Green house CO_2 controller specification

Introduction

The goal of the project is to implement a CO_2 fertilization controller system for a greenhouse. The controller keeps the CO_2 level at the level that is set from the local user interface. The settings are stored in EEPROM to make them persistent.

Controller connects to the cloud using MQTT and sends the measured CO_2 level and other environmental information to the cloud for display and/or further processing. The network parameters needed for connection can be set from the local user interface and are stored in the EEPROM.

Hardware platform

LCD display 16x2 characters:

- RS → Port 0, Pin 29
- EN → Port 0, Pin 9
- D4 → Port 0, Pin 10
- D5 → Port 0, Pin 16
- D6 → Port 1, Pin 3
- D7 → Port 0, Pin 0

Rotary encoder:

- A → Port 0, Pin 5 (SW_A3)
- B → Port 0, Pin 6 (SW_A4)
- Button → Port 1, Pin 8 (SW_A2)
- Rotary encoder signals A and B have pullups on the encoder board and can be configured as normal inputs. Encoder button is a grounding button without pullup. Configure encoder button as an input with pullup.

Relay to open the CO_2 solenoid valve:

- Port 0, Pin 27

The system is also equipped with two sensors:

- Vaisala GMP252 CO_2 probe
  - Modbus address: 240
- Vaisala HMP60 relative humidity and temperature sensor
  - Modbus address: 241

Both sensors operate at the speed of 9600 bps. See data sheets of the sensors in the course workspace for more information.

ESP-based WiFI module

- Accessed through a socket library

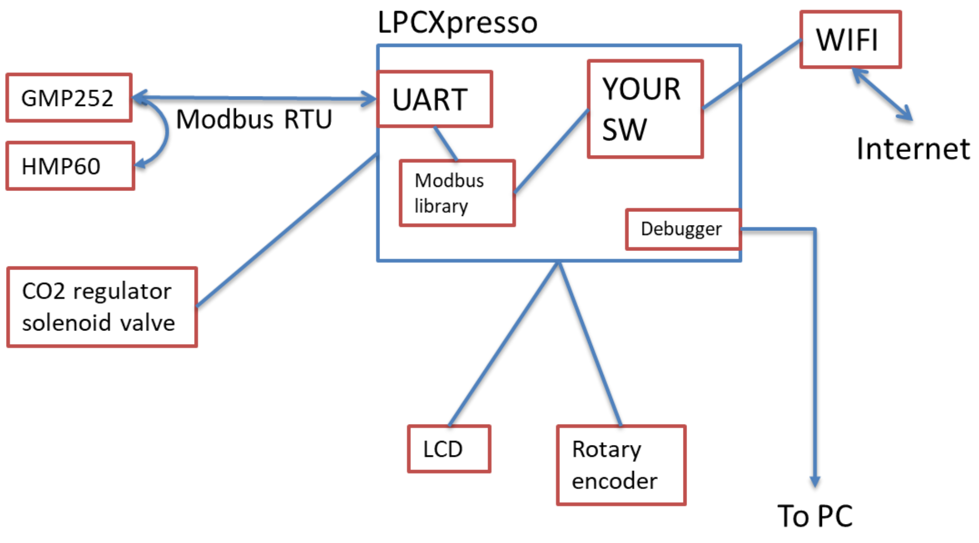

Figure 1 System diagram

Simulator system

Simulator system consists of similar components as the "big" test system with the exception that some of the components are simulated by an Arduino software. The simulator does not simulate the system accurately, it is meant for basic testing of the software. From the hardware interface perspective, the two test systems are identical and the software should run on both systems without any modification. Depending on the controller implementation the controller parameters may have to be adjusted to match the environment – see a later note about simulator short comings.

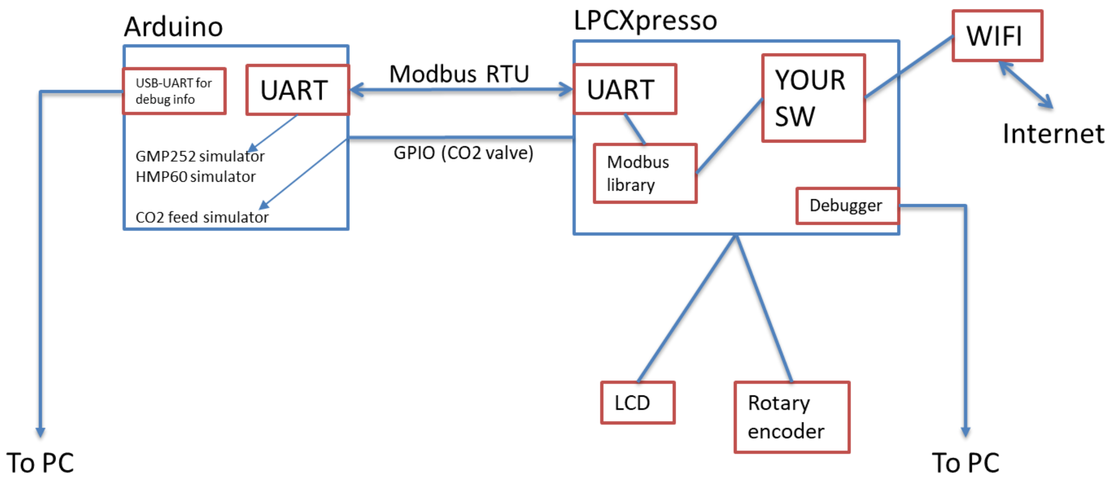

Figure 2 Miniature test system diagram

Cloud connection

The measured data and some control parameters are sent to ThingSpeak cloud service over MQTT. The following information is to be sent to the cloud at 5-minute intervals:

1. CO_2 level (ppm)
2. Relative humidity

3. Temperature
4. Valve opening percentage
5. CO_2 set point (ppm)

The data format is simple: fields are numbered as above and data fields are encoded as if the data was sent in a HTTP POST. The format is :

<name of the field>=<field value>&<name of the field>=<value>&…

Field names are: fieldX, where X is a number between 1 and 8. For example, CO_2 level of 450 ppm is sent as "field1=450". All fields should be sent in the same message. There are also additional named fields, for example time stamp or status, that can be sent along with the measurement data.

See: https://se.mathworks.com/help/thingspeak/publishtoachannelfeed.html for more detail about the data format.

Credentials and topic name that are needed to send to the cloud are given in the hackathon.

Arduino sensor/IO simulator

Arduino software simulates Vaisala sensors and simulates dissipation of CO_2 only to some extent. It should be noted that simulator has some shortcomings:

- It does not make a difference between input registers and holding registers which means that code that is able to read values from registers in the simulator may not work with the real hardware
- Simulator does not require 3.5-character delay between messages as specified in the Modbus specification because all simulated devices are on the same physical device. This means that code that is able to read sensors in the simulator may not be able to read them in the real hardware due to missing delays. The LPC Modbus library does not enforce the delays so the user must implement a delay between messages.
- CO_2 dissipation is modelled with a simple ad-hoc formula that does not match real world behaviour to any extent. It is just to help with rudimentary testing of keeping CO_2 level stable.

You can open Arduino's serial port with speed 9600 to see simulator status messages and to set values to sensors. There is no local echo, the characters you type are not echoed back to you, so you either have enable local echo on your terminal or have steady fingers. Vaisala sensor values can be set by simple commands: `co2` (or `ppm`) , `rh`, and `t` followed by value you want the sensor to report. The value that is read from the command is written to the terminal when you press enter after the command.

CO_2 dissipation is controlled with command `dec`. The command sets CO_2 decrement per second in multiples of 0,01 ppm/s. Value of 100 means 1 ppm/s, 10 is 0,1 ppm/s etc. CO_2 level is never decremented below 200 ppm.

CO_2 injected in to the green house by opening a solenoid valve that when open lets CO_2 in the green house at the rate of 17l/min. This valve is simulated by increasing CO_2 level 1 ppm/s when the valve is open. It has not been verified that the rate matches the real world case – most like it does not.

Project template

Checkout from gitlab.metropolia.fi:

```
git clone https://gitlab.metropolia.fi/lansk/mqtt_freertos.git
```

Open MCUXpresso and switch workspace to mqtt-freertos-directory.

Import existing projects into the workspace.

Minimum requirements

To have the project approved you need to implement at least the local UI with persistent CO_2 level setting. All sensors on the system must be read and displayed on the local UI. You also need to provide the following documentation:

- User manual
    - How to operate your system through the UI
- Program documentation
    - Class diagrams, block diagrams, flow charts etc.
    - Implementation principles