5. Transmit/Receive Rich Data Structure (e.g., capacitance)

Purpose: Have the ATTiny84 send a richer data structure to the RPi, a structure suitable for the capacitance moisture measurement, and have the RPi show these values on the terminal.

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."

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/nRF24-with-Rpi/CapDataReceive_02.cpp pi@192.168.1.118:/home/rf24libs/RF24/examples_linux/
```

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

06/27/2022 --

Finally! Figured it out. The 'garbage' data bring seen on the RPi side is not due to any difference in endian-ess. It is due to differing ways that the two platforms (ATTiny/Arduino and RPi) pack/unpack bytes into data structures. After a lot of hours - many of those working out how to get C++ to show me the raw bytes of variables, how to get cout to display them, and how to effect a "manual" loading of variables from raw bytes - I was able to send the dummy data structure from the ATTiny84, have the RPi receive it as a raw unit8_t [] array, and use a function I created to load the received bytes into a C struct that matches the one created on the ATTiny84 side.

The code that works for this is:

ATTiny Arduino Code: /CapMeasureAndTransmit_02

RPi Code: CapDataReceive_02c.cpp

Also for reference, see my posting about this issue on the rf24libs github "Add Info Regarding Byte Alignment or Packing Mismatches" @ https://github.com/nRF24/RF24/issues/847

06/18/2022 --

RPi Software. Starting from acknowledgementPayloads.cpp I coded up the Pi side, the receiver side. This is `CapDataReceive_02`.

ATTiny84 Software. Starting from AckPayloads_jrrMod I coded up the tiny84 side. This is `CapMeasureAndTransmit_02`. At this milestone this does NOT actually measure any capacitor. All I'm doing here is hard-coding some dummy data just to get the code data structures and architecture right.

Result for the Day:

Success!. A constant, rapid, stream of RPi scrolling on the SSH terminal window "...success..." and showing the dummy data I am sending out from the tiny84.

EXCEPT tho - the chargeTime isn't being reported out by the RPi correctly. I set this value to 0 in the tiny84; but the RPi keeps showing it as `charge time: 754974720.`

Investigation. Spent a fair bit of time trying to work this out; including adding a new float variable both before and after the chargeTime variable in the tiny84 transmit payload structure. When I did that the first float and the chargeTime both showed up correctly on the RPi side with my set values of "0." But now testFloat2 was out of wack. Then when I revised the tiny84 side to increment testFloat1 by 0.1 on each successful transmit cycle it also began to display with wacky values on the RPi side - tho chargeTime continued to properly display as "0."

Speculation: I can't really transmit data values across the airwaves with nRF24? Meaning that you have to convert them to text (e.g., HEX char strings) and send as text, then convert back on the other side. In a quick google search I did find this statement, which must be true, and would support my speculation: "The nRF24 just sends and receives bytes. It is up to the programs to interpret the bytes correctly."

Long thread about endian conversion. @ https://stackoverflow.com/questions/105252/how-do-i-convert-between-big-endian-and-little-endian-values-in-c. This may be a good quick& dirty

to try:

```
/ can be used for short, unsigned short, word, unsigned word (2-
byte types)
#define BYTESWAP16(n) (((n&0xFF00)>>8)|((n&0x00FF)<<8))

// can be used for int or unsigned int or float (4-byte types)
#define BYTESWAP32(n) ((BYTESWAP16((n&0xFFFF0000)>>16))|
((BYTESWAP16(n&0x0000FFFF))<<16))

// can be used for unsigned long long or double (8-byte types)
#define BYTESWAP64(n) ((BYTESWAP32((n&0xFFFFFFFF00000000)>>32))|
((BYTESWAP32(n&0x00000000FFFFFFFF))<<32))
```

06/17/2022 --

Milestone #3 Working Again. OK, I now have Milestone #3 working again. I might be that on 6/15 I was using the wrong sketch on the tiny84 side. I think I was running the original 'GettingStarted_jrrMod' sketch when the proper sketch to run for Milestone #3 is 'rf24ATtiny84_2rdTry_TxOnly.' Starting with rf24ATtiny84_2rdTry_TxOnly I created a slight variation to leverage the two LEDs, and my 'errorLED' subroutine. This new sketch is '`BlindTransmit`.'

BlindTransmit does report an error - RED LED on steady. This means that on the tiny84 after a transmit attempt the call to bool report = radio.write(&payload, sizeof(payload)); returns false. But in BlindTransmit I ignore that and just keep transmitting. In the scanner on the RPi I am now seeing decent hit-rates for carrier signal on channel 4C.

New Info. So I think this has also now given me a little piece of additional info - 'Report' will return false if either the nRF24 itself believes it wasn't able to send the data out for some reason OR it did send it out, but it didn't get an ACK packet back that it expected to.

Milestone #4 Working Again. But with a lot of transmission failures and time-outs. Which, maybe was true at the original run of Milestone #4 but I just wasn't paying as much attention to that fact at that point. Maybe the errors have to do with the fact that in the current code base the tiny84 side is running in low power mode. Or maybe it has to do with timing and/or delays I have in my code. But at least we're back to having some sort of back and forth conversation now.

I used AckPayloads_jrrMod_2. For this re-do of Milestone #4 I modified the tiny84 side code to use the two LEDs and my now standard errorLED routine. So this test used `AckPayloads_jrrMod_2` on the tiny84 side.

AckPayloads_jrrMod. The original I used back in Milestone #4. This does give far, far, fewer errors. So there must be some timing difference between the two programs. Tho even this one has periods of errors. Maybe due to some periodic confluence of timing delays on both side side of the code (tiny84 and RPi)? Maybe some rf interference caused by some other device? Something to watch for.

Radio 0 for RPi. Note that when running the RPi side code, the `acknowledgementPayloads` code, you must choose 'Radio 0' when it prompts for that.

Re-Coding the RPi Side. Starting from acknowledgementPayloads.cpp I coded up the Pi side, the receiver side. This is `CapDataReceive_02`.

I haven't coded up the tiny84 side yet. But just for grins I ran my newly coded CapDataReceive_02 program on the RPi and just fired up the tiny84 running the AckPayloads_jrrMod sketch. There was a back-and-forth, with the RPi showing garbage for the capacitance data, as of course it should since what it was actually receiving was the "Hello" and an integer number. But there was a steady back-and-forth between the two radios, so that is encouraging.

Result for the Day:

Hypothesis As to the 6/15 Failure. I think a few mental lapses caused the failures. First, on the RPi side I started from the manualAcknowledgements.cpp source code, while on the tiny84 I started from AckPayloads_jrrMod. These two don't match up. I should have started from the **acknowledgementPayloads.cpp** code. Then on rolling back I was also mixing up the paring of the code to run on the tiny84 / RPi. And to boot I had made some tweaks for the LED blinking so I could get better signalling on the tiny84 side of which error condition I was in.

Begin Again. So now it's time to "...begin again..." This time I will start with the code on the RPi side, starting from **`acknowledgementPayloads.cpp`**. On the tiny84 side I'll start with **`AckPayloads_jrrMod.ino`**.

Coded the Raspberry Pi Capacitance Receive Code.

06/15/2022 --

Software.

Upload the CapMeasureAndTransmit sketch to the ATTiny and see if it works.

Result for the Day:

Failure. Now nothing is working. There was a brief moment, after rolling back to milestone #4, that the two were talking to each other - tho there were a lot of error transmissions (i.e., the RPi seeing those "xxxx" packets from the tiny84); but then the RPi reported 'silence for over 6 seconds. After that point I've not been able to get anything working. I've rolled all the way back to milestone #3 - just running the scanner on RPi to see if I can even just see a signal from the ATTiny84, using the original '`GettingStarted_jrrMod`' sketch on the tiny84. Nothing. I've swapped out the nRF24 chips on both the tiny84 and RPi - nothing. I've used a new ATTiny84 chip - nothing. No clue what's going on.

In fact even the scanner running in the RPi froze and I had to reboot the Pi.

As of 2:19pm: I put a new nRF24 cip on the RPi and am running the scanner, while the 'GettingStarted_jrrMod sketch on the tiny84 - the scanner is now spitting out lines in the ssh terminal, and it ocassionally shows a signal hit of '1' for the transmitting tiny84. But mostly I'm getting '-'; no signal detected from the tiny84. The scanner is showing the usual hits from what I believe is the 2.4Ghz wifi band. Which would seem to say to me that the problem is on the ATTiny84 transmit side.

06/14/2022 --

Hardware Setup.

Using the reference design/approach from https://www.arduino.cc/en/Tutorial/Foundations/CapacitanceMeter I created a capacitor tester on a breadboad and hooked into into the ATTiny. I added a potentiometer so I could vary the calculated capacitance in order to be able to simulate that in my two transmit / receive programs.

Software.

For the ATTiny84 transmit side I started with the 'Conversation' code from my milestone #4, and modified it using the example code from the above key reference.

But - I didn't add the code to actually measure capacitance. To start with I am just hard-coding in values just to get the two nodes talking to each other and the RPi showing me the dummy received capacitance data structure it gets from the tiny84. I just want to get this part working to start with.

For the RPi side I started with the 'manualAcknowledgements.cpp code and modified it to receive and display the capacitance data structure the tiny84 is now sending out.

Result for the Day:

Created and successfully compiled the RPi side code: CapMeasureAndTransmit_Rx.cpp

Created the tiny84 side code, and successfully compiled it in the Arduino IDE, but did not upload to the ATTiny yet.

06/00/2022 --

Software.

Upload the CapMeasureAndTransmit sketch to the ATTiny and see if it works.

Result for the Day:

xxxx