Pulso corto de corriente continua

Si se aplica corriente continua sin alternar las polaridades, la excitación no debe superar los 50 ms y la lectura debe tomarse entre 100 y 200 µs después de que se aplica la excitación. Si se retrasa la lectura más de 200 µs con una fuente de corriente continua, se generará una compensación de carga en el sensor, lo que cambiará la resistencia medida y reducirá la vida útil de los electrodos. Después de la lectura, los cables deben ponerse en cortocircuito entre sí o el lado alimentado debe ponerse en cortocircuito a tierra durante 30 segundos para eliminar cualquier potencial de corriente continua acumulado en el sensor.

Aislamiento del sensor

Los dispositivos que leen sensores WATERMARK deben aislar el circuito de lectura del sensor de cualquier conexión a tierra. Los dispositivos alimentados por la red eléctrica necesitarán aislamiento por transformador para evitar un bucle a tierra a través de equipos cercanos, y cualquier línea de comunicación que se pretenda conectar permanentemente a otro dispositivo conectado a tierra requerirá aislamiento óptico. Si existe un bucle a tierra, las lecturas serán erróneas y la fuga de corriente puede destruir rápidamente los electrodos del sensor. Los dispositivos alimentados por batería están más aislados por naturaleza, pero las conexiones a tierra para protección contra rayos, etc., pueden causar el mismo problema.
Además del aislamiento general del dispositivo respecto de la tierra, al leer varios sensores, el circuito debe estar diseñado para aislar los sensores entre sí. El suelo húmedo en el que se instalan los sensores crea una

ruta conductora común entre ellos. En efecto, sin aislamiento, un dispositivo puede leer de forma parcial o total entre electrodos de diferentes sensores en lugar de entre electrodos dentro de cada sensor.
Los sensores deben recibir alimentación de forma individual y la conexión a tierra debe estar aislada entre un sensor y otro. Incluso si se alimentan de forma individual, no se puede permitir una ruta de conexión a tierra abierta hacia otro sensor que no reciba alimentación. Esto se logra mejor utilizando multiplexores para abrir y cerrar los canales adecuados.

Precaución: Dado que los sensores WATERMARK proporcionan una ruta directa desde tierra a cualquier dispositivo, se debe incluir protección mediante diodos Zener o TVS en todas las líneas de sensores. En aras de la simplicidad, no se incluyen en ninguno de los ejemplos que se incluyen a continuación, pero se recomiendan encarecidamente.

Calibración

El valor **R_x** se basa en la cantidad de humedad dentro del sensor, pero es necesaria una calibración adicional para relacionar la resistencia con la tensión del agua del suelo.

```
kPa= (-3,213*R - 4,093) / (1 - 0,009733*R - 0,01205*T)
```

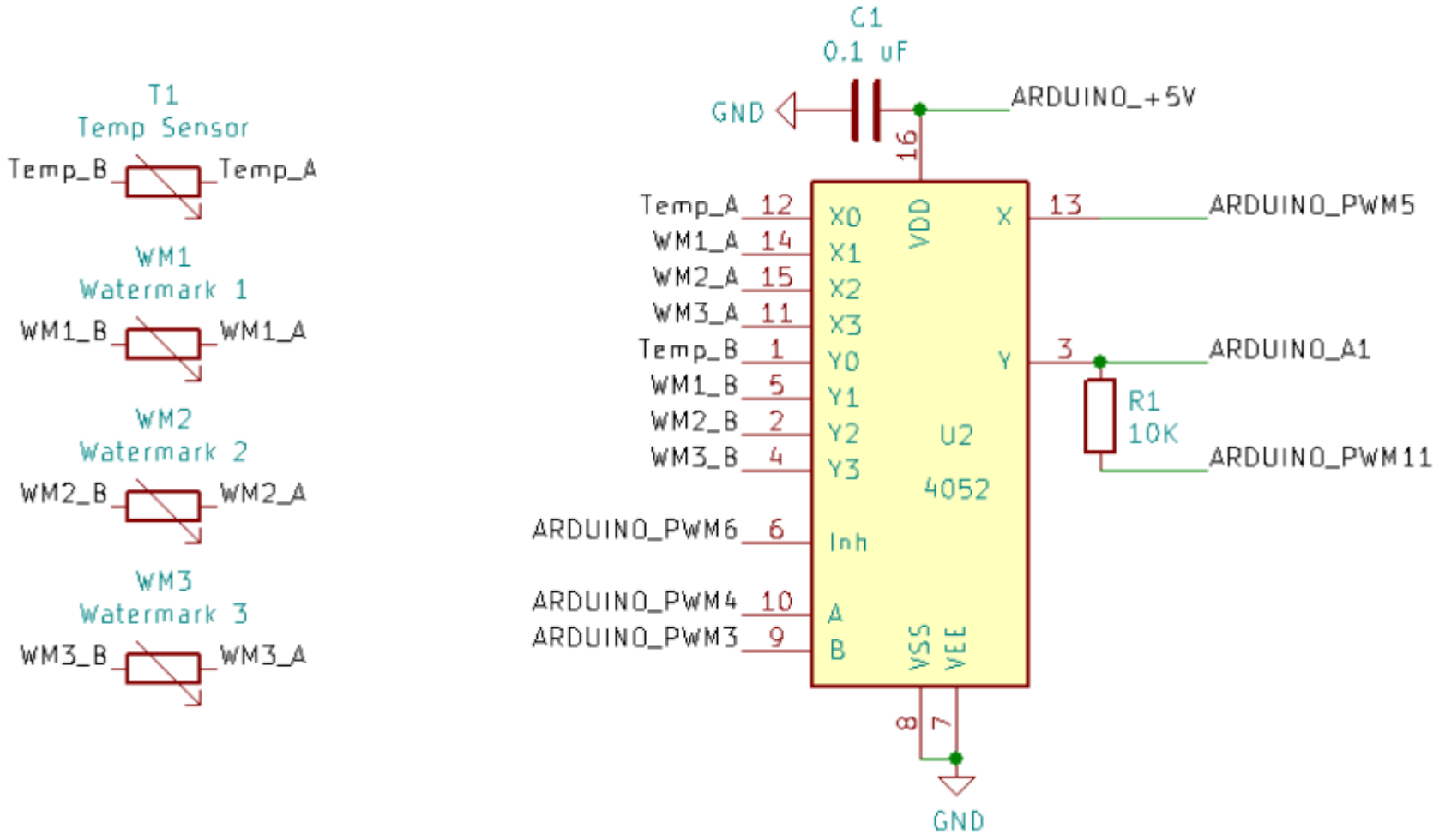
donde **R** es la `resistencia en kOhms` y **T** es la `temperatura en grados centígrados`. Esta calibración cubre el rango de 10 a 100 kPa. Normalmente se utilizan extrapolaciones lineales por debajo de 10 y por encima de 100 kPa. Tenga en cuenta que un sensor completamente húmedo mide 550 ohmios.

La resistencia medida en el sensor se ve afectada por la temperatura. Se puede utilizar una temperatura predeterminada de 24 °C en ausencia de datos de temperatura, por lo que se puede utilizar una entrada de sensor de temperatura para aumentar la precisión.

Ejemplo de Arduino

El siguiente ejemplo `Figura 1` lee un sensor de temperatura del suelo y tres sensores WATERMARK utilizando un multiplexor de canal doble 74HC4052. Se utilizan dos salidas (`PWM3 y PWM4`) para controlar los multiplexores, mientras que `PWM5 y PWM11` se alternan como potencia de excitación y GND. `A1` se utiliza para la medición analógica.
Se conecta un terminal a una entrada del grupo **X** y el otro a una del grupo **Y** para crear una diferencia de potencial a través del sensor, **permitiendo medir su resistencia**.

Figura 1: Circuito de tres sensores de potencial mátrico (watermark) con uno de temperatura y el multiplexor 74HC4052.



Pasos iniciales

1. `PWM 6` se establece en BAJO para habilitar el multiplexor
2. `PWM 3 y 4` se establecen en el canal apropiado

Lectura de los sensores

Dirección 1:

1. `PWM 5` se establece en ALTO para la excitación del sensor
2. `A1` se lee para medir la resistencia
3. `PWM 5` se establece en BAJO para finalizar la lectura

Dirección 2:

1. `PWM 11` se establece en ALTO para excitación
2. `A1` se lee para medir la resistencia
3. `PWM 11` se establece en BAJO para finalizar la lectura

`PWM 3 y 4` se configuran en el siguiente canal y la lectura se repite para el **siguiente sensor**.

Apagado:

`PWM 6` se configura en ALTO para deshabilitar el multiplexor

Calcular resistencia

donde `SenVWM1` es el voltaje analógico medido en la primera dirección, `SenVWM2` es el mismo en la segunda dirección y `R_x` es el valor de `R1`:

```
double WM_ResistanceA = (Rx * (SupplyV - SenVWM1) / SenVWM1); //voltage divider

double WM_ResistanceB = Rx * SenVWM2 / (SupplyV - SenVWM2);  // reverse

double WM_Resistance = ((WM_ResistanceA + WM_ResistanceB) / 2); //average the two
directions
```

Información adicional

- Pines de control del Arduino (p.ej. `PWM3, PWM4`) conectados a las entradas de selección A y B del 74HC4052.
- Pin ENABLE (EN) del 74HC4052 conectado a un pin digital adicional (p.ej. `PWM6`) para **habilitar o deshabilitar la lectura**.
- Una o dos líneas de excitación (p.ej. `PWM5 y PWM11`) que **alternen la polaridad** (pseudo-AC o pulsos cortos).
- Entrada analógica (`A1`) como punto de medición de voltaje.
  Cuando hay 2 o más sensores, se emplea un 74HC4052 (u otro multiplexor analógico) que permite aislar eléctricamente cada sensor en el momento de lectura y alternar entre ellos para evitar lecturas cruzadas:

1. La señal de “excitación” (pin digital) se dirige a las entradas comunes del multiplexor, y las salidas `X0-Xx` e `Y0-Yx` se conectan a los sensores.
2. De manera similar, el pin ANALÓGICO (`A1` en Arduino) se conecta a la salida analógica común del multiplexor.
3. Cada sensor `Watermark 200SS` tiene sus protecciones (diodos, resistencia de 100 Ω, capacitor 0,1 µF) en la línea.

Reversión de polaridad (pseudo-AC)

Se logra con dos pines digitales del Arduino (por ejemplo, `PWM5 y PWM11`) que se alternan como “HIGH” y “GND” mientras el otro se sitúa en alta impedancia (o en LOW), y viceversa.

Convertir a centibares (kPa)

Donde `res` es la **resistencia medida**, `TC` es la **temperatura del suelo** en **C** y `cF` es un **factor de calibración** opcional:

```
int myCBvalue(int res, float TC, float cF) {   //conversion of ohms to CB

    int WM_CB;
    float resK = res / 1000.0;
    float tempD = 1.00 + 0.018 * (TC - 24.00);
```

```
    if (res > 550.00) { //if in the normal calibration range
        if (res > 8000.00) { //above 8k
            WM_CB = (-2.246 - 5.239 * resK * (1 + .018 * (TC - 24.00)) - .06756 * resK *
resK * (tempD * tempD)) * cF;
        } else if (res > 1000.00) { //between 1k and 8k
            WM_CB = (-3.213 * resK - 4.093) / (1 - 0.009733 * resK - 0.01205 * (TC)) * cF
;
        } else { //below 1k
            WM_CB = (resK * 23.156 - 12.736) * tempD;
        }
    } else { //below normal range but above short (new, unconditioned sensors)
        if (res > 300.00)  {
            WM_CB = 0.00;
        }
        if (res < 300.00 && res >= short_resistance) { //wire short
            WM_CB = short_CB; //240 is a fault code for sensor terminal short
            Serial.print("Sensor Short WM \n");
        }
    }
    if (res >= open_resistance || res==0) {
        WM_CB = open_CB; //255 is a fault code for open circuit or sensor not present
    }
    return WM_CB;
}
```

Códigos de esta muestra

```
#include <math.h>

// Code tested on Arduino UNO R3
// Purpose of this code is to demonstrate valid WM reading code, circuitry and excitation using a voltage divider and "psuedo-ac" method
// Sensor to be energized by digital pin 11 or digital pin 5, alternating between HIGH and LOW states
//As a simplified example, this version reads one sensor only and assumes a default temperature of 24C.
//NOTE: the 0.09 excitation time may not be sufficient depending on circuit design, cable lengths, voltage, etc. Increase if necessary to get accurate readings, do not exceed 0.2
//NOTE: this code assumes a 10 bit ADC. If using 12 bit, replace the 1024 in the voltage conversions to 4096

#define num_of_read 1 // number of iterations, each is actually two reads of the sensor (both directions)
const int Rx = 10000;  //fixed resistor attached in series to the sensor and ground...the same value repeated for all WM and Temp Sensor.
const long default_TempC = 24;
const long open_resistance = 35000; //check the open resistance value by replacing sensor with an open and replace the value here...this value might vary slightly with circuit components
const long short_resistance = 200; // similarly check short resistance by shorting the sensor terminals and replace the value here.
```

```
const long short_CB = 240, open_CB = 255 ;
const int SupplyV = 5; // Assuming 5V output for SupplyV, this can be measured and
replaced with an exact value if required
const float cFactor = 1.1; //correction factor optional for adjusting curve, 1.1
recommended to match IRROMETER devices as well as CS CR1000
int i, j = 0, WM1_CB = 0;
float SenV10K = 0, SenVWM1 = 0, SenVWM2 = 0, ARead_A1 = 0, ARead_A2 = 0, WM_Resistance =
0, WM1_Resistance = 0 ;

void setup()
{
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
  // initialize the pins, 5 and 11 randomly chosen. In the voltage divider circuit example
in figure 1(www.irrometer.com/200ss.html), pin 11 is the "Output Pin" and pin 5 is the
"GND".
  // if the direction is reversed, the WM1_Resistance A and B formulas would have to be
swapped.
  pinMode(5, OUTPUT);
  pinMode(11, OUTPUT);
  //set both low
  digitalWrite(5, LOW);
  digitalWrite(11, LOW);

  delay(100);   // time in milliseconds, wait 0.1 minute to make sure the OUTPUT is
assigned
}
```

```
void loop()
{
  while (j == 0)
  {
    //Read the first Watermark sensor

    WM1_Resistance = readWMsensor();
    WM1_CB = myCBvalue(WM1_Resistance, default_TempC, cFactor);

    //*****************output*************************************

    Serial.print("WM1 Resistance(Ohms)= ");
    Serial.print(WM1_Resistance);
    Serial.print("\n");
    Serial.print("WM1(cb/kPa)= ");
    Serial.print(abs(WM1_CB));
    Serial.print("\n");

    delay(200000);
    //j=1;
  }
}

//conversion of ohms to CB
int myCBvalue(int res, float TC, float cF) {   //conversion of ohms to CB
  int WM_CB;
```

```
  float resK = res / 1000.0;
  float tempD = 1.00 + 0.018 * (TC - 24.00);

  if (res > 550.00) { //if in the normal calibration range
    if (res > 8000.00) { //above 8k
      WM_CB = (-2.246 - 5.239 * resK * (1 + .018 * (TC - 24.00)) - .06756 * resK * resK *
(tempD * tempD)) * cF;
    } else if (res > 1000.00) { //between 1k and 8k
      WM_CB = (-3.213 * resK - 4.093) / (1 - 0.009733 * resK - 0.01205 * (TC)) * cF ;
    } else { //below 1k
      WM_CB = (resK * 23.156 - 12.736) * tempD;
    }
  } else { //below normal range but above short (new, unconditioned sensors)
    if (res > 300.00)  {
      WM_CB = 0.00;
    }
    if (res < 300.00 && res >= short_resistance) { //wire short
      WM_CB = short_CB; //240 is a fault code for sensor terminal short
      Serial.print("Sensor Short WM \n");
    }
  }
  if (res >= open_resistance || res==0) {

    WM_CB = open_CB; //255 is a fault code for open circuit or sensor not present

  }
  return WM_CB;
```

```
}

//read ADC and get resistance of sensor
float readWMsensor() {  //read ADC and get resistance of sensor

  ARead_A1 = 0;
  ARead_A2 = 0;

  for (i = 0; i < num_of_read; i++) //the num_of_read initialized above, controls the
number of read successive read loops that is averaged.
  {

    digitalWrite(5, HIGH);   //Set pin 5 as Vs
    delayMicroseconds(90); //wait 90 micro seconds and take sensor read
    ARead_A1 += analogRead(A1); // read the analog pin and add it to the running total for
this direction
    digitalWrite(5, LOW);      //set the excitation voltage to OFF/LOW

    delay(100); //0.1 second wait before moving to next channel or switching MUX

    // Now lets swap polarity, pin 5 is already low

    digitalWrite(11, HIGH); //Set pin 11 as Vs
    delayMicroseconds(90); //wait 90 micro seconds and take sensor read
    ARead_A2 += analogRead(A1); // read the analog pin and add it to the running total for
this direction
    digitalWrite(11, LOW);      //set the excitation voltage to OFF/LOW
```

```
  }

  SenVWM1 = ((ARead_A1 / 1024) * SupplyV) / (num_of_read); //get the average of the
readings in the first direction and convert to volts
  SenVWM2 = ((ARead_A2 / 1024) * SupplyV) / (num_of_read); //get the average of the
readings in the second direction and convert to volts

  double WM_ResistanceA = (Rx * (SupplyV - SenVWM1) / SenVWM1); //do the voltage divider
math, using the Rx variable representing the known resistor
  double WM_ResistanceB = Rx * SenVWM2 / (SupplyV - SenVWM2);  // reverse
  double WM_Resistance = ((WM_ResistanceA + WM_ResistanceB) / 2); //average the two
directions
  return WM_Resistance;
}
```

Otro código de muestra

```
#include <math.h>

// Watermark is a resistive sensor, so in order to check the accuracy of the resistive
reading circuit a .....
// 10K resistor (1% precision) is used as a calibration reference only for prototype
purposes.
// Code tested on Arduino Uno R3
// Purpose of this code is to demonstrate valid WM reading code, circuitry and excitation.
// This program uses a modified form of Dr. Shocks 1998 calibration equation
```

```
// Circuit will use a dual channel multiplexer to isolate sensors and will alternate
polarity between the Sensor terminals for up to 4 resistive sensors( temperature/WM or
calibration resistor)
// MUX control A and B to be controlled by PWM P4 & P3
// Sensor to be energized by PWM 5 or PWM 11, alternating between HIGH and LOW states
// MUX  enable/ disable PWM 6(LOW: Enable/High: Disable)
// Analog read channels to be used during MUX active : A1
// Channel 0 of MUX used for the temp sensor, channel 1 for Watermark 1, Channel 2 for
Watermark 3 and and Channel 3 for Watermark 3
// In the absence of a Temp Sensor the program assigns 24 Degree C to temp for calibration
of soil moisture tension

//NOTE: this code assumes a 10 bit ADC. Replace 1024 with 4096 if using a 12 bit ADC.
//NOTE: the 0.09 excitation time may not be sufficient depending on circuit design, cable
lengths, voltage, etc. Increase if necessary to get accurate readings, do not exceed 0.2

//Truth table for MUX (B=PWM3 & A=PWM4)
//--B---A
//  0   0---- READ TEMPERATURE SENSOR
//  0   1---- READ WM1
//  1   0---- READ WM2
//  1   1---- READ WM3

// SET PWM5 or PWM11 HIGH/LOW ALTERNATELY TO EXCITE SENSOR

#define num_of_read 1 // number of iterations, each is actually two reads of the sensor
(both directions)
```

```
const int Rx = 10000;  //fixed resistor attached in series to the sensor and ground...the same value repeated for all WM and Temp Sensor.
const long default_TempC = 24;
const long open_resistance = 35000; //check the open resistance value by replacing sensor with an open and replace the value here...this value might vary slightly with circuit components
const long short_resistance = 200; // similarly check short resistance by shorting the sensor terminals and replace the value here.
const long short_CB = 240, open_CB = 255 ;
const int SupplyV = 5; // Assuming 5V output for SupplyV, this can be measured and replaced with an exact value if required
const float cFactor = 1.1; //correction factor optional for adjusting curves. Traditionally IRROMETER devices used a different reading method, voltage divider circuits often require this adjustment to match exactly.
int i, j = 0, WM1_CB = 0, WM2_CB = 0, WM3_CB = 0;
float SenV10K = 0, SenVTempC = 0, SenVWM1 = 0, SenVWM2 = 0, ARead_A1 = 0, ARead_A2 = 0, WM3_Resistance = 0, WM2_Resistance = 0, WM1_Resistance = 0, TempC_Resistance = 0, TempC = 0;
void setup()
{
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);

  // initialize the digital pins as outputs
  pinMode(3, OUTPUT);  //used for S0 control of the MUX 0-1
  pinMode(4, OUTPUT);  //used for S1 control of the MUX 0-1
  pinMode(5, OUTPUT);  //Sensor Vs or GND
```

```
  pinMode(6, OUTPUT);  //Enable disable MUX 0-1
  pinMode(11, OUTPUT);  //Sensor Vs or GND

  delay(100);   // time in milliseconds, wait 0.1 minute to make sure the OUTPUT is
assigned
}

void loop()
{
  while (j == 0)
  {

    //enable the MUX
    digitalWrite(6, LOW);

    //Read the temp sensor
    delay(100); //0.1 second wait before moving to next channel or switching MUX
    //address the MUX (channel 1)

    digitalWrite(3, LOW);
    digitalWrite(4, LOW);

    delay(10); //wait for MUX

    TempC_Resistance = readWMsensor();

    //Read the first Watermark sensor
```

```
    delay(100); //0.1 second wait before moving to next channel or switching MUX
    //address the MUX
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);

    delay(10); //wait for the MUX

    WM1_Resistance = readWMsensor();

    //Read the second Watermark sensor
    delay(100); //0.1 second wait before moving to next channel or switching MUX
    //address the MUX
    digitalWrite(3, HIGH);
    digitalWrite(4, LOW);

    delay(10); //wait for MUX

    WM2_Resistance = readWMsensor();

    //Read the third Watermark sensor
    delay(100); //0.1 second wait before moving to next channel or switching MUX
    //address the MUX
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);

    delay(10); //wait for the MUX
```

```
    WM3_Resistance = readWMsensor();

    delay(100); //0.1 second wait before moving to next channel or switching MUX

    digitalWrite(6, HIGH);   //Disable MUX 0-1 pair
    delay(100); //0.1 second wait before moving to next channel or switching MUX

    //convert the measured reistance to kPa/centibars of soil water tension
    TempC = myTempvalue(TempC_Resistance);
    WM1_CB = myCBvalue(WM1_Resistance, TempC, cFactor);
    WM2_CB = myCBvalue(WM2_Resistance, TempC, cFactor);
    WM3_CB = myCBvalue(WM3_Resistance, TempC, cFactor);

    //*****************output*************************************
    Serial.print("Temperature(C)= ");
    Serial.println(abs(TempC));
    Serial.print("\n");
    Serial.print("WM1 Resistance(Ohms)= ");
    Serial.println(WM1_Resistance);
    Serial.print("WM2 Resistance(Ohms)= ");
    Serial.println(WM2_Resistance);
    Serial.print("WM3 Resistance(Ohms)= ");
    Serial.println(WM3_Resistance);
    Serial.print("WM1(cb/kPa)= ");
    Serial.println(abs(WM1_CB));
    Serial.print("WM2(cb/kPa)= ");
    Serial.println(abs(WM2_CB));
```

```
    Serial.print("WM3(cb/kPa)= ");
    Serial.println(abs(WM3_CB));
    Serial.print("\n");

    delay(200000);
    //j=1;
  }
}

int myCBvalue(int res, float TC, float cF) {   //conversion of ohms to CB
  int WM_CB;
  float resK = res / 1000.0;
  float tempD = 1.00 + 0.018 * (TC - 24.00);

  if (res > 550.00) { //if in the normal calibration range
    if (res > 8000.00) { //above 8k
      WM_CB = (-2.246 - 5.239 * resK * (1 + .018 * (TC - 24.00)) - .06756 * resK * resK *
(tempD * tempD)) * cF;
    } else if (res > 1000.00) { //between 1k and 8k
      WM_CB = (-3.213 * resK - 4.093) / (1 - 0.009733 * resK - 0.01205 * (TC)) * cF ;
    } else { //below 1k
      WM_CB = (resK * 23.156 - 12.736) * tempD;
    }
  } else { //below normal range but above short (new, unconditioned sensors)
    if (res > 300.00)  {
      WM_CB = 0.00;
    }
```

```
    if (res < 300.00 && res >= short_resistance) { //wire short
      WM_CB = short_CB; //240 is a fault code for sensor terminal short
      Serial.print("Sensor Short WM \n");
    }
  }
  if (res >= open_resistance || res==0) {

    WM_CB = open_CB; //255 is a fault code for open circuit or sensor not present

  }
  return WM_CB;
}

float myTempvalue(float temp) {

    float TempC = (-23.89 * (log(TempC_Resistance))) + 246.00;
    if (TempC_Resistance < 0 || TempC_Resistance > 30000 )
    {
      Serial.print("Temperature Sensor absent, using default \n");
      TempC = 24.0;
    }
    return TempC;

}

float readWMsensor() {  //read ADC and get resistance of sensor
```

```
  ARead_A1 = 0;
  ARead_A2 = 0;

  for (i = 0; i < num_of_read; i++) //the num_of_read initialized above, controls the
number of read successive read loops that is averaged.
  {

    digitalWrite(5, HIGH);   //Set pin 5 as Vs
    delayMicroseconds(90); //wait 90 micro seconds and take sensor read
    ARead_A1 += analogRead(A1); // read the analog pin and add it to the running total for
this direction
    digitalWrite(5, LOW);      //set the excitation voltage to OFF/LOW

    delay(100); //0.1 second wait before moving to next channel or switching MUX

    // Now lets swap polarity, pin 5 is already low

    digitalWrite(11, HIGH); //Set pin 11 as Vs
    delayMicroseconds(90); //wait 90 micro seconds and take sensor read
    ARead_A2 += analogRead(A1); // read the analog pin and add it to the running total for
this direction
    digitalWrite(11, LOW);      //set the excitation voltage to OFF/LOW
  }

  SenVWM1 = ((ARead_A1 / 1024) * SupplyV) / (num_of_read); //get the average of the
readings in the first direction and convert to volts
  SenVWM2 = ((ARead_A2 / 1024) * SupplyV) / (num_of_read); //get the average of the
```

```
readings in the second direction and convert to volts

  double WM_ResistanceA = (Rx * (SupplyV - SenVWM1) / SenVWM1); //do the voltage divider
math, using the Rx variable representing the known resistor
  double WM_ResistanceB = Rx * SenVWM2 / (SupplyV - SenVWM2);  // reverse
  double WM_Resistance = ((WM_ResistanceA + WM_ResistanceB) / 2); //average the two
directions
  return WM_Resistance;
}
```