin
const int PinR3 = 4; //number of Relay 3 pin
const int PinR4 = 5; //number of Relay 4 pin
const int PinR5 = 6; //number of Relay 5 pin
const int PinR6 = 8; //number of Relay 6 pin
const int PinR7 = A5; //number of Relay 7 pin
const int PinR8 = A4; //number of Relay 8 pin
const int PinR9 = A3; //number of Relay 9 pin
const int PinR10 = A2; //number of Relay 10 pin
const int PinR11 = A1; //number of Relay 11 pin
const int PinR12 = A0; //number of Relay 12 pin
const int PinR13 = A8; //number of Relay 13 pin
const int PinR14 = A9; //number of Relay 14 pin
const int PinR15 = A10; //number of Relay 15 pin
const int PinR16 = A11; //number of Relay 16 pin

void setup()
{
  // initialize GPIO pins as output pins
  pinMode(PinR1, OUTPUT);
  pinMode(PinR2, OUTPUT);
  pinMode(PinR3, OUTPUT);
  pinMode(PinR4, OUTPUT);
  pinMode(PinR5, OUTPUT);
  pinMode(PinR6, OUTPUT);
```

```
  67:   pinMode(PinR7, OUTPUT);
  68:   pinMode(PinR8, OUTPUT);
  69:   pinMode(PinR9, OUTPUT);
  70:   pinMode(PinR10, OUTPUT);
  71:   pinMode(PinR11, OUTPUT);
  72:   pinMode(PinR12, OUTPUT);
  73:   pinMode(PinR13, OUTPUT);
  74:   pinMode(PinR14, OUTPUT);
  75:   pinMode(PinR15, OUTPUT);
  76:   pinMode(PinR16, OUTPUT);
  77:
  78:   //initialize all GPIO pin values to low
  79:   digitalWrite(PinR1, LOW);
  80:   digitalWrite(PinR2, LOW);
  81:   digitalWrite(PinR3, LOW);
  82:   digitalWrite(PinR4, LOW);
  83:   digitalWrite(PinR5, LOW);
  84:   digitalWrite(PinR6, LOW);
  85:   digitalWrite(PinR7, LOW);
  86:   digitalWrite(PinR8, LOW);
  87:   digitalWrite(PinR9, LOW);
  88:   digitalWrite(PinR10, LOW);
  89:   digitalWrite(PinR11, LOW);
  90:   digitalWrite(PinR12, LOW);
  91:   digitalWrite(PinR13, LOW);
  92:   digitalWrite(PinR14, LOW);
  93:   digitalWrite(PinR15, LOW);
  94:   digitalWrite(PinR16, LOW);
  95:
  96:   // start the Ethernet connection and the server and the serial port:
  97:   Ethernet.begin(mac, ip);
  98:   server.begin();
  99:   Serial.begin(9600);
 100:   Serial.println("Arduino Ethernet Device Switch");
 101:   Serial.println("by W3SZ");
 102:   Serial.println("Starting Server");
 103:   Serial.println (Ethernet.localIP());
 104:
 105:
 106: }
 107:
 108: //this routine reads the output pin values and reports them both through the
          serial port and to the HTML client
 109: //it also creates the HTML buttons on the web page and defines what is sent to
          the HTML server when each button is clicked
 110: void sendReply()
 111:   {
 112:
 113:     //read all output pin values
 114:             bool val = digitalRead(PinR1);
 115:             Serial.println(val);
 116:             if(val == 1)
 117:             {
 118:             serOut1 = "<input  type=button value = 'ON' style = 'background-color
                      :lime;'>";
 119:             }
 120:             else if (val == 0)
 121:             {
 122:             serOut1 = "<input  type=button value = 'OFF' style = 'background-
                      color:silver;'>";
 123:             }
 124:
 125:             val = digitalRead(PinR2);
 126:             Serial.println(val);
 127:             if(val == 1)
 128:             {
```

```
129:              serOut2 = "<input  type=button value = 'ON' style =  'background-
                      color:lime;'>";
130:              }
131:              else if(val == 0)
132:              {
133:              serOut2 = "<input  type=button value = 'OFF' style = 'background-
                      color:silver;'>";
134:              }
135:              val = digitalRead(PinR3);
136:              Serial.println(val);
137:              if(val == 1)
138:              {
139:              serOut3 = "<input  type=button value = 'ON' style = 'background-color
                      :lime;'>";
140:              }
141:              else if(val == 0)
142:              {
143:              serOut3 = "<input  type=button value = 'OFF' style = 'background-
                      color:silver;'>";
144:              }
145:              val = digitalRead(PinR4);
146:              Serial.println(val);
147:              if(val == 1)
148:              {
149:              serOut4 = "<input  type=button value = 'ON' style = 'background-color
                      :lime;'>";
150:              }
151:              else if(val == 0)
152:              {
153:              serOut4 = "<input  type=button value = 'OFF' style = 'background-
                      color:silver;'>";
154:              }
155:              val = digitalRead(PinR5);
156:              Serial.println(val);
157:              if(val == 1)
158:              {
159:              serOut5 = "<input  type=button value = 'ON' style = 'background-color
                      :lime;'>";
160:              }
161:              else if(val == 0)
162:              {
163:              serOut5 = "<input  type=button value = 'OFF' style = 'background-
                      color:silver;'>";
164:              }
165:              val = digitalRead(PinR6);
166:              Serial.println(val);
167:              if(val == 1)
168:              {
169:              serOut6 = "<input  type=button value = 'ON' style = 'background-color
                      :lime;'>";
170:              }
171:              else if(val == 0)
172:              {
173:              serOut6 = "<input  type=button value = 'OFF' style = 'background-
                      color:silver;'>";
174:              }
175:              val = digitalRead(PinR7);
176:              Serial.println(val);
177:              if(val == 1)
178:              {
179:              serOut7 = "<input  type=button value = 'ON' style = 'background-color
                      :lime;'>";
180:              }
181:              else if(val == 0)
182:              {
183:              serOut7 = "<input  type=button value = 'OFF' style = 'background-
```

```
                color:silver;'>";
184:            }
185:            val = digitalRead(PinR8);
186:            Serial.println(val);
187:            if(val == 1)
188:            {
189:            serOut8 = "<input  type=button value = 'ON' style = 'background-color
                    :lime;'>";
190:            }
191:            else if(val == 0)
192:            {
193:            serOut8 = "<input  type=button value = 'OFF' style = 'background-
                    color:silver;'>";
194:            }
195:            val = digitalRead(PinR9);
196:            Serial.println(val);
197:            if(val == 1)
198:            {
199:            serOut9 = "<input  type=button value = 'ON' style = 'background-color
                    :lime;'>";
200:            }
201:            else if(val == 0)
202:            {
203:            serOut9 = "<input  type=button value = 'OFF' style = 'background-
                    color:silver;'>";
204:            }
205:            val = digitalRead(PinR10);
206:            Serial.println(val);
207:            if(val == 1)
208:            {
209:            serOut10 = "<input  type=button value = 'ON' style = 'background-
                    color:lime;'>";
210:            }
211:            else if(val == 0)
212:            {
213:            serOut10 = "<input  type=button value = 'OFF' style = 'background-
                    color:silver;'>";
214:            }
215:            val = digitalRead(PinR11);
216:            Serial.println(val);
217:            if(val == 1)
218:            {
219:            serOut11 = "<input  type=button value = 'ON' style = 'background-
                    color:lime;'>";
220:            }
221:            else if(val == 0)
222:            {
223:            serOut11 = "<input  type=button value = 'OFF' style = 'background-
                    color:silver;'>";
224:            }
225:            val = digitalRead(PinR12);
226:            Serial.println(val);
227:            if(val == 1)
228:            {
229:            serOut12 = "<input  type=button value = 'ON' style = 'background-
                    color:lime;'>";
230:            }
231:            else if(val == 0)
232:            {
233:            serOut12 = "<input  type=button value = 'OFF' style = 'background-
                    color:silver;'>";
234:            }
235:            val = digitalRead(PinR13);
236:            Serial.println(val);
237:            if(val == 1)
238:            {
```

```
239:              serOut13 = "<input  type=button value = 'ON' style = 'background-
                      color:lime;'>";
240:              }
241:              else if(val == 0)
242:              {
243:              serOut13 = "<input  type=button value = 'OFF' style = 'background-
                      color:silver;'>";
244:              }
245:              val = digitalRead(PinR14);
246:              Serial.println(val);
247:              if(val == 1)
248:              {
249:              serOut14 = "<input  type=button value = 'ON' style = 'background-
                      color:lime;'>";
250:              }
251:              else if(val == 0)
252:              {
253:              serOut14 = "<input  type=button value = 'OFF' style = 'background-
                      color:silver;'>";
254:              }
255:              val = digitalRead(PinR15);
256:              Serial.println(val);
257:              if(val == 1)
258:              {
259:              serOut15 = "<input  type=button value = 'ON' style = 'background-
                      color:lime;'>";
260:              }
261:              else //if(val == 0)
262:              {
263:              serOut15 = "<input  type=button value = 'OFF' style = 'background-
                      color:silver;'>";
264:              }
265:              val = digitalRead(PinR16);
266:              Serial.println(val);
267:              if(val == 1)
268:              {
269:              serOut16 = "<input  type=button value = 'ON' style = 'background-
                      color:lime;'>";
270:              }
271:              else // if(val == 0)
272:              {
273:              serOut16 = "<input  type=button value = 'OFF' style = 'background-
                      color:silver;'>";
274:              }
275:
276:            client.println("HTTP/1.1 200 OK");
277:            client.println("Content-Type: text/html");
278:            client.println();
279:            client.println("<!DOCTYPE HTML>");
280:            client.println("<html>");
281:            client.println("<HEAD>");
282:            client.println("<TITLE>W3SZ Ethernet Relay Switch</TITLE>");
283:            client.println("</HEAD>");
284:            client.println("<body>");
285:            client.println("<br />");
286:            client.println("<H1>W3SZ Ethernet Relay Control</H1>");
287:            client.println("<H2>Click On Relay Buttons To Change State</H2>");
288:            client.println("<br />");
289:            client.println("<input  type=button value = 'GET STATUS' onmousedown=
                    location.href='/~STATUS$'>");
290:            client.println("<br />");
291:            client.println("<br />");
292:            client.println("<br />");
293:            client.println("<style>");
294:
295:            client.println("table, th, td {border-collapse: collapse;");
```

```
296:            client.println("}");
297:            client.println("th, td {");
298:            client.println("padding: 5px;");
299:            client.println("}");
300:
301:            client.println("table {");
302:            client.println("width: 100%;");
303:            client.println("}");
304:            client.println("</style>");
305:            client.println("<table>");
306:            client.println("<tr style='border-top:2px solid #f00; border-left:2px
                    solid #f00; border-right:2px solid #f00;'>");
307:            client.println("<td>");
308:            client.println("<input  type=button value = 'Relay 1 ON' onmousedown=
                    location.href='/~1$'>");
309:            client.println("<input  type=button value = 'Relay 1 OFF' onmousedown=
                    location.href='/~100$'>");
310:
311:            client.println("</td>");
312:            client.println("<td>");
313:            client.println("<input  type=button value = 'Relay 2 ON' onmousedown=
                    location.href='/~2$'>");
314:            client.println("<input  type=button value = 'Relay 2 OFF' onmousedown=
                    location.href='/~200$'>");
315:            client.println("</td>");
316:            client.println("<td>");
317:            client.println("<input  type=button value = 'Relay 3 ON' onmousedown=
                    location.href='/~3$'>");
318:            client.println("<input  type=button value = 'Relay 3 OFF' onmousedown=
                    location.href='/~300$'>");
319:            client.println("</td>");
320:            client.println("<td>");
321:            client.println("<input  type=button value = 'Relay 4 ON' onmousedown=
                    location.href='/~4$'>");
322:            client.println("<input  type=button value = 'Relay 4 OFF' onmousedown=
                    location.href='/~400$'>");
323:            client.println("</td>");
324:            client.println("</tr>");
325:
326:            client.println("<tr style='border-bottom:2px solid #f00; border-left:2
                    px solid #f00; border-right:2px solid #f00;'>");
327:            client.println("<td>");
328:            client.println(serOut1);
329:            client.println("</td>");
330:            client.println("<td>");
331:            client.println(serOut2);
332:            client.println("</td>");
333:            client.println("<td>");
334:            client.println(serOut3);
335:            client.println("</td>");
336:            client.println("<td>");
337:            client.println(serOut4);
338:            client.println("</td>");
339:            client.println("</tr>");
340:
341:            client.println("<tr style='border-top:2px solid #f00; border-left:2px
                    solid #f00; border-right:2px solid #f00;'>");
342:            client.println("<td>");
343:            client.println("<input  type=button value = 'Relay 5 ON' onmousedown=
                    location.href='/~5$'>");
344:            client.println("<input  type=button value = 'Relay 5 OFF' onmousedown=
                    location.href='/~500$'>");
345:            client.println("</td>");
346:            client.println("<td>");
347:            client.println("<input  type=button value = 'Relay 6 ON' onmousedown=
                    location.href='/~6$'>");
```

```
348:            client.println("<input  type=button value = 'Relay 6 OFF' onmousedown=
                    location.href='/~600$'>");
349:            client.println("</td>");
350:            client.println("<td>");
351:            client.println("<input  type=button value = 'Relay 7 ON' onmousedown=
                    location.href='/~7$'>");
352:            client.println("<input  type=button value = 'Relay 7 OFF' onmousedown=
                    location.href='/~700$'>");
353:            client.println("</td>");
354:            client.println("<td>");
355:            client.println("<input  type=button value = 'Relay 8 ON' onmousedown=
                    location.href='/~8$'>");
356:            client.println("<input  type=button value = 'Relay 8 OFF' onmousedown=
                    location.href='/~800$'>");
357:            client.println("</td>");
358:            client.println("</tr>");
359:
360:            client.println("<tr style='border-bottom:2px solid #f00; border-left:2
                    px solid #f00; border-right:2px solid #f00;'>");
361:            client.println("<td>");
362:            client.println(serOut5);
363:            client.println("</td>");
364:            client.println("<td>");
365:            client.println(serOut6);
366:            client.println("</td>");
367:            client.println("<td>");
368:            client.println(serOut7);
369:            client.println("</td>");
370:            client.println("<td>");
371:            client.println(serOut8);
372:            client.println("</td>");
373:            client.println("</tr>");
374:
375:            client.println("<tr style='border-top:2px solid #f00; border-left:2px
                    solid #f00; border-right:2px solid #f00;'>");
376:            client.println("<td>");
377:            client.println("<input  type=button value = 'Relay 9 ON' onmousedown=
                    location.href='/~9$'>");
378:            client.println("<input  type=button value = 'Relay 9 OFF' onmousedown=
                    location.href='/~900$'>");
379:            client.println("</td>");
380:            client.println("<td>");
381:            client.println("<input  type=button value = 'Relay 10 ON' onmousedown=
                    location.href='/~10$'>");
382:            client.println("<input  type=button value = 'Relay 10 OFF' onmousedown=
                    location.href='/~1000$'>");
383:            client.println("</td>");
384:            client.println("<td>");
385:            client.println("<input  type=button value = 'Relay 11 ON' onmousedown=
                    location.href='/~11$'>");
386:            client.println("<input  type=button value = 'Relay 11 OFF' onmousedown=
                    location.href='/~1100$'>");
387:            client.println("</td>");
388:            client.println("<td>");
389:            client.println("<input  type=button value = 'Relay 12 ON' onmousedown=
                    location.href='/~12$'>");
390:            client.println("<input  type=button value = 'Relay 12 OFF' onmousedown=
                    location.href='/~1200$'>");
391:            client.println("</td>");
392:            client.println("</tr>");
393:
394:            client.println("<tr style='border-bottom:2px solid #f00; border-left:2
                    px solid #f00; border-right:2px solid #f00;'>");
395:            client.println("<td>");
396:            client.println(serOut9);
397:            client.println("</td>");
```

```
398:            client.println("<td>");
399:            client.println(serOut10);
400:            client.println("</td>");
401:            client.println("<td>");
402:            client.println(serOut11);
403:            client.println("</td>");
404:            client.println("<td>");
405:            client.println(serOut12);
406:            client.println("</td>");
407:            client.println("</tr>");
408:
409:            client.println("<tr style='border-top:2px solid #f00; border-left:2px
                    solid #f00; border-right:2px solid #f00;'>");
410:            client.println("<td>");
411:            client.println("<input  type=button value = 'Relay 13 ON' onmousedown=
                    location.href='/~13$'>");
412:            client.println("<input  type=button value = 'Relay 13 OFF' onmousedown=
                    location.href='/~1300$'>");
413:            client.println("</td>");
414:            client.println("<td>");
415:            client.println("<input  type=button value = 'Relay 14 ON' onmousedown=
                    location.href='/~14$'>");
416:            client.println("<input  type=button value = 'Relay 14 OFF' onmousedown=
                    location.href='/~1400$'>");
417:            client.println("</td>");
418:            client.println("<td>");
419:            client.println("<input  type=button value = 'Relay 15 ON' onmousedown=
                    location.href='/~15$'>");
420:            client.println("<input  type=button value = 'Relay 15 OFF' onmousedown=
                    location.href='/~1500$'>");
421:            client.println("</td>");
422:            client.println("<td>");
423:            client.println("<input  type=button value = 'Relay 16 ON' onmousedown=
                    location.href='/~16$'>");
424:            client.println("<input  type=button value = 'Relay 16 OFF' onmousedown=
                    location.href='/~1600$'>");
425:            client.println("</td>");
426:            client.println("</tr>");
427:
428:            client.println("<tr style='border-bottom:2px solid #f00; border-left:2
                    px solid #f00; border-right:2px solid #f00;'>");
429:            client.println("<td>");
430:            client.println(serOut13);
431:            client.println("</td>");
432:            client.println("<td>");
433:            client.println(serOut14);
434:            client.println("</td>");
435:            client.println("<td>");
436:            client.println(serOut15);
437:            client.println("</td>");
438:            client.println("<td>");
439:            client.println(serOut16);
440:            client.println("</td>");
441:            client.println("</tr>");
442:
443:            client.println("</table>");
444:
445:
446:            client.println("</body>");
447:            client.println("</html>");
448:      // pause to give the browser time to receive the data
449:      delay(5);
450:      // close the connection:
451:      client.stop();
452:
453:
```

```
454:   }
455:
456: //this is the main program loop.
457: //it listens for an HTML client and when it gets input from the client it builds
         a string from the client's input
458: //it then parses the input and if it finds a valid command in the input, it uses
         that command to set the status of
459: //the digital output pin referenced by that command
460: //it reports the command received to the serial monitor and
461: //it calls the function sendReply which reads the output pin values and reports
         them both via serial port and HTML
462: //and creates the webpage with the buttons and the relay status displays
463: void loop()
464: {
465:   // listen for incoming client
466:   client = server.available();
467:   if (client) {
468:     while (client.connected()) {
469:       if (client.available()) {
470:         char c = client.read();
471:         commandInputString += c;  //append latest character received to string
472:       if (c == '\n')
473:       {
474:         //Checks for the URL string beginning with '~' and ending with '$'
475:           int stringStart = commandInputString.indexOf('~');
476:           int stringEnd = commandInputString.indexOf('$');
477:           String commandOut = commandInputString.substring(1 + stringStart,
                stringEnd);
478:           Serial.print("Command is: ");
479:           Serial.println(commandOut);
480:           Serial.println(" ");
481:
482:           if (commandOut  == "1") {
483:             digitalWrite(PinR1, HIGH);
484:             sendReply();
485:           }
486:           else if (commandOut  == "100") {
487:             digitalWrite(PinR1, LOW);
488:             sendReply();
489:           }
490:
491:           else if (commandOut  == "2") {
492:             digitalWrite(PinR2, HIGH);
493:             sendReply();
494:           }
495:           else if (commandOut  == "200") {
496:             digitalWrite(PinR2, LOW);
497:             sendReply();
498:           }
499:
500:           else if (commandOut  == "3") {
501:             digitalWrite(PinR3, HIGH);
502:             sendReply();
503:           }
504:           else if (commandOut  == "300") {
505:             digitalWrite(PinR3, LOW);
506:             sendReply();
507:           }
508:
509:           else if (commandOut  == "4") {
510:             digitalWrite(PinR4, HIGH);
511:             sendReply();
512:           }
513:           else if (commandOut  == "400") {
514:             digitalWrite(PinR4, LOW);
515:             sendReply();
```

```
516:            }
517:
518:            else if (commandOut  == "5") {
519:              digitalWrite(PinR5, HIGH);
520:              sendReply();
521:            }
522:            else if (commandOut  == "500") {
523:              digitalWrite(PinR5, LOW);
524:              sendReply();
525:            }
526:
527:            else if (commandOut  == "6") {
528:              digitalWrite(PinR6, HIGH);
529:              sendReply();
530:            }
531:            else if (commandOut  == "600") {
532:              digitalWrite(PinR6, LOW);
533:              sendReply();
534:            }
535:
536:            else if (commandOut  == "7") {
537:              digitalWrite(PinR7, HIGH);
538:              sendReply();
539:            }
540:            else if (commandOut  == "700") {
541:              digitalWrite(PinR7, LOW);
542:              sendReply();
543:            }
544:
545:            else if (commandOut  == "8") {
546:              digitalWrite(PinR8, HIGH);
547:              sendReply();
548:            }
549:            else if (commandOut  == "800") {
550:              digitalWrite(PinR8, LOW);
551:              sendReply();
552:            }
553:
554:            else if (commandOut  == "9") {
555:              digitalWrite(PinR9, HIGH);
556:              sendReply();
557:            }
558:            else if (commandOut  == "900") {
559:              digitalWrite(PinR9, LOW);
560:              sendReply();
561:            }
562:
563:            else if (commandOut  == "10") {
564:              digitalWrite(PinR10, HIGH);
565:              sendReply();
566:            }
567:            else if (commandOut  == "1000") {
568:              digitalWrite(PinR10, LOW);
569:              sendReply();
570:            }
571:
572:            else if (commandOut  == "11") {
573:              digitalWrite(PinR11, HIGH);
574:              sendReply();
575:            }
576:            else if (commandOut  == "1100") {
577:              digitalWrite(PinR11, LOW);
578:              sendReply();
579:            }
580:
581:            else if (commandOut  == "12") {
```

```
582:             digitalWrite(PinR12, HIGH);
583:             sendReply();
584:           }
585:           else if (commandOut  == "1200") {
586:             digitalWrite(PinR12, LOW);
587:             sendReply();
588:           }
589:
590:
591:           else if (commandOut  == "13") {
592:             digitalWrite(PinR13, HIGH);
593:             sendReply();
594:           }
595:           else if (commandOut  == "1300") {
596:             digitalWrite(PinR13, LOW);
597:             sendReply();
598:           }
599:
600:           else if (commandOut  == "14") {
601:             digitalWrite(PinR14, HIGH);
602:             sendReply();
603:           }
604:           else if (commandOut  == "1400") {
605:             digitalWrite(PinR14, LOW);
606:             sendReply();
607:           }
608:
609:           else if (commandOut  == "15") {
610:             digitalWrite(PinR15, HIGH);
611:             sendReply();
612:           }
613:           else if (commandOut  == "1500") {
614:             digitalWrite(PinR15, LOW);
615:             sendReply();
616:           }
617:
618:           else if (commandOut  == "16") {
619:             digitalWrite(PinR16, HIGH);
620:             sendReply();
621:           }
622:           else if (commandOut  == "1600") {
623:             digitalWrite(PinR16, LOW);
624:             sendReply();
625:           }
626:
627:           else if (commandOut  == "STATUS") {
628:             sendReply();
629:           }
630:           else
631:           {
632:             String HTMString = "Command Not Recognized:  ";
633:             Serial.println(commandOut);
634:             Serial.println(HTMString);
635:             sendReply();
636:           }
637:
638:           commandInputString = "";
639:           commandOut = "";
640:           c=' ';
641:
642:       }
643:       }
644:     }
645:   }
646: }
647:
```

Basic Stamp Device Control Code

```
' {$STAMP BS2p}
' {$PBASIC 2.5}
' {$PORT COM1}

' This program is supposed to take band control data from the N3FTI Bandswitch
' and use it to set the appropriate transmit and receive IF signal levels by
' setting programmable attenuators for each band from 50 MHz thru 24 GHz.
' The band-select signal is input as a 4 bit binary signal and the logic is set
' so that the appropriate signals are then sent to the programmable attenuators for
' both the transmit and receive lines.

' The input signal matrix is as follows:
' Band    A        B         C             D
' 50      0        0         0             0
' 144     1        0         0             0
' 222     0        1         0             0
' 432     1        1         0             0
' 903     0        0         1             0
' 1296    1        0         1             0
' 2304    0        1         1             0
' 3456    1        1         1             0
' 5760    0        0         0             1
' 10G     1        0         0             1
' 24G     0        1         0             1
' 47G     1        1         0             1
'
' A = LPT pin 2
' B = LPT pin 7
' C = LPT pin 8
' D = LPT pin 9
' Grnd = LPT pin 15

' Declare attenuation level variables for receive
RX50 VAR Byte
RX144  VAR Byte
RX222 VAR Byte
RX432 VAR Byte
RX903 VAR Byte
RX1296 VAR Byte
RX2304 VAR Byte
RX3G VAR Byte
RX5G VAR Byte
RX10G  VAR Byte
RX24G  VAR Byte

' Declare attenuation level variables for transmit
TX50 VAR Byte
TX144 VAR Byte
TX222 VAR Byte
TX432 VAR Byte
TX903 VAR Byte
TX1296 VAR Byte
TX2304 VAR Byte
TX3G VAR Byte
TX5G VAR Byte
TX10G VAR Byte
TX24G VAR Byte

' A Nib is 4 bits
' Declare input frequency variable from N3FTI Device
FREQ VAR Nib
' FREQ CAN BE
' 0 50 MHZ
' 1 144 MHZ
' 2 222 MHZ
' 3 432 MHZ
```

```
' 4 903 MHZ
' 5 1296 MHZ
' 6 2304 MHZ
' 7 3G
' 8 5G
' 9 10G
' 10 24G

' Declare RXOUT and TXOUT.  These are attenuation levels to be set
RXOUT VAR Byte
TXOUT VAR Byte

' Initialize receive attenuation level variables for each band
RX50 = 00
RX144 = 00
RX222 = 00
RX432 = 16
RX903 = 08
RX1296 = 0
RX2304 = 18
RX3G = 7
RX5G = 8
RX10G = 8
RX24G = 2

' Initialize transmit attenuation level variables for each band
TX50 = 0
TX144 = 17
TX222 = 11
TX432 = 04
TX903 = 13
TX1296 = 0
TX2304 = 2
TX3G = 20
TX5G = 0
TX10G = 0
TX24G = 0

' Declare control bit variables for Rx
RCV1  VAR Bit
RCV2  VAR Bit
RCV4  VAR Bit
RCV8  VAR Bit
RCV16 VAR Bit
RCV32 VAR Bit

' Declare control bit variables for Tx
TX1  VAR Bit
TX2  VAR Bit
TX4  VAR Bit
TX8  VAR Bit
TX16 VAR Bit
TX32 VAR Bit

' Define shorthand reference for input pins
A PIN 0
B PIN 1
C PIN 2
D PIN 3

' Set pins A, B, C, D to be input pins
INPUT A
INPUT B
INPUT C
INPUT D

```

```
'Set pins 4-15 as output pins
OUTPUT 4
OUTPUT 5
OUTPUT 6
OUTPUT 7
OUTPUT 8
OUTPUT 9
OUTPUT 10
OUTPUT 11
OUTPUT 12
OUTPUT 13
OUTPUT 14
OUTPUT 15

' Main program loop follows
DO


' Calculate band from BCD input
FREQ = A + (B*2) + (C*4) + (D*8)

'set RXOUT and TXOUT attenuation levels based on BCD input from N3FTI
SELECT FREQ
CASE = 0
  RXOUT = RX50
  TXOUT = TX50
CASE = 1
  RXOUT = RX144
  TXOUT = TX144
CASE = 2
  RXOUT = RX222
  TXOUT = TX222
CASE = 3
  RXOUT = RX432
  TXOUT = TX432
CASE = 4
  RXOUT = RX903
  TXOUT = TX903
CASE = 5
  RXOUT = RX1296
  TXOUT = TX1296
CASE = 6
  RXOUT = RX2304
  TXOUT = TX2304
CASE = 7
  RXOUT = RX3G
  TXOUT = TX3G
CASE = 8
  RXOUT = RX5G
  TXOUT = TX5G
CASE = 9
  RXOUT = RX10G
  TXOUT = TX10G
CASE = 10
  RXOUT = RX24G
  TXOUT = TX24G
CASE > 10
  RXOUT = RX24G
  TXOUT = TX24G
ENDSELECT

' DETERMINE RCV and TX output pin levels based on values of RXOUT and TXOUT
IF (RXOUT >= 32) THEN
  RCV32 = 1
  RXOUT = RXOUT - 32
```

```
199: ELSE
200: RCV32 = 0
201: ENDIF
202:
203: IF (RXOUT >= 16) THEN
204:   RCV16 = 1
205:   RXOUT = RXOUT - 16
206: ELSE
207: RCV16 = 0
208: ENDIF
209:
210: IF (RXOUT >= 8) THEN
211:   RCV8 = 1
212:   RXOUT = RXOUT - 8
213: ELSE
214: RCV8 = 0
215: ENDIF
216:
217: IF (RXOUT >= 4) THEN
218:   RCV4 = 1
219:   RXOUT=RXOUT - 4
220: ELSE
221: RCV4 = 0
222: ENDIF
223:
224:   IF (RXOUT >= 2) THEN
225:   RCV2 = 1
226:   RXOUT = RXOUT - 2
227: ELSE
228: RCV2 = 0
229: ENDIF
230:
231: RCV1 = RXOUT
232:
233: IF (TXOUT >= 32) THEN
234:   TX32 = 1
235:   TXOUT = TXOUT - 32
236: ELSE
237:   TX32 = 0
238: ENDIF
239:
240: IF (TXOUT  >= 16) THEN
241:   TX16 = 1
242:   TXOUT = TXOUT - 16
243: ELSE
244:   TX16 = 0
245: ENDIF
246:
247: IF (TXOUT >= 8) THEN
248:   TX8 = 1
249:   TXOUT = TXOUT - 8
250: ELSE
251:   TX8 = 0
252: ENDIF
253:
254: IF (TXOUT >= 4) THEN
255:   TX4 = 1
256:   TXOUT=TXOUT - 4
257: ELSE
258:  TX4 = 0
259: ENDIF
260:
261:   IF (TXOUT >= 2) THEN
262:   TX2 = 1
263:   TXOUT = TXOUT - 2
264: ELSE
```

```
265:   TX2 = 0
266: ENDIF
267:
268: TX1 = TXOUT
269:
270: ' Use RCV and TX levels as just determined to set output pin levels
271: OUT4 = TX1
272: OUT5 = TX2
273: OUT6 = TX4
274: OUT7 = TX8
275: OUT8 = TX16
276: OUT9 = TX32
277:
278: OUT10 = RCV1
279: OUT11 = RCV2
280: OUT12 = RCV4
281: OUT13 = RCV8
282: OUT14 = RCV16
283: OUT15 = RCV32
284:
285: ' Go back to beginning of loop and repeat
286: LOOP
287:
288: END
289:
290:
291:
```

Arduino Code Derived From Basic Stamp Device Control Code

```
// {$STAMP BS2p}
// {$PBASIC 2.5}
// {$PORT COM1}

//by W3SZ
//Arduino code tested with hardware and shown to be working 8-24-2017
// This program is supposed to take band control data from the N3FTI Bandswitch
// and use it to set the appropriate transmit and receive if signal levels by
// setting programmable attenuators for each band from 50 MHz thru 24 GHz.
// The band-select signal is input as a 4 bit binary signal and the logic is set
// so that the appropriate signals are then sent to the programmable attenuators for
// both the transmit and receive lines.

// The input signal matrix is as follows:
// Band    A        B         C             D
// 50      0        0         0             0
// 144     1        0         0             0
// 222     0        1         0             0
// 432     1        1         0             0
// 903     0        0         1             0
// 1296    1        0         1             0
// 2304    0        1         1             0
// 3456    1        1         1             0
// 5760    0        0         0             1
// 10G     1        0         0             1
// 24G     0        1         0             1
// 47G     1        1         0             1
//
// A = LPT pin 2
// B = LPT pin 7
// C = LPT pin 8
// D = LPT pin 9
// Grnd = LPT pin 15

// Declare and Initialize receive attenuation level variables for each band
int RX50 = 10;
int RX144 = 15;
int RX222 = 20;
int RX432 = 25;
int RX903 = 30;
int RX1296 = 35;
int RX2304 = 40;
int RX3G = 45;
int RX5G = 50;
int RX10G = 55;
int RX24G = 60;

// Declare and Initialize transmit attenuation level variables for each band
int TX50 = 0;
int TX144 = 2;
int TX222 = 4;
int TX432 = 8;
int TX903 = 16;
int TX1296 = 32;
int TX2304 = 1;
int TX3G = 5;
int TX5G = 9;
int TX10G = 17;
int TX24G = 33;

// Declare and initialize input frequency variable from N3FTI Device
int FREQ = 0;
// FREQ CAN BE
// 0 50 MHZ
// 1 144 MHZ
// 2 222 MHZ
```

```
 67: // 3 432 MHZ
 68: // 4 903 MHZ
 69: // 5 1296 MHZ
 70: // 6 2304 MHZ
 71: // 7 3G
 72: // 8 5G
 73: // 9 10G
 74: // 10 24G
 75:
 76: // Declare and initialize RXOUT and TXOUT.  These are attenuation levels to be set
 77: int RXOUT = 0;
 78: int TXOUT = 0;
 79:
 80: // Declare and ititialize control bit variables for Rx
 81: int RCV1 = 0;
 82: int RCV2 = 0;
 83: int RCV4 = 0;
 84: int RCV8 = 0;
 85: int RCV16 = 0;
 86: int RCV32 = 0;
 87:
 88: // Declare control bit variables for Tx
 89: int TX1 = 0;
 90: int TX2 = 0;
 91: int TX4 = 0;
 92: int TX8 = 0;
 93: int TX16 = 0;
 94: int TX32 = 0;
 95:
 96: // Define shorthand reference for input pins
 97: const int PinA = A0;
 98: const int PinB = A1;
 99: const int PinC = A2;
100: const int PinD = A3;
101:
102: //define and initialize input pin read variables
103: int A = 0;
104: int B = 0;
105: int C = 0;
106: int D = 0;
107:
108: //Define shorthand reference for output pins
109: const int TxOUT1 =4;
110: const int TxOUT2 =5;
111: const int TxOUT4 =6;
112: const int TxOUT8 =7;
113: const int TxOUT16 =8;
114: const int TxOUT32 =9;
115:
116: const int RxOUT1 =10;
117: const int RxOUT2 =11;
118: const int RxOUT4 =12;
119: const int RxOUT8 =13;
120: const int RxOUT16 =14;
121: const int RxOUT32 =15;
122:
123: void setup() {
124:   // put your setup code here, to run once:
125:   //setup input and output pins
126: pinMode(PinA, INPUT);
127: pinMode(PinB, INPUT);
128: pinMode(PinC, INPUT);
129: pinMode(PinD, INPUT);
130:
131: pinMode(TxOUT1, OUTPUT);
132: digitalWrite(TxOUT1, LOW);
```

```
133: pinMode(TxOUT2, OUTPUT);
134: digitalWrite(TxOUT2, LOW);
135: pinMode(TxOUT4, OUTPUT);
136: digitalWrite(TxOUT4, LOW);
137: pinMode(TxOUT8, OUTPUT);
138: digitalWrite(TxOUT8, LOW);
139: pinMode(TxOUT16, OUTPUT);
140: digitalWrite(TxOUT16, LOW);
141: pinMode(TxOUT32, OUTPUT);
142: digitalWrite(TxOUT32, LOW);
143:
144: pinMode(RxOUT1, OUTPUT);
145: digitalWrite(RxOUT1, LOW);
146: pinMode(RxOUT2, OUTPUT);
147: digitalWrite(RxOUT2, LOW);
148: pinMode(RxOUT4, OUTPUT);
149: digitalWrite(RxOUT4, LOW);
150: pinMode(RxOUT8, OUTPUT);
151: digitalWrite(RxOUT8, LOW);
152: pinMode(RxOUT16, OUTPUT);
153: digitalWrite(RxOUT16, LOW);
154: pinMode(RxOUT32, OUTPUT);
155: digitalWrite(RxOUT32, LOW);
156:
157: Serial.begin(9600);
158: Serial.println("Arduino IF Tx/Rx Automatic Attenuator");
159: Serial.println("Takes BCD data from N3FTI Bandswitch and");
160: Serial.println("sets programmable Attenuators for Tx and Rx 0-63 dB");
161: Serial.println("Derived from Basic Stamp Program for this purpose");
162: Serial.println("also by W3SZ");
163:
164: }
165:
166: void loop() {
167:
168: // Main program loop follows
169:
170: // Read BCD input from N3FTI controller
171: A = digitalRead(PinA);
172: B = digitalRead(PinB);
173: C = digitalRead(PinC);
174: D = digitalRead(PinD);
175:
176: // Calculate band from BCD input
177: FREQ = A + (B*2) + (C*4) + (D*8);
178: Serial.println("   ");
179: Serial.println("   ");
180: Serial.println("   ");
181: Serial.println("   ");
182: Serial.println("   ");
183: Serial.println("   ");
184: Serial.println("Starting Loop");
185: Serial.println("FREQUENCY BAND Input is: ");
186: Serial.println(FREQ);
187:
188: //set RXOUT and TXOUT attenuation levels based on BCD input from N3FTI
189: switch (FREQ)
190: {
191: case  0: {
192:   RXOUT = RX50;
193:   TXOUT = TX50;
194:   break;
195: }
196: case  1: {
197:   RXOUT = RX144;
198:   TXOUT = TX144;
```

```
199:   break;
200: }
201: case  2: {
202:   RXOUT = RX222;
203:   TXOUT = TX222;
204:   break;
205: }
206: case  3: {
207:   RXOUT = RX432;
208:   TXOUT = TX432;
209:   break;
210: }
211: case  4: {
212:   RXOUT = RX903;
213:   TXOUT = TX903;
214:   break;
215: }
216: case  5: {
217:   RXOUT = RX1296;
218:   TXOUT = TX1296;
219:   break;
220: }
221: case  6: {
222:   RXOUT = RX2304;
223:   TXOUT = TX2304;
224:   break;
225: }
226: case  7: {
227:   RXOUT = RX3G;
228:   TXOUT = TX3G;
229:   break;
230: }
231: case  8: {
232:   RXOUT = RX5G;
233:   TXOUT = TX5G;
234:   break;
235: }
236: case  9: {
237:   RXOUT = RX10G;
238:   TXOUT = TX10G;
239:   break;
240: }
241: case  10: {
242:   RXOUT = RX24G;
243:   TXOUT = TX24G;
244:   break;
245: }
246: case  11: {
247:   RXOUT = RX24G;
248:   TXOUT = TX24G;
249:   break;
250: }
251: default: {
252:   RXOUT = RX24G;
253:   TXOUT = TX24G;
254:   break;
255: }
256: }
257:
258: // DETERMINE RCV and TX output pin levels based on values of RXOUT and TXOUT
259: if (RXOUT >= 32) {
260:   RCV32 = 1;
261:   RXOUT = RXOUT - 32;
262: }
263: else {
264: RCV32 = 0;
```

```
265: }
266:
267: if (RXOUT >= 16) {
268:   RCV16 = 1;
269:   RXOUT = RXOUT - 16;
270: }
271: else {
272: RCV16 = 0;
273: }
274:
275: if (RXOUT >= 8) {
276:   RCV8 = 1;
277:   RXOUT = RXOUT - 8;
278: }
279: else {
280: RCV8 = 0;
281: }
282:
283: if (RXOUT >= 4) {
284:   RCV4 = 1;
285:   RXOUT=RXOUT - 4;
286: }
287: else {
288: RCV4 = 0;
289: }
290:
291: if (RXOUT >= 2) {
292:   RCV2 = 1;
293:   RXOUT = RXOUT - 2;
294: }
295: else {
296: RCV2 = 0;
297: }
298:
299: RCV1 = RXOUT;
300:
301: if (TXOUT >= 32) {
302:   TX32 = 1;
303:   TXOUT = TXOUT - 32;
304: }
305: else {
306:   TX32 = 0;
307: }
308:
309: if (TXOUT  >= 16) {
310:   TX16 = 1;
311:   TXOUT = TXOUT - 16;
312: }
313: else {
314:   TX16 = 0;
315: }
316:
317: if (TXOUT >= 8) {
318:   TX8 = 1;
319:   TXOUT = TXOUT - 8;
320: }
321: else {
322:   TX8 = 0;
323: }
324:
325: if (TXOUT >= 4) {
326:   TX4 = 1;
327:   TXOUT=TXOUT - 4;
328: }
329: else {
330:  TX4 = 0;
```

```
331: }
332:
333: if (TXOUT >= 2) {
334:   TX2 = 1;
335:   TXOUT = TXOUT - 2;
336: }
337: else {
338:   TX2 = 0;
339: }
340:
341: TX1 = TXOUT;
342:
343: // Use RCV and TX levels as just determined to set output pin levels
344: digitalWrite(TxOUT1, TX1);
345: digitalWrite(TxOUT2, TX2);
346: digitalWrite(TxOUT4, TX4);
347: digitalWrite(TxOUT8, TX8);
348: digitalWrite(TxOUT16, TX16);
349: digitalWrite(TxOUT32, TX32);
350:
351: digitalWrite(RxOUT1, RCV1);
352: digitalWrite(RxOUT2, RCV2);
353: digitalWrite(RxOUT4, RCV4);
354: digitalWrite(RxOUT8, RCV8);
355: digitalWrite(RxOUT16, RCV16);
356: digitalWrite(RxOUT32, RCV32);
357:
358: Serial.println("TxOUT1 is: ");
359: Serial.println(TX1);
360: Serial.println("  ");
361: Serial.println("TxOUT2 is: ");
362: Serial.println(TX2);
363: Serial.println("  ");
364: Serial.println("TxOUT4 is: ");
365: Serial.println(TX4);
366: Serial.println("  ");
367: Serial.println("TxOUT8 is: ");
368: Serial.println(TX8);
369: Serial.println("  ");
370: Serial.println("TxOUT16 is: ");
371: Serial.println(TX16);
372: Serial.println("  ");
373: Serial.println("TxOUT32 is: ");
374: Serial.println(TX32);
375: Serial.println("  ");
376:
377: Serial.println("RxOUT1 is: ");
378: Serial.println(RCV1);
379: Serial.println("  ");
380: Serial.println("RxOUT2 is: ");
381: Serial.println(RCV2);
382: Serial.println("  ");
383: Serial.println("RxOUT4 is: ");
384: Serial.println(RCV4);
385: Serial.println("  ");
386: Serial.println("RxOUT8 is: ");
387: Serial.println(RCV8);
388: Serial.println("  ");
389: Serial.println("RxOUT16 is: ");
390: Serial.println(RCV16);
391: Serial.println("  ");
392: Serial.println("RxOUT32 is: ");
393: Serial.println(RCV32);
394: Serial.println("  ");
395: }
```

Arduino and Basic Stamp Device Control Code Side By Side

```
' {$STAMP BS2p}
' {$PBASIC 2.5}
' {$PORT COM1}
‘Basic Stamp Code on this side!
' This program is supposed to take band control
‘data from the N3FTI Bandswitch
' and use it to set the appropriate transmit and
‘receive IF signal levels by
' setting programmable attenuators for each band
‘from 50 MHz thru 24 GHz.
' The band-select signal is input as a 4 bit binary
‘signal and the logic is set
' so that the appropriate signals are then sent to the
‘programmable attenuators for
' both the transmit and receive lines.

' The input signal matrix is as follows:
' Band    A       B       C           D
' 50      0       0       0           0
' 144     1       0       0           0
' 222     0       1       0           0
' 432     1       1       0           0
' 903     0       0       1           0
' 1296    1       0       1           0
' 2304    0       1       1           0
' 3456    1       1       1           0
' 5760    0       0       0           1
' 10G     1       0       0           1
' 24G     0       1       0           1
' 47G     1       1       0           1
'
' A = LPT pin 2
' B = LPT pin 7
' C = LPT pin 8
' D = LPT pin 9
' Grnd = LPT pin 15

' Declare attenuation level variables for receive
RX50 VAR Byte
RX144  VAR Byte
RX222 VAR Byte
RX432 VAR Byte
RX903 VAR Byte
RX1296 VAR Byte
RX2304 VAR Byte
RX3G VAR Byte
RX5G VAR Byte
RX10G  VAR Byte
RX24G  VAR Byte
```

```
//Arduino code on this side!
//both by W3SZ
//Arduino code tested with hardware and shown
//to be working 8-24-2017
// This program is supposed to take band control
//data from the N3FTI Bandswitch
// and use it to set the appropriate transmit and
//receive if signal levels by
// setting programmable attenuators for each
//band from 50 MHz thru 24 GHz.
// The band-select signal is input as a 4 bit
//binary signal and the logic is set
// so that the appropriate signals are then sent
//to the programmable attenuators for
// both the transmit and receive lines.

// The input signal matrix is as follows:
// Band    A       B       C           D
// 50      0       0       0           0
// 144     1       0       0           0
// 222     0       1       0           0
// 432     1       1       0           0
// 903     0       0       1           0
// 1296    1       0       1           0
// 2304    0       1       1           0
// 3456    1       1       1           0
// 5760    0       0       0           1
// 10G     1       0       0           1
// 24G     0       1       0           1
// 47G     1       1       0           1
//
// A = LPT pin 2
// B = LPT pin 7
// C = LPT pin 8
// D = LPT pin 9
// Grnd = LPT pin 15

// Declare/Initialize receive attenuation level var
int RX50 = 10;
int RX144 = 15;
int RX222 = 20;
int RX432 = 25;
int RX903 = 30;
int RX1296 = 35;
int RX2304 = 40;
int RX3G = 45;
int RX5G = 50;
int RX10G = 55;
int RX24G = 60;
```

```
' Declare attenuation level variables for transmit
TX50 VAR Byte
TX144 VAR Byte
TX222 VAR Byte
TX432 VAR Byte
TX903 VAR Byte
TX1296 VAR Byte
TX2304 VAR Byte
TX3G VAR Byte
TX5G VAR Byte
TX10G VAR Byte
TX24G VAR Byte

' A Nib is 4 bits
' Declare input frequency var from N3FTI Device
FREQ VAR Nib
' FREQ CAN BE
' 0 50 MHZ
' 1 144 MHZ
' 2 222 MHZ
' 3 432 MHZ
' 4 903 MHZ
' 5 1296 MHZ
' 6 2304 MHZ
' 7 3G
' 8 5G
' 9 10G
' 10 24G

' Declare RXOUT and TXOUT.
RXOUT VAR Byte
TXOUT VAR Byte

' Initialize receive atten level var for each band
RX50 = 00
RX144 = 00
RX222 = 00
RX432 = 16
RX903 = 08
RX1296 = 0
RX2304 = 18
RX3G = 7
RX5G = 8
RX10G = 8
RX24G = 2
```

```
// Declare/Initialize transmit attenuation level vars
int TX50 = 0;
int TX144 = 2;
int TX222 = 4;
int TX432 = 8;
int TX903 = 16;
int TX1296 = 32;
int TX2304 = 1;
int TX3G = 5;
int TX5G = 9;
int TX10G = 17;
int TX24G = 33;

// Declare/initialize input freq var from N3FTI Dev

int FREQ = 0;
// FREQ CAN BE
// 0 50 MHZ
// 1 144 MHZ
// 2 222 MHZ
// 3 432 MHZ
// 4 903 MHZ
// 5 1296 MHZ
// 6 2304 MHZ
// 7 3G
// 8 5G
// 9 10G
// 10 24G

// Declare and initialize RXOUT and TXOUT.
int RXOUT = 0;
int TXOUT = 0;
```

```
' Initialize transmit atten level var for each band
TX50 = 0
TX144 = 17
TX222 = 11
TX432 = 04
TX903 = 13
TX1296 = 0
TX2304 = 2
TX3G = 20
TX5G = 0
TX10G = 0
TX24G = 0

' Declare control bit variables for Rx
RCV1  VAR Bit
RCV2  VAR Bit
RCV4  VAR Bit
RCV8  VAR Bit
RCV16 VAR Bit
RCV32 VAR Bit

' Declare control bit variables for Tx
TX1  VAR Bit
TX2  VAR Bit
TX4  VAR Bit
TX8  VAR Bit
TX16 VAR Bit
TX32 VAR Bit

' Define shorthand reference for input pins
A PIN 0
B PIN 1
C PIN 2
D PIN 3

' Read BCD input from N3FTI controller
INPUT A
INPUT B
INPUT C
INPUT D

'Set pins 4-15 as output pins
OUTPUT 4
OUTPUT 5
OUTPUT 6
OUTPUT 7
OUTPUT 8
OUTPUT 9
```

```
// Declare and ititialize control bit variables for Rx
int RCV1 = 0;
int RCV2 = 0;
int RCV4 = 0;
int RCV8 = 0;
int RCV16 = 0;
int RCV32 = 0;

// Declare control bit variables for Tx
int TX1 = 0;
int TX2 = 0;
int TX4 = 0;
int TX8 = 0;
int TX16 = 0;
int TX32 = 0;

// Define shorthand reference for input pins
const int PinA = A0;
const int PinB = A1;
const int PinC = A2;
const int PinD = A3;

//define and initialize input pin read variables
int A = 0;
int B = 0;
int C = 0;
int D = 0;

//Define shorthand reference for output pins
const int TxOUT1 =4;
const int TxOUT2 =5;
const int TxOUT4 =6;
const int TxOUT8 =7;
const int TxOUT16 =8;
const int TxOUT32 =9;
```

OUTPUT 10
OUTPUT 11
OUTPUT 12
OUTPUT 13
OUTPUT 14
OUTPUT 15

```
const int RxOUT1 =10;
const int RxOUT2 =11;
const int RxOUT4 =12;
const int RxOUT8 =13;
const int RxOUT16 =14;
const int RxOUT32 =15;

void setup() {
  // put your setup code here, to run once:
  //setup input and output pins
pinMode(PinA, INPUT);
pinMode(PinB, INPUT);
pinMode(PinC, INPUT);
pinMode(PinD, INPUT);

pinMode(TxOUT1, OUTPUT);
digitalWrite(TxOUT1, LOW);
pinMode(TxOUT2, OUTPUT);
digitalWrite(TxOUT2, LOW);
pinMode(TxOUT4, OUTPUT);
digitalWrite(TxOUT4, LOW);
pinMode(TxOUT8, OUTPUT);
digitalWrite(TxOUT8, LOW);
pinMode(TxOUT16, OUTPUT);
digitalWrite(TxOUT16, LOW);
pinMode(TxOUT32, OUTPUT);
digitalWrite(TxOUT32, LOW);

pinMode(RxOUT1, OUTPUT);
digitalWrite(RxOUT1, LOW);
pinMode(RxOUT2, OUTPUT);
digitalWrite(RxOUT2, LOW);
pinMode(RxOUT4, OUTPUT);
digitalWrite(RxOUT4, LOW);
pinMode(RxOUT8, OUTPUT);
digitalWrite(RxOUT8, LOW);
pinMode(RxOUT16, OUTPUT);
digitalWrite(RxOUT16, LOW);
pinMode(RxOUT32, OUTPUT);
digitalWrite(RxOUT32, LOW);
}
```

```
' Main program loop follows

DO

' Calculate band from BCD input
FREQ = A + (B*2) + (C*4) + (D*8)

'set RXOUT and TXOUT attenuation levels
‘based on BCD input from N3FTI
SELECT FREQ
CASE = 0
  RXOUT = RX50
  TXOUT = TX50
CASE = 1
  RXOUT = RX144
  TXOUT = TX144
CASE = 2
  RXOUT = RX222
  TXOUT = TX222
CASE = 3
  RXOUT = RX432
  TXOUT = TX432
CASE = 4
  RXOUT = RX903
  TXOUT = TX903
CASE = 5
  RXOUT = RX1296
  TXOUT = TX1296
CASE = 6
  RXOUT = RX2304
  TXOUT = TX2304
CASE = 7
  RXOUT = RX3G
  TXOUT = TX3G
CASE = 8
  RXOUT = RX5G
  TXOUT = TX5G
```

```
void loop() {

// Read BCD input from N3FTI controller
A = digitalRead(PinA);
B = digitalRead(PinB);
C = digitalRead(PinC);
D = digitalRead(PinD);

// Calculate band from BCD input
FREQ = A + (B*2) + (C*4) + (D*8);

//set RXOUT and TXOUT attenuation levels
//based on BCD input from N3FTI
switch (FREQ) {
case  0: {
  RXOUT = RX50;
  TXOUT = TX50;
  break; }
case  1: {
  RXOUT = RX144;
  TXOUT = TX144;
  break; }
case  2: {
  RXOUT = RX222;
  TXOUT = TX222;
  break; }
case  3: {
  RXOUT = RX432;
  TXOUT = TX432;
  break; }
case  4: {
  RXOUT = RX903;
  TXOUT = TX903;
  break; }
case  5: {
  RXOUT = RX1296;
  TXOUT = TX1296;
  break; }
case  6: {
  RXOUT = RX2304;
TXOUT = TX2304;
  break; }
```

```
CASE = 9
  RXOUT = RX10G
  TXOUT = TX10G
CASE = 10
  RXOUT = RX24G
  TXOUT = TX24G
CASE > 10
  RXOUT = RX24G
  TXOUT = TX24G
ENDSELECT

' DETERMINE RCV and TX output pin levels
‘ based on values of RXOUT and TXOUT
IF (RXOUT >= 32) THEN
  RCV32 = 1
  RXOUT = RXOUT - 32
ELSE
RCV32 = 0
ENDIF

IF (RXOUT >= 16) THEN
  RCV16 = 1
  RXOUT = RXOUT - 16
ELSE
RCV16 = 0
ENDIF

IF (RXOUT >= 8) THEN
  RCV8 = 1
  RXOUT = RXOUT - 8
ELSE
RCV8 = 0
ENDIF

IF (RXOUT >= 4) THEN
  RCV4 = 1
  RXOUT=RXOUT - 4
ELSE
RCV4 = 0
ENDIF

  IF (RXOUT >= 2) THEN
  RCV2 = 1
  RXOUT = RXOUT – 2
ELSE
RCV2 = 0
ENDIF

RCV1 = RXOUT
```

```
case  7: {
  RXOUT = RX3G;
  TXOUT = TX3G;
  break; }
case  8: {
  RXOUT = RX5G;
  TXOUT = TX5G;
  break; }
case  9: {
  RXOUT = RX10G;
  TXOUT = TX10G;
  break; }
case  10: {
  RXOUT = RX24G;
  TXOUT = TX24G;
break;  }
case  11: {
  RXOUT = RX24G;
  TXOUT = TX24G;
  break; }}

/ / DETERMINE RCV and TX output pin levels
//  based on values of RXOUT and TXOUT
if (RXOUT >= 32) {
  RCV32 = 1;
  RXOUT = RXOUT – 32; }
else {
RCV32 = 0; }
if (RXOUT >= 16) {
  RCV16 = 1;
  RXOUT = RXOUT – 16; }
else {
RCV16 = 0; }
if (RXOUT >= 8) {
  RCV8 = 1;
  RXOUT = RXOUT – 8; }
else {
RCV8 = 0; }
if (RXOUT >= 4) {
  RCV4 = 1;
  RXOUT=RXOUT – 4; }
else {
RCV4 = 0; }
if (RXOUT >= 2) {
  RCV2 = 1;
  RXOUT = RXOUT – 2; }
else {
RCV2 = 0; }
RCV1 = RXOUT;
```

```
IF (TXOUT >= 32) THEN
  TX32 = 1
  TXOUT = TXOUT - 32
ELSE
  TX32 = 0
ENDIF

IF (TXOUT  >= 16) THEN
  TX16 = 1
  TXOUT = TXOUT - 16
ELSE
  TX16 = 0
ENDIF

IF (TXOUT >= 8) THEN
  TX8 = 1
  TXOUT = TXOUT - 8
ELSE
  TX8 = 0
ENDIF

IF (TXOUT >= 4) THEN
  TX4 = 1
  TXOUT=TXOUT - 4
ELSE
 TX4 = 0
ENDIF

  IF (TXOUT >= 2) THEN
  TX2 = 1
  TXOUT = TXOUT - 2
ELSE
  TX2 = 0
ENDIF

TX1 = TXOUT
```

```
if (TXOUT >= 32) {
  TX32 = 1;
  TXOUT = TXOUT - 32;
}
else {
  TX32 = 0;
}

if (TXOUT  >= 16) {
  TX16 = 1;
  TXOUT = TXOUT - 16;
}
else {
  TX16 = 0;
}

if (TXOUT >= 8) {
  TX8 = 1;
  TXOUT = TXOUT - 8;
}
else {
  TX8 = 0;
}

if (TXOUT >= 4) {
  TX4 = 1;
  TXOUT=TXOUT - 4;
}
else {
 TX4 = 0;
}

if (TXOUT >= 2) {
  TX2 = 1;
  TXOUT = TXOUT - 2;
}
else {
  TX2 = 0;
}

TX1 = TXOUT;
```

```
' Use RCV and TX levels as just determined to
‘ set output pin levels
OUT4 = TX1
OUT5 = TX2
OUT6 = TX4
OUT7 = TX8
OUT8 = TX16
OUT9 = TX32

OUT10 = RCV1
OUT11 = RCV2
OUT12 = RCV4
OUT13 = RCV8
OUT14 = RCV16
OUT15 = RCV32

' Go back to beginning of loop and repeat
LOOP

END
```

```
// Use RCV and TX levels as just determined to
// set output pin levels
digitalWrite(TxOUT1, TX1);
digitalWrite(TxOUT2, TX2);
digitalWrite(TxOUT4, TX4);
digitalWrite(TxOUT8, TX8);
digitalWrite(TxOUT16, TX16);
digitalWrite(TxOUT32, TX32);

digitalWrite(RxOUT1, RCV1);
digitalWrite(RxOUT2, RCV2);
digitalWrite(RxOUT4, RCV4);
digitalWrite(RxOUT8, RCV8);
digitalWrite(RxOUT16, RCV16);
digitalWrite(RxOUT32, RCV32);
}
```

Arduino Remote RF Power Monitor

```
  1:
  2: //    W3SZ 8-20-2017 Remote Ethernet Power Meter
  3: //    To work in conjunction with C# client also
  4: //    written by W3SZ 8-20-2017
  5:
  6: #include <Ethernet.h>  //for ethernet port
  7: #include <string.h> // for string handling
  8: #include <EthernetUdp.h>         // UDP library from: bjoern@cs.stanford.edu 12/30/
       2008
  9:
 10: //variables
 11: String commandInputString = "";
 12:
 13: // Enter MAC address and IP address for Arduino below.
 14: byte mac[] = { 0x90, 0xAA, 0xBB, 0xCC, 0xDA, 0x02 };
 15: IPAddress ip(192, 168, 10, 176); //<< ENTER YOUR IP ADDRESS HERE <<
 16:
 17: IPAddress displayIP(192,168,10,244); //IP of computer running C# program to process
       and display data
 18:
 19: unsigned int dataPort = 8888;      // local port to send and receive data on
 20:
 21: // buffers for receiving and sending data
 22: char packetBuffer[UDP_TX_PACKET_MAX_SIZE];  //buffer to hold incoming packet,
 23: char  ReplyBuffer[] = "acknowledged";       // a string to send back
 24:
 25: // An EthernetUDP instance to let us send and receive packets over UDP
 26:
 27: EthernetUDP Udp;
 28:
 29: int VoltA0 = 0;
 30: int VoltA1 = 0;
 31: int VoltA2 = 0;
 32: int VoltA3 = 0;
 33: int VoltA4 = 0;
 34: int VoltA5 = 0;
 35: int VoltA6 = 0;
 36: int VoltA7 = 0;
 37: int VoltA8 = 0;
 38: int VoltA9 = 0;
 39: int VoltA10 = 0;
 40: int VoltA11 = 0;
 41: int VoltA12 = 0;
 42: int VoltA13 = 0;
 43: int VoltA14 = 0;
 44: int VoltA15 = 0;
 45:
 46: String MeterOn = "OFF";  //turns measurement UDP server on or off
 47: String BANDA0 = "ON";     //turns sensor with this numeral on or off
 48: String BANDA1 = "ON";     //turns sensor with this numeral on or off
 49: String BANDA2 = "ON";     //turns sensor with this numeral on or off
 50: String BANDA3 = "ON";     //turns sensor with this numeral on or off
 51: String BANDA4 = "ON";     //turns sensor with this numeral on or off
 52: String BANDA5 = "ON";     //turns sensor with this numeral on or off
 53: String BANDA6 = "ON";     //turns sensor with this numeral on or off
 54: String BANDA7 = "ON";     //turns sensor with this numeral on or off
 55: String BANDA8 = "ON";     //turns sensor with this numeral on or off
 56: String BANDA9 = "ON";     //turns sensor with this numeral on or off
 57: String BANDA10 = "ON";     //turns sensor with this numeral on or off
 58: String BANDA11 = "ON";     //turns sensor with this numeral on or off
 59: String BANDA12 = "ON";     //turns sensor with this numeral on or off
 60: String BANDA13 = "ON";     //turns sensor with this numeral on or off
 61: String BANDA14 = "ON";     //turns sensor with this numeral on or off
 62: String BANDA15 = "ON";     //turns sensor with this numeral on or off
 63:
 64: // ************************************************************
```

```
 65: // ********************  S  E  T  U  P  ************************
 66: // ************************************************************
 67:
 68: void setup() {
 69:
 70:   //set pin modes to input
 71:   pinMode(A0, INPUT);
 72:   pinMode(A1, INPUT);
 73:   pinMode(A2, INPUT);
 74:   pinMode(A3, INPUT);
 75:   pinMode(A4, INPUT);
 76:   pinMode(A5, INPUT);
 77:   pinMode(A6, INPUT);
 78:   pinMode(A7, INPUT);
 79:   pinMode(A8, INPUT);
 80:   pinMode(A9, INPUT);
 81:   pinMode(A10, INPUT);
 82:   pinMode(A11, INPUT);
 83:   pinMode(A12, INPUT);
 84:   pinMode(A13, INPUT);
 85:   pinMode(A14, INPUT);
 86:   pinMode(A15, INPUT);
 87:
 88:   // start the Ethernet connection and the server and the serial port:
 89:   Ethernet.begin(mac, ip);
 90:   Udp.begin(dataPort);
 91:   Serial.begin(9600);
 92:   Serial.println("Starting Server");
 93:   Serial.println (Ethernet.localIP());
 94:
 95: // Print a message to the serial port
 96:
 97:   Serial.println("Pwr Meter");
 98:   Serial.println("1 MHz - 9 GHz");
 99:   Serial.println("W3SZ 08/2017");
100:
101:   delay (4000);
102:
103: } // end of setup
104:
105: // ************************************************************
106: // **********************  L  O  O  P  ************************
107: // ************************************************************
108: //this is the main program loop.
109: //it listens for an HTML client and when it gets input from the client it builds a
         string from the client's input
110: //it then parses the input and if it finds a valid command in the input, it uses that
         command to set each of 16 sensors
111: //(BANDS) ON or OFF or to START or STOP the measurement process altogether
112: //
113: //it reports the command received to the serial monitor and
114: //it calls the function sendReply which reads the Power/SDR values and reports them
         via UDP to C# client running on another
115: //computer
116:
117: void loop() {
118:
119:     //read sensors
120:  VoltA0 = analogRead(A0);        // Read A0 sensor voltage
121:  VoltA1 = analogRead(A1);        // Read A1 sensor voltage
122:  VoltA2 = analogRead(A2);        // Read A2 sensor voltage
123:  VoltA3 = analogRead(A3);        // Read A3 sensor voltage
124:  VoltA4 = analogRead(A4);        // Read A4 sensor voltage
125:
126:  VoltA5 = analogRead(A5);        // Read A5 sensor voltage
127:  VoltA6 = analogRead(A6);        // Read A6 sensor voltage
```

```
 128:  VoltA7 = analogRead(A7);        // Read A7 sensor voltage
 129:  VoltA8 = analogRead(A8);        // Read A8 sensor voltage
 130:  VoltA9 = analogRead(A9);        // Read A9 sensor voltage
 131:
 132:  VoltA10 = analogRead(A10);        // Read A10 sensor voltage
 133:  VoltA11 = analogRead(A11);        // Read A11 sensor voltage
 134:  VoltA12 = analogRead(A12);        // Read A12 sensor voltage
 135:  VoltA13 = analogRead(A13);        // Read A13 sensor voltage
 136:  VoltA14 = analogRead(A14);        // Read A14 sensor voltage
 137:  VoltA15 = analogRead(A15);        // Read A15 sensor voltage
 138:
 139:   // listen for incoming UDP Packet
 140:   // if there's data available, read a packet
 141:   int packetSize = Udp.parsePacket();
 142:   if (packetSize) {
 143:     Serial.print("Received packet of size ");
 144:     Serial.println(packetSize);
 145:     Serial.print("From ");
 146:     IPAddress remote = Udp.remoteIP();
 147:     for (int i = 0; i < 4; i++) {
 148:       Serial.print(remote[i], DEC);
 149:       if (i < 3) {
 150:         Serial.print(".");
 151:       }
 152:     }
 153:     Serial.print(", port ");
 154:     Serial.println(Udp.remotePort());
 155:
 156:     // read the packet into packetBufffer
 157:     Udp.read(packetBuffer, UDP_TX_PACKET_MAX_SIZE);
 158:     Serial.println("Contents:");
 159:     Serial.println(packetBuffer);
 160:
 161:           commandInputString = (String)packetBuffer;
 162:           int stringStart = commandInputString.indexOf('~');
 163:           int stringEnd = commandInputString.indexOf('$');
 164:           String commandOut = commandInputString.substring(1 + stringStart,stringEnd);
 165:           if (commandOut  == "START") {
 166:             String HTMString = "START MEASUREMENT";
 167:             Serial.println(HTMString);
 168:             MeterOn = "ON";
 169:           }
 170:           else if (commandOut  == "STOP") {
 171:             String HTMString = "STOP MEASUREMENT";
 172:             Serial.println(HTMString);
 173:             MeterOn = "OFF";
 174:           }
 175:
 176:           else if (commandOut  == "BANDA0ON") {
 177:             String HTMString = "BAND A0 is ON";
 178:             Serial.println(HTMString);
 179:             BANDA0 = "ON";
 180:           }
 181:           else if (commandOut  == "BANDA0OFF") {
 182:             String HTMString = "BAND A0 is OFF";
 183:             Serial.println(HTMString);
 184:             BANDA0 = "OFF";
 185:           }
 186:
 187:           else if (commandOut  == "BANDA1ON") {
 188:             String HTMString = "BAND A1 is ON";
 189:             Serial.println(HTMString);
 190:             BANDA1 = "ON";
 191:           }
 192:           else if (commandOut  == "BANDA1OFF") {
 193:             String HTMString = "BAND A1 is OFF";
```

```
194:              Serial.println(HTMString);
195:              BANDA1 = "OFF";
196:            }
197:
198:            else if (commandOut  == "BANDA2ON") {
199:              String HTMString = "BAND A2 is ON";
200:              Serial.println(HTMString);
201:              BANDA2 = "ON";
202:            }
203:            else if (commandOut  == "BANDA2OFF") {
204:              String HTMString = "BAND A2 is OFF";
205:              Serial.println(HTMString);
206:              BANDA2 = "OFF";
207:            }
208:
209:            else if (commandOut  == "BANDA3ON") {
210:              String HTMString = "BAND A3 is ON";
211:              Serial.println(HTMString);
212:              BANDA3 = "ON";
213:            }
214:            else if (commandOut  == "BANDA3OFF") {
215:              String HTMString = "BAND A3 is OFF";
216:              Serial.println(HTMString);
217:              BANDA3 = "OFF";
218:            }
219:
220:            else if (commandOut  == "BANDA4ON") {
221:              String HTMString = "BAND A4 is ON";
222:              Serial.println(HTMString);
223:              BANDA4 = "ON";
224:            }
225:            else if (commandOut  == "BANDA4OFF") {
226:              String HTMString = "BAND A4 is OFF";
227:              Serial.println(HTMString);
228:              BANDA4 = "OFF";
229:            }
230:
231:            else if (commandOut  == "BANDA5ON") {
232:              String HTMString = "BAND A5 is ON";
233:              Serial.println(HTMString);
234:              BANDA5 = "ON";
235:            }
236:            else if (commandOut  == "BANDA5OFF") {
237:              String HTMString = "BAND A5 is OFF";
238:              Serial.println(HTMString);
239:              BANDA5 = "OFF";
240:            }
241:
242:            else if (commandOut  == "BANDA6ON") {
243:              String HTMString = "BAND A6 is ON";
244:              Serial.println(HTMString);
245:              BANDA6 = "ON";
246:            }
247:            else if (commandOut  == "BANDA6OFF") {
248:              String HTMString = "BAND A6 is OFF";
249:              Serial.println(HTMString);
250:              BANDA6 = "OFF";
251:            }
252:
253:            else if (commandOut  == "BANDA7ON") {
254:              String HTMString = "BAND A7 is ON";
255:              Serial.println(HTMString);
256:              BANDA7 = "ON";
257:            }
258:            else if (commandOut  == "BANDA7OFF") {
259:              String HTMString = "BAND A7 is OFF";
```

```
260:             Serial.println(HTMString);
261:             BANDA7 = "OFF";
262:           }
263:
264:           else if (commandOut  == "BANDA8ON") {
265:             String HTMString = "BAND A8 is ON";
266:             Serial.println(HTMString);
267:             BANDA8 = "ON";
268:           }
269:           else if (commandOut  == "BANDA8OFF") {
270:             String HTMString = "BAND A8 is OFF";
271:             Serial.println(HTMString);
272:             BANDA8 = "OFF";
273:           }
274:
275:           else if (commandOut  == "BANDA9ON") {
276:             String HTMString = "BAND A9 is ON";
277:             Serial.println(HTMString);
278:             BANDA9 = "ON";
279:           }
280:           else if (commandOut  == "BANDA9OFF") {
281:             String HTMString = "BAND A9 is OFF";
282:             Serial.println(HTMString);
283:             BANDA9 = "OFF";
284:           }
285:
286:           else if (commandOut  == "BANDA10ON") {
287:             String HTMString = "BAND A10 is ON";
288:             Serial.println(HTMString);
289:             BANDA10 = "ON";
290:           }
291:           else if (commandOut  == "BANDA10OFF") {
292:             String HTMString = "BAND A10 is OFF";
293:             Serial.println(HTMString);
294:             BANDA10 = "OFF";
295:           }
296:
297:           else if (commandOut  == "BANDA11ON") {
298:             String HTMString = "BAND A11 is ON";
299:             Serial.println(HTMString);
300:             BANDA11 = "ON";
301:           }
302:           else if (commandOut  == "BANDA11OFF") {
303:             String HTMString = "BAND A11 is OFF";
304:             Serial.println(HTMString);
305:             BANDA11 = "OFF";
306:           }
307:
308:           else if (commandOut  == "BANDA12ON") {
309:             String HTMString = "BAND A12 is ON";
310:             Serial.println(HTMString);
311:             BANDA12 = "ON";
312:           }
313:           else if (commandOut  == "BANDA12OFF") {
314:             String HTMString = "BAND A12 is OFF";
315:             Serial.println(HTMString);
316:             BANDA12 = "OFF";
317:           }
318:
319:           else if (commandOut  == "BANDA13ON") {
320:             String HTMString = "BAND A13 is ON";
321:             Serial.println(HTMString);
322:             BANDA13 = "ON";
323:           }
324:           else if (commandOut  == "BANDA13OFF") {
325:             String HTMString = "BAND A13 is OFF";
```

```
            Serial.println(HTMString);
            BANDA13 = "OFF";
          }

          else if (commandOut  == "BANDA14ON") {
            String HTMString = "BAND A14 is ON";
            Serial.println(HTMString);
            BANDA14 = "ON";
          }
          else if (commandOut  == "BANDA14OFF") {
            String HTMString = "BAND A14 is OFF";
            Serial.println(HTMString);
            BANDA14 = "OFF";
          }

          else if (commandOut  == "BANDA15ON") {
            String HTMString = "BAND A15 is ON";
            Serial.println(HTMString);
            BANDA15 = "ON";
          }
          else if (commandOut  == "BANDA15OFF") {
            String HTMString = "BAND A15 is OFF";
            Serial.println(HTMString);
            BANDA15 = "OFF";
          }
          commandInputString = "";
   } // end if UDP data received

   //send Sensor Data
    String data = "DATA";

     if(BANDA0 =="ON"){
       data = data + ",A00=" +String(VoltA0);
     }
     if(BANDA1 =="ON"){
       data = data + ",A01=" +String(VoltA1);
     }
     if(BANDA2 =="ON"){
       data = data + ",A02=" +String(VoltA2);
     }
     if(BANDA3 =="ON"){
       data = data + ",A03=" +String(VoltA3);
     }
     if(BANDA4 =="ON"){
       data = data + ",A04=" +String(VoltA4);
     }
     if(BANDA5 =="ON"){
       data = data + ",A05=" +String(VoltA5);
     }
     if(BANDA6 =="ON"){
       data = data + ",A06=" +String(VoltA6);
     }
     if(BANDA7 =="ON"){
       data = data + ",A07=" +String(VoltA7);
     }
     if(BANDA8 =="ON"){
       data = data + ",A08=" +String(VoltA8);
     }
     if(BANDA9 =="ON"){
       data = data + ",A09=" +String(VoltA9);
     }
     if(BANDA10 =="ON"){
       data = data + ",A10=" +String(VoltA10);
     }
     if(BANDA11 =="ON"){
       data = data + ",A11=" +String(VoltA11);
```

```
392:     }
393:     if(BANDA12 =="ON"){
394:       data = data + ",A12=" +String(VoltA12);
395:     }
396:     if(BANDA13 =="ON"){
397:       data = data + ",A13=" +String(VoltA13);
398:     }
399:     if(BANDA14 =="ON"){
400:       data = data + ",A14=" +String(VoltA14);
401:     }
402:     if(BANDA15 =="ON"){
403:       data = data + ",A15=" +String(VoltA15);
404:     }
405:
406:    if(MeterOn == "ON")
407:    {
408:     int datalength = 1 + data.length();
409:     char databuf[datalength];
410:     data.toCharArray(databuf, datalength);
411:     // send a reply to the IP address and port that sent us the packet we received
412:     Udp.beginPacket(displayIP, dataPort);
413:     Udp.write(databuf);
414:     Udp.endPacket();
415:  //   Serial.println(datalength);
416:   //  Serial.print("DATA IS: ");
417:  //   Serial.println(data);
418:  //   Serial.print("DATABUF IS: ");
419:  //   Serial.println(databuf);
420:    }
421:   delay(50);
422: } //end loop
423:
424:
```

Arduino Device Band Switch

```
  1: //By W3SZ
  2: //to take UDP input from N1MM and perform device switching of
  3: //Microphone, TxDigitalAudio, CW_Key, Two Footswitches, Two Receive Audio Channels
  4: //Designed for SO1V use with N1MM
  5: //this is small enough to run on Uno, etc.
  6:
  7: //Derived from python code by W3SZ for ethernet N1MM IF/xvtr
  8: //bandswitching
  9:
 10:
 11: // Import Libraries
 12:
 13: #include <Ethernet.h>     //for ethernet port
 14: #include <string.h>      // for string handling
 15:
 16: //define constant pin aliases
 17: const int MicPin = 2; //number of Microphone pin
 18: const int TxDigitalAudioPin = 3; //number of Main Digital Audio pin
 19: const int CW_KeyPin = 4; //number of CW Key pin
 20: const int LeftFootswitchPin = 5; //number of Left Footswitch pin
 21: const int RightFootswitchPin = 6; //number of Right Footswitch pin
 22: const int ReceiveAudioOnePin = 7; //number of Receive Audio One pin
 23: const int ReceiveAudioTwoPin = 8; //number of Receive Audio Two pin
 24:
 25: //Define and Initialize variables and Constants
 26:
 27: // Enter MAC address and IP address for Arduino below.
 28: byte mac[] = { 0x90, 0xAA, 0xBB, 0xCC, 0xDA, 0x02 };
 29: IPAddress ip(192, 168, 10, 176); //<< ENTER YOUR IP ADDRESS HERE <<
 30:
 31: unsigned int dataPort = 13063;      // UDP port as specified in N1MM Setup
 32:
 33: // An EthernetUDP instance to let us send and receive packets over UDP
 34: EthernetUDP Udp;
 35:
 36: // buffers for receiving data
 37:
 38: String commandInputString = "";  //input string created from UDP read buffer
 39: String commandOut = "";  //parsed 2 digit band data, used by SetBand procedure to set
         relays
 40:
 41: void setup() {
 42:   // put your setup code here, to run once:
 43:   // initialize GPIO pins as output pins
 44:   pinMode(MicPin, OUTPUT);
 45:   pinMode(TxDigitalAudioPin, OUTPUT);
 46:   pinMode(CW_KeyPin, OUTPUT);
 47:   pinMode(LeftFootswitchPin, OUTPUT);
 48:   pinMode(RightFootswitchPin, OUTPUT);
 49:   pinMode(ReceiveAudioOnePin, OUTPUT);
 50:   pinMode(ReceiveAudioTwoPin, OUTPUT);
 51:
 52:   //initialize all GPIO pin values to low
 53:   digitalWrite(MicPin, LOW);
 54:   digitalWrite(TxDigitalAudioPin, LOW);
 55:   digitalWrite(CW_KeyPin, LOW);
 56:   digitalWrite(LeftFootswitchPin, LOW);
 57:   digitalWrite(RightFootswitchPin, LOW);
 58:   digitalWrite(ReceiveAudioOnePin, LOW);
 59:   digitalWrite(ReceiveAudioTwoPin, LOW);
 60:
 61:   // start the Ethernet connection and the server and the serial port:
 62:   Ethernet.begin(mac, ip);
 63:   Udp.begin(dataPort);
 64:   Serial.begin(9600);
 65:   Serial.println("Starting Server");
```

```
  66:   Serial.println (Ethernet.localIP());
  67:
  68:   //send some code to serial port so know when Arduino powers up
  69:   Serial.println("N1MM Arduino Device Bandswitch");
  70:   Serial.println("W3SZ 08/2017");
  71:
  72:   delay (4000);
  73: }
  74:
  75: void loop() {
  76:
  77: char packetBuffer[500];  //buffer to hold incoming packet
  78:   // listen for incoming UDP Packet
  79:   // if there's data available, read a packet
  80: int packetSize = Udp.parsePacket();
  81:   if (packetSize) {
  82:      while(Udp.available())
  83:      {
  84:     // read the packet into packetBuffer
  85:
  86:     Udp.read(packetBuffer, packetSize);
  87:
  88:    // create command string from input buffer
  89:     commandInputString = (String)packetBuffer;
  90:     Serial.println(commandInputString);
  91:
  92:     //string2 is used to pick off the radio frequency that is used for setting the
           band and thus the device relays
  93:     int string2 = commandInputString.indexOf("<Freq>");
  94:     commandOut = commandInputString.substring(string2 + 6, string2 + 8);
  95:     SetBand(commandOut);
  96:
  97:
  98:     //this duplicate code is because the loop seems to process new data only every
           other new-data-event
  99:     Udp.read(packetBuffer, packetSize);
 100:     commandInputString = (String)packetBuffer;
 101:     string2 = commandInputString.indexOf("<Freq>");
 102:     commandOut = commandInputString.substring(string2 + 6, string2 + 8);
 103:     SetBand(commandOut);
 104:
 105:     commandInputString = "";
 106:     commandOut = "";
 107:
 108:     Udp.flush();
 109:
 110:      } //end of while available
 111:   } // end "if UDP data received"
 112:
 113:
 114: }
 115:
 116: void SetBand(String commandOut)
 117: {
 118:   if(commandOut == "50" || commandOut == "14" || commandOut == "22" || commandOut == "
          43")
 119:   {
 120:     //This is low band radio, set relays off to connect to this radio
 121:
 122:   digitalWrite(MicPin, LOW);
 123:   digitalWrite(TxDigitalAudioPin, LOW);
 124:   digitalWrite(CW_KeyPin, LOW);
 125:   digitalWrite(LeftFootswitchPin, LOW);
 126:   digitalWrite(RightFootswitchPin, LOW);
 127:   digitalWrite(ReceiveAudioOnePin, LOW);
 128:   digitalWrite(ReceiveAudioTwoPin, LOW);
```

```
129:   }
130:   else if (commandOut == "90" || commandOut == "12" || commandOut == "23" ||
           commandOut == "34" || commandOut == "57" || commandOut == "10" || commandOut ==
           "24")
131:   {
132:     //This is microwave radio, set telays to ON to connect to this radio
133:   digitalWrite(MicPin, HIGH);
134:   digitalWrite(TxDigitalAudioPin, HIGH);
135:   digitalWrite(CW_KeyPin, HIGH);
136:   digitalWrite(LeftFootswitchPin, HIGH);
137:   digitalWrite(RightFootswitchPin, HIGH);
138:   digitalWrite(ReceiveAudioOnePin, HIGH);
139:   digitalWrite(ReceiveAudioTwoPin, HIGH);
140:   }
141:
142: }
```


Super MEGA SO2RDuino
BLEND
1
2
CW
MIC
RX
PTT A
PTT B


1
2
AUX
RADIO
2
1
PTT
A
CW
PTT
B

```
// SO2Rduino - An SO2R Box built on an Arduino clone
//
// Copyright 2010, Paul Young

// This is a simple implementation of an OTRSP SO2R Device.
// See http://www.k1xm.org/otrsp/ for information on the
// protocol.

// This implementation does not use the Arduino libraries
// for much.  It also does not use interrupts except for
// the one ms timer set up by the Arduino environment.
// As long as the main loop takes less than about 1 ms in
// the worst case it can keep up with the UART which is
// run at 9600 baud.

// Modifications made by W3SZ August 2017 to permit use with
// modern Arduino IDEs, and to correct bug where Aux2 values displayed
// with ?AUX2 command were actually AUX1's values
// further modifications by W3SZ September 2017:
// use with MEGA:
// eliminated shift register for aux outputs;
// each output (aux1 and aux2) has 16 GPIO pins for output values 0-15
// separated tx2 into cw2, mic2, ptt_a_2, ptt_b_2
// I have found this extremely helpful for vhr/uhf/microwave operation
// over the past 15 years or so.  There are now
// separate manual spdt toggles with center off position for:
// rcv, cw, mic, ptt_a, ptt_b.
// removed rx = !tx function as N1MM supplies this with dueling CQs and
// some settings of this could really mess up dueling CQs.
// I just commented it out so it can be reinstituted if desired.

#include <EEPROM.h>
#include "uart.h"

// Misc. definitions
#define DEBOUNCE 10                     // Debounce time (ms)

#define EEPROM_RESERVED 0
#define EEPROM_MONO     1
#define EEPROM_LATCH    2

// The front panel switches have three values.  They are
// declared as byte because the compiler generates better
// code that way.
typedef enum {
    RADIO_AUTO = 0,
    RADIO_1,
    RADIO_2
} radio_switch;

// Switches and inputs
static boolean          ptt_computer;   // Computer asserts PTT
static byte             ptt_debounce;   // PTT switch debounce (ms)

static byte             cw_switch;      // Front panel cw switch
static byte             mic_switch;      // Front panel mic switch
static byte             ptt_a_switch;      // Front panel ptt_a switch
static byte             ptt_b_switch;      // Front panel ptt_b switch
static byte             rx_switch;      // Front panel RX switch
static byte             cw_debounce;    // CW switch debounce (ms)
static byte             mic_debounce;    // MIC switch debounce (ms)
static byte             ptt_a_debounce;    // PTT_A switch debounce (ms)
static byte             ptt_b_debounce;    // PTT_B switch debounce (ms)
static byte             rx_debounce;    // RX switch debounce (ms)
static byte             debounce;       // Debounce timer

```

```
 67: // Outputs
 68: static boolean          cw2;            // Radio 2 selected for CW
 69: static boolean          mic2;            // Radio 2 selected for MIC
 70: static boolean          ptt_a_2;            // Radio 2 selected for PTT_A
 71: static boolean          ptt_b_2;            // Radio 2 selected for PTT_B
 72: static boolean          rx2;            // Listening to radio 2
 73: static boolean          stereo;         // Listening in stereo
 74: static boolean          tx2_computer;   // Computer says select TX 2
 75: static boolean          rx2_computer;   // Computer says select RX 2
 76: static boolean          stereo_computer;// Computer says RX stereo
 77:
 78: // Aux outputs
 79: static byte             aux1;           // Aux 1 output
 80: static byte             aux2;           // Aux 2 output
 81:
 82: // OTRSP events
 83: static boolean          event_tx;       // TX event is enabled
 84: static boolean          event_rx;       // RX event is enabled
 85: static boolean          event_ab;       // Abort event is enabled
 86: static boolean          event_cr0;      // PTT switch event is enabled
 87:
 88: // Special and misc state
 89: static boolean          mono;           // Stereo is not allowed
 90: static boolean          latch;          // Move phones to non-TX radio
 91:
 92: // Forward references
 93: static boolean check_switches();
 94: static void do_relays();
 95: static void do_ptt();
 96: static void do_aux();
 97: static void check_key_ptt();
 98: static void do_command();
 99:
100:
101: //define constant pin aliases
102: //output pins
103: const int RX2 = 10; //RX2 OUTPUT (SET_RX2, CLEAR_RX2) //was D13
104: const int STEREO = 11; //STEREO OUTPUT (SET_STEREO, CLEAR_STEREO)
105: const int MIC2 = 9; //MIC2 RELAY AND LED OUTPUT (SET_TX2, CLEAR_TX2)
106: const int MIC1_LED = 8; //MIC2 LED OUTPUT (SET_TX1, CLEAR_TX1)
107: const int RX1_LED = A4; //RX1 LED OUTPUT (SET_RX1_LED, CLEAR_RX1_LED)
108: const int RX2_LED = A5; //RX2 LED OUTPUT (SET_RX2_LED, CLEAR_RX2_LED)
109: const int PTT1 = 4; //PTT 1 OUTPUT (SET_PTT1)
110: const int PTT2 = 5; //PTT 2 OUTPUT (SET_PTT2)
111: const int CW1 = 6; //CW 1 OUTPUT (SET_CW1)
112: const int CW2 = 7; //CW 2 OUTPUT (SET_CW2)
113:
114: //input pins
115: const int GET_PTT_A_SW = 12; //PTT_A FOOTSWITCH INPUT (GET_PTT_SWITCH)
116: const int GET_PTT_B_SW = 13; //PTT_B FOOTSWITCH INPUT (GET_PTT_SWITCH)
117: const int GET_PTT_DTR = 2; //PTT DTR INPUT (GET_PTT_DTR) (RTS FROM PIN 14 OF CH340G)
118: const int GET_CW_KEY = 3; //CW KEY INPUT (GET_CW_KEY)
119: const int CW1_MAN = A0; //TX1 SWITCH INPUT (GET_TX1_SWITCH)
120: const int CW2_MAN = A1; //TX2 SWITCH INPUT (GET TX2_sWITCH)
121: const int RX1_MAN = A2; //RX1 SWITCH INPUT (GET_RX1_SWITCH)
122: const int RX2_MAN = A3; //RX2 SWITCH INPUT (GET_RX2_SWITCH)
123:
124: //constants below added with conversion to MEGA with
125: //conversion of serial aux outputs to parallel
126: //both AUX1 and AUX outputs provided
127: //Two PTTs provided for: PTT_A and PTT_B
128: //each can be computer controlled or manual depending on
129: //setting of manual PTT_A 1-off-2 switch
130: //and manual PTT_B 1-off-2 switch
131: //GET_TX1 split into CW1_MAN and MIC1_MAN and PTT1_MAN
132: //GET_TX2 split into CW2_MAN and MIC2_MAN and PTT2_MAN
```

```
133: //All 3 mech switches are center off DPDT toggles
134: const int MIC1_MAN = 14;
135: const int MIC2_MAN = 15;
136: const int PTT_A_1_MAN = 16;
137: const int PTT_A_2_MAN = 17;
138: const int PTT_B_1_MAN = 18;
139: const int PTT_B_2_MAN = 19;
140:
141: //below are output pins for AUX1 and AUX2
142: const int AUX1_0 = 22;
143: const int AUX1_1 = 24;
144: const int AUX1_2 = 26;
145: const int AUX1_3 = 28;
146: const int AUX1_4 = 30;
147: const int AUX1_5 = 32;
148: const int AUX1_6 = 34;
149: const int AUX1_7 = 36;
150: const int AUX1_8 = 38;
151: const int AUX1_9 = 40;
152: const int AUX1_10 = 42;
153: const int AUX1_11 = 44;
154: const int AUX1_12 = 46;
155: const int AUX1_13 = 48;
156: const int AUX1_14 = 50;
157: const int AUX1_15 = 52;
158:
159: const int AUX2_0 = 23;
160: const int AUX2_1 = 25;
161: const int AUX2_2 = 27;
162: const int AUX2_3 = 29;
163: const int AUX2_4 = 31;
164: const int AUX2_5 = 33;
165: const int AUX2_6 = 35;
166: const int AUX2_7 = 37;
167: const int AUX2_8 = 39;
168: const int AUX2_9 = 41;
169: const int AUX2_10 = 43;
170: const int AUX2_11 = 45;
171: const int AUX2_12 = 47;
172: const int AUX2_13 = 49;
173: const int AUX2_14 = 51;
174: const int AUX2_15 = 53;
175:
176: void
177: setup()
178: //------------------------------------------------------------------------
179: // Initialize the SO2RDuino
180: //------------------------------------------------------------------------
181: {
182:     cw_switch = RADIO_AUTO;
183:     mic_switch = RADIO_AUTO;
184:     ptt_a_switch = RADIO_AUTO;
185:     ptt_b_switch = RADIO_AUTO;
186:     rx_switch = RADIO_AUTO;
187:     debounce = millis();
188:     rx_debounce = 0;
189:     cw_debounce = 0;
190:     mic_debounce = 0;
191:     ptt_a_debounce = 0;
192:     ptt_b_debounce = 0;
193:
194: //DEFINE GPIO PIN MODES
195: //OUTPUT pins
196: pinMode(RX2,OUTPUT);
197: pinMode(STEREO,OUTPUT);
198: pinMode(MIC2,OUTPUT);
```

```
199: pinMode(MIC1_LED,OUTPUT);
200: pinMode(RX1_LED,OUTPUT);
201: pinMode(RX2_LED,OUTPUT);
202: pinMode(PTT1,OUTPUT);
203: pinMode(PTT2,OUTPUT);
204: pinMode(CW1,OUTPUT);
205: pinMode(CW2,OUTPUT);
206:
207: //INPUT pins
208: pinMode(GET_PTT_A_SW,INPUT);
209: pinMode(GET_PTT_B_SW,INPUT);
210: pinMode(GET_PTT_DTR,INPUT);
211: pinMode(GET_CW_KEY,INPUT);
212: pinMode(CW1_MAN,INPUT);
213: pinMode(CW2_MAN,INPUT);
214: pinMode(RX1_MAN,INPUT);
215: pinMode(RX2_MAN,INPUT);
216:
217: pinMode(MIC1_MAN,INPUT);
218: pinMode(MIC2_MAN,INPUT);
219: pinMode(PTT_A_1_MAN,INPUT);
220: pinMode(PTT_A_2_MAN,INPUT);
221: pinMode(PTT_B_1_MAN,INPUT);
222: pinMode(PTT_B_2_MAN,INPUT);
223:
224: //AUXILIARY (BAND DATA) OUTPUT pins
225: pinMode(AUX1_0,OUTPUT);
226: pinMode(AUX1_1,OUTPUT);
227: pinMode(AUX1_2,OUTPUT);
228: pinMode(AUX1_3,OUTPUT);
229: pinMode(AUX1_4,OUTPUT);
230: pinMode(AUX1_5,OUTPUT);
231: pinMode(AUX1_6,OUTPUT);
232: pinMode(AUX1_7,OUTPUT);
233: pinMode(AUX1_8,OUTPUT);
234: pinMode(AUX1_9,OUTPUT);
235: pinMode(AUX1_10,OUTPUT);
236: pinMode(AUX1_11,OUTPUT);
237: pinMode(AUX1_12,OUTPUT);
238: pinMode(AUX1_13, OUTPUT);
239: pinMode(AUX1_14, OUTPUT);
240: pinMode(AUX1_15, OUTPUT);
241:
242: pinMode(AUX2_0, OUTPUT);
243: pinMode(AUX2_1, OUTPUT);
244: pinMode(AUX2_2, OUTPUT);
245: pinMode(AUX2_3, OUTPUT);
246: pinMode(AUX2_4, OUTPUT);
247: pinMode(AUX2_5, OUTPUT);
248: pinMode(AUX2_6, OUTPUT);
249: pinMode(AUX2_7, OUTPUT);
250: pinMode(AUX2_8, OUTPUT);
251: pinMode(AUX2_9, OUTPUT);
252: pinMode(AUX2_10, OUTPUT);
253: pinMode(AUX2_11, OUTPUT);
254: pinMode(AUX2_12, OUTPUT);
255: pinMode(AUX2_13, OUTPUT);
256: pinMode(AUX2_14, OUTPUT);
257: pinMode(AUX2_15, OUTPUT);
258:
259: //INITIALIZE GPIO OUTPUT PINS TO LOW
260: digitalWrite(RX2,LOW);
261: digitalWrite(STEREO,LOW);
262: digitalWrite(MIC2,LOW);
263: digitalWrite(MIC1_LED,LOW);
264: digitalWrite(RX1_LED,LOW);
```

```
265: digitalWrite(RX2_LED,LOW);
266: digitalWrite(PTT1,LOW);
267: digitalWrite(PTT2,LOW);
268: digitalWrite(CW1,LOW);
269: digitalWrite(CW2,LOW);
270:
271: digitalWrite(AUX1_0,LOW);
272: digitalWrite(AUX1_1,LOW);
273: digitalWrite(AUX1_2,LOW);
274: digitalWrite(AUX1_3,LOW);
275: digitalWrite(AUX1_4,LOW);
276: digitalWrite(AUX1_5,LOW);
277: digitalWrite(AUX1_6,LOW);
278: digitalWrite(AUX1_7,LOW);
279: digitalWrite(AUX1_8,LOW);
280: digitalWrite(AUX1_9,LOW);
281: digitalWrite(AUX1_10,LOW);
282: digitalWrite(AUX1_11,LOW);
283: digitalWrite(AUX1_12,LOW);
284: digitalWrite(AUX1_13, LOW);
285: digitalWrite(AUX1_14, LOW);
286: digitalWrite(AUX1_15, LOW);
287:
288: digitalWrite(AUX2_0, LOW);
289: digitalWrite(AUX2_1, LOW);
290: digitalWrite(AUX2_2, LOW);
291: digitalWrite(AUX2_3, LOW);
292: digitalWrite(AUX2_4, LOW);
293: digitalWrite(AUX2_5, LOW);
294: digitalWrite(AUX2_6, LOW);
295: digitalWrite(AUX2_7, LOW);
296: digitalWrite(AUX2_8, LOW);
297: digitalWrite(AUX2_9, LOW);
298: digitalWrite(AUX2_10, LOW);
299: digitalWrite(AUX2_11, LOW);
300: digitalWrite(AUX2_12, LOW);
301: digitalWrite(AUX2_13, LOW);
302: digitalWrite(AUX2_14, LOW);
303: digitalWrite(AUX2_15, LOW);
304:
305: //SET PULLUP RESISTORS ON INPUTS
306: digitalWrite(GET_PTT_A_SW,HIGH);
307: digitalWrite(GET_PTT_B_SW,HIGH);
308: digitalWrite(GET_PTT_DTR,LOW);
309: digitalWrite(GET_CW_KEY,HIGH);
310: digitalWrite(CW1_MAN,HIGH);
311: digitalWrite(CW2_MAN,HIGH);
312: digitalWrite(RX1_MAN,HIGH);
313: digitalWrite(RX2_MAN,HIGH);
314:
315: digitalWrite(MIC1_MAN,HIGH);
316: digitalWrite(MIC2_MAN,HIGH);
317: digitalWrite(PTT_A_1_MAN,HIGH);
318: digitalWrite(PTT_A_2_MAN,HIGH);
319: digitalWrite(PTT_B_1_MAN,HIGH);
320: digitalWrite(PTT_B_2_MAN,HIGH);
321:
322:
323:
324:
325:     // Initialize to TX1, RX1, mono, not transmitting
326:     cw2 = false;
327:     mic2 = false;
328:     ptt_a_2 = false;
329:     ptt_b_2 = false;
330:     rx2 = false;
```

```
331:     stereo = false;
332:     tx2_computer = false;
333:     rx2_computer = false;
334:     stereo_computer = false;
335:
336:     event_tx = false;
337:     event_rx = false;
338:     event_ab = false;
339:     event_cr0 = false;
340:
341:     aux1 = 0;
342:     aux2 = 0;
343:
344:     // The values for mono and latch are stored in EEPROM
345:     mono = (bool)EEPROM.read(EEPROM_MONO);
346:     latch = (bool)EEPROM.read(EEPROM_LATCH);
347:
348:     // Initialize the uart
349:     uart_init();
350:
351:     // Set up the aux port
352:     do_aux();
353:
354:     // Wait for the switches to settle
355:     delay(DEBOUNCE);
356:
357:     check_switches();
358:     do_relays();
359:
360: }
361:
362: void
363: loop()
364: //------------------------------------------------------------------------
365: // Main loop
366: //------------------------------------------------------------------------
367: {
368:     if (check_switches()) {
369:         do_relays();
370:     }
371:
372:     check_key_ptt();
373:
374:     if (do_uart()) {
375:         do_command();
376:         uart_clear_buffer();
377:     }
378:
379:     check_key_ptt();
380: }
381:
382: static boolean check_switches()
383: //------------------------------------------------------------------------
384: // Check front panel switches and the PTT switch
385: //------------------------------------------------------------------------
386: {
387:
388:     // When a switch opens or closes the contacts can bounce a
389:     // few times, opening and closing before it settles.  To
390:     // eliminate this the code ignores the switch for a few
391:     // milliseconds after it changes.
392:     byte now = millis();
393:     boolean changed = false;
394:
395:     // Switches are closed when LOW
396:
```

```
397:     // Only do once per millisecond max
398:     if (now == debounce) {
399:         return false;
400:     }
401:     debounce = now;
402:
403:     // Check the front-panel CW switch
404:     if (cw_debounce) {
405:         cw_debounce --;
406:     }
407:     else {
408:         if (!digitalRead(CW1_MAN)) {
409:             if (cw_switch != RADIO_1) {
410:                 cw_debounce = DEBOUNCE;
411:                 cw_switch = RADIO_1;
412:                 changed = true;
413:             }
414:         }
415:         else {
416:             if (!digitalRead(CW2_MAN)) {
417:                 if (cw_switch != RADIO_2) {
418:                     cw_debounce = DEBOUNCE;
419:                     cw_switch = RADIO_2;
420:                     changed = true;
421:                 }
422:             }
423:             else {
424:                 if (cw_switch != RADIO_AUTO) {
425:                     cw_debounce = DEBOUNCE;
426:                     cw_switch = RADIO_AUTO;
427:                     changed = true;
428:                 }
429:             }
430:         }
431:     }
432:
433:
434:     // Check the front-panel MIC switch
435:     if (mic_debounce) {
436:         mic_debounce --;
437:     }
438:     else {
439:         if (!digitalRead(MIC1_MAN)) {
440:             if (mic_switch != RADIO_1) {
441:                 mic_debounce = DEBOUNCE;
442:                 mic_switch = RADIO_1;
443:                 changed = true;
444:             }
445:         }
446:         else {
447:             if (!digitalRead(MIC2_MAN)) {
448:                 if (mic_switch != RADIO_2) {
449:                     mic_debounce = DEBOUNCE;
450:                     mic_switch = RADIO_2;
451:                     changed = true;
452:                 }
453:             }
454:             else {
455:                 if (mic_switch != RADIO_AUTO) {
456:                     mic_debounce = DEBOUNCE;
457:                     mic_switch = RADIO_AUTO;
458:                     changed = true;
459:                 }
460:             }
461:         }
462:     }
```

```
463:
464:
465:     // Check the front-panel PTT_A switch
466:     if (ptt_a_debounce) {
467:         ptt_a_debounce --;
468:     }
469:     else {
470:         if (!digitalRead(PTT_A_1_MAN)) {
471:             if (ptt_a_switch != RADIO_1) {
472:                 ptt_a_debounce = DEBOUNCE;
473:                 ptt_a_switch = RADIO_1;
474:                 changed = true;
475:             }
476:         }
477:         else {
478:             if (!digitalRead(PTT_A_2_MAN)) {
479:                 if (ptt_a_switch != RADIO_2) {
480:                     ptt_a_debounce = DEBOUNCE;
481:                     ptt_a_switch = RADIO_2;
482:                     changed = true;
483:                 }
484:             }
485:             else {
486:                 if (ptt_a_switch != RADIO_AUTO) {
487:                     ptt_a_debounce = DEBOUNCE;
488:                     ptt_a_switch = RADIO_AUTO;
489:                     changed = true;
490:                 }
491:             }
492:         }
493:     }
494:
495:
496:     // Check the front-panel PTT_B switch
497:     if (ptt_b_debounce) {
498:         ptt_b_debounce --;
499:     }
500:     else {
501:         if (!digitalRead(PTT_A_1_MAN)) {
502:             if (ptt_b_switch != RADIO_1) {
503:                 ptt_b_debounce = DEBOUNCE;
504:                 ptt_b_switch = RADIO_1;
505:                 changed = true;
506:             }
507:         }
508:         else {
509:             if (!digitalRead(PTT_A_2_MAN)) {
510:                 if (ptt_b_switch != RADIO_2) {
511:                     ptt_b_debounce = DEBOUNCE;
512:                     ptt_b_switch = RADIO_2;
513:                     changed = true;
514:                 }
515:             }
516:             else {
517:                 if (ptt_b_switch != RADIO_AUTO) {
518:                     ptt_b_debounce = DEBOUNCE;
519:                     ptt_b_switch = RADIO_AUTO;
520:                     changed = true;
521:                 }
522:             }
523:         }
524:     }
525:
526:     // Check the front-panel RX switch
527:     if (rx_debounce) {
528:         rx_debounce--;
```

```
529:     }
530:     else {
531:         if (!digitalRead(RX1_MAN)) {
532:             if (rx_switch != RADIO_1) {
533:                 rx_debounce = DEBOUNCE;
534:                 rx_switch = RADIO_1;
535:                 changed = true;
536:             }
537:         }
538:         else {
539:             if (!digitalRead(RX2_MAN)) {
540:                 if (rx_switch != RADIO_2) {
541:                     rx_debounce = DEBOUNCE;
542:                     rx_switch = RADIO_2;
543:                     changed = true;
544:                 }
545:             }
546:             else {
547:                 if (rx_switch != RADIO_AUTO) {
548:                     rx_debounce = DEBOUNCE;
549:                     rx_switch = RADIO_AUTO;
550:                     changed = true;
551:                 }
552:             }
553:         }
554:     }
555:
556:     // Check the PTT_A footswitch
557:     // Note:  PTT_A switch is LOW when active.
558:     if (ptt_a_debounce) {
559:         ptt_a_debounce--;
560:     }
561:     else {
562:         if (digitalRead(GET_PTT_A_SW)) {
563:             // Switch is open
564:             if (ptt_a_switch) {
565:                 // Switch was closed
566:                 ptt_a_debounce = DEBOUNCE;
567:                 ptt_a_switch = false;
568:                 do_ptt();
569:                 if (event_cr0) {
570:                     uart_send_prog_string(PSTR("$CR00\r"));
571:                 }
572:             }
573:         }
574:         else {
575:             // Switch is closed
576:             if (!ptt_a_switch) {
577:                 // Switch was closed
578:                 ptt_a_debounce = DEBOUNCE;
579:                 ptt_a_switch = true;
580:                 do_ptt();
581:                 if (event_cr0) {
582:                     uart_send_prog_string(PSTR("$CR01\r"));
583:                 }
584:             }
585:         }
586:     }
587:
588:     // Check the PTT_B footswitch
589:     // Note:  PTT_B switch is LOW when active.
590:     if (ptt_b_debounce) {
591:         ptt_b_debounce--;
592:     }
593:     else {
594:         if (digitalRead(GET_PTT_B_SW)) {
```

```
595:             // Switch is open
596:             if (ptt_b_switch) {
597:                 // Switch was closed
598:                 ptt_b_debounce = DEBOUNCE;
599:                 ptt_b_switch = false;
600:                 do_ptt();
601:                 if (event_cr0) {
602:                     uart_send_prog_string(PSTR("$CR00\r"));
603:                 }
604:             }
605:         }
606:         else {
607:             // Switch is closed
608:             if (!ptt_b_switch) {
609:                 // Switch was closed
610:                 ptt_b_debounce = DEBOUNCE;
611:                 ptt_b_switch = true;
612:                 do_ptt();
613:                 if (event_cr0) {
614:                     uart_send_prog_string(PSTR("$CR01\r"));
615:                 }
616:             }
617:         }
618:     }
619: 
620:     return changed;
621: }
622: 
623: static void do_ptt()
624: //---------------------------------------------------------------------
625: // Set or clear PTT
626: //---------------------------------------------------------------------
627: {
628: 
629:   //deal with PTT FROM COMPUTER
630:     if (ptt_computer) //if computer is asserting PTT
631:     {
632:       if((tx2_computer && digitalRead(PTT_A_1_MAN)) || !digitalRead(PTT_A_2_MAN))
633:           {
634:             digitalWrite(PTT2, HIGH);
635:           }
636:       if ((!tx2_computer && digitalRead(PTT_A_2_MAN)) || !digitalRead(PTT_A_1_MAN))
637:           {
638:             digitalWrite(PTT1, HIGH);
639:           }
640:     }
641:     else // if not ptt_computer
642:     {
643:       if((tx2_computer && digitalRead(PTT_A_1_MAN)) || !digitalRead(PTT_A_2_MAN) )
644:           {
645:             digitalWrite(PTT2, LOW);
646:           }
647:       if ((!tx2_computer && digitalRead(PTT_A_2_MAN)) || !digitalRead(PTT_A_1_MAN))
648:           {
649:             digitalWrite(PTT1, LOW);
650:           }
651:     }
652: 
653:     //DEAL WITH PTT FROM PTT_A
654:     if (!digitalRead(GET_PTT_A_SW)) //if PTT_A is asserting PTT
655:     {
656:       if((tx2_computer && digitalRead(PTT_A_1_MAN)) || !digitalRead(PTT_A_2_MAN))
657:           {
658:             digitalWrite(PTT2, HIGH);
659:           }
660:       if ((!tx2_computer && digitalRead(PTT_A_2_MAN)) || !digitalRead(PTT_A_1_MAN))
```

```
661:            {
662:              digitalWrite(PTT1, HIGH);
663:            }
664:     }
665:     else  // if PTT_A not asserting PTT
666:     {
667:        if((tx2_computer && digitalRead(PTT_A_1_MAN)) || !digitalRead(PTT_A_2_MAN))
668:            {
669:              digitalWrite(PTT2, LOW);
670:            }
671:        if ((!tx2_computer && digitalRead(PTT_A_2_MAN)) || !digitalRead(PTT_A_1_MAN))
672:            {
673:              digitalWrite(PTT1, LOW);
674:            }
675:     }
676:
677:    //DEAL WITH PTT FROM PTT_B
678:
679:     if (!digitalRead(GET_PTT_B_SW)) //if PTT_B is asserting PTT
680:     {
681:         if((tx2_computer && digitalRead(PTT_B_1_MAN)) || !digitalRead(PTT_B_2_MAN))
682:       {
683:              digitalWrite(PTT2, HIGH);
684:       }
685:          if ((!tx2_computer && digitalRead(PTT_B_2_MAN)) || !digitalRead(PTT_B_1_MAN))
686:           {
687:             digitalWrite(PTT1, HIGH);
688:           }
689:     }
690:     else  // if PTT_B not asserting PTT
691:     {
692:         if((tx2_computer && digitalRead(PTT_B_1_MAN)) || !digitalRead(PTT_B_2_MAN))
693:           {
694:             digitalWrite(PTT2, LOW);
695:           }
696:          if ((!tx2_computer && digitalRead(PTT_B_2_MAN)) || !digitalRead(PTT_B_1_MAN))
697:           {
698:             digitalWrite(PTT1, LOW);
699:           }
700:     }
701:
702: }
703:
704: static void do_relays()
705: //---------------------------------------------------------------------------
706: // Set the relays as appropriate
707: //---------------------------------------------------------------------------
708: {
709:     boolean tx2_current;
710:     boolean rx2_current;
711:     boolean stereo_current;
712:
713:     // The switches get priority.  Otherwise do what the computer says.
714:     if (mic_switch == RADIO_AUTO) {
715:         tx2_current = tx2_computer;
716:     }
717:     else {
718:         tx2_current = (mic_switch == RADIO_2);
719:     }
720:
721:     // RX is similar to TX except that the computer can say stereo.
722:     // The switches get priority.  Otherwise do what the computer says.
723:     if (rx_switch == RADIO_AUTO) {
724:         rx2_current = rx2_computer;
725:         stereo_current = stereo_computer && !mono;
726:     }
```

```
727:     else {
728:         rx2_current = (rx_switch == RADIO_2);
729:         stereo_current = false;
730:     }
731:
732:     if (mic2 != tx2_current) {
733:         // If the abort event is going to be sent do it before sending
734:         // the transmit changed event.  This is because it takes a few
735:         // milliseconds to send an event and the abort is more important.
736:         if (ptt_a_switch ||ptt_b_switch || ptt_computer || (!digitalRead(GET_CW_KEY)))
                 {
737:             // Transmitting while changing the transmitter.  Do what is possible
738:             // to stop this and send an abort if the event is turned on.
739:
740:         digitalWrite(CW1,LOW);
741:         digitalWrite(CW2,LOW);
742:         digitalWrite(PTT1,LOW);
743:         digitalWrite(PTT2,LOW);
744:             if (event_ab) {
745:                 uart_send_prog_string(PSTR("$AB\r"));
746:             }
747:         }
748:         if (event_tx) {
749:            uart_send_prog_string((tx2_current) ? PSTR("$TX2\r") : PSTR("$TX1\r"));
750:         }
751:     }
752:
753:     // Send a receive changed event if desired
754:     if (event_rx && ((rx2 != rx2_current) || (stereo != stereo_current))) {
755:         uart_send_prog_string((rx2_current) ? PSTR("$RX2") : PSTR("$RX1"));
756:         uart_send_prog_string((stereo_current) ? PSTR("S\r") : PSTR("\r"));
757:     }
758:
759:     mic2 = tx2_current;
760:     rx2 = rx2_current;
761:     stereo = stereo_current;
762:
763:     //below is commented out because
764:     //it is not necessary with N1MM
765:     //and in fact messes up the receiver
766:     //switching with dueling CQ
767:     /*
768:     // Do the latch after checking because it is vendor-specific and
769:     // does not cause an event.
770:     if ((rx_switch == RADIO_AUTO) && ptt_computer && latch) {
771:         rx2_current = !tx2_current;
772:         stereo_current = false;
773:     }
774:     */
775:
776:     // Now set the relays and the LEDs
777:
778:     if (tx2_current) {
779:         digitalWrite(MIC2,HIGH);
780:         digitalWrite(MIC1_LED,LOW);
781:     }
782:     else {
783:         digitalWrite(MIC1_LED,HIGH);
784:         digitalWrite(MIC2,LOW);
785:     }
786:
787:     //For stereo The RX2 relay must not be set.
788:     if (stereo_current) {
789:         digitalWrite(RX2,LOW);
790:         digitalWrite(STEREO,HIGH);
791:         digitalWrite(RX1_LED,HIGH);
```

```
792:         digitalWrite(RX2_LED,HIGH);
793:     }
794:     else {
795:         if (rx2_current) {
796:         digitalWrite(RX2,HIGH);
797:         digitalWrite(STEREO,LOW);
798:         digitalWrite(RX1_LED,LOW);
799:         digitalWrite(RX2_LED,HIGH);
800:         }
801:         else {
802:         digitalWrite(RX2,LOW);
803:         digitalWrite(STEREO,LOW);
804:         digitalWrite(RX1_LED,HIGH);
805:         digitalWrite(RX2_LED,LOW);
806:         }
807:     }
808: }
809:
810:
811: static void check_key_ptt()
812: //---------------------------------------------------------------------------
813: // Check the CW key and computer PTT
814: //---------------------------------------------------------------------------
815: {
816:     // If the key is closed the result is LOW
817:     if (digitalRead(GET_CW_KEY)) {
818:       if((tx2_computer && digitalRead(CW1_MAN)) || !digitalRead(CW2_MAN))
819:       {
820:         digitalWrite(CW2,LOW);
821:         cw2 = true;
822:       }
823:       if((!tx2_computer && digitalRead(CW2_MAN)) || !digitalRead(CW1_MAN))
824:       {
825:         digitalWrite(CW1,LOW);
826:         cw2 = false;
827:       }
828:
829:     }
830:     else {
831:       if((tx2_computer && digitalRead(CW1_MAN)) || !digitalRead(CW2_MAN))
832:       {
833:         digitalWrite(CW2,HIGH);
834:         cw2 = true;
835:       }
836:       if((!tx2_computer && digitalRead(CW2_MAN)) || !digitalRead(CW1_MAN))
837:       {
838:         digitalWrite(CW1,HIGH);
839:         cw2 = false;
840:       }
841:     }
842:     // If DTR is set the result is LOW
843:     if (digitalRead(GET_PTT_DTR)) {
844:         if (ptt_computer) {
845:             ptt_computer = false;
846:   //          do_ptt();
847:         }
848:     }
849:     else {
850:         if (!ptt_computer) {
851:             ptt_computer = true;
852:   //          do_ptt();
853:         }
854:     }
855:
856:     do_ptt();
857: }
```

```
858:
859: static void do_aux()
860: //------------------------------------------------------------------------
861: // Set the Auxiliary ports
862: // AUX1 is for Radio 1
863: // AUX2 is for Radio 2
864: //------------------------------------------------------------------------
865: {
866:     int data;
867:
868:     data = (int)aux2;
869:     switch(data)
870:     {
871:       case 0:
872:       {
873:         digitalWrite(AUX2_0, HIGH);
874:         digitalWrite(AUX2_1, LOW);
875:         digitalWrite(AUX2_2, LOW);
876:         digitalWrite(AUX2_3, LOW);
877:         digitalWrite(AUX2_4, LOW);
878:         digitalWrite(AUX2_5, LOW);
879:         digitalWrite(AUX2_6, LOW);
880:         digitalWrite(AUX2_7, LOW);
881:         digitalWrite(AUX2_8, LOW);
882:         digitalWrite(AUX2_9, LOW);
883:         digitalWrite(AUX2_10, LOW);
884:         digitalWrite(AUX2_11, LOW);
885:         digitalWrite(AUX2_12, LOW);
886:         digitalWrite(AUX2_13, LOW);
887:         digitalWrite(AUX2_14, LOW);
888:         digitalWrite(AUX2_15, LOW);
889:         break;
890:       }
891:       case 1:
892:       {
893:         digitalWrite(AUX2_0, LOW);
894:         digitalWrite(AUX2_1, HIGH);
895:         digitalWrite(AUX2_2, LOW);
896:         digitalWrite(AUX2_3, LOW);
897:         digitalWrite(AUX2_4, LOW);
898:         digitalWrite(AUX2_5, LOW);
899:         digitalWrite(AUX2_6, LOW);
900:         digitalWrite(AUX2_7, LOW);
901:         digitalWrite(AUX2_8, LOW);
902:         digitalWrite(AUX2_9, LOW);
903:         digitalWrite(AUX2_10, LOW);
904:         digitalWrite(AUX2_11, LOW);
905:         digitalWrite(AUX2_12, LOW);
906:         digitalWrite(AUX2_13, LOW);
907:         digitalWrite(AUX2_14, LOW);
908:         digitalWrite(AUX2_15, LOW);
909:         break;
910:       }
911:
912:       case 2:
913:       {
914:         digitalWrite(AUX2_0, LOW);
915:         digitalWrite(AUX2_1, LOW);
916:         digitalWrite(AUX2_2, HIGH);
917:         digitalWrite(AUX2_3, LOW);
918:         digitalWrite(AUX2_4, LOW);
919:         digitalWrite(AUX2_5, LOW);
920:         digitalWrite(AUX2_6, LOW);
921:         digitalWrite(AUX2_7, LOW);
922:         digitalWrite(AUX2_8, LOW);
923:         digitalWrite(AUX2_9, LOW);
```

```
924:          digitalWrite(AUX2_10, LOW);
925:          digitalWrite(AUX2_11, LOW);
926:          digitalWrite(AUX2_12, LOW);
927:          digitalWrite(AUX2_13, LOW);
928:          digitalWrite(AUX2_14, LOW);
929:          digitalWrite(AUX2_15, LOW);
930:          break;
931:        }
932:
933:        case 3:
934:        {
935:          digitalWrite(AUX2_0, LOW);
936:          digitalWrite(AUX2_1, LOW);
937:          digitalWrite(AUX2_2, LOW);
938:          digitalWrite(AUX2_3, HIGH);
939:          digitalWrite(AUX2_4, LOW);
940:          digitalWrite(AUX2_5, LOW);
941:          digitalWrite(AUX2_6, LOW);
942:          digitalWrite(AUX2_7, LOW);
943:          digitalWrite(AUX2_8, LOW);
944:          digitalWrite(AUX2_9, LOW);
945:          digitalWrite(AUX2_10, LOW);
946:          digitalWrite(AUX2_11, LOW);
947:          digitalWrite(AUX2_12, LOW);
948:          digitalWrite(AUX2_13, LOW);
949:          digitalWrite(AUX2_14, LOW);
950:          digitalWrite(AUX2_15, LOW);
951:          break;
952:        }
953:
954:        case 4:
955:        {
956:          digitalWrite(AUX2_0, LOW);
957:          digitalWrite(AUX2_1, LOW);
958:          digitalWrite(AUX2_2, LOW);
959:          digitalWrite(AUX2_3, LOW);
960:          digitalWrite(AUX2_4, HIGH);
961:          digitalWrite(AUX2_5, LOW);
962:          digitalWrite(AUX2_6, LOW);
963:          digitalWrite(AUX2_7, LOW);
964:          digitalWrite(AUX2_8, LOW);
965:          digitalWrite(AUX2_9, LOW);
966:          digitalWrite(AUX2_10, LOW);
967:          digitalWrite(AUX2_11, LOW);
968:          digitalWrite(AUX2_12, LOW);
969:          digitalWrite(AUX2_13, LOW);
970:          digitalWrite(AUX2_14, LOW);
971:          digitalWrite(AUX2_15, LOW);
972:          break;
973:        }
974:
975:        case 5:
976:        {
977:          digitalWrite(AUX2_0, LOW);
978:          digitalWrite(AUX2_1, LOW);
979:          digitalWrite(AUX2_2, LOW);
980:          digitalWrite(AUX2_3, LOW);
981:          digitalWrite(AUX2_4, LOW);
982:          digitalWrite(AUX2_5, HIGH);
983:          digitalWrite(AUX2_6, LOW);
984:          digitalWrite(AUX2_7, LOW);
985:          digitalWrite(AUX2_8, LOW);
986:          digitalWrite(AUX2_9, LOW);
987:          digitalWrite(AUX2_10, LOW);
988:          digitalWrite(AUX2_11, LOW);
989:          digitalWrite(AUX2_12, LOW);
```

```
 990:          digitalWrite(AUX2_13, LOW);
 991:          digitalWrite(AUX2_14, LOW);
 992:          digitalWrite(AUX2_15, LOW);
 993:          break;
 994:        }
 995:
 996:        case 6:
 997:        {
 998:          digitalWrite(AUX2_0, LOW);
 999:          digitalWrite(AUX2_1, LOW);
1000:          digitalWrite(AUX2_2, LOW);
1001:          digitalWrite(AUX2_3, LOW);
1002:          digitalWrite(AUX2_4, LOW);
1003:          digitalWrite(AUX2_5, LOW);
1004:          digitalWrite(AUX2_6, HIGH);
1005:          digitalWrite(AUX2_7, LOW);
1006:          digitalWrite(AUX2_8, LOW);
1007:          digitalWrite(AUX2_9, LOW);
1008:          digitalWrite(AUX2_10, LOW);
1009:          digitalWrite(AUX2_11, LOW);
1010:          digitalWrite(AUX2_12, LOW);
1011:          digitalWrite(AUX2_13, LOW);
1012:          digitalWrite(AUX2_14, LOW);
1013:          digitalWrite(AUX2_15, LOW);
1014:          break;
1015:        }
1016:
1017:        case 7:
1018:        {
1019:          digitalWrite(AUX2_0, LOW);
1020:          digitalWrite(AUX2_1, LOW);
1021:          digitalWrite(AUX2_2, LOW);
1022:          digitalWrite(AUX2_3, LOW);
1023:          digitalWrite(AUX2_4, LOW);
1024:          digitalWrite(AUX2_5, LOW);
1025:          digitalWrite(AUX2_6, LOW);
1026:          digitalWrite(AUX2_7, HIGH);
1027:          digitalWrite(AUX2_8, LOW);
1028:          digitalWrite(AUX2_9, LOW);
1029:          digitalWrite(AUX2_10, LOW);
1030:          digitalWrite(AUX2_11, LOW);
1031:          digitalWrite(AUX2_12, LOW);
1032:          digitalWrite(AUX2_13, LOW);
1033:          digitalWrite(AUX2_14, LOW);
1034:          digitalWrite(AUX2_15, LOW);
1035:          break;
1036:        }
1037:
1038:        case 8:
1039:        {
1040:          digitalWrite(AUX2_0, LOW);
1041:          digitalWrite(AUX2_1, LOW);
1042:          digitalWrite(AUX2_2, LOW);
1043:          digitalWrite(AUX2_3, LOW);
1044:          digitalWrite(AUX2_4, LOW);
1045:          digitalWrite(AUX2_5, LOW);
1046:          digitalWrite(AUX2_6, LOW);
1047:          digitalWrite(AUX2_7, LOW);
1048:          digitalWrite(AUX2_8, HIGH);
1049:          digitalWrite(AUX2_9, LOW);
1050:          digitalWrite(AUX2_10, LOW);
1051:          digitalWrite(AUX2_11, LOW);
1052:          digitalWrite(AUX2_12, LOW);
1053:          digitalWrite(AUX2_13, LOW);
1054:          digitalWrite(AUX2_14, LOW);
1055:          digitalWrite(AUX2_15, LOW);
```

```
1056:         break;
1057:       }
1058:
1059:       case 9:
1060:       {
1061:         digitalWrite(AUX2_0, LOW);
1062:         digitalWrite(AUX2_1, LOW);
1063:         digitalWrite(AUX2_2, LOW);
1064:         digitalWrite(AUX2_3, LOW);
1065:         digitalWrite(AUX2_4, LOW);
1066:         digitalWrite(AUX2_5, LOW);
1067:         digitalWrite(AUX2_6, LOW);
1068:         digitalWrite(AUX2_7, LOW);
1069:         digitalWrite(AUX2_8, LOW);
1070:         digitalWrite(AUX2_9, HIGH);
1071:         digitalWrite(AUX2_10, LOW);
1072:         digitalWrite(AUX2_11, LOW);
1073:         digitalWrite(AUX2_12, LOW);
1074:         digitalWrite(AUX2_13, LOW);
1075:         digitalWrite(AUX2_14, LOW);
1076:         digitalWrite(AUX2_15, LOW);
1077:         break;
1078:       }
1079:
1080:       case 10:
1081:       {
1082:         digitalWrite(AUX2_0, LOW);
1083:         digitalWrite(AUX2_1, LOW);
1084:         digitalWrite(AUX2_2, LOW);
1085:         digitalWrite(AUX2_3, LOW);
1086:         digitalWrite(AUX2_4, LOW);
1087:         digitalWrite(AUX2_5, LOW);
1088:         digitalWrite(AUX2_6, LOW);
1089:         digitalWrite(AUX2_7, LOW);
1090:         digitalWrite(AUX2_8, LOW);
1091:         digitalWrite(AUX2_9, LOW);
1092:         digitalWrite(AUX2_10, HIGH);
1093:         digitalWrite(AUX2_11, LOW);
1094:         digitalWrite(AUX2_12, LOW);
1095:         digitalWrite(AUX2_13, LOW);
1096:         digitalWrite(AUX2_14, LOW);
1097:         digitalWrite(AUX2_15, LOW);
1098:         break;
1099:       }
1100:
1101:       case 11:
1102:       {
1103:         digitalWrite(AUX2_0, LOW);
1104:         digitalWrite(AUX2_1, LOW);
1105:         digitalWrite(AUX2_2, LOW);
1106:         digitalWrite(AUX2_3, LOW);
1107:         digitalWrite(AUX2_4, LOW);
1108:         digitalWrite(AUX2_5, LOW);
1109:         digitalWrite(AUX2_6, LOW);
1110:         digitalWrite(AUX2_7, LOW);
1111:         digitalWrite(AUX2_8, LOW);
1112:         digitalWrite(AUX2_9, LOW);
1113:         digitalWrite(AUX2_10, LOW);
1114:         digitalWrite(AUX2_11, HIGH);
1115:         digitalWrite(AUX2_12, LOW);
1116:         digitalWrite(AUX2_13, LOW);
1117:         digitalWrite(AUX2_14, LOW);
1118:         digitalWrite(AUX2_15, LOW);
1119:         break;
1120:       }
1121:
```

```
1122:       case 12:
1123:       {
1124:         digitalWrite(AUX2_0, LOW);
1125:         digitalWrite(AUX2_1, LOW);
1126:         digitalWrite(AUX2_2, LOW);
1127:         digitalWrite(AUX2_3, LOW);
1128:         digitalWrite(AUX2_4, LOW);
1129:         digitalWrite(AUX2_5, LOW);
1130:         digitalWrite(AUX2_6, LOW);
1131:         digitalWrite(AUX2_7, LOW);
1132:         digitalWrite(AUX2_8, LOW);
1133:         digitalWrite(AUX2_9, LOW);
1134:         digitalWrite(AUX2_10, LOW);
1135:         digitalWrite(AUX2_11, LOW);
1136:         digitalWrite(AUX2_12, HIGH);
1137:         digitalWrite(AUX2_13, LOW);
1138:         digitalWrite(AUX2_14, LOW);
1139:         digitalWrite(AUX2_15, LOW);
1140:         break;
1141:       }
1142:
1143:       case 13:
1144:       {
1145:         digitalWrite(AUX2_0, LOW);
1146:         digitalWrite(AUX2_1, LOW);
1147:         digitalWrite(AUX2_2, LOW);
1148:         digitalWrite(AUX2_3, LOW);
1149:         digitalWrite(AUX2_4, LOW);
1150:         digitalWrite(AUX2_5, LOW);
1151:         digitalWrite(AUX2_6, LOW);
1152:         digitalWrite(AUX2_7, LOW);
1153:         digitalWrite(AUX2_8, LOW);
1154:         digitalWrite(AUX2_9, LOW);
1155:         digitalWrite(AUX2_10, LOW);
1156:         digitalWrite(AUX2_11, LOW);
1157:         digitalWrite(AUX2_12, LOW);
1158:         digitalWrite(AUX2_13, HIGH);
1159:         digitalWrite(AUX2_14, LOW);
1160:         digitalWrite(AUX2_15, LOW);
1161:         break;
1162:       }
1163:
1164:       case 14:
1165:       {
1166:         digitalWrite(AUX2_0, LOW);
1167:         digitalWrite(AUX2_1, LOW);
1168:         digitalWrite(AUX2_2, LOW);
1169:         digitalWrite(AUX2_3, LOW);
1170:         digitalWrite(AUX2_4, LOW);
1171:         digitalWrite(AUX2_5, LOW);
1172:         digitalWrite(AUX2_6, LOW);
1173:         digitalWrite(AUX2_7, LOW);
1174:         digitalWrite(AUX2_8, LOW);
1175:         digitalWrite(AUX2_9, LOW);
1176:         digitalWrite(AUX2_10, LOW);
1177:         digitalWrite(AUX2_11, LOW);
1178:         digitalWrite(AUX2_12, LOW);
1179:         digitalWrite(AUX2_13, LOW);
1180:         digitalWrite(AUX2_14, HIGH);
1181:         digitalWrite(AUX2_15, LOW);
1182:         break;
1183:       }
1184:
1185:       case 15:
1186:       {
1187:         digitalWrite(AUX2_0, LOW);
```

```
1188:         digitalWrite(AUX2_1, LOW);
1189:         digitalWrite(AUX2_2, LOW);
1190:         digitalWrite(AUX2_3, LOW);
1191:         digitalWrite(AUX2_4, LOW);
1192:         digitalWrite(AUX2_5, LOW);
1193:         digitalWrite(AUX2_6, LOW);
1194:         digitalWrite(AUX2_7, LOW);
1195:         digitalWrite(AUX2_8, LOW);
1196:         digitalWrite(AUX2_9, LOW);
1197:         digitalWrite(AUX2_10, LOW);
1198:         digitalWrite(AUX2_11, LOW);
1199:         digitalWrite(AUX2_12, LOW);
1200:         digitalWrite(AUX2_13, LOW);
1201:         digitalWrite(AUX2_14, LOW);
1202:         digitalWrite(AUX2_15, HIGH);
1203:         break;
1204:       }
1205:     }
1206:
1207:
1208:     data = (int)aux1;
1209:     switch(data)
1210:     {
1211:       case 0:
1212:       {
1213:         digitalWrite(AUX1_0, HIGH);
1214:         digitalWrite(AUX1_1, LOW);
1215:         digitalWrite(AUX1_2, LOW);
1216:         digitalWrite(AUX1_3, LOW);
1217:         digitalWrite(AUX1_4, LOW);
1218:         digitalWrite(AUX1_5, LOW);
1219:         digitalWrite(AUX1_6, LOW);
1220:         digitalWrite(AUX1_7, LOW);
1221:         digitalWrite(AUX1_8, LOW);
1222:         digitalWrite(AUX1_9, LOW);
1223:         digitalWrite(AUX1_10, LOW);
1224:         digitalWrite(AUX1_11, LOW);
1225:         digitalWrite(AUX1_12, LOW);
1226:         digitalWrite(AUX1_13, LOW);
1227:         digitalWrite(AUX1_14, LOW);
1228:         digitalWrite(AUX1_15, LOW);
1229:         break;
1230:       }
1231:       case 1:
1232:       {
1233:         digitalWrite(AUX1_0, LOW);
1234:         digitalWrite(AUX1_1, HIGH);
1235:         digitalWrite(AUX1_2, LOW);
1236:         digitalWrite(AUX1_3, LOW);
1237:         digitalWrite(AUX1_4, LOW);
1238:         digitalWrite(AUX1_5, LOW);
1239:         digitalWrite(AUX1_6, LOW);
1240:         digitalWrite(AUX1_7, LOW);
1241:         digitalWrite(AUX1_8, LOW);
1242:         digitalWrite(AUX1_9, LOW);
1243:         digitalWrite(AUX1_10, LOW);
1244:         digitalWrite(AUX1_11, LOW);
1245:         digitalWrite(AUX1_12, LOW);
1246:         digitalWrite(AUX1_13, LOW);
1247:         digitalWrite(AUX1_14, LOW);
1248:         digitalWrite(AUX1_15, LOW);
1249:         break;
1250:       }
1251:
1252:       case 2:
1253:       {
```

```
1254:         digitalWrite(AUX1_0, LOW);
1255:         digitalWrite(AUX1_1, LOW);
1256:         digitalWrite(AUX1_2, HIGH);
1257:         digitalWrite(AUX1_3, LOW);
1258:         digitalWrite(AUX1_4, LOW);
1259:         digitalWrite(AUX1_5, LOW);
1260:         digitalWrite(AUX1_6, LOW);
1261:         digitalWrite(AUX1_7, LOW);
1262:         digitalWrite(AUX1_8, LOW);
1263:         digitalWrite(AUX1_9, LOW);
1264:         digitalWrite(AUX1_10, LOW);
1265:         digitalWrite(AUX1_11, LOW);
1266:         digitalWrite(AUX1_12, LOW);
1267:         digitalWrite(AUX1_13, LOW);
1268:         digitalWrite(AUX1_14, LOW);
1269:         digitalWrite(AUX1_15, LOW);
1270:         break;
1271:       }
1272:
1273:       case 3:
1274:       {
1275:         digitalWrite(AUX1_0, LOW);
1276:         digitalWrite(AUX1_1, LOW);
1277:         digitalWrite(AUX1_2, LOW);
1278:         digitalWrite(AUX1_3, HIGH);
1279:         digitalWrite(AUX1_4, LOW);
1280:         digitalWrite(AUX1_5, LOW);
1281:         digitalWrite(AUX1_6, LOW);
1282:         digitalWrite(AUX1_7, LOW);
1283:         digitalWrite(AUX1_8, LOW);
1284:         digitalWrite(AUX1_9, LOW);
1285:         digitalWrite(AUX1_10, LOW);
1286:         digitalWrite(AUX1_11, LOW);
1287:         digitalWrite(AUX1_12, LOW);
1288:         digitalWrite(AUX1_13, LOW);
1289:         digitalWrite(AUX1_14, LOW);
1290:         digitalWrite(AUX1_15, LOW);
1291:         break;
1292:       }
1293:
1294:       case 4:
1295:       {
1296:         digitalWrite(AUX1_0, LOW);
1297:         digitalWrite(AUX1_1, LOW);
1298:         digitalWrite(AUX1_2, LOW);
1299:         digitalWrite(AUX1_3, LOW);
1300:         digitalWrite(AUX1_4, HIGH);
1301:         digitalWrite(AUX1_5, LOW);
1302:         digitalWrite(AUX1_6, LOW);
1303:         digitalWrite(AUX1_7, LOW);
1304:         digitalWrite(AUX1_8, LOW);
1305:         digitalWrite(AUX1_9, LOW);
1306:         digitalWrite(AUX1_10, LOW);
1307:         digitalWrite(AUX1_11, LOW);
1308:         digitalWrite(AUX1_12, LOW);
1309:         digitalWrite(AUX1_13, LOW);
1310:         digitalWrite(AUX1_14, LOW);
1311:         digitalWrite(AUX1_15, LOW);
1312:         break;
1313:       }
1314:
1315:       case 5:
1316:       {
1317:         digitalWrite(AUX1_0, LOW);
1318:         digitalWrite(AUX1_1, LOW);
1319:         digitalWrite(AUX1_2, LOW);
```

```
1320:           digitalWrite(AUX1_3, LOW);
1321:           digitalWrite(AUX1_4, LOW);
1322:           digitalWrite(AUX1_5, HIGH);
1323:           digitalWrite(AUX1_6, LOW);
1324:           digitalWrite(AUX1_7, LOW);
1325:           digitalWrite(AUX1_8, LOW);
1326:           digitalWrite(AUX1_9, LOW);
1327:           digitalWrite(AUX1_10, LOW);
1328:           digitalWrite(AUX1_11, LOW);
1329:           digitalWrite(AUX1_12, LOW);
1330:           digitalWrite(AUX1_13, LOW);
1331:           digitalWrite(AUX1_14, LOW);
1332:           digitalWrite(AUX1_15, LOW);
1333:           break;
1334:         }
1335:
1336:         case 6:
1337:         {
1338:           digitalWrite(AUX1_0, LOW);
1339:           digitalWrite(AUX1_1, LOW);
1340:           digitalWrite(AUX1_2, LOW);
1341:           digitalWrite(AUX1_3, LOW);
1342:           digitalWrite(AUX1_4, LOW);
1343:           digitalWrite(AUX1_5, LOW);
1344:           digitalWrite(AUX1_6, HIGH);
1345:           digitalWrite(AUX1_7, LOW);
1346:           digitalWrite(AUX1_8, LOW);
1347:           digitalWrite(AUX1_9, LOW);
1348:           digitalWrite(AUX1_10, LOW);
1349:           digitalWrite(AUX1_11, LOW);
1350:           digitalWrite(AUX1_12, LOW);
1351:           digitalWrite(AUX1_13, LOW);
1352:           digitalWrite(AUX1_14, LOW);
1353:           digitalWrite(AUX1_15, LOW);
1354:           break;
1355:         }
1356:
1357:         case 7:
1358:         {
1359:           digitalWrite(AUX1_0, LOW);
1360:           digitalWrite(AUX1_1, LOW);
1361:           digitalWrite(AUX1_2, LOW);
1362:           digitalWrite(AUX1_3, LOW);
1363:           digitalWrite(AUX1_4, LOW);
1364:           digitalWrite(AUX1_5, LOW);
1365:           digitalWrite(AUX1_6, LOW);
1366:           digitalWrite(AUX1_7, HIGH);
1367:           digitalWrite(AUX1_8, LOW);
1368:           digitalWrite(AUX1_9, LOW);
1369:           digitalWrite(AUX1_10, LOW);
1370:           digitalWrite(AUX1_11, LOW);
1371:           digitalWrite(AUX1_12, LOW);
1372:           digitalWrite(AUX1_13, LOW);
1373:           digitalWrite(AUX1_14, LOW);
1374:           digitalWrite(AUX1_15, LOW);
1375:           break;
1376:         }
1377:
1378:         case 8:
1379:         {
1380:           digitalWrite(AUX1_0, LOW);
1381:           digitalWrite(AUX1_1, LOW);
1382:           digitalWrite(AUX1_2, LOW);
1383:           digitalWrite(AUX1_3, LOW);
1384:           digitalWrite(AUX1_4, LOW);
1385:           digitalWrite(AUX1_5, LOW);
```

```
1386:         digitalWrite(AUX1_6, LOW);
1387:         digitalWrite(AUX1_7, LOW);
1388:         digitalWrite(AUX1_8, HIGH);
1389:         digitalWrite(AUX1_9, LOW);
1390:         digitalWrite(AUX1_10, LOW);
1391:         digitalWrite(AUX1_11, LOW);
1392:         digitalWrite(AUX1_12, LOW);
1393:         digitalWrite(AUX1_13, LOW);
1394:         digitalWrite(AUX1_14, LOW);
1395:         digitalWrite(AUX1_15, LOW);
1396:         break;
1397:       }
1398:
1399:       case 9:
1400:       {
1401:         digitalWrite(AUX1_0, LOW);
1402:         digitalWrite(AUX1_1, LOW);
1403:         digitalWrite(AUX1_2, LOW);
1404:         digitalWrite(AUX1_3, LOW);
1405:         digitalWrite(AUX1_4, LOW);
1406:         digitalWrite(AUX1_5, LOW);
1407:         digitalWrite(AUX1_6, LOW);
1408:         digitalWrite(AUX1_7, LOW);
1409:         digitalWrite(AUX1_8, LOW);
1410:         digitalWrite(AUX1_9, HIGH);
1411:         digitalWrite(AUX1_10, LOW);
1412:         digitalWrite(AUX1_11, LOW);
1413:         digitalWrite(AUX1_12, LOW);
1414:         digitalWrite(AUX1_13, LOW);
1415:         digitalWrite(AUX1_14, LOW);
1416:         digitalWrite(AUX1_15, LOW);
1417:         break;
1418:       }
1419:
1420:       case 10:
1421:       {
1422:         digitalWrite(AUX1_0, LOW);
1423:         digitalWrite(AUX1_1, LOW);
1424:         digitalWrite(AUX1_2, LOW);
1425:         digitalWrite(AUX1_3, LOW);
1426:         digitalWrite(AUX1_4, LOW);
1427:         digitalWrite(AUX1_5, LOW);
1428:         digitalWrite(AUX1_6, LOW);
1429:         digitalWrite(AUX1_7, LOW);
1430:         digitalWrite(AUX1_8, LOW);
1431:         digitalWrite(AUX1_9, LOW);
1432:         digitalWrite(AUX1_10, HIGH);
1433:         digitalWrite(AUX1_11, LOW);
1434:         digitalWrite(AUX1_12, LOW);
1435:         digitalWrite(AUX1_13, LOW);
1436:         digitalWrite(AUX1_14, LOW);
1437:         digitalWrite(AUX1_15, LOW);
1438:         break;
1439:       }
1440:
1441:       case 11:
1442:       {
1443:         digitalWrite(AUX1_0, LOW);
1444:         digitalWrite(AUX1_1, LOW);
1445:         digitalWrite(AUX1_2, LOW);
1446:         digitalWrite(AUX1_3, LOW);
1447:         digitalWrite(AUX1_4, LOW);
1448:         digitalWrite(AUX1_5, LOW);
1449:         digitalWrite(AUX1_6, LOW);
1450:         digitalWrite(AUX1_7, LOW);
1451:         digitalWrite(AUX1_8, LOW);
```

```
1452:         digitalWrite(AUX1_9, LOW);
1453:         digitalWrite(AUX1_10, LOW);
1454:         digitalWrite(AUX1_11, HIGH);
1455:         digitalWrite(AUX1_12, LOW);
1456:         digitalWrite(AUX1_13, LOW);
1457:         digitalWrite(AUX1_14, LOW);
1458:         digitalWrite(AUX1_15, LOW);
1459:         break;
1460:       }
1461:
1462:       case 12:
1463:       {
1464:         digitalWrite(AUX1_0, LOW);
1465:         digitalWrite(AUX1_1, LOW);
1466:         digitalWrite(AUX1_2, LOW);
1467:         digitalWrite(AUX1_3, LOW);
1468:         digitalWrite(AUX1_4, LOW);
1469:         digitalWrite(AUX1_5, LOW);
1470:         digitalWrite(AUX1_6, LOW);
1471:         digitalWrite(AUX1_7, LOW);
1472:         digitalWrite(AUX1_8, LOW);
1473:         digitalWrite(AUX1_9, LOW);
1474:         digitalWrite(AUX1_10, LOW);
1475:         digitalWrite(AUX1_11, LOW);
1476:         digitalWrite(AUX1_12, HIGH);
1477:         digitalWrite(AUX1_13, LOW);
1478:         digitalWrite(AUX1_14, LOW);
1479:         digitalWrite(AUX1_15, LOW);
1480:         break;
1481:       }
1482:
1483:       case 13:
1484:       {
1485:         digitalWrite(AUX1_0, LOW);
1486:         digitalWrite(AUX1_1, LOW);
1487:         digitalWrite(AUX1_2, LOW);
1488:         digitalWrite(AUX1_3, LOW);
1489:         digitalWrite(AUX1_4, LOW);
1490:         digitalWrite(AUX1_5, LOW);
1491:         digitalWrite(AUX1_6, LOW);
1492:         digitalWrite(AUX1_7, LOW);
1493:         digitalWrite(AUX1_8, LOW);
1494:         digitalWrite(AUX1_9, LOW);
1495:         digitalWrite(AUX1_10, LOW);
1496:         digitalWrite(AUX1_11, LOW);
1497:         digitalWrite(AUX1_12, LOW);
1498:         digitalWrite(AUX1_13, HIGH);
1499:         digitalWrite(AUX1_14, LOW);
1500:         digitalWrite(AUX1_15, LOW);
1501:         break;
1502:       }
1503:
1504:       case 14:
1505:       {
1506:         digitalWrite(AUX1_0, LOW);
1507:         digitalWrite(AUX1_1, LOW);
1508:         digitalWrite(AUX1_2, LOW);
1509:         digitalWrite(AUX1_3, LOW);
1510:         digitalWrite(AUX1_4, LOW);
1511:         digitalWrite(AUX1_5, LOW);
1512:         digitalWrite(AUX1_6, LOW);
1513:         digitalWrite(AUX1_7, LOW);
1514:         digitalWrite(AUX1_8, LOW);
1515:         digitalWrite(AUX1_9, LOW);
1516:         digitalWrite(AUX1_10, LOW);
1517:         digitalWrite(AUX1_11, LOW);
```

```
1518:         digitalWrite(AUX1_12, LOW);
1519:         digitalWrite(AUX1_13, LOW);
1520:         digitalWrite(AUX1_14, HIGH);
1521:         digitalWrite(AUX1_15, LOW);
1522:         break;
1523:       }
1524:
1525:       case 15:
1526:       {
1527:         digitalWrite(AUX1_0, LOW);
1528:         digitalWrite(AUX1_1, LOW);
1529:         digitalWrite(AUX1_2, LOW);
1530:         digitalWrite(AUX1_3, LOW);
1531:         digitalWrite(AUX1_4, LOW);
1532:         digitalWrite(AUX1_5, LOW);
1533:         digitalWrite(AUX1_6, LOW);
1534:         digitalWrite(AUX1_7, LOW);
1535:         digitalWrite(AUX1_8, LOW);
1536:         digitalWrite(AUX1_9, LOW);
1537:         digitalWrite(AUX1_10, LOW);
1538:         digitalWrite(AUX1_11, LOW);
1539:         digitalWrite(AUX1_12, LOW);
1540:         digitalWrite(AUX1_13, LOW);
1541:         digitalWrite(AUX1_14, LOW);
1542:         digitalWrite(AUX1_15, HIGH);
1543:         break;
1544:       }
1545:     }
1546: }
1547:
1548:
1549: static void do_command()
1550: //------------------------------------------------------------------------
1551: // Parse and handle a command from the computer
1552: //------------------------------------------------------------------------
1553: {
1554:     // Commands are not checked very thoroughly - the computer
1555:     // should not send garbage.
1556:
1557: #define COMPARE(command)  (memcmp_P(in_buf, PSTR(command), \
1558:                                     sizeof(command)-1) == 0)
1559:
1560: #define QCOMPARE(command) (memcmp_P(in_buf+1, PSTR(command), \
1561:                                     sizeof(command)-1) == 0)
1562:
1563: #define AFTER(command) (sizeof(command)-1)
1564:
1565: // Parse the commands
1566:
1567:     // Handle the queries
1568:     if (in_buf[0] == '?') {
1569:
1570:         // Check for 'ping' - just the ? by itself
1571:         if (in_buf[AFTER("?")] == '\0') {
1572:             uart_send_prog_string(PSTR("?\r"));
1573:             return;
1574:         }
1575:
1576:         // Return TX1 or TX2
1577:         if (QCOMPARE("TX")) {
1578:             uart_send_prog_string((mic2) ? PSTR("TX2\r") : PSTR("TX1\r"));
1579:            return;
1580:         }
1581:
1582:         // Return RX1 or RX2 or RX1S or RX2S
1583:         if (QCOMPARE("RX")) {
```

```
1584:             uart_send_prog_string((rx2) ? PSTR("RX2") : PSTR("RX1"));
1585:             uart_send_prog_string((stereo) ? PSTR("S\r") : PSTR("\r"));
1586:             return;
1587:         }
1588:
1589:         // Return AUX1 value
1590:         if (QCOMPARE("AUX1")) {
1591:             uart_send_prog_string(PSTR("AUX1"));
1592:             if (aux1 >= 10) {
1593:                 uart_send_char('1');
1594:                 uart_send_char((aux1-10) + '0');
1595:             }
1596:             else {
1597:                 uart_send_char(aux1 + '0');
1598:             }
1599:             uart_send_char('\r');
1600:             return;
1601:         }
1602:
1603:         // Return AUX2 value
1604:         if (QCOMPARE("AUX2")) {
1605:             uart_send_prog_string(PSTR("AUX2"));
1606:             if (aux2 >= 10) {
1607:                 uart_send_char('1');
1608:                 uart_send_char((aux2-10) + '0');
1609:             }
1610:             else {
1611:                 uart_send_char(aux2 + '0');
1612:                 uart_send_char('\r');
1613:              }
1614:             return;
1615:         }
1616:
1617:         // Return the SO2R device's name
1618:         if (QCOMPARE("NAME")) {
1619:             uart_send_prog_string(PSTR("NAME_SUPER_MEGA_SO2Rduino\r"));
1620:             return;
1621:         }
1622:
1623:         // Return the state of the footswitch
1624:         if (QCOMPARE("CR0")) {
1625:             uart_send_prog_string((ptt_a_switch || ptt_b_switch)
1626:                 ? PSTR("CR01\r") : PSTR("CR00\r"));
1627:             return;
1628:         }
1629:
1630:         // Return whether footswitch events are enabled
1631:         if (QCOMPARE("ECR0")) {
1632:             uart_send_prog_string((event_cr0)
1633:                 ? PSTR("ECR01\r") : PSTR("ECR00\r"));
1634:             return;
1635:         }
1636:
1637:         // Return whether receiver events are enabled
1638:         if (COMPARE("ERX")) {
1639:             uart_send_prog_string((event_rx)
1640:                 ? PSTR("ERX1\r") : PSTR("ERX0\r"));
1641:             return;
1642:         }
1643:
1644:         // Return whether transmitter events are enabled
1645:         if (COMPARE("ETX")) {
1646:             uart_send_prog_string((event_tx)
1647:                 ? PSTR("ETX1\r") : PSTR("ETX0\r"));
1648:             return;
1649:         }
```

```
1650:
1651:         // Return whether mono is forced (no stereo allowed)
1652:         if (COMPARE("VMONO")) {
1653:             uart_send_prog_string((mono)
1654:                 ? PSTR("VMONO1\r") : PSTR("VMONO0\r"));
1655:             return;
1656:             }
1657:
1658:         // Return whether latch feature is on
1659:         if (COMPARE("VLATCH")) {
1660:             uart_send_prog_string((latch)
1661:                 ? PSTR("VLATCH1\r") : PSTR("VLATCH0\r"));
1662:             return;
1663:         }
1664:
1665:         // Unknown query command
1666:         uart_send_string(in_buf);
1667:         uart_send_char('\r');
1668:         return;
1669:     }
1670:
1671:     // Handle transmitter change
1672:     if (COMPARE("TX")) {
1673:         tx2_computer = (in_buf[AFTER("TX")] == '2');
1674:         do_relays();
1675:        return;
1676:     }
1677:
1678:     // Handle receiver change
1679:     if (COMPARE("RX")) {
1680:         rx2_computer = (in_buf[AFTER("TX")] == '2');
1681:         stereo_computer = ((in_buf[AFTER("TXn")] == 'S') ||
1682:                   (in_buf[AFTER("TXn")] == 'R'));
1683:         do_relays();
1684:         return;
1685:     }
1686:
1687:     // Handle aux output 1 - values are four bits
1688:     if (COMPARE("AUX1")) {
1689:         aux1 = atoi((char *)&in_buf[AFTER("AUXn")]) & 15;
1690:         do_aux();
1691:         return;
1692:     }
1693:
1694:     // Handle aux output 2 - values are four bits
1695:     if (COMPARE("AUX2")) {
1696:         aux2 = atoi((char *)&in_buf[AFTER("AUXn")]) & 15;
1697:         do_aux();
1698:         return;
1699:     }
1700:
1701:     // Turn on or off PTT footswitch events
1702:     if (COMPARE("ECR0")) {
1703:       event_cr0 = (in_buf[AFTER("ECRn")] == '1');
1704:         if (event_cr0) {
1705:             uart_send_prog_string((ptt_a_switch || ptt_b_switch)
1706:                 ? PSTR("$CR01\r") : PSTR("$CR00\r"));
1707:         }
1708:         return;
1709:     }
1710:
1711:     // Turn on or off receiver events
1712:     if (COMPARE("ERX")) {
1713:         event_rx = (in_buf[AFTER("ERX")] == '1');
1714:         if (event_rx) {
1715:             uart_send_prog_string((rx2) ? PSTR("$RX2") : PSTR("$RX1"));
```

```
1716:             uart_send_prog_string((stereo) ? PSTR("S\r") : PSTR("\r"));
1717:         }
1718:         return;
1719:     }
1720:
1721:     // Turn on or off transmitter events
1722:     if (COMPARE("ETX")) {
1723:         event_tx = (in_buf[AFTER("ETX")] == '1');
1724:         if (event_tx) {
1725:              uart_send_prog_string((mic2) ? PSTR("$TX2\r") : PSTR("$TX1\r"));
1726:         }
1727:         return;
1728:     }
1729:
1730:     // Turn on or off forced mono feature
1731:     if (COMPARE("VMONO")) {
1732:         boolean n;
1733:         n = (in_buf[AFTER("VMONO")] == '1');
1734:         if (n != mono) {
1735:             mono = n;
1736:             EEPROM.write(EEPROM_MONO, mono);
1737:             do_relays();
1738:         }
1739:         return;
1740:     }
1741:
1742:     // Turn on or off latch feature
1743:     if (COMPARE("VLATCH")) {
1744:         boolean n;
1745:         n = (in_buf[AFTER("VLATCH")] == '1');
1746:         if (n != latch) {
1747:             latch = n;
1748:             EEPROM.write(EEPROM_LATCH, latch);
1749:             do_relays();
1750:         }
1751:         return;
1752:     }
1753:
1754:     // Unknown commands are ignored, as are commands that try to
1755:     // set something which is read-only such as NAME or CR0.
1756:     return;
1757: }
```

```
  1: // SO2Rduino - An SO2R Box built on an Arduino clone
  2: //
  3: // Copyright 2010, Paul Young
  4: //
  5: // This file contains the UART routines
  6:
  7: #include "Arduino.h"
  8: #include "avr/pgmspace.h"
  9: #include "uart.h"
 10:
 11: char                    in_buf[UBLEN];  // UART input buffer
 12: static byte             in_len;         // Number of chars in buffer
 13: static char             out_buf[UBLEN]; // UART output circular buffer
 14: static byte             out_add;        // Place to add a character
 15: static byte             out_remove;     // Place to remove a character
 16:
 17: void
 18: uart_send_char(const char c)
 19: //--------------------------------------------------------------------------
 20: // Add a character to the UART output buffer
 21: //--------------------------------------------------------------------------
 22: {
 23:
 24:     // Check for space in the buffer
 25:     if ((out_remove == (out_add + 1)) ||
 26:         ((out_remove == 0) && out_add == (UBLEN-1))) {
 27:             return;
 28:     }
 29:
 30:     out_buf[out_add++] = c;
 31:     if (out_add == UBLEN) {
 32:         out_add = 0;
 33:     }
 34: }
 35:
 36: void
 37: uart_send_string(const char* bp)
 38: //--------------------------------------------------------------------------
 39: // Add a character string to the UART output buffer
 40: //--------------------------------------------------------------------------
 41: {
 42:     while (*bp) {
 43:         uart_send_char(*bp++);
 44:     }
 45: }
 46:
 47: void
 48: uart_send_prog_string(const char * bp)
 49: //--------------------------------------------------------------------------
 50: // Add a character string from flash to the UART output buffer
 51: //--------------------------------------------------------------------------
 52: {
 53:     byte c;
 54:     while ((c=pgm_read_byte(bp++))) {
 55:         uart_send_char(c);
 56:     }
 57: }
 58:
 59: void
 60: uart_init()
 61: //--------------------------------------------------------------------------
 62: // Set up the UART
 63: //--------------------------------------------------------------------------
 64: {
 65:
 66:     in_len = 0;
```

```
// SO2Rduino - An SO2R Box built on an Arduino clone
//
// Copyright 2010, Paul Young
//
// This file contains the UART prototypes
//

#ifndef UART_H

// The uart file is C, not C++ so this is needed to cause the
// compiler to generate the correct routine names.
#ifdef __cplusplus
extern "C"{
#endif

// The input buffer is used in command parsing.
#define UBLEN 64                        // UART Buffer length
extern char in_buf[UBLEN];  // UART input buffer

extern void uart_send_char(const char c);
extern void uart_send_string(const char* bp);
extern void uart_send_prog_string(const char * bp);
extern void uart_init(void);
extern void uart_clear_buffer(void);
extern boolean do_uart(void);

#ifdef __cplusplus
}
#endif

#endif
```

```
 67:     out_add = 0;
 68:     out_remove = 0;
 69:
 70:     // Set the UART to 9600 baud
 71: #define MYUBRR 103
 72:
 73:     UBRR0H = (MYUBRR>>8);
 74:     UBRR0L = (unsigned char)MYUBRR;
 75:
 76:     // Enable receiver and transmitter
 77:     UCSR0B = _BV(RXEN0)| _BV(TXEN0);
 78:
 79:     // Set 8 bit, one stop bit
 80:     UCSR0C = _BV(UCSZ00) | _BV(UCSZ01);
 81: }
 82:
 83: void
 84: uart_clear_buffer()
 85: //-------------------------------------------------------------------------
 86: // Clear the UART input buffer
 87: //-------------------------------------------------------------------------
 88: {
 89:     in_len = 0;
 90: }
 91:
 92: boolean
 93: do_uart()
 94: //-------------------------------------------------------------------------
 95: // Send and receive characters
 96: //-------------------------------------------------------------------------
 97: {
 98:     // Check for characters to send and the UART being ready
 99:     if ((out_add != out_remove) && (UCSR0A & _BV(UDRE0))) {
100:         UDR0 = out_buf[out_remove++];
101:         if (out_remove == UBLEN) {
102:             out_remove = 0;
103:         }
104:     }
105:
106:     // Check for UART having a character to input
107:     if (UCSR0A & _BV(RXC0)) {
108:         char c = UDR0;
109:         // Special handling for return, means end of command
110:         if (c == '\r') {
111:             in_buf[in_len] = '\0';
112:             return true;
113:         }
114:         if (in_len < (UBLEN-2)) {
115:             in_buf[in_len++] = c;
116:         }
117:     }
118:
119:     return false;
120: }
121:
```