Electronic Supplementary Material (ESI) for Environmental Science: Water Research & Technology.
This journal is © The Royal Society of Chemistry 2016

Affordable, Flexible, and Modular: A Guide to Open-Source Membrane-Based Water Treatment Systems

Adam Slade, David Jassby*

Department of Chemical and Environmental Engineering, University of California – Riverside, Riverside, CA 92521, USA

**Corresponding author email: djassby@engr.ucr.edu*

Appendix A

Arduino Code

The following is the Arduino sketch code in its entirety. It can be modified to accept additional commands, such as LCD display, as necessary. The contents are to be saved in a plain-text **.ino* file and uploaded onto the Arduino hardware using the Arduino IDE.

A.1 Constants

```
// constants - for tuning
unsigned long dtmax = 1000000; // [us] - max sample collection time
unsigned long cutoff = 4294967295 - dtmax; // [us] - cutoff time
word smax = 500; // maximum number of samples to collect


// autonomous shutdown
unsigned long lastContact = 0; // [ms] - the last contact time
unsigned long timeout = 10000; // [ms] - max no-contact time
```

A.2 Setup

```
// arduino controller setup (runs once)
void setup()
{
    Serial.begin(500000); // 500 kbps baud rate
    delay(1000); // wait for communication
    SerialWrite(1,2,3); // report connection to PC
}
```

A.3 Loop

```
// arduino controller infinite loop (runs indefinitely)
void loop()
{
    // check for command
    if (Serial.available() >= 4){
        byte DataIn[] = { 0, 0, 0, 0 }; // {command, pin, data, checksum}
        Serial.readBytes(DataIn, 4);
        if (isValid(DataIn) == true){
            word result = DoWork(DataIn); // get result
            SerialWrite(DataIn[0],DataIn[1],result); // send result
        }
        else{
```

```
                        Serial.flush(); // clear corrupted data
                }
          lastContact = millis(); // record last PC communication
        }
        else{
          // check for timeout
          if((millis()-lastContact)>timeout){

           //shutdown everything
           for(int i=0;i<54;i++){
             digitalWrite(i,LOW);
           }
          }
          else{
           delay(1); // wait for command
          }
        }

}
```

A.4 Checksum

```
// generate and compare the checksum for incoming data
boolean isValid(byte DataIn[]){
        if (DataIn[3] == byte(ceil((255.0 + DataIn[0] + DataIn[1] + DataIn[2]) / 4.0))){
                return true;
```

```
    }
    else{
        return false;
    }
}

// generate the checksum for outgoing data
byte getCheckSum(byte a, byte b, byte c, byte d){
    return byte(ceil((255.0 + a + b + c + d) / 5.0));
}
```

A.5 Write Data

```
// add checksum and write data to port
void SerialWrite(byte command, byte pin, word val){

    // initialize output
    byte DataOut[] = { 0, 0, 0, 0, 0 }; // {command, pin, data, data, checksum}

    // assign values
    DataOut[0] = command;
    DataOut[1] = pin;

    // convert word to two bytes
    byte* w = (byte*)&val;
    DataOut[2] = w[0];
    DataOut[3] = w[1];

    // add checksum
    DataOut[4] = getCheckSum(DataOut[0], DataOut[1], DataOut[2], DataOut[3]);

    // write to serial port
    Serial.write(DataOut, 5);
}
```

A.6 Execute Command

```
// manipulate pins, based on input
word DoWork(byte DataIn[]){

  // data holders
    byte pin = DataIn[1];
    byte data = DataIn[2];
    long sum = 0;

  // execute command
    switch (DataIn[0]){
    case 1: // digital read
        if (digitalRead(pin) == HIGH){
            return 1;
        }
        else{
            return 0;
        }
        break;
case 2: // digital write
        if (data == 1){
                digitalWrite(pin, HIGH);
                return 1;
        }
        else{
                digitalWrite(pin, LOW);
                return 0;
        }
        break;
case 3: // analog read
        for (int i = 0; i < data; i++){
                sum += analogRead(pin);
        }
        return word(float(sum) / float(data));
        break;
case 4: // analog write (digital PWM)
        analogWrite(pin, data);
        return data;
        break;
case 5: // pin mode
        switch (data){
        case 1: // digital read
                pinMode(pin, INPUT);
```

```
            return 1;
            break;
        case 2: // digital write
            pinMode(pin, OUTPUT);
            return 0;
            break;
        default:
            // do nothing
            return 0;
            break;
        }
    case 6: // pulse frequency
        return getPulsesPerSecond(pin);
        break;
    }
}
```

A.7 Pulse Frequency

```
// return the pulse frequency for the selected pin
word getPulsesPerSecond(byte pin){

    // wait for overflow to occur, if pending
    while (micros() > cutoff){
        delay(1); // wait for overflow
    }
```

```
// initialize counters
word sum = 0; // detected signal changes
byte value = digitalRead(pin); // current value
byte last = value; // previous value

// timers
float elapsed;
unsigned long start = micros();
unsigned long maxmicros = start + dtmax; // max time to record samples

// count signal changes
while (true){

    // check for change
    value = digitalRead(pin);
    if (value != last){
        sum++;
        last = value;
    }

    // check for time
    if (micros()>=maxmicros){
        break;
    }
```

```
        // check for samples
        if (sum >= smax){
            break;
        }

    }

    // get elapsed time
    elapsed = float(micros() - start) / 1000000.0;

    // convert to frequency (pps)
    float freq = (float(sum) / float(elapsed)) / 2.0;
    return word(freq);
}
```