Remote Monitoring Example
Remote RF Power Output Monitor

Remote RF Power Output Monitoring

- Monitor RF power output via the Ethernet
- We will use an Arduino to gather the RF power data and send it to a remote computer
  - Arduino UDP packet server at transmitter site
  - C# client to display data on computer off-site

Remote RF Power Output Monitoring

- Sensor to provide voltage based on RF power level
- Arduino or other MCU with analog inputs and Ethernet capability

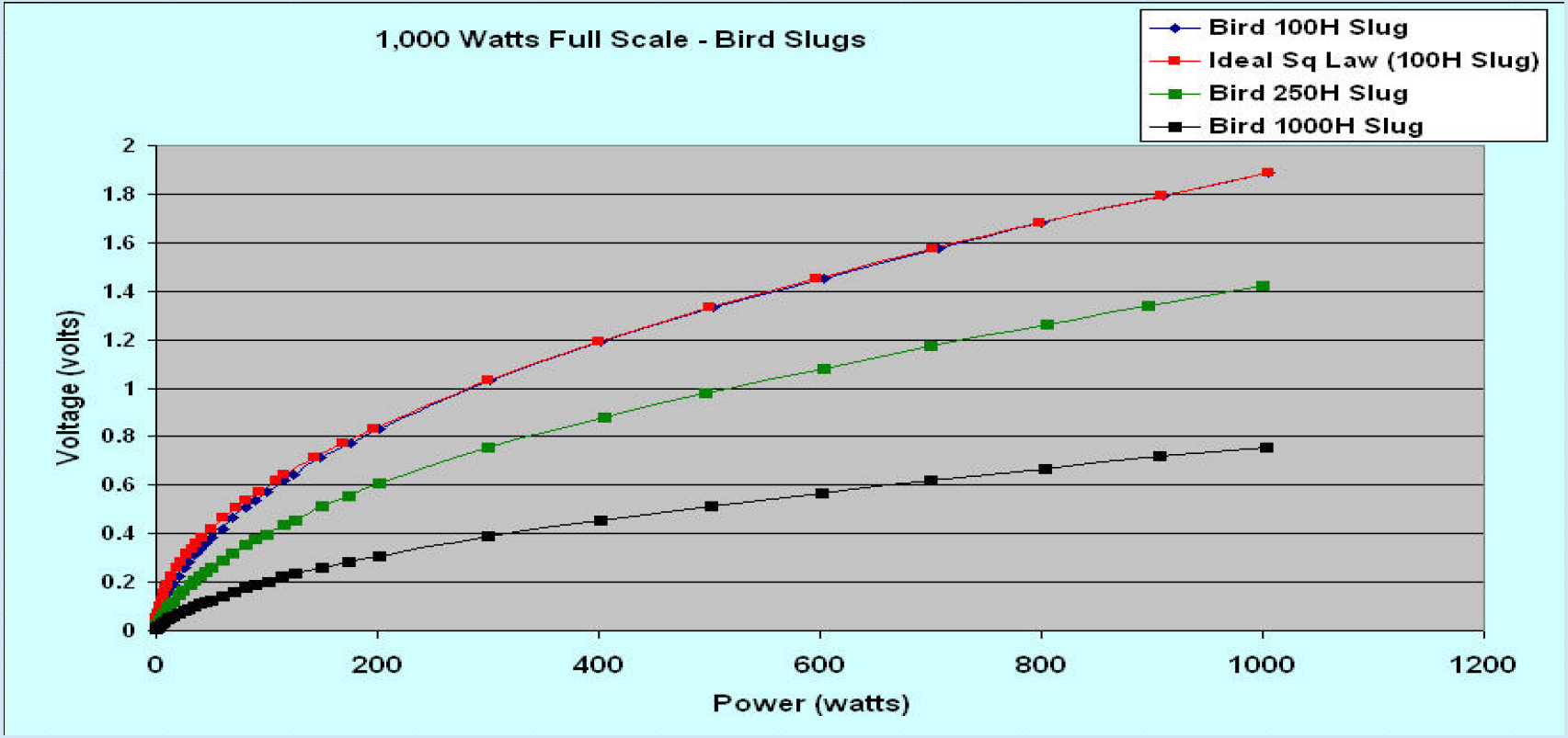
RF Power Output Monitoring Possible Sensors

- Bird Wattmeter line sections and elements
  - Output voltage depends on element (next slide)
- W1GHz power meter
  - http://www.w1ghz.org/new/portable_powermeter.pdf
  - Output voltage range 0.25 – 2.5 V
- Analog Devices Power Detectors

RF Power Output Monitoring
Bird Elements

- Output voltage depends on element
  - 100H Element @ 100 watts: 0.6V (unterminated)
  - 250H Element @ 250 watts: 0.7V (unterminated)
  - 1000H Element @ 1000 watts: 0.75V (unterminated)
  - http://www.meterbuilder.com/mb1/bird-line-sections.html
    - Recommends using 100H up to full legal limit
- Voltage/power relationship is non-linear
  - Software approach is perfect for this circumstance (you can calibrate using a known calibration curve)

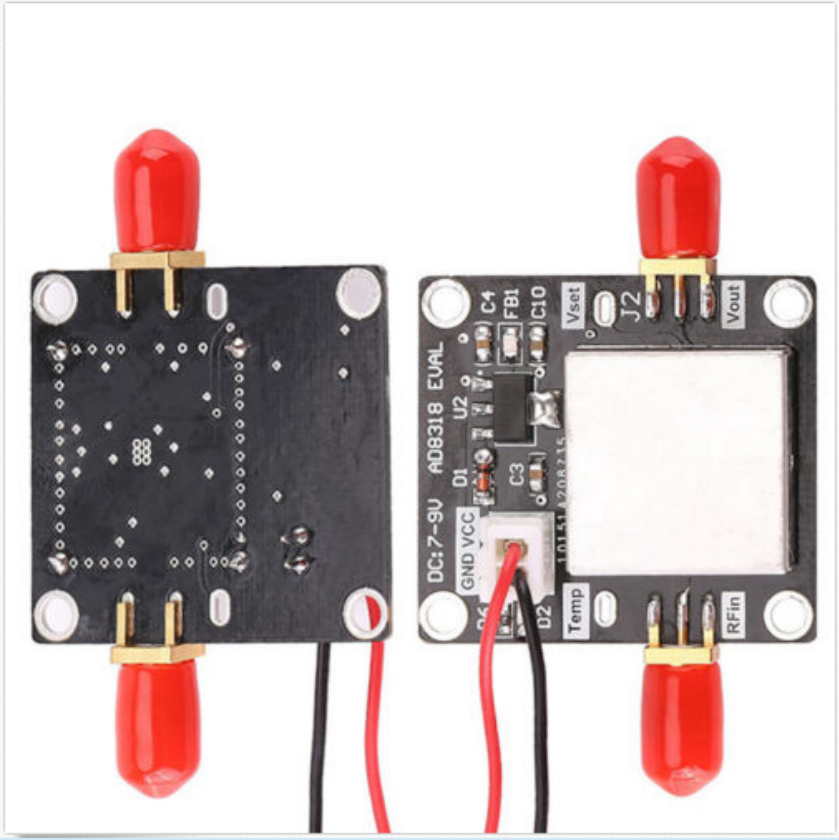
RF Power Output Monitoring Bird Elements



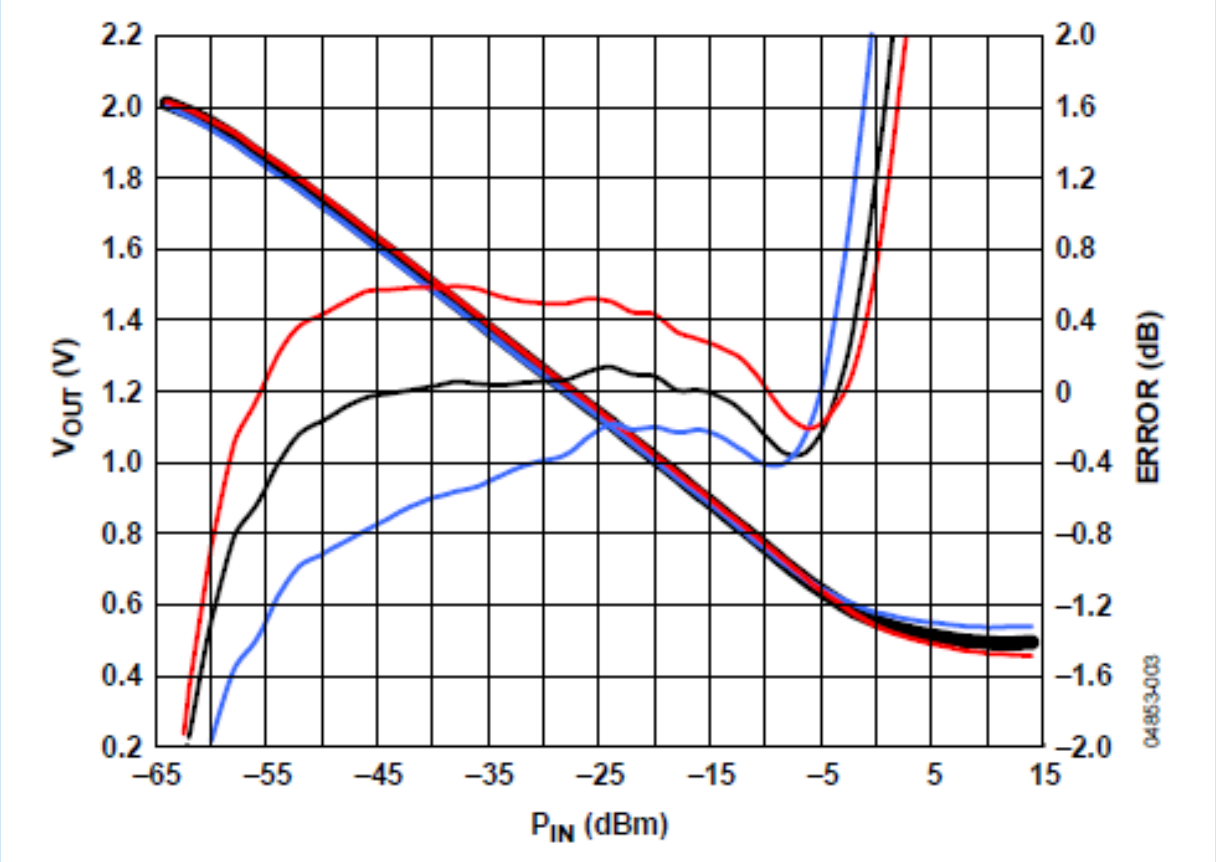
Analog Devices Power Detectors

Log Detectors

- AD8318
  - $13.37 on eBay



AD8318 Performance



Analog Devices Power Detectors

- Analog Devices ADL5XDETECTRKIT Evaluation Board
  - 3 detectors: ADL5511, ADL5513, ADL5902
  - DC – 9 GHz
  - -60 to 0 dBm or -30 to +30 dBm depending on detector
  - $10 per board; currently may be unavailable
  - http://www.richardsonrfpd.com/Pages/Product-Details.aspx?productId=1090721
  - Output voltage range 26 mV to 3.5 V (peak V varies among 3 detectors)

RF Power Output Monitor

MFG Part Number: ADL5XDETECTRKIT

RoHS

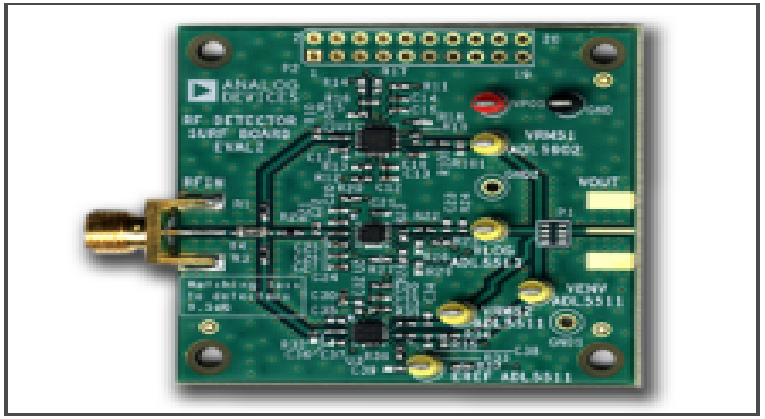
This RF Detector kit is a simple way to determine which of three popular detectors would work best for a specific RF applications, from cellular infrastructure and repeaters, to A&D, test and measurement, WiFi and more.

The kit contains three detectors mounted on an evaluation board, with one test signal input and three outputs to compare the performance of the three detectors simultaneously:

ADL5511: DC - 6 GHz, 47 dB Envelope and TruPwr™ RMS Detector

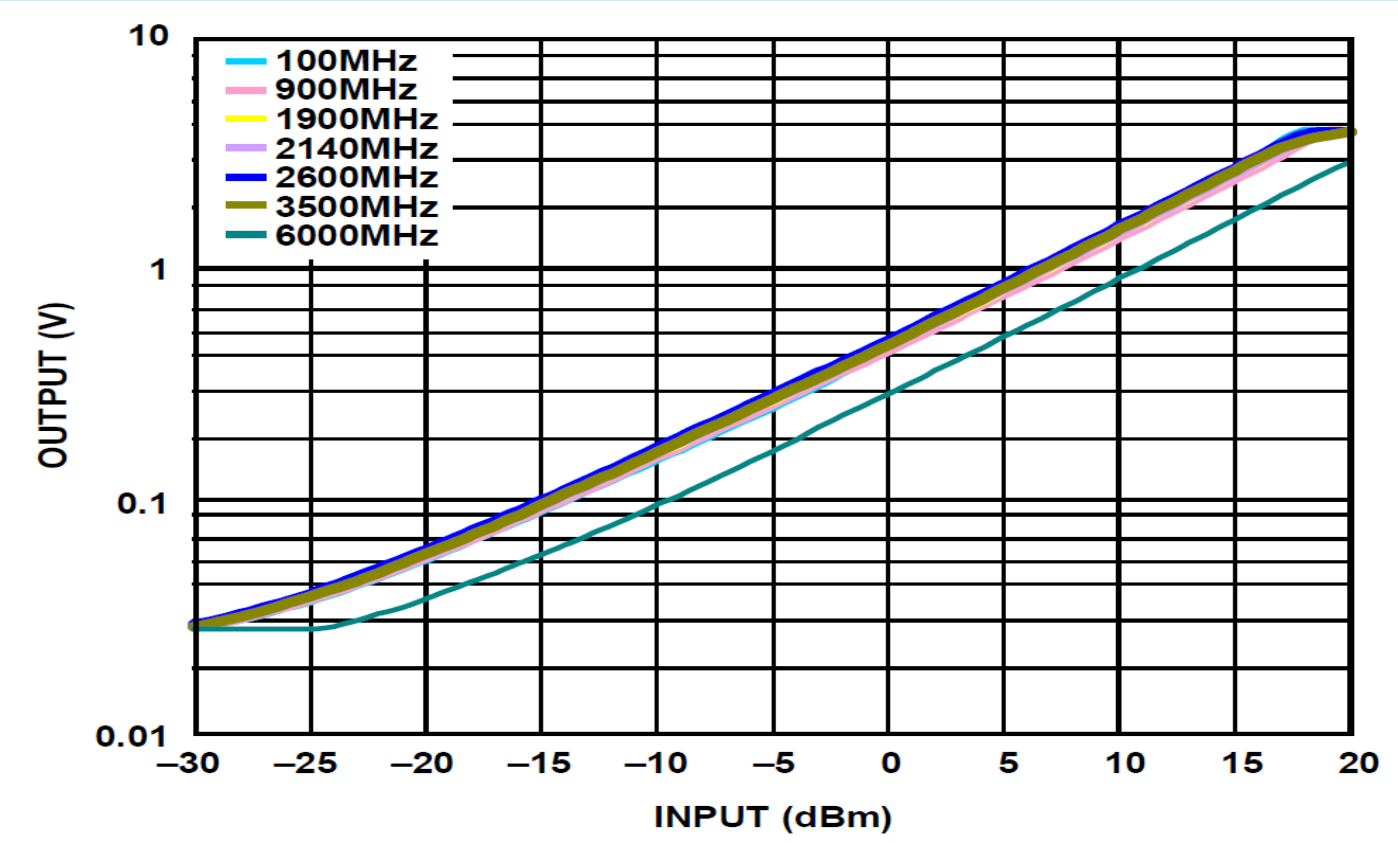
ADL5513: 1 MHz - 4 GHz, 80 dB Logarithmic Detector/Controller

ADL5902: 50 MHz - 9 GHz, 65 dB TruPwr™ RMS Detector



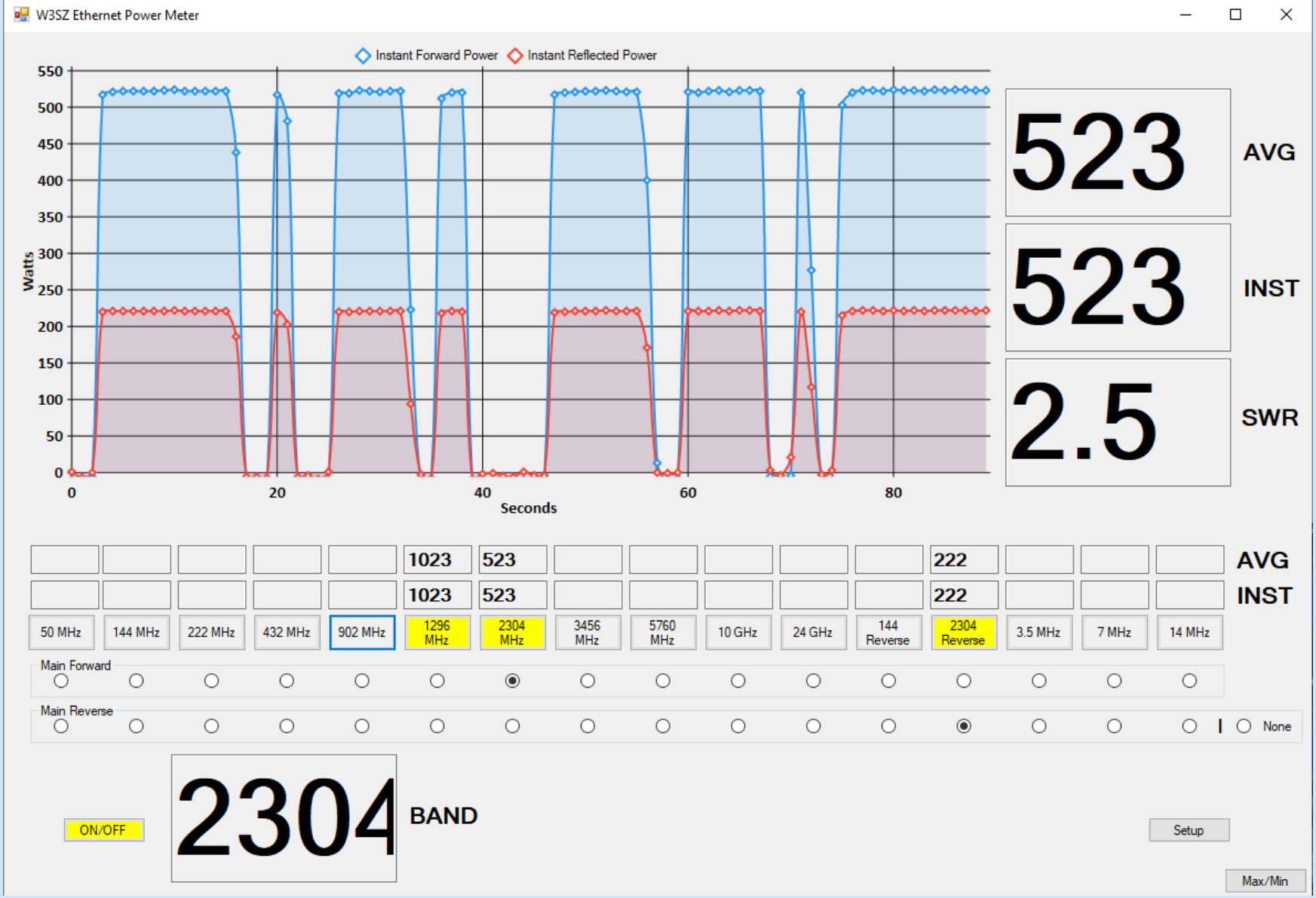
Enlarge Photo

ADL5511 Volts Out vs Power In



RF Power Output Monitor
Our Design Specs

- 16 input channels
- Graphic Display
- Remote selection of channel for graphic display, SWR measurement
- Inexpensive:
  - $10 for Arduino, $10 for the Analog Devices Evaluation Board or $13.37 for AD8318 Detector


W3SZ Ethernet Power Meter
Instant Forward Power
Instant Reflected Power
Watts
550
500
450
400
350
300
250
200
150
100
50
0
0
20
40
60
80
Seconds
523
AVG
523
INST
2.5
SWR
1023
523
222
AVG
1023
523
222
INST
50 MHz
144 MHz
222 MHz
432 MHz
902 MHz
1296 MHz
2304 MHz
3456 MHz
5760 MHz
10 GHz
24 GHz
144 Reverse
2304 Reverse
3.5 MHz
7 MHz
14 MHz
Main Forward
Main Reverse
None
ON/OFF
2304
BAND
Setup
Max/Min

SetupForm

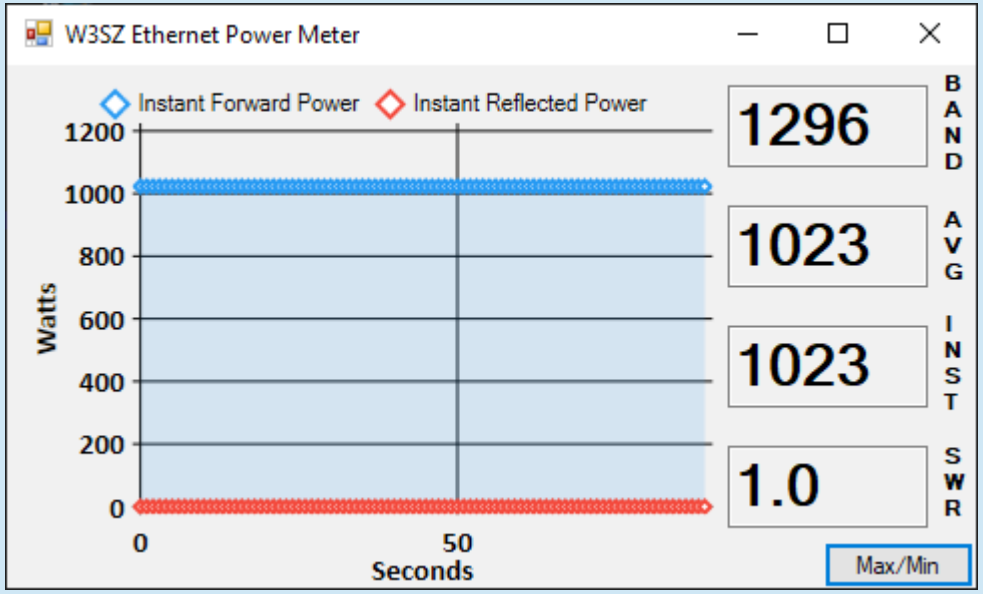
		REVERSE CHANNEL
50 MHz	BUTTON A0 LABEL	none
144 MHz	BUTTON A1 LABEL	none
222 MHz	BUTTON A2 LABEL	none
432 MHz	BUTTON A3 LABEL	none
902 MHz	BUTTON A4 LABEL	none
1296 MHz	BUTTON A5 LABEL	none
2304 MHz	BUTTON A6 LABEL	A12
3456 MHz	BUTTON A7 LABEL	none
5760 MHz	BUTTON A8 LABEL	none
10 GHz	BUTTON A9 LABEL	none
24 GHz	BUTTON A10 LABEL	none
144 Reverse	BUTTON A11 LABEL	none
2304 Reverse	BUTTON A12 LABEL	none
3.5 MHz	BUTTON A13 LABEL	none
7 MHz	BUTTON A14 LABEL	none
14 MHz	BUTTON A15 LABEL	none

192	168	10	244	This Computer's IP Address
192	168	10	177	Arduino's IP Address

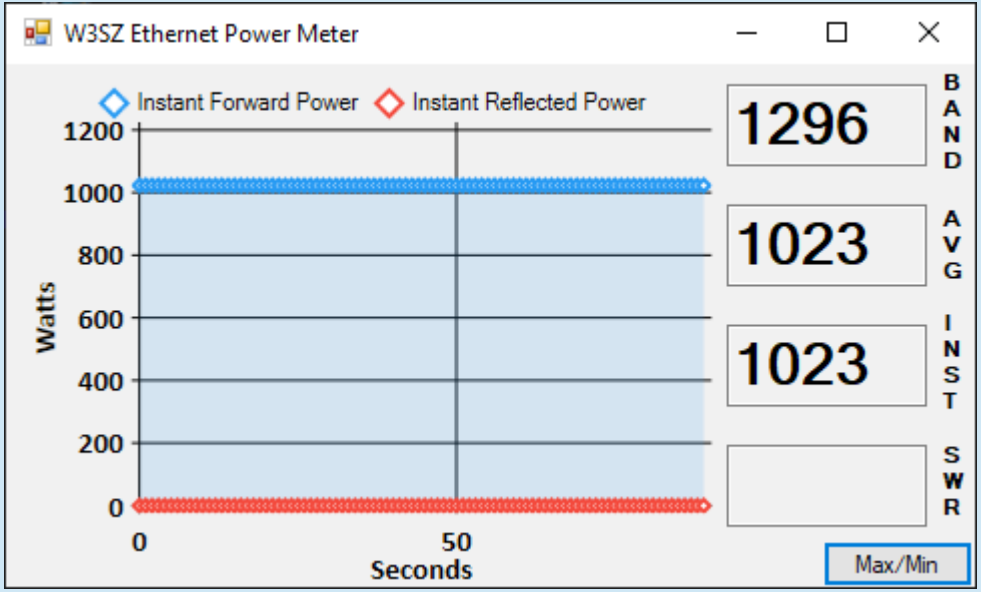
NOTE:
If you change IP address you will need to exit and restart program for new IP address to take effect.

Mini Window

Automatically Selects Input Channel with Largest Signal



With reverse channel selected in setup window



With no reverse channel selected in setup window

Remote RF Power Meter Demo

Remote Power Meter Code

- Arduino sketch is here:

  http://w3sz.com/W3SZ_Simple_Remote_PowerMeter.ino

- Zip file of C# source and binaries is here:

  http://w3sz.com/W3SZ_Remote_PowerMeter.zip

RF Power Output Monitor Code at the Arduino End

1) Include Libraries
2) Define and initialize constants and variables
3) Setup()
    - Define analog input pins
4) Loop()
    - Read voltage inputs from sensors via the analog input pins
    - Send selected voltage values to PC for display
    - Receive commands from PC
        - Turn measurement process on or off
        - Select channels to send to PC (up to 16 simultaneous channels)

Include Libraries, Define Variables

Preprocessor directives to include libraries

```
#include <Ethernet.h>  //for ethernet port
#include <string.h> // for string handling
#include <EthernetUdp.h>          // UDP library from: bjoern@cs.stanford.edu 12/30/2008

//variables
String commandInputString = "";
```

Define Ethernet-related Constants and Variables

```
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 pro
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;
```

Ethernet.h

- Library to work with Ethernet Shield, Ethernet Shield 2, and Leonardo Ethernet. Contains the classes:

  Ethernet: members begin, localIP, maintain

  IPAddress: member IPAddress(address)

  Server: members Server, EthernetServer, begin, available, write, print, println

  Client: members Client, EthernetClient, if(EthernetClient), connected, connect, write, print, println, available, read, flush, stop

  EthernetUdp members begin, read, write, beginPacket, endPacket, parsePacket, available, stop, remoteIP, remotePort

Define Ethernet-related Constants and Variables

```
// Enter MAC address and IP address for Arduino below.
byte mac[] = { 0x90, 0xAA, 0xBB, 0xCC, 0xDA, 0x02 };
IPAddress ip(192, 168, 10, 176);

IPAddress displayIP(192,168,10,244); //IP of computer running C# program to pro

unsigned int dataPort = 8888;      // local port to send and receive data on

// buffers for receiving and sending data
char packetBuffer[UDP_TX_PACKET_MAX_SIZE];  //buffer to hold incoming packet,
char  ReplyBuffer[] = "acknowledged";       // a string to send back

// An EthernetUDP instance to let us send and receive packets over UDP

EthernetUDP Udp;
```

Ethernet.IPAddress defines an IP address

EthernetUdp.h

- Library to send/receive UDP packets with Arduino. Contains the class EthernetUdp
  - Members include:
    - begin(uint16_t)
    - beginMulticast(IPAddress, uint16_t)
    - beginPacket(IPAddress ip, uint16_t, port)
    - endPacket()
    - write(uint16_t)
    - write(const uint8_t *buffer, size_t size)
    - parsePacket()
    - available()
    - read()
    - read(unsigned char* buffer, size_t len)
    - peek()
    - flush()
    - remoteIP()
    - remotePort()
    -

EthernetUDP.begin
EthernetUDP.beginMulticast
EthernetUDP.beginPacket
EthernetUDP.endPacket
EthernetUDP.write
EthernetUDP.parsePacket
.
.
.
.

Define Ethernet-related Constants and Variables

```
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 pro
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;
```

UDP_TX_PACKET_MAX_SIZE is defined as 24 bytes in EthernetUdp.h

We are defining the object Udp that is an instance of the class EthernetUdp

Define / Initialize Sensor Input Variables

```
int VoltA0 = 0;
int VoltA1 = 0;
int VoltA2 = 0;
int VoltA3 = 0;
int VoltA4 = 0;
int VoltA5 = 0;
int VoltA6 = 0;
int VoltA7 = 0;
int VoltA8 = 0;
int VoltA9 = 0;
int VoltA10 = 0;
int VoltA11 = 0;
int VoltA12 = 0;
int VoltA13 = 0;
int VoltA14 = 0;
int VoltA15 = 0;
```

Define/Initialize Control Parameters

```
String MeterOn = "OFF";  //turns measurement UDP server on or off
String BANDA0 = "ON";     //turns sensor with this numeral on or off
String BANDA1 = "ON";     //turns sensor with this numeral on or off
String BANDA2 = "ON";     //turns sensor with this numeral on or off
String BANDA3 = "ON";     //turns sensor with this numeral on or off
String BANDA4 = "ON";     //turns sensor with this numeral on or off
String BANDA5 = "ON";     //turns sensor with this numeral on or off
String BANDA6 = "ON";     //turns sensor with this numeral on or off
String BANDA7 = "ON";     //turns sensor with this numeral on or off
String BANDA8 = "ON";     //turns sensor with this numeral on or off
String BANDA9 = "ON";     //turns sensor with this numeral on or off
String BANDA10 = "ON";     //turns sensor with this numeral on or off
String BANDA11 = "ON";     //turns sensor with this numeral on or off
String BANDA12 = "ON";     //turns sensor with this numeral on or off
String BANDA13 = "ON";     //turns sensor with this numeral on or off
String BANDA14 = "ON";     //turns sensor with this numeral on or off
String BANDA15 = "ON";     //turns sensor with this numeral on or off
```

Setup Pin Modes, Start Ethernet and Serial Port

```
void setup() {

  //set pin modes to input
  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(A2, INPUT);
  pinMode(A3, INPUT);
  pinMode(A4, INPUT);
  pinMode(A5, INPUT);
  pinMode(A6, INPUT);
  pinMode(A7, INPUT);
  pinMode(A8, INPUT);
  pinMode(A9, INPUT);
  pinMode(A10, INPUT);
  pinMode(A11, INPUT);
  pinMode(A12, INPUT);
  pinMode(A13, INPUT);
  pinMode(A14, INPUT);
  pinMode(A15, INPUT);

  // start the Ethernet connection and the server and the serial port:
  Ethernet.begin(mac, ip);
  Udp.begin(dataPort);
  Serial.begin(9600);
  Serial.println("Starting Server");
  Serial.println (Ethernet.localIP());
```

Ethernet.begin(mac, ip) Initializes the ethernet library and network settings. mac is array of 6 bytes. ip is array of 4 bytes. Returns nothing.

EthernetUdp.begin(port) Initialize, start listening on specified port. Returns 1 if successful, 0 if there are no sockets (unsuccessful)

Ethernet.localIP() Obtains the IP address of the Ethernet shield. Returns the IP address.

Send Startup Message to Serial Port

```
// Print a message to the serial port

  Serial.println("Pwr Meter");
  Serial.println("1 MHz - 9 GHz");
  Serial.println("W3SZ 08/2017");

  delay (4000);

} // end of setup

```

Start Loop, Read Voltages

```
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
```

UDP.parsePacket checks for packet and reports size

```
int packetSize = Udp.parsePacket();
if (packetSize) {
  Serial.print("Received packet of size ");
  Serial.println(packetSize);
  Serial.print("From ");
  IPAddress remote = Udp.remoteIP();
  for (int i = 0; i < 4; i++) {
    Serial.print(remote[i], DEC);
    if (i < 3) {
      Serial.print(".");
    }
  }
  Serial.print(", port ");
  Serial.println(Udp.remotePort());
```

EthernetUDP.parsePacket(): Returns the size of the packet in bytes or 0 if no packets are available

EthernetUDP.remoteIP(): Returns the IP address of the host who sent the current incoming packet

EthernetUDP.remotePort(): Return the port of the host who sent the current incoming packet

Read packet and parse string to extract commands sent from PC

```
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           }
```

EthernetUDP.read(buffer, len): Read up to len characters from the current packet and place them into buffer, Returns the number of characters read, or 0 if none are available

UDP_TX_PACKET_MAX_SIZE is defined as 24 bytes in EthernetUdp.h

Arduino String class

- Members include:

  charAt
  compareTo
  concat
  c_str
  endsWith
  equals
  equalsIgnoreCase
  getBytes
  indexOf
  lastIndexOf
  length
  remove
  replace
  reserve
  setCharAt
  startsWith
  substring
  toCharArray
  toInt
  toFloat
  toLowerCase
  toUpperCase
  trim

Read packet and parse string to extract commands sent from PC

```
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           }
```

String.indexOf(val) Locates a character or String val within another String. Returns the index of val within the String, or -1 if not found.

String.substring(val1, val2) Gets a substring of a String. The starting index val1 is inclusive (the corresponding character is included in the substring), but the optional ending index val2 is exclusive. Returns the substring.

Continue parsing string to extract commands sent from PC

```
        else if (commandOut  == "BANDA0OFF") {
          String HTMString = "BAND A0 is OFF";
          Serial.println(HTMString);
          BANDA0 = "OFF";
        }

        else if (commandOut  == "BANDA1ON") {
          String HTMString = "BAND A1 is ON";
          Serial.println(HTMString);
          BANDA1 = "ON";
        }
        else if (commandOut  == "BANDA1OFF") {
          String HTMString = "BAND A1 is OFF";
          Serial.println(HTMString);
          BANDA1 = "OFF";
        }

        else if (commandOut  == "BANDA2ON") {
          String HTMString = "BAND A2 is ON";
          Serial.println(HTMString);
          BANDA2 = "ON";
        }
        else if (commandOut  == "BANDA2OFF") {
          String HTMString = "BAND A2 is OFF";
          Serial.println(HTMString);
          BANDA2 = "OFF";
```

More parsing string to extract commands sent from PC

```
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         }
```

Finish reading commands
Start reading sensor data
Form data string

```
        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);
```

Continue forming data string

```
    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);
    }
```

Finish forming data string

```
    if(BANDA11 =="ON"){
      data = data + ",A11=" +String(VoltA11);
    }
    if(BANDA12 =="ON"){
      data = data + ",A12=" +String(VoltA12);
    }
    if(BANDA13 =="ON"){
      data = data + ",A13=" +String(VoltA13);
    }
    if(BANDA14 =="ON"){
      data = data + ",A14=" +String(VoltA14);
    }
    if(BANDA15 =="ON"){
      data = data + ",A15=" +String(VoltA15);
    }
```

Data string example
All 16 channels ON

```
DATA,A00=157,A01=168,A02=243,A03=256,A04=270,A05=279,A06=288,
```

```
A07=289,A08=292,A09=302,A10=303,A11=304,A12=302,A13=305,A14=306,A15=297
```

Send Data String And End Loop

```
  if(MeterOn == "ON")
  {
   int datalength = 1 + data.length();
   char databuf[datalength];
   data.toCharArray(databuf, datalength);
   // send a reply to the IP address and port that sent us the packet we received
   Udp.beginPacket(displayIP, dataPort);
   Udp.write(databuf);
   Udp.endPacket();
//    Serial.println(datalength);
 //   Serial.print("DATA IS: ");
//    Serial.println(data);
//    Serial.print("DATABUF IS: ");
//    Serial.println(databuf);
  }
 delay(50);
} //end loop
```

Arduino String class

- Members include:

  charAt
  compareTo
  concat
  c_str
  endsWith
  equals
  equalsIgnoreCase
  getBytes
  indexOf
  lastIndexOf
  length
  remove
  replace
  reserve
  setCharAt
  startsWith
  substring
  toCharArray
  toInt
  toFloat
  toLowerCase
  toUpperCase
  trim

Send Data String And End Loop

string.length() Returns the length of the String in characters.

string.toCharArray(buf, len) Copies the String's characters to the supplied buffer buf of size len. Returns nothing.

```
  if(MeterOn == "ON")
  {
   int datalength = 1 + data.length();
   char databuf[datalength];
   data.toCharArray(databuf, datalength);
   // send a reply to the IP address and port that sent us the packet we received
   Udp.beginPacket(displayIP, dataPort);
   Udp.write(databuf);
   Udp.endPacket();
//   Serial.println(datalength);
 //  Serial.print("DATA IS: ");
//   Serial.println(data);
//   Serial.print("DATABUF IS: ");
//   Serial.println(databuf);
  }
 delay(50);
} //end loop
```

Send Data String And End Loop

```
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
```

EthernetUDP.beginPacket(remoteIP, remotePort): Starts a connection to write UDP data to the remote connection. Returns 1 if successful, 0 if there was a problem resolving the hostname or port.

EthernetUDP.write(message) Writes UDP data to the remote connection. Returns the number of characters sent.

EthernetUDP.endPacket(): Called after writing UDP data to the remote connection. Returns 1 if the packet was sent successfully, 0 if there was an error.

What happens at the other end?

- C# program gets data string

```
DATA,A00=157,A01=168,A02=243,A03=256,A04=270,A05=279,A06=288,
```

```
A07=289,A08=292,A09=302,A10=303,A11=304,A12=302,A13=305,A14=306,A15=297
```

- C# program parses data
- C# program displays data
- C# program sends channel On/Off commands to Arduino

Remote RF Power Monitor Coding

- Very Straightforward:
  - Got Some Input from analog input pins
  - Did Something With It (formed data string to send to PC)
  - Produced Some Output (UDP packet of data)

Programming Steps

1) Included libraries containing external functions
   - Ethernet.h
   - string.h
   - EthernetUDP.h
2) Defined variables and constants
3) Setup ()
   - Defined and initialized Analog I/O pins
   - Defined, started serial port, Ethernet port
4) Loop()
   - Received input from Ethernet port / Analog pins
   - Parsed / processed data to extract desired information
   - Used information derived from data to perform desired task (e.g. switch channels on or off) and to send RF Power Data to client computer
5) From within Loop(), called other functions() as needed (e.g. Serial.x, Udp.x, data.toCharArray, delay)

Wrap-up

What Now?

- Pick a Project
- Choose “best” device for project
- Use Google and code examples from this seminar to get started and write the code
- Have fun!

THERE'S BEEN A LOT OF CONFUSION OVER 1024 vs 1000, KBYTE vs KBIT, AND THE CAPITALIZATION FOR EACH.

HERE, AT LAST, IS A SINGLE, DEFINITIVE STANDARD:

SYMBOL	NAME	SIZE	NOTES
kB	KILOBYTE	1024 BYTES OR 1000 BYTES	1000 BYTES DURING LEAP YEARS, 1024 OTHERWISE
KB	KELLY-BOOTLE STANDARD UNIT	1012 BYTES	COMPROMISE BETWEEN 1000 AND 1024 BYTES
KiB	IMAGINARY KILOBYTE	1024 √-1 BYTES	USED IN QUANTUM COMPUTING
kb	INTEL KILOBYTE	1023.937528 BYTES	CALCULATED ON PENTIUM F.P.U.
Kb	DRIVEMAKER'S KILOBYTE	CURRENTLY 908 BYTES	SHRINKS BY 4 BYTES EACH YEAR FOR MARKETING REASONS
KBa	BAKER'S KILOBYTE	1152 BYTES	9 BITS TO THE BYTE SINCE YOU'RE SUCH A GOOD CUSTOMER

I would take 'kibibyte' more seriously if it didn't sound so much like 'Kibbles N Bits'.