```
/*
01.05.16
 CH_timer_28
 change of MAC for actual timer rather than test unit

 TIMER PROGRAM FOR ARDUINO CENTRAL HEATING

 Credit to Michael Margolis and John Boxall
 temp102 by Arduino Playground author
 timer control developed 4.7.15 onwards by Julian Rogers

 */

//timer with two ons and offs for water and heat manual advance and manual over-ride
//Arduino Uno with Ethernet Shield with SD card
//temperature by TMP102 (sockets for 2 wired sensors - could use network connected sensors
//clock is DS3231
//clock should be set to GMT, conversion to BST is by software
//reads manual switch positions
//turns on status LEDs as appropriate
//water and heat LEDs will flash if manual off during an "on" timed period
//on off times and thermostat value stored on SD card
//communication via UDP
//timer times, clock setting, thermostat setting set remotely by UDP
//includes data logger function now enough RAM available


/*
  STATUS LIGHTS

  Switch | Timed On | Timed Off
  _______________________________
    T    | Cont Red | Faint Green
  _______________________________
    O    |Flash Red |   Off
  _______________________________
    C    |Cont Green| Cont Green
  _______________________________

  OUTPUT FROM SERIAL MONITOR
  Cryptic to reduce RAM usage!
  T plus temp * 10, S or G (Summer or GM time) plus hh:mm, on/off times, ON or OFF depending
  whether roomstat setting is satisfied.

  OUTPUT FROM UDP
  Cryptic to reduce RAM usage!
  time in minutes, / plus on/off times, T plus temp * 10, S plus switch code,
  A plus advance code

*/

//Assignment of Arduino pins:

//D0 = serial RX (not connected)
//D1 = "running" (serial comms TX)
//D2 = advance LED - will flash if advance is activated
//D3 = water LED
//D4 = used by Ethernet Shield, SD card
//D5 = heat LED
//D6 = water advance button
//D7 = heat advance button
//D8 = heat relay
//D9 = water relay
//D10= SPI (Ethernet Shield, SD card)
//D11= SPI
//D12= SPI
//D13= SPI
//D16 = output to turn off motorised valve
//D17 = output to self-reset (via an NPN transistor)

//A0 = gives heat switch position
//A1 = gives water switch position
//A2, A3 = unallocated sensor inputs or digital input/outputs
//                  (eg. to turn off power to motorised valve when boiler not operating)
//A2 becomes D16 and A3 becomes D17
//A2 now allocated as digital output to turn off motorised valve.
//A4 = I2C SDA (clock and temp sensors}
//A5 = I2C SCL

//Libraries:

#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h> // UDP library from: bjoern@cs.stanford.edu 12/30/2008
#define UDP_TX_PACKET_MAX_SIZE 64 //increase UDP size
#include <Wire.h>
#define DS3231_I2C_ADDRESS 0x68
#define TMP102_I2C_ADDRESS 0x48 // I2C address TMP102 A0 to GND (0x48 = 72 = 1001000 for GND, 73 for vcc)
#include <SD.h>
#define  SERIAL_BUFFER_SIZE 16 // reduce  buffer size
```

```
//global variables:
/*
int waterOn1;
int waterOff1;
int waterOn2;
int waterOff2;

int heatingOn1;
int heatingOff1;
int heatingOn2;
int heatingOff2;
*/
byte advCode; //holds manual advance status

int tim;      //current (hours x 60 + minutes) for daily timed periods
int setTemp = 180;  //180 is default temp for thermostat if SD card fails
const byte hysteresis = 3;  //used in thermostat function - is this enough?

char gmtBst[2];   //holds GMT or BST

//char tNowStr[7];

//boolean sndRec = false;

boolean newChangeH = false;
boolean newChangeW = false;
boolean oldChangeH = false;
boolean oldChangeW = false;
boolean advanceH = false;
boolean advanceW = false;

boolean heat1 = false;
boolean heat2 = false;
boolean water1 = false;
boolean water2 = false;
boolean heat = false;
boolean water = false;

boolean dataAdded = true;
//needed for data logging which does not work!


// MAC address for "genuine" ethernet shield is 90:A2:DA:0D:2C:EC
// MAC address for knock off shield is DE:AD:BE:EF:FE:ED
// The IP address will be dependent on the local network:
  //byte mac[] = {
 // 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };   // test Ethernet Shield

  byte mac[] = {
  0x90, 0xA2, 0xDA, 0x0D, 0x2C, 0xEC };   // test Ethernet Shield

IPAddress ip(192, 168, 1, 177);

unsigned int localPort = 8888;      // local port to listen on

char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; //buffer to hold incoming packet,
//char timStr[7];
char hourStr[7];
char dayStr[7];
char monthStr[7];
char clockSetString[18];
char onOffString[48];
char newonOffString[64];
char tempSetString[4];
char tNowStr[7];
byte sStart[5];
byte sEnd[5];
File myFile;


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


//////////////////////////////////////////////////////////////////////

void setup() {

  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(6, INPUT);
  pinMode(7, INPUT);
  pinMode(16, OUTPUT);  // formally analog 2
  pinMode(17, OUTPUT);  // formally analog 3
  digitalWrite(6, HIGH);
  digitalWrite(7, HIGH);
  //digitalWrite(16, HIGH); // high only at end of day to deactivate valve
```

```
// load dates for summer time
// start 2015, end 2018
// start in March, end in October

sStart[0] = 29;
sStart[1] = 27;
sStart[2] = 26;
sStart[3] = 25;

sEnd[0] = 25;
sEnd[1] = 30;
sEnd[2] = 29;
sEnd[3] = 28;

// start the Ethernet, UDP, Serial and I2C:
Ethernet.begin(mac,ip);
Udp.begin(localPort);
Wire.begin();
Serial.begin(9600);
//Serial.println("Initializing SD card...");
// On the Ethernet Shield, CS is pin 4. It's set as an output by default.
// Note that even if it's not used as the CS pin, the hardware SS pin
// (10 on most Arduino boards, 53 on the Mega) must be left as an output
// or the SD library functions will not work.
pinMode(10, OUTPUT);

SD.begin(4);


//////////////////////////////////////////////////
 // open file and get values
myFile = SD.open("TEMPROG1.txt");
if (myFile) {
  //Serial.println("TEMPROG1.txt:");

  // read from the file until there's nothing else in it:
  byte index = 0;
  while (myFile.available()) {

    tempSetString[index] = myFile.read();
    index++;
    //Serial.println("reading..");
  }
  // close the file:
  myFile.close();
  /*
  if(index != 3){
    Serial.println("SD data error");
  }
  else{
    setTemp = atoi(tempSetString);

    }
  */
  setTemp = atoi(tempSetString);
  }
//////////////////////////////////////////////////
// open file and get values
myFile = SD.open("CHPROG1.txt");
if (myFile) {
  //Serial.println("CHPROG1.txt:");

  // read from the file until there's nothing else in it:
  byte index = 0;
  while (myFile.available()) {

    newonOffString[index] = myFile.read();
    index++;
    //Serial.println("reading..");
  }
  // close the file:
  myFile.close();
  /*
  if(index != 47){
    Serial.println("SD data error");
  }
  else{
    for(index = 0; index < 48; index++){
      onOffString[index] = newonOffString[index];

    }
  */
  for(index = 0; index < 48; index++){
      onOffString[index] = newonOffString[index];
  }


  //////////////////////////////////////////////////////
  // set the initial time here ONLY USE GMT!!:
  // DS3231 seconds, minutes, hours, day, date, month, year
```

```
    //setDS3231time(0,14,22,2,17,8,15);
  }
}

// End of setup()


//Functions start here

/////////////////////////////////////////////////////////////////////////////////

// Convert normal decimal numbers to binary coded decimal
byte decToBcd(byte val)
{
  return( (val/10*16) + (val%10) );
}

//////////////////////////////////////////////////////////////////////////////////

// Convert binary coded decimal to normal decimal numbers
byte bcdToDec(byte val)
{
  return( (val/16*10) + (val%16) );
}

//////////////////////////////////////////////////////////////////////////////

/*
 // doesn't work if parameters are variables - why?!!
void setDS3231time(byte second, byte minute, byte hour, byte dayOfWeek, byte
dayOfMonth, byte month, byte year)
{
  // sets time and date data to DS3231
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // set next input to start at the seconds register
  Wire.write(decToBcd(second)); // set seconds
  Wire.write(decToBcd(minute)); // set minutes
  Wire.write(decToBcd(hour)); // set hours
  Wire.write(decToBcd(dayOfWeek)); // set day of week (1=Sunday, 7=Saturday)
  Wire.write(decToBcd(dayOfMonth)); // set date (1 to 31)
  Wire.write(decToBcd(month)); // set month
  Wire.write(decToBcd(year)); // set year (0 to 99)
  Wire.endTransmission();
}
*/
////////////////////////////////////////////////////////////////////////////////////

void readDS3231time(byte *second,
byte *minute,
byte *hour,
byte *dayOfWeek,
byte *dayOfMonth,
byte *month,
byte *year)
{
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // set DS3231 register pointer to 00h
  Wire.endTransmission();
  Wire.requestFrom(DS3231_I2C_ADDRESS, 7);
  // request seven bytes of data from DS3231 starting from register 00h
  *second = bcdToDec(Wire.read() & 0x7f);
  *minute = bcdToDec(Wire.read());
  *hour = bcdToDec(Wire.read() & 0x3f);
  *dayOfWeek = bcdToDec(Wire.read());
  *dayOfMonth = bcdToDec(Wire.read());
  *month = bcdToDec(Wire.read());
  *year = bcdToDec(Wire.read());
}



/////////////////////////////////////////////////////////////////////////////////

//function to extract values from onOffString and convert to minutes
int getTimes(byte index) {
  char selectorString[3];
  selectorString[0] = onOffString[index];
  index++;
  selectorString[1] = onOffString[index];
  int result = atoi(selectorString);
  index++;
  index++;
  selectorString[0] = onOffString[index];
  index++;
  selectorString[1] = onOffString[index];
  result = result*60 + atoi(selectorString);
  return result;
}


/////////////////////////////////////////////////////////////////////////
```

```
//function to extract values from clockSetString
//see function "adjClock()"
byte getClock(byte index) {
  char selectorString[3];
  selectorString[0] = clockSetString[index];
  index++;
  selectorString[1] = clockSetString[index];
  int i = atoi(selectorString);
  byte result = (byte) i;

  return result;
}
////////////////////////////////////////////////////////////////////////

//function to return the two switch positions coded to numbers 0 to 8
//see documentation / circuit diagram for explanation
//analog val for continuous is >500
//analog val for off is < 50
//analog value for timed is somewhere in between 50 and 500 (approx 317)

byte getSwitchPositions() {
  int valHeat = analogRead(0);
  int valWater = analogRead(1);
  if(valHeat < 50){
    valHeat = 2; //off
  }
  else if(valHeat > 500){
    valHeat = 3; // continuous
  }
  else {
    valHeat = 1; // timed
  }
  if(valWater < 50){
    valWater = 2;
  }
  else if(valWater > 500){
    valWater = 3;
  }
  else {
    valWater = 1;
  }

  return valWater * 3 + valHeat - 4;
}

////////////////////////////////////////////////////////////////////////////
// determine whether it's BST or GMT
boolean isItSummer(byte year, byte month, byte day, byte hour) {
  boolean summer = false;
  year = year - 15; // list of dates starts in 2015, array index starts at 0
  byte startDate = sStart[year];
  byte endDate = sEnd[year];

  if(month > 3 && month < 10) {
    summer = true;
  }
  if(month == 3 && day > startDate) {
    summer = true;
  }
  if(month == 3 && day == startDate && hour > 1){
    summer = true;
  }
  if(month == 10 && day < endDate){
    summer = true;
  }
  if(month == 10 && day == endDate && hour < 2){
    summer = true;
  }
  return summer;

}
//////////////////////////////////////////////////////////////////////////

int getTemp102(){
  byte firstbyte, secondbyte; //these are the bytes we read from the TMP102 temperature registers
  int val; /* an int is capable of storing two bytes, this is where we "chuck" the two bytes together. */

  float convertedtemp; /* We then need to multiply our two bytes by a scaling factor, mentioned in the datasheet. */

  //float correctedtemp;
  // The sensor overreads? I don't think it does!


  /* Reset the register pointer (by default it is ready to read temperatures)
   You can alter it to a writeable register and alter some of the configuration -
   the sensor is capable of alerting you if the temperature is above or below a specified threshold. */

  Wire.beginTransmission(TMP102_I2C_ADDRESS); // start talking to sensor
  Wire.write(0x00);
  Wire.endTransmission();
  Wire.requestFrom(TMP102_I2C_ADDRESS, 2);
  Wire.endTransmission();


  firstbyte      = (Wire.read());
  /*read the TMP102 datasheet - here we read one byte from
   each of the temperature registers on the TMP102*/
  secondbyte     = (Wire.read());
  /*The first byte contains the most significant bits, and
   the second the less significant */
  val = firstbyte;
  if ((firstbyte & 0x80) > 0) {
    val |= 0x0F00;
  }
  val <<= 4;
  /* MSB */
  val |= (secondbyte >> 4);
  // LSB is ORed into the second 4 bits of our byte.

  convertedtemp = val*0.625; // temp x 10
  //correctedtemp = convertedtemp - 0;   //should be 5 according to playground author




  int temp = (int)convertedtemp;
  return temp;
}

/////////////////////////////////////////////////////////////////////////////
//function incorporates a thermostatic function

boolean thermostat(int targetT){
  boolean heating;

  int tmp = getTemp102();
  //Serial.println(tmp);
  if(tmp >= (targetT + hysteresis)){
    heating = false;
  }
  if(tmp < (targetT - hysteresis)){
    heating = true;
  }
  return heating;
}

//////////////////////////////////////////////////////////////////////////////
// adjusts clock
void adjClock(){
  byte minutes = getClock(0);
  byte hours = getClock(3);
  byte day = getClock(6);
  byte date = getClock(9);
  byte month = getClock(12);
  byte year = getClock(15);
 /*
  Serial.println(minutes);
  Serial.println(hours);
  Serial.println(day);
  Serial.println(date);
  Serial.println(month);
  Serial.println(year);
 */

  minutes = minutes/10 * 16 + minutes % 10;
  hours = hours /10 * 16 + hours % 10;
  day = day / 10 * 16 + day % 10;
  date = date / 10 * 16 + date % 10;
  month = month / 10 * 16 + month % 10;
  year = year / 10 * 16 + year % 10;

  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // set next input to start at the seconds register
  Wire.write(0); // set seconds
  Wire.write(minutes); // set minutes
  Wire.write(hours); // set hours
  Wire.write(day); // set day of week (1=Sunday, 7=Saturday)
  Wire.write(date); // set date (1 to 31)
  Wire.write(month); // set month
  Wire.write(year); // set year (0 to 99)
  Wire.endTransmission();




  //Serial.println(minutes);
  //setDS3231time(0, minutes, hours, day, date, month, year);
  //setDS3231time(0, minutes, 22, 6, 14, 8, 15);
}

//////////////////////////////////////////////////////////////////////////////
```

```
void loop() {

  int tempNow = getTemp102();
  Serial.print("T");
  Serial.print(tempNow);
  //Serial.print(" deg C");
  itoa(tempNow, tNowStr, 10);
  //Serial.print(tNowStr);
  // Here is where the progam checks to see if data is being sent
  // and sends back status data

  // if there's data available, read a packet
  int packetSize = Udp.parsePacket();

  if(packetSize)
  {
    //Serial.print("Received packet of size ");
    //Serial.println(packetSize);
    //Serial.print("From ");
    //sndRec = true;
    IPAddress remote = Udp.remoteIP();
    for (byte i =0; i < 4; i++)
    {
      Serial.print(remote[i], DEC);
      if (i < 3)
      {
        Serial.print(".");
      }
    }
    Serial.print(",P");
    Serial.println(Udp.remotePort());

    if(packetSize == 47 || packetSize == 17 || packetSize == 4 || packetSize == 3 || packetSize == 2){

      // read the packet into packetBufffer
      Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);
      //Serial.println("Contents:");
      Serial.println(packetBuffer);
      //get new timer on/off times and save to SD card
      if(packetSize == 47){

      //save to SD card
      SD.remove("CHPROG1.txt");    //first delete previous file

        myFile = SD.open("CHPROG1.txt", FILE_WRITE);
      if (myFile) {
        myFile.print(packetBuffer);
        //int numChars;
        //numChars = myFile.print(packetBuffer);
        //Serial.write("number of characters written is ");
        //Serial.println(numChars);
        myFile.close();
        for (byte x = 0; x<48; x++){
          onOffString[x] = packetBuffer[x];
        }


      }
       /*
      else {
        Serial.println("SD - failed to write data!");
        //add error to UDP transmission
      }
      */
     }

     ////////////////////////////////////////////////////////////
     //get new clock settings
     //format: mm,hh,dd,dd,mm,yy
     if(packetSize == 17){
       for(byte x = 0; x < 18; x++){
         clockSetString[x] = packetBuffer[x];
       }
       //Serial.println(clockSetString);
       //already printed!
      adjClock();
     }
     //////////////////////////////////////////////////////////

     //reset Arduino
     if(packetSize == 4){
      digitalWrite(17, HIGH);  // thats all folks!
     }

     ///////////////////////////////////////////////////////////
     //get new thermostat setting
     //format: ttt (TdegC x 10)
     if(packetSize == 3){
       for(byte x = 0; x < 4; x++){
         tempSetString[x] = packetBuffer[x];
       }
      setTemp = atoi(tempSetString);
     // atoi(packetBuffer) ?

      //save to SD card
      SD.remove("TEMPROG1.txt");    //first delete previous file

        myFile = SD.open("TEMPROG1.txt", FILE_WRITE);
      if (myFile) {

        myFile.print(tempSetString);

        myFile.close();
        //for (byte x = 0; x<48; x++){
          //onOffString[x] = packetBuffer[x];
        //}


      }

      /*
      else {
        Serial.println("SD - failed to write data!");
        //add error to UDP transmission
      }

      */
     }
////////////////////////////////////////////////////////////////////////
     //respond to advance command
     if(packetSize == 2){

      if(packetBuffer[0] == 'Y'){
        advanceH = true;
      }
      else{
        advanceH = false;
      }

      if(packetBuffer[1] == 'Y'){
        advanceW = true;
      }
      else{
        advanceW = false;
      }

     }

    }
    // send a reply, to the IP address and port that sent us the packet we received
    int switches = getSwitchPositions();
    char swStr[7];
    itoa(switches, swStr, 10);
    char tNowStr[7];
    itoa(tempNow, tNowStr, 10);
    char setTempStr[7];
    itoa(setTemp, setTempStr, 10);
    char advCodeStr[7];
    itoa(advCode, advCodeStr, 10);


    char timStr[7];
    itoa(tim, timStr, 10);


    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
    //Udp.write("*");
    Udp.write(timStr);
    Udp.write("/");
    Udp.write(onOffString);
    Udp.write("T");
    Udp.write(tNowStr);
    Udp.write("S");
    Udp.write(swStr);
    Udp.write("A");
    Udp.write(advCodeStr);
    Udp.write("t");
    Udp.write(setTempStr);

    //should develop this to report status, errors etc
    Udp.endPacket();

  }

  // End of section checking whether data is being sent
  ////////////////////////////////////////////////////////////////////


  //displayTime();
  Serial.println();
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
  // retrieve data from DS3231
  readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);



  boolean summer;
  summer = isItSummer(year,month,dayOfMonth,hour);
  if(summer == true){


    strcpy(gmtBst, "S");

    if(hour == 23){
      hour = 0;
    }
    else {
      hour = hour + 1;
    }

  }
  else{

    strcpy(gmtBst, "G");

  }
  Serial.print(gmtBst);
  Serial.print(hour);
  Serial.print(":");
  Serial.print(minute);
  Serial.println();
  Serial.println(onOffString);

  tim = hour*60 + minute;
  //itoa(tim, timStr, 10);

  //char hourStr[7];
  itoa(hour, hourStr, 10);
  itoa(dayOfMonth, dayStr, 10);
  itoa(month, monthStr, 10);
  //char minuteStr[7];
  //itoa(minute, minuteStr, 10);
  /*
  if(sndRec){
    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
    Udp.write(" T: ");
    Udp.write(hourStr);
    Udp.write(":");
    Udp.write(minuteStr);
    Udp.write(" ");
    Udp.write(gmtBst);
    Udp.endPacket();
    sndRec = false;
  }
  */
  /////////////////////////////////////////////////////////////////

  //data log section
  //data logged every hour, one minute past hour
  if(minute == 1 && dataAdded == false){
//save to SD card


        myFile = SD.open("DATALOG1.txt", FILE_WRITE);

      if (myFile) {
        myFile.print(hour);
        myFile.print(":");
        myFile.print(dayStr);
        myFile.print(":");
        myFile.print(monthStr);
        myFile.print(":");
        myFile.print(tNowStr);
        myFile.print(",");


        }

        dataAdded = true;
      /*
       if(!myFile) {
        Serial.println("SD - DLog failed to write!");
        //add error to UDP transmission
      }
      */

    }
    myFile.close();

  if(minute != 1){
    dataAdded = false;
  }
```

```
//////////////////////////////////////////////////////////////
//Main logic for interaction with hardware starts here!

// ensures valve is deactivated at end of day

if(tim == 0){
  digitalWrite(16, HIGH);
}
else{
  digitalWrite(16, LOW);
}

//initialise variables holding on/off times


int waterOn1 = getTimes(0);
int waterOff1 = getTimes(6);
int waterOn2 = getTimes(12);
int waterOff2 = getTimes(18);

int heatingOn1 = getTimes(24);
int heatingOff1 = getTimes(30);
int heatingOn2 = getTimes(36);
int heatingOff2 = getTimes(42);

digitalWrite (2, LOW);        //advance LED
// get switch positions
// if off or cont, LEDs will flash during a timed period set in memory
int valHeat = analogRead(0);
int valWater = analogRead(1);

if(valHeat < 50 || valHeat > 500)     //off or cont
{

  digitalWrite(5,LOW);      //heat LED off - LED will flash during an "on" timed period
  delay(200);
}
if(valWater < 50 || valWater > 500)   //off or cont
{

  digitalWrite(3,LOW);    //water LED off - LED will flash during an "on" timed period
  delay(200);
}

//////////////////////////////////////////////////////////////

//Check to see if advance button (heating) is pressed and
//check times to see if boiler should be on
//set various boolean flags accordingly viz. heat1, heat2 advanceH, oldChangeH and newChangeH

byte val = digitalRead(7);    //heat advance button
delay(100);

if (val == LOW){
  advanceH = true;

}
oldChangeH = newChangeH; //used to detect status chages due to the timer
                         //to decide if advance is valid

if (tim >= heatingOn1 && tim < heatingOff1)
{
   heat1 = true;
}
else
{
  heat1 = false;
}

if (tim >= heatingOn2 && tim < heatingOff2)
{
   heat2 = true;
}
else
{
  heat2 = false;
}

///////////////////////////////////////////////////////////////////
//Do the same for water

val = digitalRead(6);   //water advance button
delay(100);
if (val == LOW){
  advanceW = true;

}
oldChangeW = newChangeW;
```

```
if (tim >= waterOn1 && tim < waterOff1)
{
   water1 = true;
}
else
{
  water1 = false;
}
if (tim >= waterOn2 && tim < waterOff2)
{
   water2 = true;
}
else
{
  water2 = false;
}

////////////////////////////////////////////////////
//Work out if heat should be on according to timed periods and advance

if (heat1 == true || heat2 == true)
{
   heat = true;
}
else
{
  heat = false;
}
newChangeH = heat;

if (newChangeH != oldChangeH)       //on/off status has changed
{
  advanceH = false;
}
if (advanceH)
{
  heat = !heat;     //if advance is true, negate the boiler state set by the state of the timer
}

// check to see if boiler should be on comparing temp sensor against target temp

if (heat)
{
  digitalWrite(5, HIGH); // heat LED
  boolean thermo = thermostat(setTemp);
  if(thermo){         // if temperature below target less hysteresis
    digitalWrite(8,HIGH); // heat relay on
    Serial.println("ON");
   }

  if(!thermo) {
    digitalWrite(8, LOW); // heat relay off
    Serial.println("OFF");
  }

}
if(!heat)
{
  digitalWrite(8, LOW); // heat relay off
  digitalWrite(5, LOW); // heat LED off
}
///////////////////////////////////////////////
//Similarly for water except that, with regard to temperature,
//the themostat is wired into the boiler circuitry
//and not subject to computer control

if (water1 == true || water2 == true)
{
   water = true;
}

else
{
  water = false;
}
newChangeW = water;
//check to see if advance is still valid
if (newChangeW != oldChangeW)
{
  advanceW = false;
}
if (advanceW)
{
  water = !water;

}
if (water)
{
  digitalWrite(9, HIGH);
  digitalWrite(3, HIGH);
}
```

```
  else
  {
    digitalWrite(9, LOW);
    digitalWrite(3, LOW);
  }

  if (advanceH || advanceW)
  {
    digitalWrite (2, HIGH);
  }

//encode the switch positions ready to broadcast over the network

    advCode = 0;
    if(advanceH){
    advCode = 1;
  }
    if(advanceW){
    advCode = 2;
  }
    if(advanceH && advanceW){
    advCode = 3;
  }



}

//end of loop
//and program!
```