12. Conversational Software

Purpose: Take the software to the next level so that it is reasonably suitable for long-term field test monitoring.

Goal: Create software in both the ATTiny84 (ATtiny) and the Raspberry Pi (RPi) that enables the RPi to send commands to the ATTiny for it to carry out. Establish a comms protocol that enables a back-and-forth 'conversation' of sorts between the two devices.

> **Sub-Goal**: Redesign the code on the ATTiny84 side to make it 'event-driven' and 'non-blocking' in order to facilitate it being responsive to commands from the RPi. This will also allow for more responsive health and error signalling via LEDs on the sensor board.

Design Notes:

Assumptions:

1. The raspberry pi will be plugged in to the mains and always on and available as seen from the perspective of the moisture sensors.

2. Raspberry Pi functions as central server for the sensor network.

3. Raspberry Pi provides instructions to the sensors. A very limited set of possible instructions, but the RPi is in control. Possible instructions:

    * Send capacitance reading.

    * Got to sleep for x minutes.

    * Send capacitance readings continuously until I say to stop.

    * Take, and send me, average cap reading over 20 sample points.

Communications Structure:

Here is how I'm thinking it goes. We start with the raspberry pi sniffing the airwaves just listening for a moisture sensor to tell it: "Hey raspberry pi I'm here, I'm awake, and I'm awaiting instructions." Then the RPi sends an instruction. The moisture sensor acknowledges the instruction, which might include in the ACK results from performing that instruction - e.g., a moisture reading. Then the RPi acknowledges the moisture sensor's completion of the instruction. Then the cycle can then begin again.

Sensor: I'm awake.
RPi: Acknowledged, send me capacitance reading.
Moisture sensor: [performs a soil moisture reading action] Capacitance is ###.
RPi: Acknowledged, sleep for 1 hour.
Moisture sensor: Acknowledged. [Goes to sleep]

Considerations:

* Comms Errors.Now what I also have to consider is that either device may not hear the commands or acknowledgements. So there has to be some facility for repetition and time-outs. But I also know that the RPi seems to pretty much always get the messages from the ATtiny even though the ATTiny will often not hear the acknowledgement back from the RPi. So I have to account for that situation as well.

    - The fall-back for the moisture sensor sleep time, in the event of an ack timout, will simply be that the moisture sensor will use its last known sleep time. On the RPi side it will also know that value so if the RPi never gets an acknowledgement from the moisture sensor that it heard the latest sleep command the raspberry pi will also fall back to the prior one.

Key References --

Raspberry Pi Login --

Pi user is: pi. So SSH login is:

```
ssh pi@192.168.1.118
password: pi9012
```

And then once in, get into the working directory for the nRF24 utilities and programs:

```
cd /home/rf24libs/RF24/examples_linux/build
```

The run the RPi receiving program:

```
./RPi_CapDataReceive
```

To Modify RPi side .cpp code --

Use Kate on my desktop.

Then copy to the RPi using below command:

```
scp /home/jroc/Dropbox/projects/MoistureSensor/CapSensor
/RPi/RPi_CapDataReceive.cpp pi@192.168.1.118:/home/rf24libs
/RF24/examples_linux/
```

Then on the RPi, cd into the build directory and execute 'make'

BUT if this is the first iteration of the program, you need to first add it to CMakeLists.txt in the examples_linux directory (above the build directory).

NOTE: All the code is under git version control. So the executables for both the ATTiny and RPi are on my desktop.

Next Step: Several steps upon completion of this milestone: One, Fabricate a few different capacitor sensors to test. Two, modify code to measure and TX moisture (capacitance) data once/hour instead of continuously. Three, setup sensor in a real flower/plant pot in my house and run field tests on the sensors.

7/18/2023 --

Results: **See Below**.

ATTiny Code: .

> * Created and successfully tested the `CapSensor class`. A non-blocking object to provide a sensor capacitor reading that is averaged out over 10 measurements. Sucessfully passed `TC02_CapClass`.

RPI Code:

> * No work on this as yet.

Documentation: I also created a 'protocol' for managing test cases for Arduino projects, and documented that in: `/Software/tiny84/Readme-TESTcases.md`.

7/09/2023 --

Results: **See Below**.

ATTiny Code: For this milestone the ATTiny84 sketch, or application, is: **`tiny84_SensorAsSlave`**. Since the start of July, on an as time permits basis, I have been re-architecting the code for the ATTiny84 sensor driver to be 'event driven' and 'non-blocking.' Non-blocking meaning that we don't invoke the delay() function; plus we don't perform a long series of time-consuming instructions as a single block. In effect I am simulating a multi-threaded environment. I am doing this, first, as an experiment and learning experience. But if it works out I expect it to be very useful for future projects.

> * I have created and tested a 'HeartBeat' class that continuously flashes an LED (in this case it's a green LED) just to let me know that the processor is alive and running through the loop() function. It tells me that we are caught in some infinite loop elsewhere or that the processor has crashed. In either of those two cases that green LED will either be continuously OFF or continuously ON. See: `/Software/tiny84/tiny84_SensorAsSlave/HeartBeat.h` & `/HeartBeat.ino`.
>
> * I have also created, and tested, an LED based error-reporting function. The main sketch can 'report' an error condition to the ErrorFlash object and that object will report it out by flashing it's LED (in this case a red LED) the number of times of the error #. See: `/Software/tiny84/tiny84_SensorAsSlave/ErrorFlash.h` & `/ErrorFlash.ino`.
>
> Each of these classes are good candidates to be in my personal Arduino library; but for now I am just treating them as part of the code base for the ATTiny84 sensor driver application.

RPI Code:

> * No work on this side as of this date.