7. Measure and Transmit Capacitance Value

Purpose: Get the ATTiny84 to determine the value of an external capacitor and send that data over to the RPi.

Goal: From the monitor on the Raspberry Pi, see the capacitor value. Ideally, see varying values on the RPi, in real-time, as I vary a pot.

Key References --

For NOT using a 555 timer to do this - much simpler:

https://www.arduino.cc/en/Tutorial/Foundations/CapacitanceMeter

For WHY not to use a 555 timer:

Is the NE555 the IC I need, and if not, what do I replace it with? @ https://electronics.stackexchange.com/questions/486156/is-the-ne555-the-ic-i-need-and-if-not-what-do-i-replace-it-with/486157#486157

> "Modern microcontrollers work at lower voltages, and typical everything enabled, even if not used (example datasheet) at say 12 MHz clock rate (which should be way more than enough to synthesize any signal shape that the NE555 might create) half of that; but realistically, you'd run your MCU at a lower frequency, and let it sleep most of the time when using it to replace an NE555 in most applications."

Terminal Cursor Control Escape Sequences:

See @ https://tldp.org/HOWTO/Bash-Prompt-HOWTO/x361.html

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`

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).

10/06/2022 --

Update: Success! I now have the ATTiny84 successfully reading the 10mFD capacitor I've connected to it, to stand in for the eventual moisture sensing capacitance. Tho, of course, the moisture device will be in the pico / nano farad range.

I created a new branch, 'blown-pin-test,' to confirm my theory that I had blown pin #6 in setting both charge and discharge pins to OUTPUT / LOW, per the notes of 10/3 below. By using an LED, I confirmed that pin#6 no longer works. I moved the discharge pin to physical pin #2 (PB0) and it 'works as designed.' I also changed the charge capacitor from a 1K to a 100K resistor. I set that value in the Arduino code: [`#define resistorValue 100000.0F`] and, volia! - the console display shows me "10.46 mFD" for the cap measurement. It's magic!

Software: Committed and checked in branch: 'blown-pin-test.' Checked out 'main' and applied the changes to the tiny84 sketch: dischargePin to PB0 and resistor value to 100000F.

Next Step: Fabricate a candidate capacitance probe for the soil.

10/04/2022--

Update: Per 10/3, in double-checking cap measurement wiring and code - one must set the charge and discharge pins to INPUT while discharging / charging the capacitor. Can't leave them both in OUTPUT mode at the same time. So the alogrithm for capacitance meter sketch is (see @ https://www.arduino.cc/en/Tutorial/Foundations/CapacitanceMeter):

Set discharge pin to INPUT (so it can't discharge the capacitor)

Record the start time with millis()

Set charge pin to OUTPUT and make it HIGH

Check the voltage repeatedly in a loop until it gets to 63.2% of total voltage.

After the cap is charged, subtract the current time from the start time to find out how long the capacitor took to charge.

Divide the Time in seconds by the charging Resistance in ohms to find the Capacitance.

Report the value (with serial.print or otherwise)

Discharge the capacitor. To do this:

Set the charge pin to Input

Set the discharge pin to OUTPUT and make it LOW

Read the voltage to make sure the capacitor is fully discharged

Loop and do it again

Result: **At first blush, this version of the code appears to be working, from a logic path-execution perspective**. The cap measurement result is wrong, as I did expect because I don't even know I'm using for resistance values or even if I'm calculating the 63% thing correctly. But the display did cycling through charging / measurement / discharging. But after which I'm only seeing 'Success' and 'Error' text in the status message field; so I do need to remove those so I can more clearly see, now, the measurement process.

BUT: From what I'm seeing I do think that after the initial cycle, the cap is not discharging, so it's stuck in State #3.

What I should do next is place the voltage info into the 'Capacitance' field; and put 'State' into the chargeTime field.

Result: I did the above. Behavior is that it initiates the discharge cycle; and I can see the voltage starts at the top, 1023; then very slowly falls down to 1008/1009 and holds at that point. So gotta figure out why that is so.

GIT: NOTE - I figure that I have now learned enough to now consider myself in the 'building a sensor prototype' stage. So I have created a new working folder for the 'Capacitance Moisture Sensor' prototype; and put that folder under git version control. My intent is put all new assets pertaining to the remaining build-out of this first prototype into this location, and under git version management. This folder is at: `/home/jroc/Dropbox /projects/MoistureSensor/CapSensor`.

RPi Software: `RPi_CapDataReceive.cpp` <- Using Git, so this should be the same till further notice.

ATTiny84 Software: `tiny84_CapMeasureAndTx` <- Using Git, so this should be the same till further notice.

10/03/2022 --

Update: **Somewhat Solved**. I rolled the tiny84 sketch back to just reading the voltage in order to get back to a known good working state, and with just the simple voltage read function. This is sketch: `MS-07_CapMeasureAndTx_01-i01`. When I uncomment the pin setups for capacitor charge/discharge, with no other changes, then this sketch appears to be acting as if the cap measure function is the void loop() function. This is occurring when the first line under the comment: "/* Init capacitance measurement pins." is uncommented. Un-commenting the other three lines is fine, the code works just fine with those, as long as that first line is commented out. The four lines in this troublesome code are:

```
/* Init pines needed to manage & measure the capacitor. */

//pinMode(chargePin, OUTPUT);     // set chargePin to output |<- !FAILS IF
UNCOMMENTED! (VER 00-05)

digitalWrite(chargePin, LOW);    // Set chargePin LOW |<- WORKS OK UNCOMMENTED
(VER 00-02)

pinMode(dischargePin, OUTPUT);   // Set discharge pin to output |<- WORKS OK
UNCOMMENTED (VER 00-03)

digitalWrite(dischargePin, LOW); // Init discharge pin to LOW |<- WORKS OK
UNCOMMENTED (VER 00-04)
```

The failure mode for the above is that both LEDs are lit, stay lit for the delay(5000) I have at the start of

setup(); then there is a very brief (sub-second) blinking out of both LEDs simultanously, then they both light up again....repeat endlessly. I get NO nRF24 transmissions over to the RPi (the RPi times out, as designed, after the 12 seconds I coded it to do when no packets received from the tiny84). My guess is that this LED pattern is telling me that the CPU is continuously rebooting.

I think I have it. I initilized both the charge and discharge pins to an OUTPUT value of LOW. When I, instead, initilize the charge pin to a value of HIGH the sketch runs OK. So my thought is that setting both pins at once to LOW creates an internally seen short-circut condition - which the CPU trapped and self-initated a reset. So this code now works, in terms of the tiny84 taking the intended execution path; looping through loop(), with a call out to void handleCapacitor() each loop cycle, and successful Tx over to the RPi, which shows the latest updates to the tiny84's data structure:

```
/* Init pines needed to manage & measure the capacitor. */

pinMode(chargePin, OUTPUT); // set chargePin to output

//digitalWrite(chargePin, LOW); // Set chargePin LOW <-- can't have both pins LOW at once!

digitalWrite(chargePin, HIGH); // Set chargePin HIGH

pinMode(dischargePin, OUTPUT); // Set discharge pin to output

digitalWrite(dischargePin, LOW); // Init discharge pin to LOW
```

BUT this then begs the question as to why this would be so. You can definitely have two pins set to LOW at once. And, hmmm..., don't have I have to set them both to LOW in order to discharge the cap? **So I'm going to have to go back to the cap measurement example and double check everything; including my breadboard wiring and pin assignments**.

RPi Software: `MS-07_CapDataReceive_01.cpp`.

ATTiny84 Software: `MS-07_CapMeasureAndTx_01-i01`.

10/02/2022 --

Update: No Go. When I tried to implement cap reading, at first the sketch was running, and it seems all I needed to do was work the correct params for getting a good calculation. But then I discovered that I still had the lines of code for initializing the cap charge/discharge commented out. So I uncommented them. As soon as I did that the sketch failed to work properly. What I saw on the RPi console with only the very first, initial, transmission from the top of the loop() function - the one that only had the data structure's initilized data in it, with "Initial" in the status message. Then absolutely nothing else appeared on the RPi console. The LED behavior gave the impression that the tiny84 was looping endlessly through the capacitor() function and not the loop() function.

RPi Software: `MS-07_CapDataReceive_01.cpp`. On this date, this is simply a copy of `MS-06_VoltsRead_01.cpp` without any any changes. Seems my console display of incoming data from the Tiny84 is beginning to stabilize.

ATTiny84 Software: `MS-07_CapMeasureAndTx_01-i01`.

Result: xx.