Embedded systems programming

Project requirements

Requirements

- Control the speed of a ventilation fan
- Fan is connected to an ABB frequency converter
  - Converter is controlled using Modbus protocol
- Two operating modes:
  - Manual mode
  - Automatic mode
- LCD user interface

Automatic mode

- Set the pressure level in the ventilation duct (0 – 120 pa) in the UI
  - Pressure level is pressure difference between the room and the ventilation duct
- Controller measures pressure level and keeps the level at the required setting by adjusting the fan speed
  - If required level can’t be reached within a reasonable time user is notified on the UI

Manual mode

- Set the speed of the fan in UI (0 – 100%)
- Display current fan setting and pressure level in the UI

Documentation

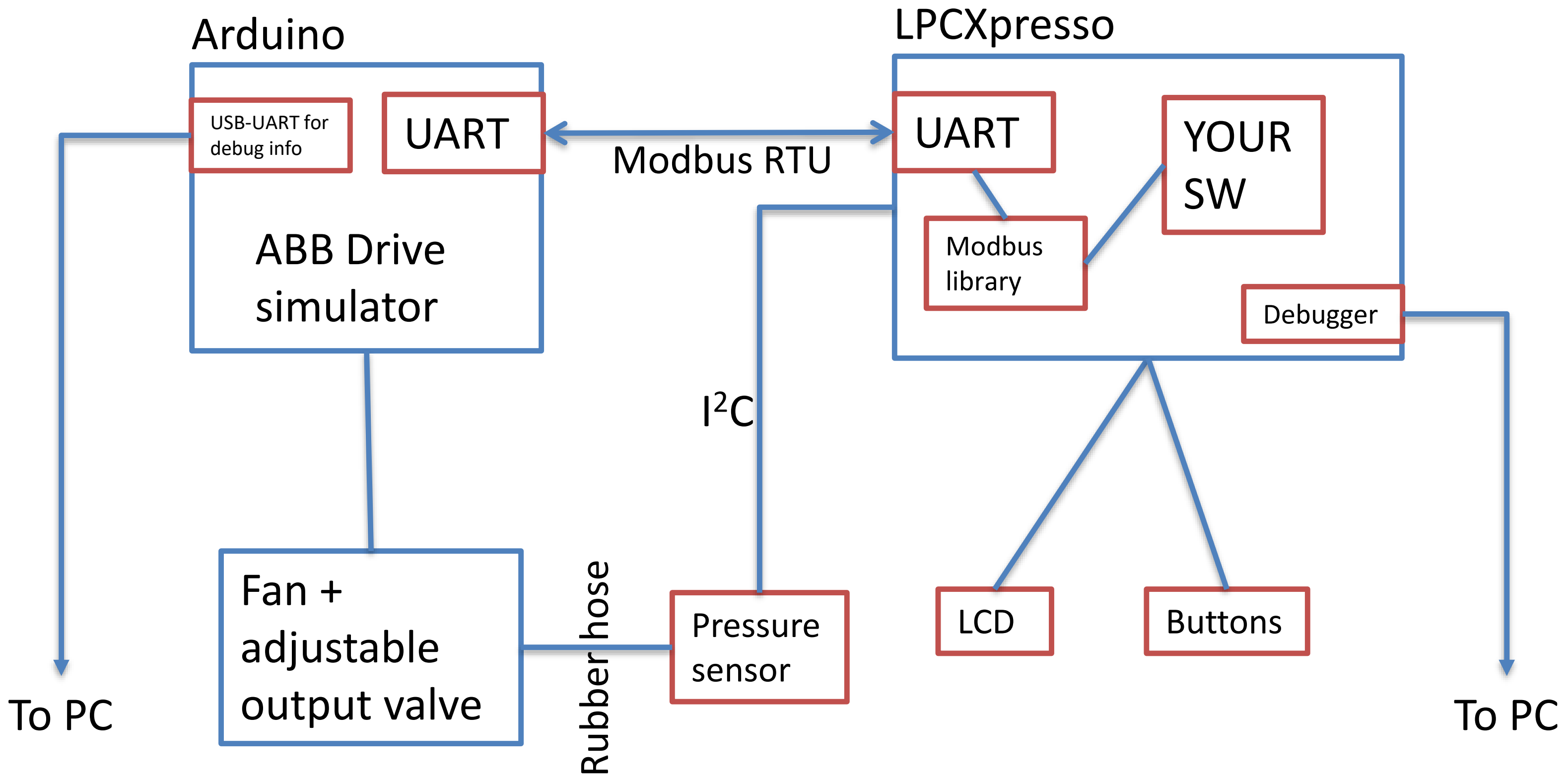
- User manual
- Wiring diagrams
- Program documentation

Some help and instructions

Wiring instructions, sensors, system diagram, etc.

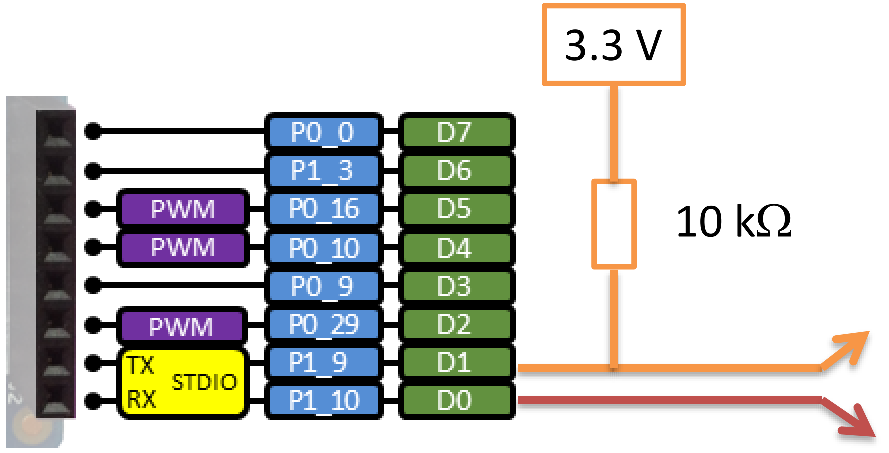
System diagram

ABB drive is controlled by reading/writing Modbus registers.
Modbus register is a ”variable” that can be accessed using Modbus protocol.



Simulator wiring

- This is for connecting LPCXpresso to frequency converter simulator



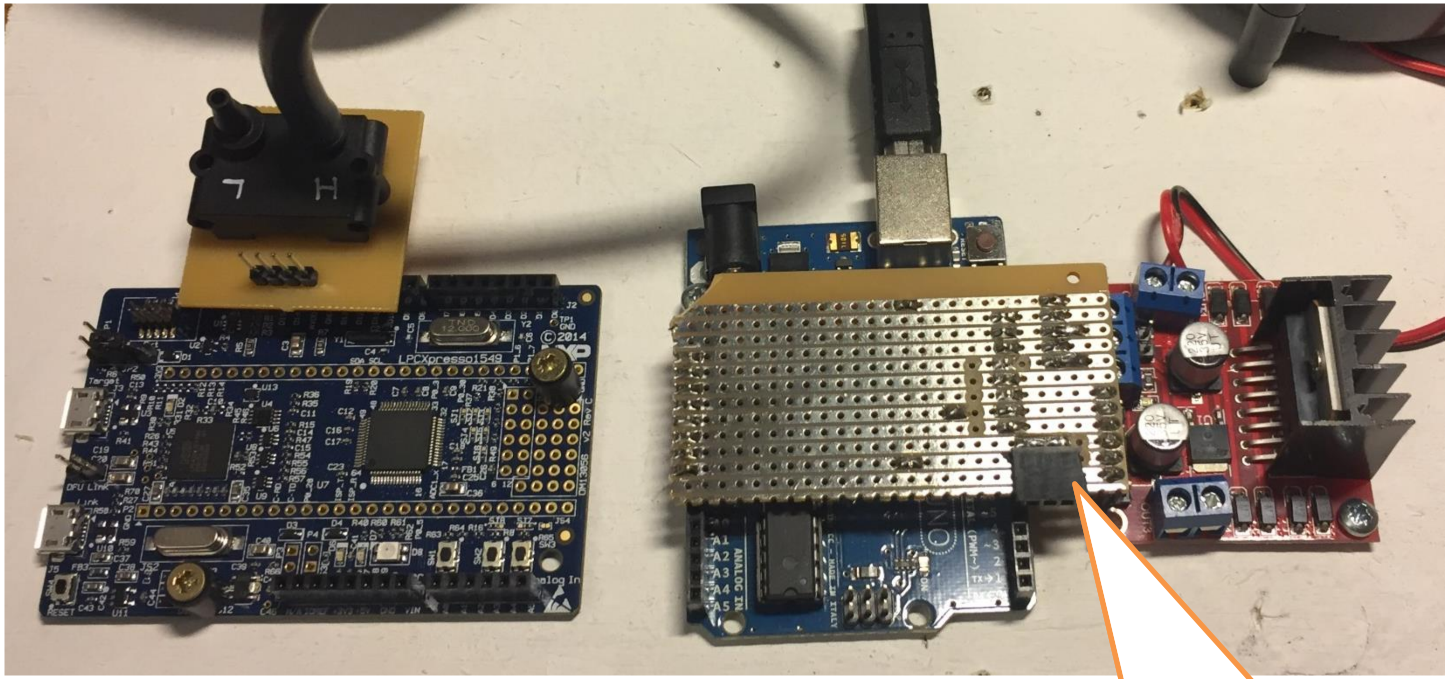
Pull-up resistor is required on P1_9 (D1) to boot CPU into correct mode.

Arduino pin Digital 2

Arduino pin Digital 3

- Connect a wire from Arduino ground to LPCXpresso ground

Simulator wiring



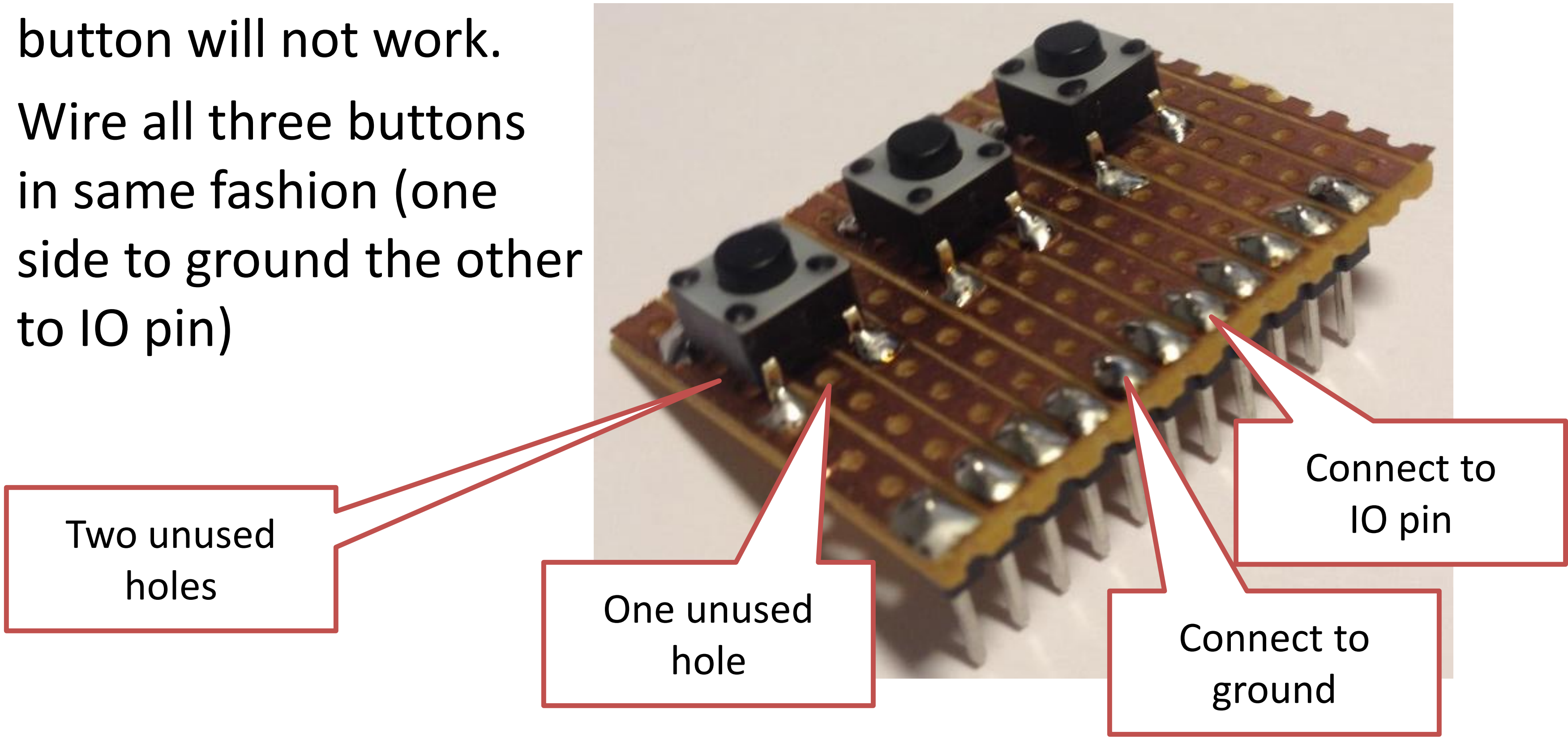
Arduino ground is available in this connector.
Connect Arduino ground to LPCXpresso ground!

LPCXpresso buttons

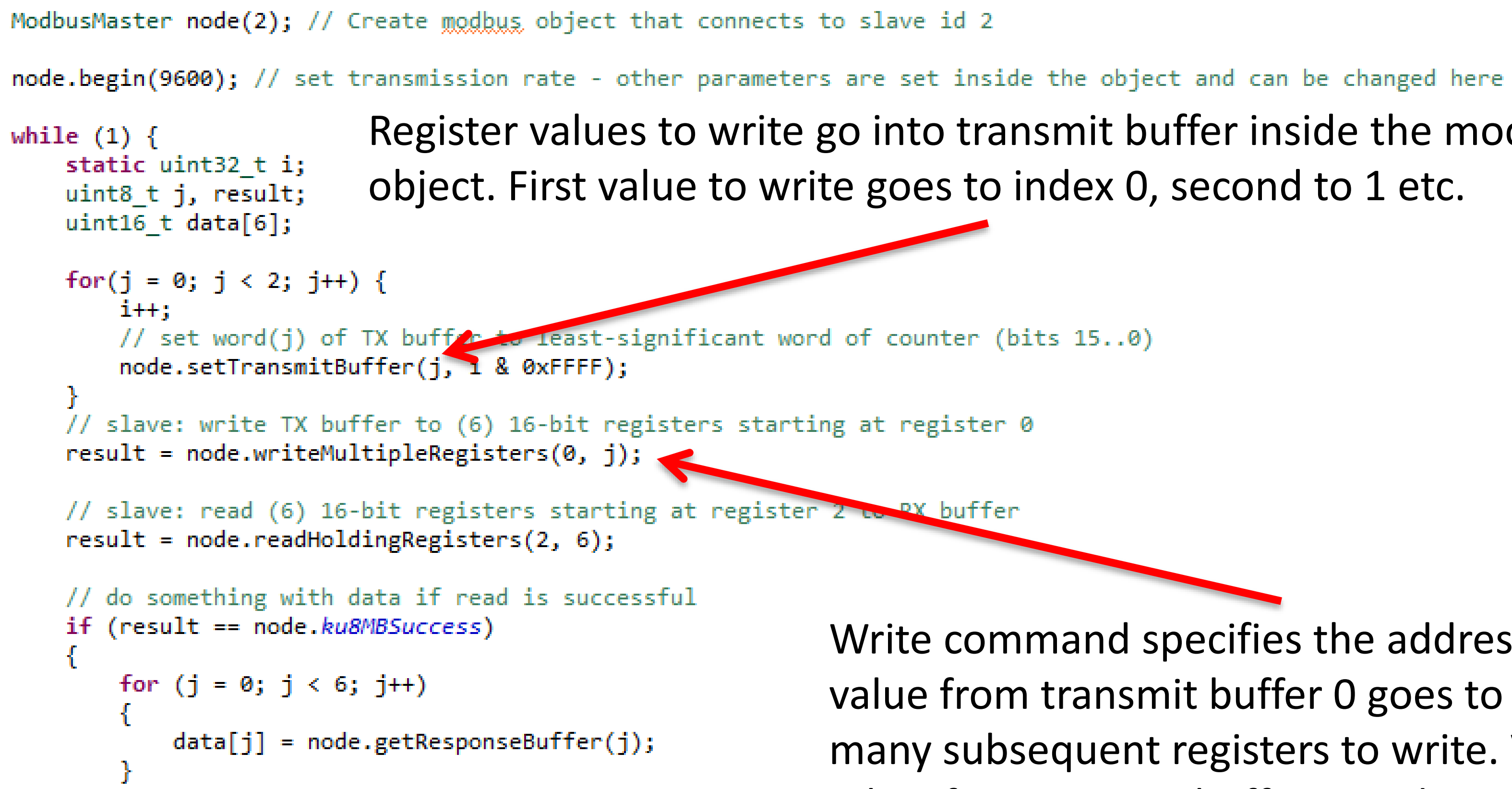
- SW3 on LPCXpresso may not be used for UI
  - SW3 is connected to a pin that is needed for Modbus communication
  - Pressing SW3 will corrupt modbus data!
- Use external buttons for UI

External buttons

- The buttons are soldered to a piece of strip board (vero board). Pay attention to the orientation of the buttons. Incorrectly oriented button will create a short circuit and your button will not work.
- Wire all three buttons in same fashion (one side to ground the other to IO pin)



Writing Modbus registers

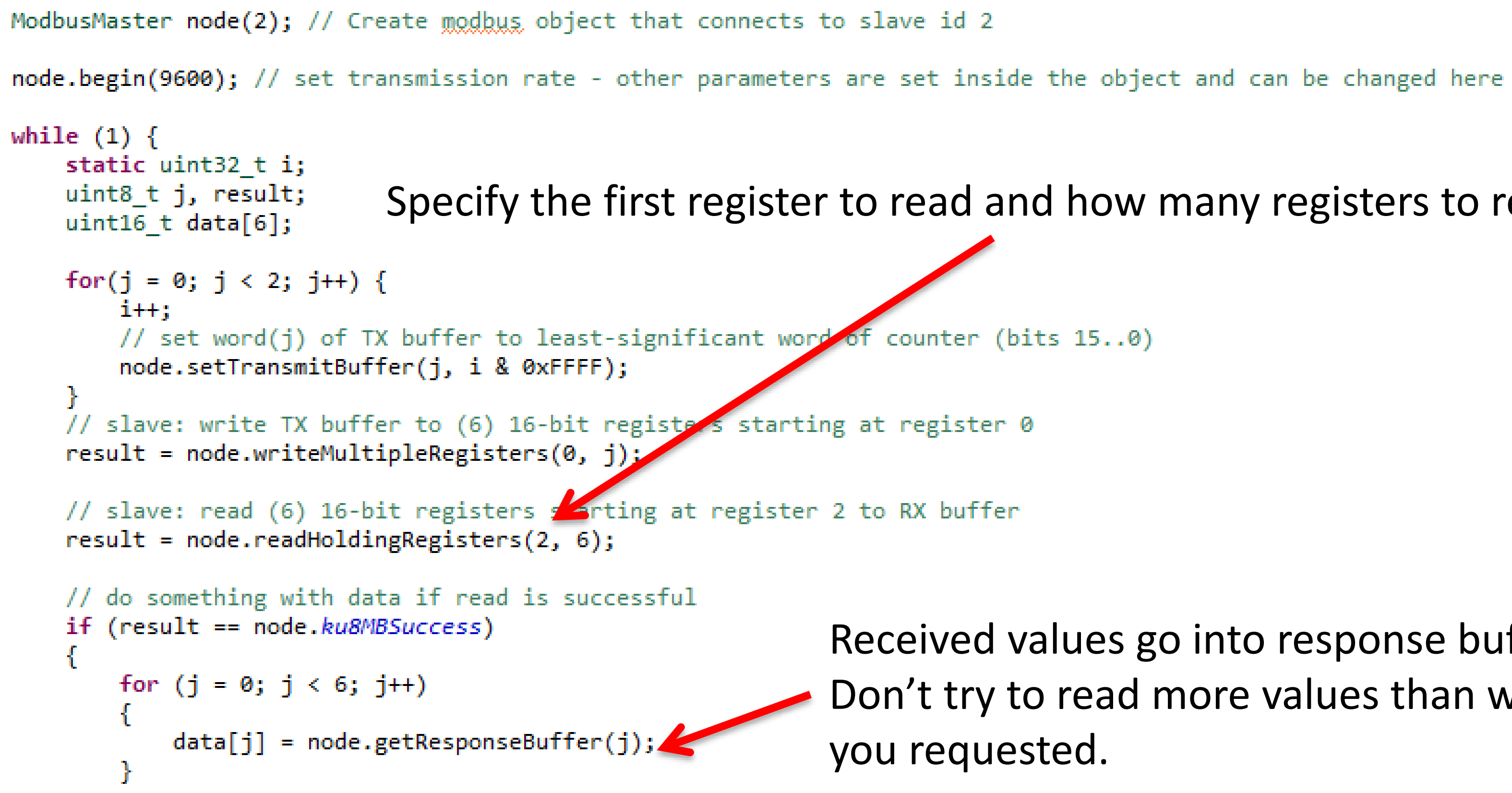


```
ModbusMaster node(2); // Create modbus object that connects to slave id 2

node.begin(9600); // set transmission rate - other parameters are set inside the object and can be changed here

while (1) {
    static uint32_t i;
    uint8_t j, result;
    uint16_t data[6];

    for(j = 0; j < 2; j++) {
        i++;
        // set word(j) of TX buffer to least-significant word of counter (bits 15..0)
        node.setTransmitBuffer(j, i & 0xFFFF);
    }
    // slave: write TX buffer to (6) 16-bit registers starting at register 0
    result = node.writeMultipleRegisters(0, j);

    // slave: read (6) 16-bit registers starting at register 2 to RX buffer
    result = node.readHoldingRegisters(2, 6);

    // do something with data if read is successful
    if (result == node.ku8MBSuccess)
    {
        for (j = 0; j < 6; j++)
        {
            data[j] = node.getResponseBuffer(j);
        }
```

Register values to write go into transmit buffer inside the modbus object. First value to write goes to index 0, second to 1 etc.

Write command specifies the address where value from transmit buffer 0 goes to and how many subsequent registers to write. Values are taken from tranmit buffer in order starting with index 0.

Reading Modbus registers



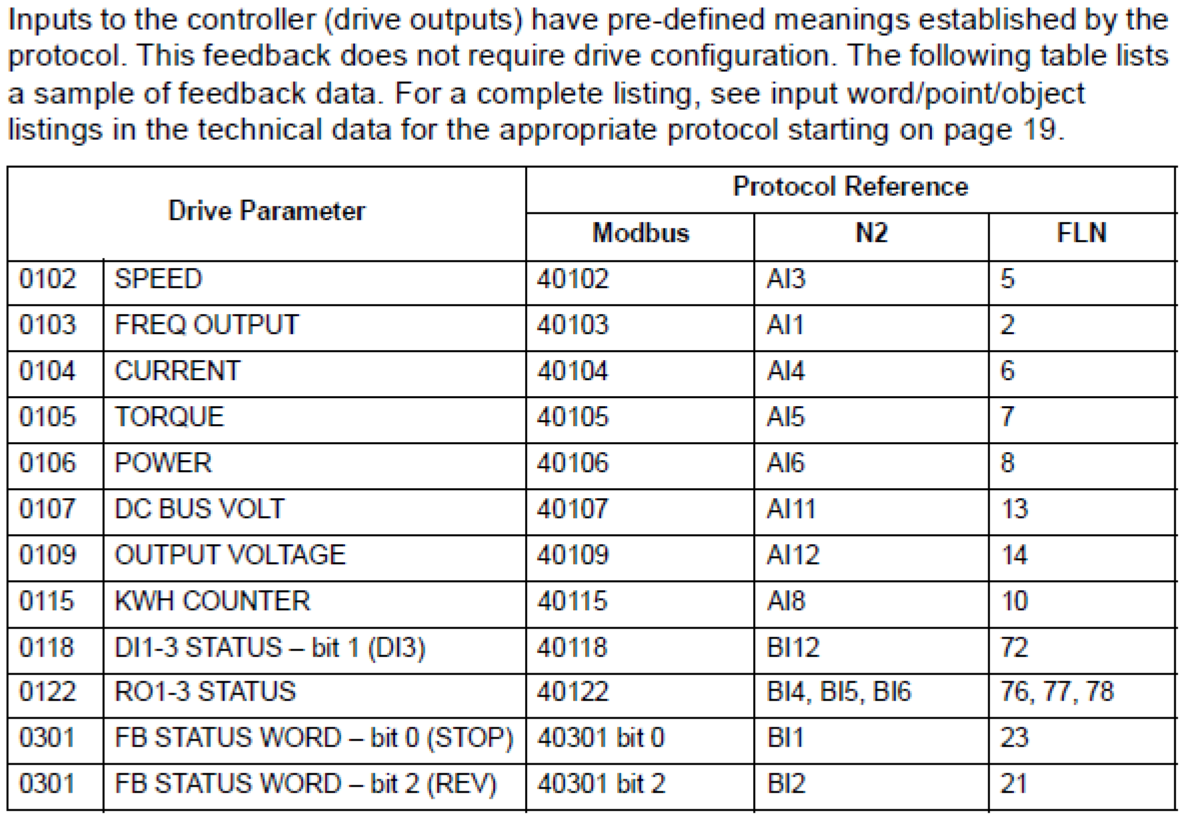
```
ModbusMaster node(2); // Create modbus object that connects to slave id 2

node.begin(9600); // set transmission rate - other parameters are set inside the object and can be changed here

while (1) {
    static uint32_t i;
    uint8_t j, result;
    uint16_t data[6];

    for(j = 0; j < 2; j++) {
        i++;
        // set word(j) of TX buffer to least-significant word of counter (bits 15..0)
        node.setTransmitBuffer(j, i & 0xFFFF);
    }
    // slave: write TX buffer to (6) 16-bit registers starting at register 0
    result = node.writeMultipleRegisters(0, j);

    // slave: read (6) 16-bit registers starting at register 2 to RX buffer
    result = node.readHoldingRegisters(2, 6);

    // do something with data if read is successful
    if (result == node.ku8MBSuccess)
    {
        for (j = 0; j < 6; j++)
        {
            data[j] = node.getResponseBuffer(j);
        }
```

Specify the first register to read and how many registers to read.

Received values go into response buffer. Don’t try to read more values than what you requested.

ACH550 and Modbus

- To read these using modbus you must subtract one from the parameter number (Drive parameter 0103 can be read from Modbus holding register address 102)



Inputs to the controller (drive outputs) have pre-defined meanings established by the protocol. This feedback does not require drive configuration. The following table lists a sample of feedback data. For a complete listing, see input word/point/object listings in the technical data for the appropriate protocol starting on page 19.

Drive Parameter		Protocol Reference		
		Modbus	N2	FLN
0102	SPEED	40102	AI3	5
0103	FREQ OUTPUT	40103	AI1	2
0104	CURRENT	40104	AI4	6
0105	TORQUE	40105	AI5	7
0106	POWER	40106	AI6	8
0107	DC BUS VOLT	40107	AI11	13
0109	OUTPUT VOLTAGE	40109	AI12	14
0115	KWH COUNTER	40115	AI8	10
0118	DI1-3 STATUS – bit 1 (DI3)	40118	BI12	72
0122	RO1-3 STATUS	40122	BI4, BI5, BI6	76, 77, 78
0301	FB STATUS WORD – bit 0 (STOP)	40301 bit 0	BI1	23
0301	FB STATUS WORD – bit 2 (REV)	40301 bit 2	BI2	21

ABB modbus registers

- The most important modbus registers are:
  - Control word
  - Reference 1
  - Status word
- See page 24 in the Embedded field bus document
- Study the sample codes in OMA

ABB simulator

- **The simulator must be powered from the DC power supply**
  - It will not work properly with USB power
- Check the state diagram in EN_ACH550_EFB_D.pdf on page 35 it tells how ABB drive should be controlled
- The profile that we use is ABB DRV FULL
  - The state diagram does not show control word bit 4. CW bit 4 needs also to be set to 1 to enable operation
- The simulator outputs debug information on the USB port of Arduino
  - Speed is 9600 bps
  - Simulator shows the current state (according to the state diagram in the pdf)

Remote (Modbus) mode

- One oddity of this device is that Modbus control needs to be enabled with a Modbus message
  - Remote control mode needs also to be activated with a Modbus message
    - Bit 10 in control word must be set to 1 make ABB to accept remote commands over Modbus
  - The first writes to register 0 in the example code do that

Modbus heart beat

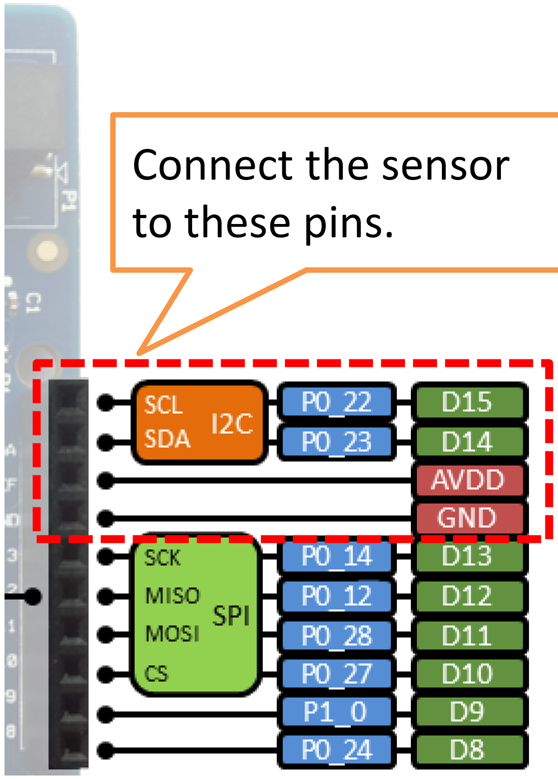
- When the frequency converter is controlled with Modbus a heart beat signal is required to inform the frequencey converter that master is still ’alive’
- Any Modbus message to the converter (simulator) works as a heart beat signal. For example reading the current frequency value keeps the frequency converter running.
  - You can see the converter simulator status in the Arduino output window. Simulator prints ”Inactive” when the heart beat signal is lost

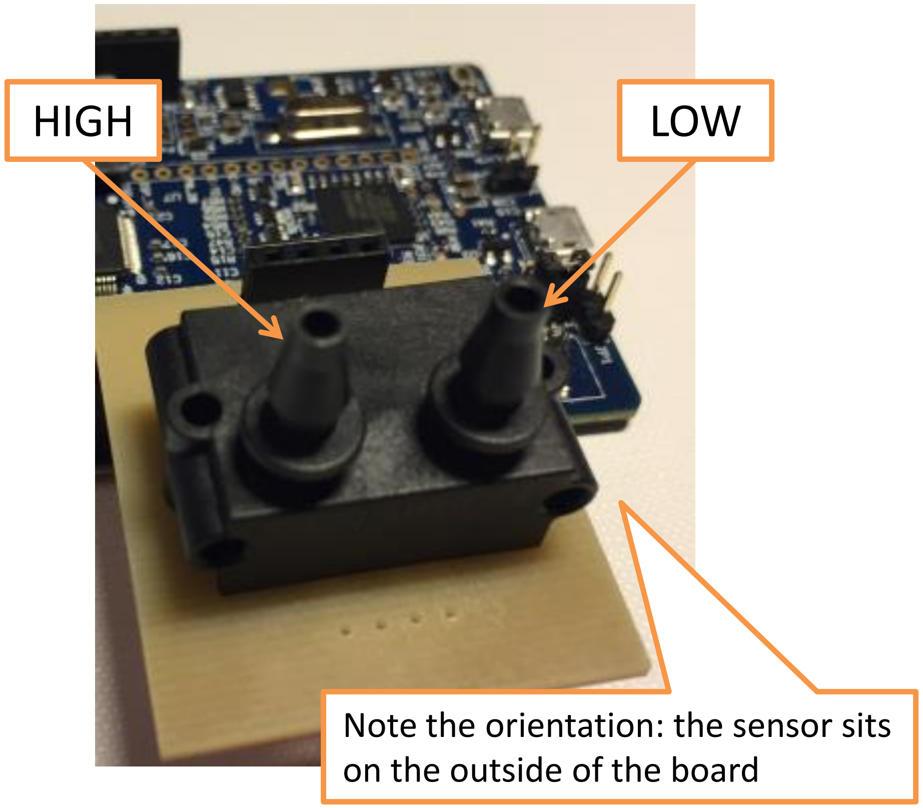
Interface

- Pressure sensor has an I^2C interface and operates on 3.3 V
- I^2C interface
  - Two signals: SCL (clock) and SDA (data)
  - SDA is bidirectional (both input and output)
  - Requires pull-up resistors of specific size → can't use built in pull-ups
    - Our board has suitable external pull-ups installed on P0-22 (SCL/D15) and P0-23 (SDA/D14)

Pressure sensor board

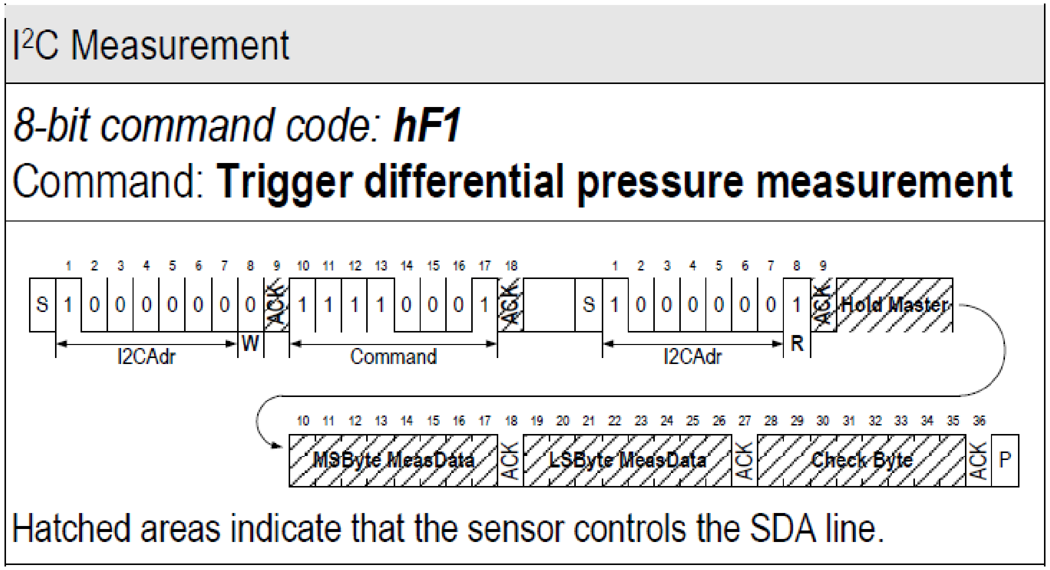
- To measure the pressure in the ducts connect a measuring hose to HIGH and the other end of the measuring hose to a connector in the ventilation duct




HIGH
LOW
Note the orientation: the sensor sits on the outside of the board

Measurement

- The address of the sensor is 0x40
- The command for reading value is 0xF1
- Sensor returns 3 bytes: 2 bytes of data and 1 byte CRC


I²C Measurement

8-bit command code: ***hF1***

Command: **Trigger differential pressure measurement**

S 1 0 0 0 0 0 0 0 ACK 1 1 1 1 0 0 0 1 ACK
I2CAdr
W
Command
S 1 0 0 0 0 0 0 1 ACK Hold Master
I2CAdr
R
MSByte MeasData
ACK
LSByte MeasData
ACK
Check Byte
ACK
P

Hatched areas indicate that the sensor controls the SDA line.

Measurement

- Sensor returns a signed 16 bit value
- The value must be converted to physical value (pascals)
  - Scale factor (see data sheet chapter 2)
  - Altitude correction (see data sheet chapter 5)
  - Hose length compensation (see data sheet chapter 8) – not needed for hose length up to 1 m