```
/*****************
This program was written by WS on 13/11/2018

This program is intended for Arduino UNO R3 or Leonardo R3
(or Olimexino, which
is a Leonardo clone). Edit #defines to set which one. If
using Arduino UNO R3 or
Leonardo R3, the different communication logic levels should
be respected. Arduino
is 5v, OPC-N3 is 3.3v. The OPC-N3 PIC is 5v tolerant on most
of the SPI pins, but
not the Slave Select pin. A simple solution is to use a
resistor pair to divide the
voltage on the SS pin appropriately. If an Olimexino device
is used, it has a
jumper switch to set it to either 5v or 3.3v. Set to 3.3v to
connect to OPC-N3
without need for logic level conversion.
The program is for use with OPC-N3 devices with standard
v1.17a firmware.

This program configures the Arduino to put an OPC-N3 through
1 minute cycles
taking one set of measurements per second during the 10s
active part of the cycle
and turning the fan and laser off for the remainder of the
cycle.
The data taken from the OPC-N3 via the SPI interface is
processed
and the processed data fed to the serial interface (the USB
port) so that it can be displayed on a computer using
suitable terminal software.
Usual serial port settings should be set in the terminal
software (8 data bits,
1 stop bit, no parity, no flow control). Set to baud rate to
match the value
defined in this program.

Adding more Slave Select pins will allow multiple OPC-N3
devices on one SPI bus.
Only one Slave Select pin should be active at any time.
*****************/


#include <SPI.h>
#include <avr/wdt.h>


#define FirmwareVer "OPC-N3-02(UNOr3)(res divider on SS)"

#define ArduinoUNO
//#define ArduinoLeonardo //(or Olimexino)

#define opSerial Serial
//#define opSerial Serial1 //(On Leonardo or Olimexino,
'Serial' is distinct from 'Serial1'. 'Serial' uses the USB
link and 'Serial1' uses the serial io pins D0 and D1. On Uno,
'Serial' uses both simultaneously.)
```

```
#define BaudRate 9600

#define SPI_OPC_busy 0x31
#define SPI_OPC_ready 0xF3

/*
SPI (on ICSP) pins on UNO-R3, Leonardo-R3 and Olimexino
1-MISO oo 2-+Vcc
3-SCK  oo 4-MOSI
5-Res  oo 6-Gnd

SPI pins for UNO-R3 only
19/D13 = SCK
18/D12 = MISO
17/D11 = MOSI
16/D10 = /SS
*/

unsigned long currentTime;
unsigned long cloopTime;
unsigned char SPI_in[86], SPI_in_index, ssPin_OPC;


void setup()
{
  wdt_reset(); //Reset watchdog timer
  wdt_enable(WDTO_8S); //Enable watchdog timer, countdown 8s
(max)

  //Set IOs
  //Set all the pins available for use as SS pins to outputs
and set HIGH
  for (unsigned char i=2;i<11;i++)
  {
    digitalWrite(i, HIGH); //Initiate pin HIGH
    pinMode(i, OUTPUT); //Set pin as output
  }

  delay(1000); //delay in case of noise on power connection.
Also allows OPC to boot up.

  //Start serial port
  opSerial.begin(BaudRate);

  #if defined (ArduinoLeonardo)
    if (opSerial == Serial)
    {
      //Wait until USB CDC port connects (only necessary with
Arduino Leonardo or Olimexino)
      while (!opSerial) wdt_reset(); //Reset watchdog timer
    }
  #endif

  PrintFirmwareVer(opSerial);

  // start the SPI library:
  SPI.begin(); //Enable SPI for OPC comms
```

```
  //Device #1 (ssPin_OPC = 10)
  ssPin_OPC = 10;
  wdt_reset(); //Reset watchdog timer
  InitDevice();
  wdt_reset(); //Reset watchdog timer
  //END Device #1

  PrintDataLabels(opSerial); //Print labels to serial port

  currentTime = millis();
  cloopTime = currentTime;
}


void PrintFirmwareVer (Stream &port)
{
  port.print(F("Datalogger firmware ver "));
  port.println(FirmwareVer);
}


void InitDevice (void)
{
  wdt_reset(); //Reset watchdog timer

  ReadOPCstring(0x10); //Get serialstr from OPC device
  ReadOPCstring(0x3F); //Get infostr from OPC device

  StartOPC(); //Switch on power to fan and laser
  wdt_reset(); //Reset watchdog timer
  ReadOPCconfig(opSerial); //Get Config data (bin boundaries
etc.) from OPC device
}


// Main Loop
void loop()
{
  wdt_reset(); //Reset watchdog timer

  // This is the main loop which should do the following:
  // Switch ON fan and laser
  // Get 10 histogram data sets, one per second (don't record
the first one)
  // Switch OFF fan and laser
  // Repeat every 60s

  currentTime = millis(); //millis count will reset on sketch
restart
  if (currentTime >= cloopTime)
  {
    cloopTime += 60000;  // Updates cloopTime

    wdt_reset(); //Reset watchdog timer

    //Device #1 (ssPin_OPC = 10)
      ssPin_OPC = 10;
```

```
      //Switch power ON to fan and laser
      StartOPC();

      wdt_reset(); //Reset watchdog timer

      //Get 10 histogram data sets (don't record the first
one as it will contain invalid data)
      unsigned long GetHistTime = millis(); //Set initial
GetHistTime
      for (byte i=0; i<10; i++)
      {
        delay(1000);
        ReadOPChist(); //Read OPC histogram data
        if (i != 0) {
          //Print time since start (millis() returns an
unsigned long of number of ms since program started. It wraps
around in ~50 days.)
          opSerial.print(millis());
          PrintData(opSerial); //Print data to serial
        }
        wdt_reset(); //Reset watchdog timer
      }

      //Switch power OFF to fan and laser
      StopOPC();
    //END Device #1

    opSerial.println("Waiting until next cycle");
  }
}


//Get string (serialstr or infostr) from OPC device
void ReadOPCstring (unsigned char SPIcommand)
{
  GetReadyResponse(SPIcommand);
  for (SPI_in_index=0; SPI_in_index<60; SPI_in_index++)
  {
    delayMicroseconds(10);
    SPI_in[SPI_in_index] = SPI.transfer(0x01); //Value of
outgoing byte doesn't matter
  }

  SetSSpin(HIGH);
  SPI.endTransaction();

  PrintOPCstring(opSerial);
}

void PrintOPCstring (Stream &port)
{
  port.write(SPI_in, 60); //print 60 characters from SPI_in[]
array
  port.println("");
  port.flush();
}

void ReadOPChist (void)
```

```
{
  GetReadyResponse(0x30);
  for (SPI_in_index=0; SPI_in_index<86; SPI_in_index++)
  {
    delayMicroseconds(10);
    SPI_in[SPI_in_index] = SPI.transfer(0x01); //Value of
outgoing byte doesn't matter
  }
  SetSSpin(HIGH);
  SPI.endTransaction();
  delay(10);
}


void DiscardSPIbytes (byte NumToDiscard)
{
  for (SPI_in_index=0; SPI_in_index<NumToDiscard;
SPI_in_index++)
  {
    delayMicroseconds(10);
    SPI.transfer(0x01); //Value of outgoing byte doesn't
matter
  }
}


//Get Config data (bin boundaries etc.) from OPC device
void ReadOPCconfig (Stream &port)
{
  unsigned int *pUInt16;
  float *pFloat;

  //Have to read config from OPC device in this 'chunks'
manner as Arduino buffer isn't big enough to hold all config
data at once and OPC could timeout if Arduino took time to
print/save data from the buffer during the SPI transfer
sequence.
  //Instead, config data is read several times, and each time
a different chunk is saved to the Arduino buffer and printed.
This way there is no delay during each individual SPI
transfer sequence.

  //Get config from OPC device (Bin Boundaries ADC)
    GetReadyResponse(0x3C);
    for (SPI_in_index=0; SPI_in_index<50; SPI_in_index++)
    {
      delayMicroseconds(10);
      SPI_in[SPI_in_index] = SPI.transfer(0x01); //Value of
outgoing byte doesn't matter
    }
    //Throw away any remaining bytes OPC expects to transfer.
Although not putting it in buffer yet, must complete SPI
transfer of all config bytes as OPC is expecting this
    DiscardSPIbytes(118);
    SetSSpin(HIGH);
    SPI.endTransaction();
    delay(10);
```

```
    port.print(F("BinBoundaries(ADC)"));
    for (SPI_in_index=0; SPI_in_index<50; SPI_in_index+=2)
    {
      AddDelimiter(port);
      pUInt16 = (unsigned int *)&SPI_in[SPI_in_index];
      port.print(*pUInt16, DEC);
    }
    port.println("");
    port.flush();
  //END Get config from OPC device (Bin Boundaries ADC)

  //Get config from OPC device (Bin Boundaries um)
    GetReadyResponse(0x3C);
    //Throw away bytes until reaching desired bytes in config
    DiscardSPIbytes(50);
    //Put required config bytes in buffer
    for (SPI_in_index=0; SPI_in_index<50; SPI_in_index++)
    {
      delayMicroseconds(10);
      SPI_in[SPI_in_index] = SPI.transfer(0x01); //Value of
outgoing byte doesn't matter
    }
    //Throw away any remaining bytes OPC expects to transfer
    DiscardSPIbytes(68);
    SetSSpin(HIGH);
    SPI.endTransaction();
    delay(10);

    port.print(F("BinBoundaries(um)"));
    for (SPI_in_index=0; SPI_in_index<50; SPI_in_index+=2)
    {
      AddDelimiter(port);
      pUInt16 = (unsigned int *)&SPI_in[SPI_in_index];
      port.print((float)*pUInt16/100, 2); //print to 2dp
    }
    port.println("");
    port.flush();
  //END Get config from OPC device (Bin Boundaries um)

  //Get config from OPC device (Bin Weightings)
    GetReadyResponse(0x3C);
    //Throw away bytes until reaching desired bytes in config
    DiscardSPIbytes(100);
    //Put required config bytes in buffer
    for (SPI_in_index=0; SPI_in_index<48; SPI_in_index++)
    {
      delayMicroseconds(10);
      SPI_in[SPI_in_index] = SPI.transfer(0x01); //Value of
outgoing byte doesn't matter
    }
    //Throw away any remaining bytes OPC expects to transfer
    DiscardSPIbytes(20);
    SetSSpin(HIGH);
    SPI.endTransaction();
    delay(10);

    port.print(F("BinWeightings"));
    for (SPI_in_index=0; SPI_in_index<48; SPI_in_index+=2)
```

```
    {
      AddDelimiter(port);
      pUInt16 = (unsigned int *)&SPI_in[SPI_in_index];
      port.print((float)*pUInt16/100, 2); //print to 2dp
    }
    port.println("");
    port.flush();
  //END Get config from OPC device (Bin Weightings)

  //Get config from OPC device (Misc)
    GetReadyResponse(0x3C);
    //Throw away bytes until reaching desired bytes in config
    DiscardSPIbytes(148);
    //Put required config bytes in buffer
    for (SPI_in_index=0; SPI_in_index<20; SPI_in_index++)
    {
      delayMicroseconds(10);
      SPI_in[SPI_in_index] = SPI.transfer(0x01); //Value of
outgoing byte doesn't matter
    }
    SetSSpin(HIGH);
    SPI.endTransaction();
    delay(10);

    port.print(F("M_A(um),"));
    pUInt16 = (unsigned int *)&SPI_in[0];
    port.println((float)*pUInt16/100, 2); //print to 2dp

    port.print(F("M_B(um),"));
    pUInt16 = (unsigned int *)&SPI_in[2];
    port.println((float)*pUInt16/100, 2); //print to 2dp

    port.print(F("M_C(um),"));
    pUInt16 = (unsigned int *)&SPI_in[4];
    port.println((float)*pUInt16/100, 2); //print to 2dp

    port.print(F("MaxTOF(us),"));
    pUInt16 = (unsigned int *)&SPI_in[6];
    port.println((float)*pUInt16/48, 2); //print to 2dp

    port.print(F("AMSamplingIntervalCount,"));
    pUInt16 = (unsigned int *)&SPI_in[8];
    port.println(*pUInt16, DEC); //print value

    port.print(F("AMIdleIntervalCount,"));
    pUInt16 = (unsigned int *)&SPI_in[10];
    port.println(*pUInt16, DEC); //print value

    port.print(F("AMMaxDataArraysInFile,"));
    pUInt16 = (unsigned int *)&SPI_in[12];
    port.println(*pUInt16, DEC); //print value

    port.print(F("AMOnlySavePMData,"));
    port.println(SPI_in[14], DEC); //print value

    port.print(F("AMFanOnInIdle,"));
    port.println(SPI_in[15], DEC); //print value
```

```
    port.print(F("AMLaserOnInIdle,"));
    port.println(SPI_in[16], DEC); //print value

    port.print(F("TOFtoSFRfactor,"));
    port.println(SPI_in[17], DEC);

    port.print(F("PVP(us),"));
    port.println((float)SPI_in[18]/48, 2); //print to 2dp

    port.print(F("BinWeightingIndex,"));
    port.println(SPI_in[19], DEC);
    port.flush();
  //END Get config from OPC device (Misc)

  //Get DAC and power status from OPC device (this is a
different command to the 'get config' one)
    GetReadyResponse(0x13);

    //Put required status bytes in buffer
    for (SPI_in_index=0; SPI_in_index<6; SPI_in_index++)
    {
      delayMicroseconds(10);
      SPI_in[SPI_in_index] = SPI.transfer(0x01); //Value of
outgoing byte doesn't matter
    }
    SetSSpin(HIGH);
    SPI.endTransaction();
    delay(10);

    port.print(F("FanDAC_ON,"));
    port.println(SPI_in[0], DEC); //print value

    port.print(F("LaserDAC_ON,"));
    port.println(SPI_in[1], DEC); //print value

    port.print(F("FanDACval,"));
    port.println(SPI_in[2], DEC); //print value

    port.print(F("LaserDACval,"));
    port.println(SPI_in[3], DEC); //print value

    port.print(F("LaserSwitch,"));
    port.println(SPI_in[4], DEC); //print value

    port.print(F("AutoGainToggle,"));
    port.println(SPI_in[5]>>1, DEC); //print value
    port.flush();
  //END Get DAC and power status from OPC device (this is a
different command to the 'get config' one)
}


void StartOPC (void)
{
  //Turn ON fan and laser

  //Laser power ON
  GetReadyResponse(0x03);
```

```
  SPI.transfer(0x07); //Turn ON laser power
  SetSSpin(HIGH);
  SPI.endTransaction();
  delay(10);

  delay(1000); //Delay may be necessary to seperate power ON
of fan and laser

  //Fan DAC ON
  GetReadyResponse(0x03);
  SPI.transfer(0x03); //Turn ON fan DAC
  SetSSpin(HIGH);
  SPI.endTransaction();
  delay(10);

  //Wait for fan to reach full speed (and for multiple
attempts by OPC firmware to turn on fan)
  for (byte i=0; i<5; i++)
  {
    wdt_reset(); //Reset watchdog timer
    delay(1000);
  }
}


void StopOPC (void)
{
  //Turn OFF fan and laser.

  //Laser power OFF
  GetReadyResponse(0x03);
  SPI.transfer(0x06); //Turn OFF laser power
  SetSSpin(HIGH);
  SPI.endTransaction();
  delay(10);

  //Fan DAC OFF
  GetReadyResponse(0x03);
  SPI.transfer(0x02); //Turn OFF fan DAC
  SetSSpin(HIGH);
  SPI.endTransaction();
  delay(10);
}


void GetReadyResponse (unsigned char SPIcommand)
{
  unsigned char Response;

  SPI.beginTransaction(SPISettings(300000, MSBFIRST,
SPI_MODE1));

  //Try reading a byte here to clear out anything remnant of
SD card SPI activity (WORKS!)
  Response = SPI.transfer(SPIcommand);
  delay(1);  //wait 1ms

  do
```

```
  {
    SetSSpin(LOW);
    unsigned char Tries = 0;
    do
    {
      Response = SPI.transfer(SPIcommand);
      if (Response != SPI_OPC_ready) delay(1); //wait 1ms
    }
    while ((Tries++ < 20) && (Response != SPI_OPC_ready));

    if (Response != SPI_OPC_ready)
    {
      if (Response == SPI_OPC_busy)
      {
        SetSSpin(HIGH);
        Serial.println(F("ERROR Waiting 2s (for OPC comms
timeout)")); //signal user
        Serial.flush();
        wdt_reset();
        delay(2000); //wait 2s
      }
      else
      {
        /*
        //Can just wait for WDT to timeout and SPI to be
restablished on reset
        SetSSpin(HIGH);
        Serial.println(F("ERROR Waiting for UNO WDT
timeout")); //signal user
        Serial.flush();
        while(1);
        */

        //End SPI and wait a few seconds for it to be cleared
        SetSSpin(HIGH);
        Serial.println(F("ERROR Resetting SPI")); //signal
user
        Serial.flush();
        SPI.endTransaction();
        //Wait 6s here for buffer to be cleared
        wdt_reset();
        delay(6000);
        wdt_reset();
        SPI.beginTransaction(SPISettings(300000, MSBFIRST,
SPI_MODE1));
      }
    }
  }
  while ((Response != SPI_OPC_ready) && (Serial.available()==
0)); //don't hang on this if data is coming in on serial
interface
  delay(10);

  wdt_reset();
}


unsigned int MODBUS_CalcCRC(unsigned char data[], unsigned
```

```
char nbrOfBytes)
{
  #define POLYNOMIAL_MODBUS 0xA001 //Generator polynomial for
MODBUS crc
  #define InitCRCval_MODBUS 0xFFFF //Initial CRC value

  unsigned char _bit; // bit mask
  unsigned int crc = InitCRCval_MODBUS; // initialise
calculated checksum
  unsigned char byteCtr; // byte counter

  // calculates 16-Bit checksum with given polynomial
  for(byteCtr = 0; byteCtr < nbrOfBytes; byteCtr++)
  {
    crc ^= (unsigned int)data[byteCtr];
    for(_bit = 0; _bit < 8; _bit++)
    {
      if (crc & 1) //if bit0 of crc is 1
      {
        crc >>= 1;
        crc ^= POLYNOMIAL_MODBUS;
      }
      else
        crc >>= 1;
    }
  }
  return crc;
}


//Convert SHT31 ST output to Temperature (C)
float ConvSTtoTemperature (unsigned int ST)
{
  return -45 + 175*(float)ST/65535;
}


//Convert SHT31 SRH output to Relative Humidity (%)
float ConvSRHtoRelativeHumidity (unsigned int SRH)
{
  return 100*(float)SRH/65535;
}


//Process OPC data and print
void PrintData (Stream &port)
{
  unsigned char i;
  unsigned int *pUInt16;
  float *pFloat;
  float Afloat;

  //Histogram bins (UInt16) x16
  for (i=0; i<48; i+=2)
  {
    AddDelimiter(port);
    pUInt16 = (unsigned int *)&SPI_in[i];
    port.print(*pUInt16, DEC);
```

```
  }

  //MToF bytes (UInt8) x4
  for (i=48; i<52; i++)
  {
    AddDelimiter(port);
    Afloat = (float)SPI_in[i];
    Afloat /= 3; //convert to us
    port.print(Afloat, 2);
  }

  //Sampling period(s) (UInt16) x1
  AddDelimiter(port);
  pUInt16 = (unsigned int *)&SPI_in[52];
  port.print((float)*pUInt16/100, 3); //print to 3dp

  //SFR (UInt16) x1
  AddDelimiter(port);
  pUInt16 = (unsigned int *)&SPI_in[54];
  port.print((float)*pUInt16/100, 3); //print to 3dp

  //Temperature (UInt16) x1
  AddDelimiter(port);
  pUInt16 = (unsigned int *)&SPI_in[56];
  port.print(ConvSTtoTemperature(*pUInt16), 1); //print to
1dp

  //Relative humidity (UInt16) x1
  AddDelimiter(port);
  pUInt16 = (unsigned int *)&SPI_in[58];
  port.print(ConvSRHtoRelativeHumidity(*pUInt16), 1); //print
to 1dp

  //PM values(ug/m^3) (4-byte float) x3
  for (i=60; i<72; i+=4)
  {
    AddDelimiter(port);
    pFloat = (float *)&SPI_in[i];
    port.print(*pFloat, 3); //print to 3dp
  }

  //Reject count Glitch (UInt16) x1
  AddDelimiter(port);
  pUInt16 = (unsigned int *)&SPI_in[72];
  port.print(*pUInt16, DEC);

  //Reject count LongTOF (UInt16) x1
  AddDelimiter(port);
  pUInt16 = (unsigned int *)&SPI_in[74];
  port.print(*pUInt16, DEC);

  //Reject count Ratio (UInt16) x1
  AddDelimiter(port);
  pUInt16 = (unsigned int *)&SPI_in[76];
  port.print(*pUInt16, DEC);

  //Reject count OutOfRange (UInt16) x1
  AddDelimiter(port);
```

```
  pUInt16 = (unsigned int *)&SPI_in[78];
  port.print(*pUInt16, DEC);

  //Fan rev count (UInt16) x1 (Not using)
  //AddDelimiter(port);
  //pUInt16 = (unsigned int *)&SPI_in[80];
  //port.print(*pUInt16, DEC);

  //Laser status (UInt16) x1
  AddDelimiter(port);
  pUInt16 = (unsigned int *)&SPI_in[82];
  port.print(*pUInt16, DEC);

  //Checksum (UInt16) x1
  AddDelimiter(port);
  pUInt16 = (unsigned int *)&SPI_in[84];
  port.println(*pUInt16, DEC);

  //Compare recalculated Checksum with one sent
  if (*pUInt16 != MODBUS_CalcCRC(SPI_in, 84)) //if checksums
aren't equal
    port.println(F("Checksum error in line above!"));

  port.flush();
}


//Print data labels
void PrintDataLabels (Stream &port)
{
  unsigned char i;

  port.print(F("Time(ms)"));

  for (i=0; i<24; i++)
  {
    port.print(F(",Bin"));
    if (i < 10) port.print(F("0")); //leading 0 for single
digit bin numbers
    port.print(i, DEC);
  }

  for (i=1; i<9; i+=2)
  {
    port.print(F(",MToFBin"));
    port.print(i, DEC);
    if (i == 1) port.print(F("(us)")); //print units for
first value of this type
  }

  port.println(F(",SampPrd(s),SFR(ml/s),T(C),RH(%),PM_A(ug/m^
3),PM_B,PM_C,#RejectGlitch,#RejectLongTOF,#RejectRatio,
#RejectCountOutOfRange,LaserStatus,Checksum")); //(Not using
FanRevCount)

  port.flush();
}
```

```
void AddDelimiter (Stream &port)
{
  port.print(F(",")); //delimiter
}


void SetSSpin (bool pinState) //pinState is HIGH or LOW
{
  digitalWrite(ssPin_OPC, pinState); //Set output to pinState
}
```