CIWS-EWM Datalogger

Software Documentation

Software ver. 1.0.0

Overview

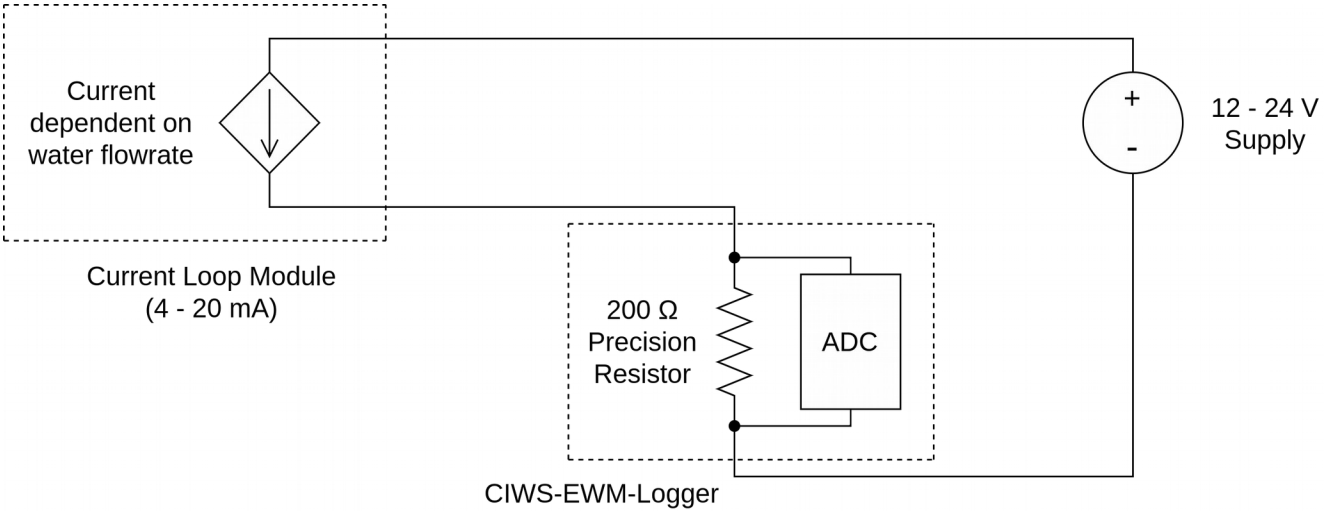
The CIWS-EWM-Logger device is a datalogger designed for gathering water flow data from residential buildings on Utah State University's campus with 4 - 20 mA current loop output meters. Two configurations appear in this repository:

- A configuration which reads a single 4 - 20 mA current loop output meter.
- A configuration which reads two 4 - 20 mA current loop output meters and a pulsed-output meter, along with three DS18B20 temperature sensors.

These devices are based on Raspberry Pi 3 Model B embedded computers. Additional electronics are assembled on an Adafruit Perma-Proto HAT (Adafruit product ID 2310). Both devices make use of the ADS1015 Analog-to-Digital Converter (ADC), which is also available from Adafruit (product ID 1083).

Principle of Functioning

Both devices are designed to measure the output of 4 - 20 mA current loops. The current loop module used is designed for Master Meter's Octave ultrasonic water meters. A diagram of the current loop module connected to the datalogger is shown here:



The current loop module, powered by a 12 - 24 V power supply, regulates current through the circuit based on the water flow rate. A 200 Ω precision resistor is placed in series with the loop module. Current through the resistor drops a voltage across the resistor's terminals. The relationship for voltage, current, and resistance is given in the following equation:

$$v = i \cdot R$$

In other words, the voltage across the resistor terminals is equal to the resistor's resistance times the amount of current flowing through the resistor. In this way, the current loop output is converted into a value detectable by the ADC on the datalogger. From the voltage value, it is then converted to a value corresponding to the meter flow rate in gallons per minute (GPM).

Software

The CIWS-EWM-Logger software is written in Python version 2.7, though a conversion to Python version 3.7 would not be difficult. There are two flavors of the software; one for the configuration measuring one current loop module, and the other for the configuration measuring multiple sensors.

Single-Sensor Configuration: `meterlog.py`

The Python script `meterlog.py` depends on the modules `time` and `Adafruit_ADS1x15`. After these modules are imported in the script, a few configuration variables are set. The user is free to change these as needed.

Configuration Variable	Descriptions
`siteCode`	String describing the site in which the datalogger is installed.
`scanInterval`	Time between scans within the main loop (seconds)
`recordInterval`	Time between recorded values (seconds)
`maxFlowRate`	Maximum flow rate of meter (GPM), corresponding to 20 mA from loop.
`calibrationFactor`	Scales the output voltages based on simple one-point calibration

After these variables are set, the script initializes more variables dealing with the ADC, timing, and filenaming, and a data output file is generated. The further intitialization is not documented here, but is well documented through commenting in the script file.

The script then enters the main program loop. The main program loop repeats until the script is terminated, and does the following:

- Reads the current time
- If a scan interval has elapsed:
    - Increment the sample count
    - Get a correctly formatted date/time string
    - Read the ADC and convert reading to a sensor voltage.
    - Calculate the flowrate from the sensor voltage
    - Calculate total flow for the scan interval
    - Calculate total flow for the record interval
    - Calculate total flow for the logging session
    - Calculate average flow
    - Print the data to the screen
    - If a record interval has elapsed

- Write the data to file
- Increment the sample number
- Reset record variables

Reading the ADC in this script is a single-ended measurement, meaning that the voltage being read is actually the voltage across one of the resistor terminals and ground. This works, because the other resistor terminal is also connected to ground; however, error is introduced using this method because there is a non-zero resistance between the resistor and ground. This extra resistance must be accounted for; otherwise, the measurement will be inaccurate. The current loop design is modified in the next Python script, helping to improve accuracy.

Multi-Sensor Configuration:

`Integratable_multimeter_logger_with_temperature_and_chunkingTest.py`

The Python script `Integratable_multimeter_logger_with_temperature_and_chunkingTest.py` depends on the modules `time`, `Adafruit_ADS1x15`, `sys`, `RPi.GPIO`, `w1thermsensor`, `threading`, and `os`. The structure of this script is similar to the single-sensor configuration, `meterlog.py`. This script differs from the single-sensor configuration in that it reads a total of six sensors: two current-loop modules, one pulse-output module, and three DS18B20 temperature sensors. This script also starts a new data file every day at midnight, making data collection more manageable.

DS18B20 temperature sensors communicate with a host controller using the Dallas Semiconductor 1-Wire interface. This interface is very similar to the I^2C protocol, or 2-Wire interface. Each DS18B20 sensor has a unique 64-bit address, allowing there to be a network of sensors on a single 1-wire bus. In order for the script to run properly, the address of each temperature sensor, along with its location, must be listed in a configuration file called `thermConfig.txt`, as shown here:

```
00000588806a HOT_SUPPLY
00000588806b HOT_RETURN
00000588806c COLD_SUPPLY
```

The w1thermsensor python package includes methods to retrieve sensor IDs you can use to generate `thermConfig.txt`. The three sensors are read using the following methods:

```
read_Hot_Supply_Temp()
read_Hot_Return_Temp()
read_Cold_Supply_Temp()
```

These methods are actually ran as threads, using the Python `threading` module. This is done because reading all three sensors in order takes longer than one second, the scan interval being used. Running the three methods in parallel with the rest of the script allows for all three sensors to be read in time. The low speed may be due to the library implementation or the kernel module.

Pulsed-output from the hot return meter is logged by reading GPIO pin 23 (Broadcom mode). The script tracks if the low input level is a new pulse or not, thus only counting new pulses instead of counting a pulse every time the script reads the GPIO pin and detects a logic low level.

In this configuration, both current-loop modules are read by performing a differential measurement across the terminals of the 200 Ω shunt resistor. This is done by attaching one resistor between terminals A0 and A1 of the ADC, and the other resistor between terminals A2 and A3. The ADC performs a differential measurement by measuring the voltage on both pins and keeping the difference. This eliminates the issue described in the single-sensor configuration. Please see the hardware documentation for more information.