Wood, Erica
CS49N
12/8/21

CS49N Final Project Writeup

What I Did

The primary goal of my project was to implement a networking system between two nrf24L01+ transceivers. One Raspberry Pi controls one transceiver and a second controls the other. In addition, one of the Raspberry Pi's is hooked up to four buttons and a joystick, such that data from those inputs is then sent using the transceiver to the other Raspberry Pi's transceiver, where the Raspberry Pi processes it and outputs it to the light strip. There are red, green, and blue buttons, each of which controls its respective output color on the light strip (clicking each button once increments the RGB output of the light strip by a set factor). There is also a grey button, which turns the light strip and the system off when the user is finished. The value from the joystick (only using the y-axis), which utilizes the analog to digital converter, ranges from 0 to 100, each representing a percent of the light strip's LEDs that are turned on.

I used the CS 140E nrf24L01p lab as the backbone of my NRF code. While I did use some of the prebuilt functions for assigning specific values to registers, I spent a significant amount of time reading about possible values for the various registers in the source documentation.

Challenges

- I spent a long time working on sending packets with acknowledgments. After the send, it uses a while loop to prepare to receive the acknowledgement. However, I spent a long time trying to debug an issue where it would receive the acknowledgement, then stay in the loop because I forgot to break afterwards. This was a challenging issue to debug because the first send would go through, then the send from the client side would get stuck at max interrupt. However, it wasn't obvious that the reason it wasn't getting received was because it had gone back into TX mode after getting the acknowledgement and hadn't ever reached the line putting it back into RX mode.
- I also had a problem where I was trying to use put8 to set the addresses. This was unsuccessful because the addresses are all three bytes, not one.
- When I was wiring up the buttons, I was surprised to learn that they default to an "on" state, so if you want unpressed to be 0 and pressed to be 1, you have to invert the output. You also have to pull up the pins.
- At first, I took a period of samples with the joystick to ensure that it wouldn't be a noisy reading. However, this created issues when it led to blocking with the transceivers and larger delays from reading the buttons to them being output on the light strip (a full second, and sometimes the button wouldn't register). I switched it to not taking the average of several samples, instead giving just one joystick output. The readings weren't overly noisy and led to a much more consistent output overall.

Quickstart Guide

1. Follow the steps in the CS140E nrf24L01p lab to get the transceiver working. This will take a significant amount of time, but I think it is unnecessary to rewrite the steps. Make sure to write the function to get packets before writing the function to send with acknowledgements, as the latter calls the former. I found the diagram on page 22 very helpful for visualizing how the chip

works. I didn't use standby II, only standby I. I would also recommend using the initialization settings from the staff binaries as a starting point to work off of.

2. Wire up a joystick. I used the Adafruit Analog Thumb Joystick and hooked it up to a breadboard using a similar wiring pattern to the one from the ADC lab.
3. Update your ADC code to return the joystick value. Specifically, determine the minimum and maximum outputs, then scale them based on those values so that the joystick ranges from 0-100 (not using negatives will be easier for transmitting data). Remember that you can't divide by variables, so you will have to hard code a constant factor there. (You can do this either in a separate file, or, what I did, which is just add another function to the existing one).
4. Wire up four buttons. You don't hook up voltage to these, only a pin and ground, on opposite sides of the four pins (diagonals). Also, you want these to straddle the middle of the breadboard.
5. Write a button.c and button.h that takes the pin and both structures it for inputting a button value (set input and pullup – might need to update gpio.c for this as well) and for getting the value from the Raspberry Pi and converting it into the desired form.
6. Write a package_data.c and package_data.h that controls taking button and joystick input and formatting it into a 2-byte packet. I recommend sending the button values rather than the desired RGB color, since it is a smaller packet. Also, only send a packet if the values have changed. This should protect you from adding more increments than desired to the color (along with clearing out the button section of the packet after each receive).
7. Write an unpack.c and an unpack.h that unpacks the values sent over the transceiver (that were packaged in the previous step). You might want a struct representing an RGB color that you can use to manage all of your RGB values.
8. Write lightstrip.c and lightstrip.h for controlling the light strip, using the neopixel interface.
9. Write a client.c and server.c (based on 2-client-pingpong.c and 2-server-pingpong.c) that controls the buttons, joystick, NRFs, and light strip.