6. Measure Voltage on Analog Pin

Purpose: Figure out how to use an analog pin to measure voltage - which we'll need to do in order to measure capacitance.

Goal: From the monitor on the Raspberry Pi, see the value of a voltage applied to one analog pin.

Key References --

SpenceKonde Arduino Core for ATTiny family:

https://github.com/SpenceKonde/ATTinyCore

Sample code showing how to measure voltage on an analog pin:

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

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`

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/MS-
06_VoltsRead_01.cpp pi@192.168.1.118:/home/rf24libs/RF24/examples_linux/
```

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

09/29/2022 --

Update: **Success!** I created a simple voltage divider, using a pot, and connected the output voltage to ATTiny84 physical pin #6. This is "analog channel" #7. In Arduino sketch `MS-06_VoltsReadAndTx_01-i01` I reference the analog as per the below code snip, and on the Raspberry Pi console output I do get an integer

value that varies nicely as I adjust the pot.

```
#define analogPin A7 // Attempting form implied by line 65 on pins_arduino.h
at SpenceKonde github. | Physical pin#6 - analog pin for measuring voltage
```

and then, to read the voltage:

```
void getVolts() {

    uint32_t iVolts;
    iVolts = analogRead(analogPin);
    txPayload.capacitance = 0; // calculated capacitance
    txPayload.chargeTime = iVolts; // The volts on pin #6, as converted by
    ATTiny into integer relative value
    memcpy(txPayload.units, "---", 3); // capacitance units
    memcpy(txPayload.statusText, "Volts ", 8); // status message

}
```

RPi Software: `MS-06_VoltsRead_01.cpp`

NOTE: The tx/rx success/fail rate thing. What this success has shown me is that the TX's have an actualy high success rate. What I'm seeing as a high 'failure' count is the ATTiny not receiving acknowledgement responses back from the Pi. But dispite this, as I turn the pot I see the voltage TX values change on the RPi console screen nearly instantaneously. So that tells me the transmissions from the tiny84, and receipt by the RPi, **are** reliable in the current setup.

Key Learning: In digging into the SpenceKonde attiny core github, as well as a bunch of Google searches for how to use the analog pins, I have learned that you have to reference the analog pins as "channels," using the 'channel number.' And as with the digital pins, how you reference them in the Arduino IDE code depends on how the core you are using for the ATTiny chip has defined (i.e., exposed) them to the Arduino platform. It appears this is done via a header file that the arduino IDE uses, which may be called "pins_arduino.h," tho I don't know where it's located. I am assuming it gets installed somewhere as part of the initial process of downloading and installing the 'core,' which I did way back when in getting myself set up with the Arduino UNO as a programmer, the IDE and burning the SpenceKonde core onto my ATTiny84. Now, all that being said, the documentation is still confusing to me on just how to define and reference the analog pins in the code. On the SpenceKonde github, in the (huge!) readme file it talks about using a macro. And in pins_arduino.h there are variable definitions for them, e.g.: | `static const uint8_t A7 = ADC_CH(7)`. So I did expect a lot of trial-and-error on this. But just using 'A7' worked the first time. And that is, after all, implied by the `static const uint8_t A7` definition in the header file. `ADC_CH(7)` is a macro in the SpenceKonde core, per documention - see below notes for 09/27/2022.

09/27/2022 --

Update: Back from Instanbul, back from the Washington, DC trip. House cleaned. So now getting back to this.

Started by googling around for how to measure voltage using ATtiny84. Eventually went back to the 'core' I am using for Arduino IDE/coding [https://github.com/SpenceKonde/ATTinyCore] and found the following statement in the README.txt file:

> **ADC Support** || ATTinyCore 2.0.0 introduces a major enhancement to the handling of analog and digital pin numbers: Now, in all the #defined constants that refer to an analog channel, the high bit is set. (ie, ADC channel 4, A4, is defined by a line #define A4 (0x80 | 4)); (actually, we also define ADC_CH() macro as shorthand for the bitwise or with 0x80. This advantage of this that it makes it more obvious why we're doing this to the number; if you see (0x80 | 4) you'd be like "wtf is this for? what does 0x80 have to do with anything?", whereas if you hadn't read this, and you saw ADC_CH(4) - you might not know exactly what's going on, but just from the name you'd know it was something to do with an analog reading, maybe of channel 4). Because all the analog channel number defines are all distinct from things that aren't analog channel numbers, the core's analogRead and digitalRead functions can tell the two apart; digitalRead(A3) will now look up what digital pin analog channel 3 is on, and use digitalRead on that, while analogRead(7) will now go look up what analog channel is on digital pin 7, and use analogRead on that.

xxxx

07/23/2022 --

Update: Between 7/8 and today I have been fiddling about with the software on both sides, but primarily on the RPi side - to improve my debugging capability. In MS-06_CapDataReceive_06.cpp on the RPi side I have been able to implement a 'static' display of the received packets from the tiny84 so that the screen doesn't scroll; and in the tiny84 I am keeping track of the 'success' and 'errors' and passing those counters in the packets. So I have been able to see the relative rate of success vs errors, as seen by the tiny84. What I've seen so far is that it varies, seemingly randomly. Sometimes I run the tiny84 and I get mostly successes, sometimes mostly errors, sometimes an even balance. I can't really discern, so far, why the variation; yet in my head it still "feels" like a timing-sync thing.
RPi Software: On this date I created MS-06_CapDataReceive_07.cpp, in which I created a class to handle the receive packet display. I ended up doing this as part of working out how to move the cursor around on the console output; including how to best deal with the first time listing out so that I don't overwrite the lines output from the 'prettyprint' config info of the nRF24 chip. And, really, because I just wanted to re-learn how to create a class object in C++. But I haven't yet copied this code over to the RPi and tried to compile it. Saving that for after I get back from the Istanbul trip.

07/08/2022 --

RPi Software: ATTiny84 Software: MS-06_CapMeasureAndTx_04, Increment #1. Simply modify the TxPayload struct for the fields we'll use for the live capacator measurement.
Result: Works. However, what I mostly get from the ATTiny84 is "ERROR 3" status messages back. So for my next increment I'd like to smooth out the back and forth timing so that I'm getting almost all "Success." In doing this I feel I need to insert some delay on the tiny84 side. One, to simulate the time delay of making a cap measurement. And, two, to perhaps allow the two devices to better settle into a back and forth cycle.

07/08/2022 --

RPi Software: MS-06_CapDataReceive_04.cpp.
ATTiny84 Software: CapMeasureAndTransmit_03.
Result: No Go. I could not get this to work at all. The RPi side was picking up no transmissions at all. And Scanner also didn't seen any RF signal in the air for the channel this was supposedly transmitting on. So I rolled back to CapMeasureAndTransmit_02 - dummy data only - and confirmed that version's transmissions were being picked up by my MS-06_CapDataReceive_04.cpp program on the RPi side.
So I have rolled back to CapMeasureAndTransmit_02 and will start all over again - tho this time taking it in smaller increments, not advancing until each increment is shown to be working. So the new starting point is MS-06_CapMeasureAndTx_04.