ECE353 In-Class Exercise

SPI Bus

Problem Objectives

- Initialize a SPI peripheral to a specified data rate and SPI mode
- Implement routines that send data to the Nordic NRF24L01+ register set

Overview

This in class exercise will require you to work with a partner to wirelessly transmit data using the Nordic nRF24L01+ radio and the ECE353 development platform. Most of the driver for the nRF24L01+ has been provided for you.

The TM4C123 can access registers in the nRF24L01+ by sending formatted packets of data via the SPI interface. The format of the data packets is specified in the nRF24L01+ data sheet. We will only be implementing 5 of the different data packets.

When writing data to the nRF24L01+, we must transmit one of the specified write commands followed by the data we want to write. Any data returned will be discarded.

When reading data to the nRF24L01+, we must transmit one of the specific read command followed by the number of bytes we want to read. The contents of the data being transmitted **after** the command can be anything.

1. Run the Demo

You can find the Demo executable in the demo directory. **Make sure you do not re-compile the demo project or the executable will be lost!**

You will need to perform this lab in groups of two. Each student should load the demo project on their board. The demo project requires you to provide a unique ID for each board. Use the last two digits of the PCs you and your partner are using to ensure both of your boards have a unique set of IDs.

The first time you run the DEMO, you will need to set the ID of each board using the serial debug interface. Once set, the IDs for both boards should be printed out on the LCD screen. If you need to change the ID of the demo board, press the up directional button. This will generate a message on the serial debug port to enter a set of new IDs for the demo board.

The Demo program also allows you to toggle between transmitting and receiving mode. You can toggle between the two modes by pressing the down button. You will need to press the down button on one of the boards in order to place one board into receive mode. After you have done this, you should be able to observe that one board is transmitting data and the other is receiving data.

For the remainder of the lab, one board should continue to use the Demo image. You will develop software to test on the other board. It is important that you use matching device IDs in order to get the boards to communicate correctly.

2. Modify main.c

Modify the arrays `myID` so that the 5th byte matches the last two digits of the PC you are working on. Also, set the 4th byte to your favorite 2-digit, non-zero, hexadecimal number (for additional uniqueness). Share this information with your partner and have them set the `remoteID` to your value of `myID`. (Your `remoteID` should match your partners `myID`) . DO NOT modify the first three bytes! The demo code expects that the first three bytes are '3', '5', '3'.

3. Modify `initialize_spi()` in `spi.c`

A. Set the SPI clock so that it operates at 10MHz
B. Set the SPI device so that it is configured for the SPI mode that is passed into the function.

4. Modify `wireless.c`

A. Complete **`wireless_reg_read()`.**
B. Complete **`wireless_reg_write()`.**
C. Complete **`wireless_set_tx_addr()`.**
D. Complete **`wireless_tx_data_payload()`.**
E. Complete **`wireless_rx_data_payload()`.**

5. Observe Receiving Data

In order to observe the correct output, you and another person will need to communicate using a pair of Nordic nRF24L01+ radios. Once both have been programmed, observe that the Demo board transmits a count once per second. The device running your code will receive the data and print it out to the serial debug interface.

6. Observe Transmitting Data

Modify main.c by setting **`tx_mode`** to be true. Press the down directional button on the demo board to place the demo board into receive mode. Once the device running your code has been re-programmed, observe that the Demo receives data once per second.

7. What to Turn In

Turn in **`wireless.c`** to the dropbox on the course website.

NOTES:

Most of the functions in **`wireless.c`** are declared as **`static __INLINE.`** The **`static`** keyword makes these functions private to the file in which they are written. Only functions written in **`wireless.c`** have access to these functions. It helps to protect the interworking's of the driver code from the rest of the application.

The **`__INLINE`** keyword tells the compiler that code found in the function should be inserted directly into the function that calls it. This helps to reduce branches (increased performance) at the cost of increasing the overall size of the application image. Performance is increased by removing branch instructions and possibly stack operations. The application image gets larger because each time the inline function is called, the assembler inserts a copy of the resulting assembly code directly into the calling function.