----------------------------------------

/\*  
Version 9, The changes to this device to stop using XBee broadcast fixed all the  
problems with it failing over time.  A month ago I discovered the XBee library  
had been updated to support the SoftwareSerial library and this means I can now  
use the library to simplify the code I have to write, and rejoin the mainstream  
of XBee work.  This version has the library associated and it made the XBee send  
much, much simpler.  I don't check the transmit response frame to see what happened  
with the transmission.  I will if there seems to be a problem at any point.  However,  
that always brings up the problem of what to do with an error report.  I have  
nowhere to send it.   
  
Version 8, Well, as I add devices to the network, I have to stop using broadcast.   
Each device is echoing the broadcast and totally clogging up the network with  
retransmissions.  So, this version is the same as 7 except it sends the packets to  
the house controller.  The house controller is responsible for updating status now,  
so that isn't too bad.  
  
Version 7 of the monitor; still trying to get around the problem of short lines.  
I got a clue when it stopped a couple of days ago.  Monitoring the XBee traffic, it  
was stopping in exactly the same place each time.  Then after a second or two it would  
continue.  The lines were always complete, but characters inside were separated by a  
couple of seconds.  This meant that the clock or something else was able to get in  
the middle of the transmission.  This was happening on every single line, so the  
controller couldn't decode the transmission.  I switched the XBee to API mode and put  
code in to support it so that each packet was complete.  Now to wait a week or so to  
see if this gets rid of the problem.  Months of working just fine and then this shows  
up.  Funny how something like this hides forever.  
  
Version 6 of the monitor; I had the device hang up for around 12 hours not  
reporting the proper power over the XBee network.  The house controller was seeing a  
power usage of zero and assumed (correctly) that this was invalid and didn't  
update pachube.  I didn't think to get enough data from the various devices around  
the house before resetting the monitor, so I don't know exactly what happened.  
  
However, the monitor doesn't reset itself to clean out any problems, and since the  
reset fixed it; it must have been a software problem.  The easiest way to fix this  
without information is to just have the board reset itself periodically.  So, I'm  
just setting the time to something and having it reboot every 24 hours.  That will  
restart all the stuff and allow it come up cleanly.  To do this I added the timeAlarm  
library.  There are two timers, one to report the power and the other to cause the  
reset.  I also moved the reporting function into it's own routine partly because that  
made it easier to hook it to a timer and partly to clean up the display logic.  Since  
I have lots of memory on this device, I used sprintf() to format the string for transmission  
This can also provide an example for folks trying to do something similar.  
  
  
Version 5 of the monitor.  The first version was ethernet enabled and  
used wifi to communicate.  Later, I added an XBee and transmitted the data over  
it also.  Now, I removed the WiShield that allowed connection over the ethernet.  
I was getting more and more hangups and resets over time.  I suspect there is a  
problem in the code with large packets that occasionally travel around and the  
device isn't supported anymore, why spend time fixing the code?  
So the device transmits only over the XBee now.  Which makes it a remote power  
monitor that forwards data to another device that can listen to it.  
  
Actually, this isn't a bad idea at all.  I can pick up the data from any other  
device around the house and do whatever I want with it.  
/\*  
 \* Credits:  
 \* Most of the power measurement and calculations were invented by Trystan Lea and documented at  
 \* http://openenergymonitor.org/.  I modified it for the split phase 240 in residential use in the U.S.  
 \* I'm using a ladyada boot rom (purchased directly from her site) to overcome unexpected problems  
 \* and multiple vulnerable arduinos.  A lot of the remaining code was taken from example sketches  
 \* supplied by the Arduino site www.arduino.cc  
 \* Basic energy monitoring sketch plus kwh and frequency calc - by Trystan Lea  
 \* Licenced under GNU General Public Licence more details here  
 \* openenergymonitor.org  
 \*  
 \* Open source rules!  
 \* Pin usage:  
 \*   3,4 for newsoftserial XBee comm  
 \*   Analog 0 current sensor  
 \*   Analog 1 voltage sensor  
 \*/  
  
//Sketch measures voltage and current.  
//and then calculates useful values like real power,  
//apparent power, powerfactor, Vrms, Irms, frequency and kwh.  
  
#include <avr/wdt.h>  
#include <MemoryFree.h>  
#include <avr/pgmspace.h>  
#include <SoftwareSerial.h>  
#include <Time.h>  
#include <TimeAlarms.h>  
#include <XBee.h>  
  
char verNum[] = "Version 9";  
  
SoftwareSerial xbeeSerial = SoftwareSerial(3,4);  
XBee xbee = XBee();  
  
char Dbuf[100];   // general purpose buffer  
//Setup variables  
int numberOfSamples = 3000;  
  
//Set Voltage and current input pins  
int inPinV = 4;  
int inPinI = 5;  
  
//Calibration coeficients  
double VCAL = 0.592;  
double ICAL = 03.2;  
double PHASECAL = 0.1;  
  
//Sample variables  
int lastSampleV,lastSampleI,sampleV,sampleI;  
  
//Filter variables  
double lastFilteredV, lastFilteredI, filteredV, filteredI = 0;  
double filterTemp;  
  
//Stores the phase calibrated instantaneous voltage.  
double calibratedV;  
  
//Power calculation variables  
double sqI,sqV,instP,sumI,sumV,sumP;  
  
//Useful value variables  
double realPower,  
       apparentPower,  
       powerFactor,  
       Vrms,  
       Irms;  
  
  //--ENERGY MEASURMENT VARIABLES------------------------------  
    //Calculation of kwh  
  
    //time taken since last measurment timems = tmillis - ltmillis;  
    unsigned long ltmillis, tmillis, timems;  
    //time when arduino is switched on... is it 0?  
    unsigned long startmillis;  
     
  //--FREQUENCY MEASURMENT VARIABLES---------------------------   
    //time in microseconds when the voltage waveform  
    //last crossed zero.  
    unsigned long vLastZeroMsec;  
    //Micro seconds since last zero-crossing  
    unsigned long vPeriod;  
    //Sum of vPeriod's to obtain an average.  
    unsigned long vPeriodSum;  
    //Number of periods summed  
    unsigned long vPeriodCount;  
     
    //Frequency  
    float freq;  
     
    //Used to filter out fringe vPeriod readings.  
    //Configured for 50Hz  
    //- If your 60Hz set expPeriod = 16666  
    unsigned long expPeriod = 16666;  
    unsigned long filterWidth = 2000;  
  //-----------------------------------------------------------  
  
/\* Measurement calculations \*/   
  
void PwrCalcs()  
{  
  for (int n=0; n<numberOfSamples; n++) // gather samples  
  {  
     //Used for offset removal  
     lastSampleV=sampleV;  
     lastSampleI=sampleI;  
     //Read in voltage and current samples.    
     sampleV = analogRead(inPinV);  
     sampleI = analogRead(inPinI);  
  
     //Used for offset removal  
     lastFilteredV = filteredV;  
     lastFilteredI = filteredI;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
     //Digital high pass filters to remove 2.5V DC offset.  
     filteredV = 0.996 \* (lastFilteredV+sampleV-lastSampleV);  
     filteredI = 0.996 \* (lastFilteredI+sampleI-lastSampleI);  
  
     //Phase calibration goes here.  
     calibratedV = lastFilteredV + PHASECAL \* (filteredV - lastFilteredV);  
   
     //Root-mean-square method voltage  
     //1) square voltage values  
     sqV= calibratedV \* calibratedV;  
     //2) sum  
     sumV += sqV;  
    
     //Root-mean-square method current  
     //1) square current values  
     sqI = filteredI \* filteredI;  
     //2) sum  
     sumI += sqI;  
  
     //Instantaneous Power  
     instP = abs(calibratedV \* filteredI);  
     //Sum  
     sumP += instP;  
    
     //--FREQUENCY MEASURMENT---------------------------            
     if (n==0) vLastZeroMsec = micros();  
    
     //Check for zero crossing from less than zero to more than zero  
     if (lastFilteredV < 0 && filteredV >= 0 && n>1)  
     {  
       //period of voltage waveform  
       vPeriod = micros() - vLastZeroMsec;  
  
       //Filteres out any erronous period measurments  
       //Increases accuracy considerably  
       if (vPeriod > (expPeriod-filterWidth) && vPeriod<(expPeriod+filterWidth))  
       {  
          vPeriodSum += vPeriod;  
          vPeriodCount++;  
       }  
       vLastZeroMsec = micros();  
     }  
  } //end of sample gathering  
  
  //Calculation of the root of the mean of the voltage and current squared (rms)  
  //Calibration coeficients applied.  
  Vrms = VCAL\*sqrt(sumV / numberOfSamples);  
  Irms = ICAL\*sqrt(sumI / numberOfSamples);  
  
  //Calculation power values  
  realPower = VCAL\*ICAL\*sumP / numberOfSamples;  
  apparentPower = Vrms \* Irms;  
  powerFactor = realPower / apparentPower;  
  
  //FREQUENCY CALCULATION--------------------------  
  freq = (1000000.0 \* vPeriodCount) / vPeriodSum;  
       
  vPeriodSum=0;  
  vPeriodCount=0;  
  //------------------------------------------------  
  
  //--ENERGY MEASURMENT CALCULATION----------------      
         
  //Calculate amount of time since last realpower measurment.  
  ltmillis = tmillis;  
  tmillis = millis();  
  timems = tmillis - ltmillis;  
       
  //Reset accumulators  
  sumV = 0;  
  sumI = 0;  
  sumP = 0;  
}  
  
// this function outputs the current free memory to the serial port  
// really nice to use in debugging and making sure the board doesn't  
// fail running out of memory  
void showMem(){  
  strcpy\_P(Dbuf,PSTR("Mem = "));  
  Serial.print(Dbuf);  
  Serial.println(freeMemory());  
}  
  
// this set of functions are for a software reset of the board  
// the reset function allows a call to location zero which will emulate a reset  
// the resetMe funtion allows a normal call from the timer routines  
void(\* resetFunc) (void) = 0; //declare reset function @ address 0  
  
void resetMe(){  // for periodic resets to be sure nothing clogs it up  
  Serial.println("Periodic Reset - Normal Operation");  
  resetFunc();  
}  
  
// this little function will return the first two digits after the decimal  
// point of a float as an int to help with sprintf() (won't work for negative values)  
int frac(float num){  
  return( (num - (int)num) \* 100);  
}  
  
// report the power usage over XBee network and out the Serial Port  
void reportPower(){   
  memset(Dbuf,0,sizeof(Dbuf));  
  Serial.print("Broadcast--");  
  // first construct the payload line  
  sprintf(Dbuf,"Power,%d.%02d,%d.%02d,%d.%02d,%d.%02d,%d.%02d,%d.%02d\r",  
       (int)realPower, frac(realPower),  
       (int)apparentPower, frac(apparentPower),  
       (int)powerFactor, frac(powerFactor),  
       (int)Vrms, frac(Vrms),  
       (int)Irms, frac(Irms),  
       (int)freq, frac(freq));  
  // Display it on the serial monitor for debugging  
  Serial.print(Dbuf);  
  Serial.print("\n");  
  sendStatusXbee(Dbuf);  
}  
  
void setup()  
{  
  Serial.begin(9600);  
  Serial.println(verNum);  
  xbeeSerial.begin(9600);  
  // This sets the XBee library to use the software serial port  
  xbee.setSerial(xbeeSerial);  
  
  //--ENERGY MEASURMENT SETUP--------------------------------  
  tmillis = millis();  
  startmillis=tmillis;  
  //---------------------------------------------------------  
  Serial.println("I'm alive ");  
  Serial.println("Setting timer for reporting");  
  /\* I really don't care what time it is on this device  
     it just measure time and reports.  But, I want the timer capability  
     to allow a reset every 24 hours and to handle the reporting function  
     so I just set the time to something reasonable and get on with the  
     rest of the work.  
  \*/  
  setTime(0,0,0,1,1,12);  
  Alarm.timerRepeat(5, reportPower);   // report the power usage every 5 seconds  
  Alarm.alarmRepeat(23,59,0,resetMe);  // periodic reset to keep things cleaned up  
                                       // I use a lot of libraries and sometimes they have bugs  
                                       // as well as hang ups from various hardware devices  
  showMem();                           // to make sure I don't make it too big to fit in ram reliably  
  pinMode(6,OUTPUT);  
  digitalWrite(6,LOW);  
  Serial.println("Init done");  
  wdt\_enable(WDTO\_8S);          // No more than 8 seconds of inactivity  
   
}  
  
/\*  
  The loop() just calculates power over and over again.  There is a timer  
  set in setup() that causes the device to report every few seconds.  
  The loop() also resets the watchdog timer so it doesn't time out.  
\*/  
void loop()  
{  
  // get power calcs into variables  
  PwrCalcs();  
  wdt\_reset();                   // watchdog timer set back to zero  
  Alarm.delay(0);                // This causes the alarm timer to update  
}