3. nRF24 on ATTiny84 - Blind Data Send

Purpose: Having successfully got the nRF24 working on a Raspberry Pi, now it's time to get it working on the ATTiny84. (See log entry titled "**1. Install or Update RF24 Libraries and Repo on Raspberry Pi.**")

Goal: ATTiny84 continuously sending out a byte of data, on a given channel, and those transmissions being picked up by the nRF24 connected to the Raspberry Pi.

Key References --

What I finally discovered on 3/25/2022 is the below site is the key resource for the nRF24:

nRF24 / RF24 github repo @ https://github.com/nRF24/RF24

However, you have to be careful about the PIN mappings and connections. There are subtle variations by device and by which document you are reading. Some of the documentation isn't "aware" that a PIN or two on the ATTiny84 needs to be mapped differently than, say, an Arduino or ATTiny85, etc.

I have created a worksheet that, for me, holds the pin mapping that actually works when using an ATTiny84. See the document in this repo titled: **PinMappings.ods**.

Another key reference is the so-called Arduino Core for the ATTiny family. Above reference says the code there is based on the 'SpenceKonde' core. The home for this is @ https://github.com/SpenceKonde/ATTinyCore. See the readme file there for documentation, including how to reference pins inside the code that match up to the processor physical pins. It says to use the Port and Number as reference. E.g., ATTiny84 physical pin 12 would be referenced as 'PA-1' in the code. As in: `const int ledPin = PIN_PA1; // the pin that the led is wired to`

"Addressing" with the RF24's is confusing to me. So called 'pipes' are opened for reading and writing; and there are notes in the Arduino RF24.h file about addresses associated with these pipes. So far I have not been able to work out what's really going on. In RF24.h there was a reference to the below web page re addressing:

Improve RF24 Radio Performance With Proper Addressing Schemes @ http://maniacalbits.blogspot.com/2013/04/rf24-addressing-nrf24l01-radios-require.html

04/26/2022 --

First light on the ATTiny!

There is a long story about creating a costume version of the scanner.cpp program on the Raspberry Pi and getting that to compile and link. Many days past, but with the help of Brendan over at the github repo @ https://github.com/nRF24/RF24 I started from scratch and reinstalled and built the RF24 libaries and code base on the Pi. And now I am able to write and compile my own programs for the RF24 on the Pi.

See log entry: "**1. Install or Update RF24 Libraries and Repo on Raspberry Pi.**" about all that.

So I:

* Created a modified version of scanner.cpp, named scanner_ch4C-highlighted.cpp, so that it highlights the channel my RF24 modules are transmitting on - channel 0x4C.

* Created a new sketch in Arduino IDE for the ATTiny to have it continuously transmit - this sketch is named: rf24ATtiny84_2rdTry_TxOnly.

* Ran my modified scanner program - and now I'm seeing the scanner show data, consistently, on channel 0x4C.

04/22/2022 --

I switched the nRF24 chip modules between the Pi and the ATTiny84 to see what the 'scanner' program on the pi says the RF channel is set to: for both it is set to 0x4c = 76.

The nRF24 module on the pi shows the following for it's configuration:

```
================ NRF Configuration ================
STATUS = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=7 TX_FULL=0
RX_ADDR_P0-1 = 0xe7e7e7e7e7 0xc2c2c2c2c2
RX_ADDR_P2-5 = 0xc3 0xc4 0xc5 0xc6
TX_ADDR = 0xe7e7e7e7e7
RX_PW_P0-6 = 0x20 0x20 0x20 0x20 0x20 0x20
EN_AA = 0x00
EN_RXADDR = 0x03
RF_CH = 0x4c
RF_SETUP = 0x07
CONFIG = 0x0e
DYNPD/FEATURE = 0x00 0x00
Data Rate = 1 MBPS
Model = nRF24L01+
CRC Length = 16 bits
PA Power = PA_MAX
ARC = 0
```

04/21/2022 --

I have made a 2nd nRF24 carrier board that I can stick into a breadboard. I've done that and, I believe, wired it up to a 'zero-insertion force' socket also on the breadboard that will hold the ATTiny84 after I've written the Arduino program to it. So with this arrangement I can easily move the '84 from the Arduino programmer to the 'in service' breadboard as I keep trying stuff.

Now I'll just try to compile, write, and run the 'rf24pingAttiny84' Arduino sketch which I have obtained from the above referenced Key References. Tho I'd like to add a blinking LED just so I can see that the program is running on the '84. However, I can't seem to actually find a good reference that tells me the pin mapping for the 'core' I am using. So what I'm going to do is use the pin reference that I used in the 'blinky' sketch I first wrote to the ATTiny to prove I could program it. The code for blinking the LED in that sketch is:

```
const int ledPin = PIN_PA1; // the pin that the led is wired to
// the setup function runs once when you press reset or power the board
void setup() {
// initialize LED digital pin as an output.
pinMode(ledPin, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
digitalWrite(ledPin, HIGH); // turn the LED on (HIGH is the voltage level)
delay(1000); // wait for a second
digitalWrite(ledPin, LOW); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
}
```

Ok, first attempt did not work. The LED I added in is not lighting. I have the nRF24 that is connected to the pi up and running the scanner program; but it doesn't see any transmitted data from the ATTiny.

I modified the sketch to add a unique LED flash pattern in the code where it goes into an infinte loop if the code is unable to initilize the nRF24 radio. And now I am getting the flashing LED pattern. So on my ATTiny breadboard, something is amiss in terms of the '84 being able to see the radio. Most likely, of course, would be the wiring.

So - The code and libraries I am using, per the Key Reference at the top of this note, say to use the "" Arduino core for the ATTiny family of processors. And that is what I used right from the get-go. Now, the documentation for that core, to get the pin number references to use in the code, is at: https://github.com/SpenceKonde/ATTinyCore . Which I am now adding to the Key References section as well.

> So with this I'm now going to try to rename the pin references, as described in the section of the Readme doc, "Refer to pins by port/pin" and see if I can get it working.

Alright, I've changed two pin references in my code -

```
#define CE_PIN PIN_PA2

#define CSN_PIN PIN_PA3
```

And now I'm getting some different behaviour from the LED. A steady, even, flashing. Tho I can't quite tell from how I coded the LED to flash what it tells me about which parts of the code it's passing through; tho it appears to me that the blinking is due to looping in the 'transmit' section of the loop.

However, I'm not seeing anything on the Pi's scanner terminal that would indicate that it is seeing any transmitted data. Reasons I can think of for this:

> * No data actually being sent out.
>
> * Data is being sent out; but not in any manner that the scanner can 'see.'
>
> * Data is being sent out, scanner can see it, but it's in a band or 'channel' that overlaps with my WiFi, wireless mouse, or bluetooth such that I can't see it.
>
> * Data is being sent out, but the interval it is being sent out on isn't matching up to the intervals that scanner is performing it's channel-scan. If I wait long enough I'll see it - tho how i'd differentiate it from noise I don't know.

It is very much unclear to me what/ where/ how in the Arduino code the "channel" is set for transmissions (in such a way that it'd match up to what the scanner is scanning for). There is a reference to "`radioNumber = 1`," "`uint8_t address[][6] = {"1Node", "2Node"};`," and "`memcpy(payload.message, "Hello ", 6); // set the outgoing message`" <- does that '6' there mean channel 6?

A bit of reverse engineering of the code and figure out what is really going on in terms of what data is actually being sent out, and how the 'channel' is specified. This page has a lot of information that could be a good start - @ https://arduinoinfo.mywikis.net/wiki/Nrf24L01-2.4GHz-HowTo

> In particular it says this:
>
> > `radio.setChannel(108);`
> >
> > Which RF channel to communicate on, 0-124 Can operate on frequencies from 2.400GHz to 2.524GHz.
> >
> > Programming resolution of channel frequency is 1Mhz This is the same unlicensed band WiFi operates in (WiFi uses 2.400 to 2.500 gHz). Usually frequencies above channel 100 are best. NOTE: In most countries the allowed frequencies are from 2.400GHz to 2.483.5GHz which you must not exceed. In USA it's best to use channels from 70 to 80. You can scan the channels in your environment to find a channel that is clear... See the Scanner sketch.
>
> In looking through the sketch's code I don't see any call to `radio.setChannel()`; nor do I see any default being set in the 'RF24.cpp' object that initializes and uses the radio.

> When I look at the header info that the scanner program is running on the raspberry pi, it does show this: `RF_CH = 0x4c`, Hex 4C is 76. And that reference "`RF_CH`" is used in RF24.cpp, in `void RF24::setChannel(uint8_t channel)`, as the chip's register which holds the channel number. So I'm thinking that the nRF24 ships with a default channel set in it. And that this channel is channel 76. Tho of course they could all be randomly set. One thing I could do is swap my two nRF24 chips between the R-Pi and ATtiny84 to see if they both report the same channel.

I'm going to call this a day. Next time I think I'll reverse the chips to see what the scanner reports as the channel for the chip I'm using on the ATTiny. Then I'll see if I can mod the scanner code to more clearly show the channel the ATTiny's radio is broadcasting on. And then from there perhaps focus the scanner on that channel and have it scan more frequently. And also I should mod the transmit sketch to have it transmit only, bypassing the section where it goes into listen mode.

.