Tutorial 3: Working with Multiple Pipes

Written by Brennen Ball, © 2007

Introduction

In this tutorial, I am going to show you how to set up your nRF24L01 to use all 6 of its pipes. This could come in handy for those of you who wish to set up mesh networks and many other applications. Just like before, it is very important that you be familiar with the previous tutorials (Tutorial 0 and 1, in particular) before you continue on.

What the Heck Are Pipes?

Pipes are pretty easy to understand. Each pipe is essentially just an address that you can receive data on (when in RX mode). This means that any 24L01 can actually act as 6 different receivers at the same time!

So what good is that for you, you might ask? There are several uses that you could use multiple pipes for, but there are a few that come to my mind immediately. You could set up one pipe to be a data channel and another to be a control channel. You might also have a receiver that did several different functions, and would do the task that corresponds to the channel it received data on. Your imagination is the limit.

There are a couple of things to keep in mind when you are using pipes. First, **all pipes use the same RF channel** (whatever the channel is set in the RF_CH register). If you are using Enhanced Shockburst, the transmitter's TX address and pipe 0 receive address (TX_ADDR and RX_ADDR_P0, respectively) must be the same to get auto-acknowledgements properly. This can be a pain since every time you want to send data to a different address you have to change two addresses, but with the nRF24L01 library it only takes one more line of code to change the RX_ADDR_P0 register if you're already changing the TX address. Just as an aside, this tutorial does not use Enhanced Shockburst just to illustrate things as simply as possible.

The Ins and Outs

In this tutorial, everything is setup exactly as in Tutorials 1 & 2. This includes connecting one node (uC and 24L01) up to a serial port on your PC at 9600-N-1 and no flow control. The other node is totally standalone.

In this tutorial, our aim is to ping each pipe on the remote unit from the local unit and report the statistics. In operation, the local unit will set its TX address to the RX address of pipe 0 on the remote unit. It will then send one byte and wait for the byte to be returned. If the returned byte is the same as the sent byte, then we have a successful ping. This is done five times, and the statistics are recorded. This entire process is then done sequentially for each of the other 5 pipes on the 24L01. Finally, the overall statistics are

calculated and the 24L01 prints the statistics for each of the data pipes and the overall tally to the screen.

The Hardware I Used for this Tutorial

I used the exact same hardware in this tutorial as in Tutorials 1 & 2. Nothing added, nothing subtracted, so as long as you were able to get those guys working, you should certainly be able to get this one going.

Tutorial Software Description/Requirements

Similar to the hardware required, the software for this tutorial requires the same support as in Tutorials 1 & 2. If you were able to get them going, then you're in the clear for this tutorial.

Local Main File: maintutorial2local.c

This is our main driver file for the local unit. It is based off the local main file found in Tutorial 1, but there are a considerable number of changes in this one to allow for the use of multiple pipes.

First off, the header files and the defines for Jim Lynch's functions are still there and are the same. Beginning at line 34, you can see a few additions. First is the Ping() function, which does exactly what you would imagine – it sends the receiver data (1 byte) and then waits on the receiver to return the data. The code in the function is actually the main code from Tutorial 1, except that it instead of reading the data byte to send from the UART, the byte 0xFF is sent. I have also included defines to set up how many pings you will use in lines 36 and 37 to allow you to set how many pings per pipe to send and how many seconds to wait between pinging sessions, respectively.

Once in the main function, we set up several variables to act as counters, storage for successful pings for each pipe, the current TX address, and the total of the successful pings. The trusty old Initialize() function is called, and we set up the local 24L01 the same as in Tutorial 1 (TX, 1 data byte, no Enhanced Shockburst).

After the call to Initialize(), we get into the main program loop. First, we clear the number of successful pings for each pipe to zero. Next, we print a message to let the user know we are beginning the pinging process.

The following for loop is the loop that will execute the pings for each pipe. The loop will execute 6 times, once for each pipe. First, the switch statement takes the iteration we're on and sets the local unit's TX address to the appropriate RX pipe address of the remote unit. Notice that the full address of pipes 0 and 1 are set, but only the least-significant byte (LSB) of the address for pipes 2 through 5 is sent, since they share the same most-significant bytes with pipe 1 and the address for pipe 1 is already loaded into TX_ADDR. After the TX address is set, the pipe is pinged the number of times specified in NUM_PINGS (default is 5) and if the ping is successful, the counter is incremented. Finally, the statistics for the pipe are printed out.

Once all of the pipes have been pinged, we print out a message showing how many successful pings we had versus total pings. Then we toggle the onboard LED to

indicate the loop has completed and we delay the number of seconds in PING_DELAY. The main loop is then repeated.

Remote Main File: maintutorial2remote.c

Similar to Tutorial 2, maintutorial3remote.c is almost identical to the one used in Tutorial 1. The only difference here is that we enable all six data pipes so that we can communicate with them all.

You can find this difference on lines 71 through 91. This is where we call the nrf24l01_initialize() function. In the first line of the call, we set up the CONFIG register up for CRC enabled/1-byte, powered up, and RX. The next line is "true" so that we can enable RX functionality immediately. The third line sets up the pipes with no auto-ack, and the fourth line enables all data pipes. We leave all remaining registers in their default states except for the RX_PW_P* registers. We set all of them up with a 1-byte payload so that we can receive data on them.

As for the program, if you didn't read Tutorial 1, here is a rundown. The program waits until it receives data at the 24L01. When it does, it sends the data back to the local node's pipe 0. It then loops back around and waits for data again.

Remembering that this remote node can receive data on all pipes, it is possible to process data differently depending on the pipe it was received on (it is not done in this example, however). This is possible to do with my library, since the function nrf24l01_read_rx_payload() returns the value of the status register, which contains the data pipe number for the current payload. Using this value, you can then use the function nrf24l01_get_rx_pipe_from_status() to return the value of the pipe from the payload. Consider the following code as an example:

```
unsigned char data, status, pipe;

status = nrf24l01_read_rx_payload(&data, 1);
pipe = nrf24l01_get_rx_pipe_from_status(status);

switch(pipe)
{
case 0:
      //process case 0
      break;
case 1:
      //process case 1
      break;
case 2:
      //process case 2
      break;
default:
      //process any other pipe
}
```

This code will allow you to read a data payload (it assumes that one exists – you should add code to ensure that there indeed is some data in the RX FIFO) and get the current value of the status register. You then get the pipe number based on the status register, and use a switch statement to process the data. I am a big fan of using switch

statements when using discrete number values, but you might not be. If not, you could also use if statements or any of the other niceties built into the C language.

Actual Program Output

In Figure 1, you can see the window in AbsoluteTelnet that shows the program working. The 24L01s are in very close proximity to one another (around 6" apart), so there is an extremely high percentage of successful packets (the same as in Tutorials 1 & 2). You can see the statistics for each channel and overall for each of two sessions.

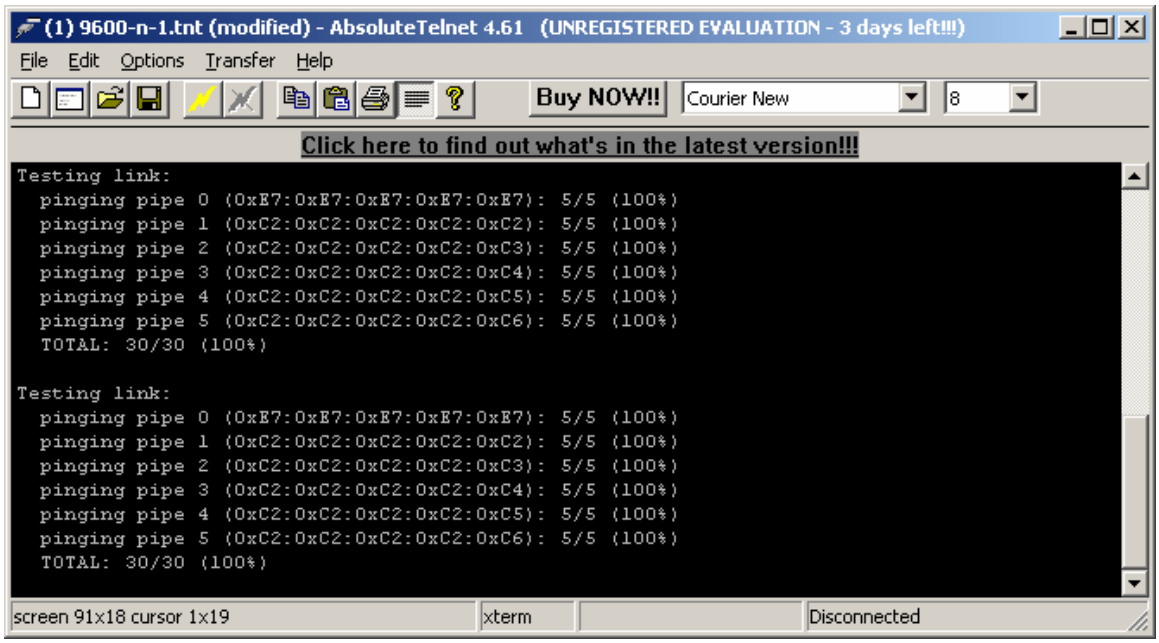

Figure 1. Output window from AbsoluteTelnet

Concluding Remarks

At this point, you should know how to communicate on any or all of the 6 RX data pipes that the 24L01 offers. If you have any questions, feel free to email me at brennen@diyembedded.com.