ECE381 – Lab 6 – SPI Radio

DUE: 28 March 2023

Introduction:

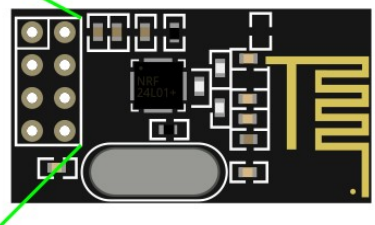
Learn the SPI bus protocol and programming by interfacing with an NRF24 wireless radio. You will configure the radio as specified to send and receive a commands consisting of an LED color and PWM rate to and from neighboring groups.

Hardware Configuration:

Connect the following pins for the NRF24 Radio:

- P6[0]: MOSI
- P6[1]: MISO
- P6[2]: SCK
- P2[7]: CSN
- P5[3]: CE
- P5[5]: IRQ

Top View:	
GND	VCC
CE	CSN
SCK	MOSI
MISO	IRQ



The LCD should be connected on P2[6:0], the Encoder on P[1:0], a button on P4[5], and the RGBY LEDs on P3[1:0] and P0[3:2].

REMEMBER: VCC on the NRF24 is 3.3V NOT 5V!!! The pins ARE 5V compatible though.

PSoC Configuration:

- Place a Communications/SPI (SCB) Module. Right-click to configure. Set the mode to "Master". Set the Data rate to 1 Mbps. Click on the Advanced tab. Set the RX and TX buffer sizes to minimum 17 bytes (I would recommend 32 bytes 'cause powers of 2 are more fun!). You might also want to rename it to SPI or NRF24.

- Verify that MISO, MOSI, SCK, and CSN are on P6[2:0] and P2[7] in the Pins dialog, CE is on P5[3], and IRQ is on P5[5].
- Place a Display/Character LCD. Right-click to configure, change the baud rate to 57000 and click “OK”
- Connect Channel A & Channel B of your encoder to P0[0] and P0[1] respectively.
- Connect a push-button to P4[5]. (It can’t be on Port0, since the encoder will be using that interrupt vector)
- Place a “Digital Input Pin” onto the schematic. Right-click and select “Configure”. Rename the pin to ENC. Set the number of pins to be 2. Uncheck “HW connection”. Click on the Input tab and change the Interrupt to “Either Edge”, “Rising Edge”, or “Falling Edge”, depending on how you implement your Encoder detection logic. Make sure you check “Dedicated Interrupt”. If you are not using an external pull-down resistor, change the drive mode to be “Resistive Pull-Down” for BOTH pins. Click OK.
- From System, grab the Interrupt and drag it into the schematic window. Connect the irq pin on ENC to the input of the Interrupt module. Right-click on the Interrupt module and select “Configure”. Rename the module to ENC_ISR, make sure the InterruptType is DERIVED, and click OK.
- Place a “Digital Input Pin” onto the schematic. Right-click and select “Configure”. Rename the pin to BTN. Set the number of pins to be 1. Uncheck “HW connection”. Click on the Input tab and change the Interrupt to “Rising Edge”, or “Falling Edge”, depending on what kind of pull-up/pull-down resistor you use. Make sure you check “Dedicated Interrupt”.
- Place a “Digital/Functions/PWM (TCPWM mode)”. Place a “System/Clock” and connect it to the input.

Software

A skeleton project should be given for you. The basic outline of the project goes like this:

1. Turn on all the Modules (SPI, LCD, ISRs, etc)
    a. Make sure CE is low to begin with as well.
    b. You should also delay for 100ms for the radio to settle
2. Call the Receiver() function to put your NRF24 in RX mode for your assigned address & channel
3. Poll for the switch, while using the Encoder to enter a target Group ID on the LCD
    a. IF IRQ goes low while polling for the switch, someone sent you a command!
        i. Get the message from the NRF24
        ii. Parse the LED color and PWM duty cycle from it
        iii. Select your R/G/B/Y LED to output that PWM and turn it on using the target duty cycle
        iv. You don’t have to mix or cycle through colors here, just output the last received color and duty cycle
4. IF the user enters a valid group ID:
    a. Use the encoder to select ‘R’, ‘G’, ‘B’, or ‘Y’ on the LCD until the user presses the button.

    b. After the users presses the button, keep the choice on the LCD, then let the user choose a number between 0 to 100 for the PWM duty cycle on the LCD (you can split rows if needed or input on the same row)
    c. IF IRQ goes low while polling for the switch, someone sent you a command!
        i. Get the message from the NRF24
        ii. Parse the LED color and PWM rate from it
        iii. Select your R/G/B/Y LED to output that PWM
        iv. You don't have to mix or cycle through colors here, just output the last received color and duty cycle
5. After the user enters a valid color and PWM duty cycle
    a. Call the Transmitter() function to put the NRF24 in TX mode
    b. Send the selected color and PWM rate to the target radio in the format "R/G/B/Y *PWM_Rate(0-100)*" where R/G/B/Y is a single char, followed by a space, followed by a number from 0 to 100 in ASCII. For example:
    "R 65"
    would be the command to tell the target group to turn on their red LED with a PWM duty cycle of 65%. The command:
    "B 80"
    would be the command the tell the target group to turn on their blue LED with a PWM duty cycle of 80%
    c. Once you transmit the payload, wait for IRQ to go low. Once it does, verify whether the command was sent successfully or whether it failed.
    d. Notify the user of successful transmission or failure on the LCD and delay 1 second
    e. Go back to Step 2 to get a new target group and repeat the process

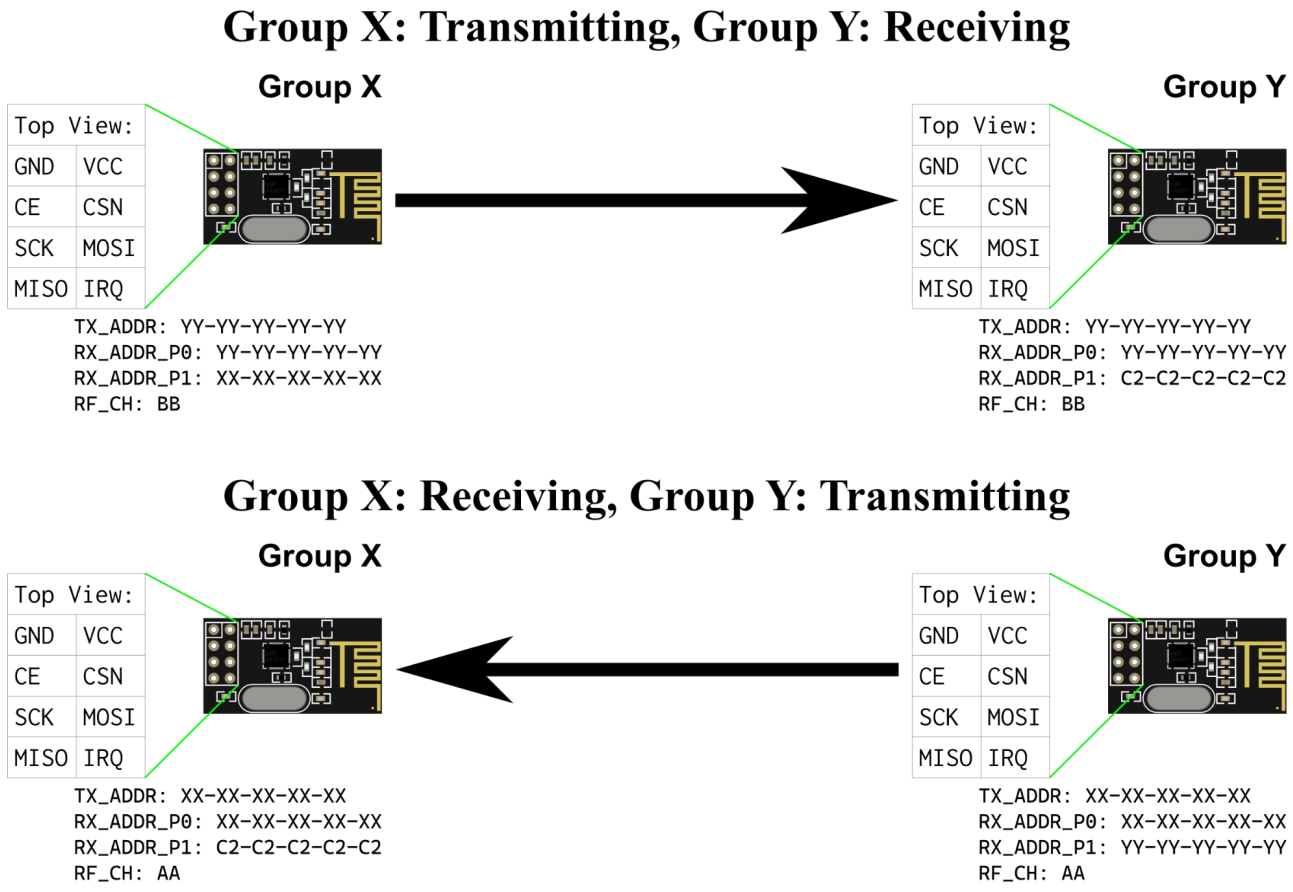
The NRF24 radios are controlled through the SPI interface. Configuration is done by writing specific values in registers defined in Chapter 9 of the NRF24L01+ datasheet. Pg. 51 of the datasheet tells us that to modify a register, the command (1st SPI byte on wire) is 001A AAAA, where A AAAA is the 5-bit register address. To read the value of a register back, the command is 000A AAAA, where A AAAA is again the 5-bit register address. Some registers are 1 byte, some are multi-byte, so the values that need to be written are register specific.

Since the configuration registers control things like RF channel, error checking size, data rate, etc., it is crucial that both sending and receiving radios match in those parameters in order to be successful. Therefore, to ensure everyone is using consistent configuration information, both Transmitter() and Receiver() functions should:

- Set the number or retransmissions to 15 with a 4000 us delay between retransmissions
- Set the data rate to 250 kbps at maximum power level
- Set the payload size to be fixed at 16 bytes
- Turn on auto-acknowledge for your data pipes
- Turn on your data pipes

- Use 2-byte CRCs for the auto-acknowledges

What will be different for each device is the address and channel. Each group will be assigned an address and channel. The channel shouldn't overlap with another group (to minimize interference). The address is used to allow multiple transmitters to talk to the same receiver. Therefore, one of the two RX pipe addresses should at all times be your assigned address. The difference of which one depends on if your radio is transmitting (TX mode) or receiving (RX mode), which are controlled by calling the Transmitter() or Receiver() functions respectively. If you are receiving, the Receiver() function should set both your TX_ADDR and RX_ADDR_P0 to your assigned address, and the RF channel should be your assigned channel. If you have a message to send, then you will put your radio in TX mode by calling the Transmitter() function, which should set TX_ADDR and RX_ADDR_P0 to the target group's address (and for sanity's sake, set the RX_ADDR_P1 to your group's assigned address so you can still receive messages even while transmitting). The figure below shows an example, with XX-XX-XX-XX-XX as the address of Group X, YY-YY-YY-YY-YY as the address of Group Y, AA as the channel for Group X, and BB as the channel for Group Y:



The address C2-C2-C2-C2-C2 will be used as a broadcast address (as it is the default TX address on reset) for RX_ADDR_P1 in receiver mode.

The next thing to sort is how to actually send and receive. This is where the IRQ pin comes into play. IRQ should normally be high. When you need to transmit a message, you use the W_TX_PAYLOAD command, followed by your message (16 bytes for us since we are using fixed 16 byte payloads). One of two things will happen. When you transmit, your NRF24 should compute a 2-byte CRC on your message before sending, sends this CRC along with the message, and starts a timer. The receiver, when it detects a transmission on its channel, should also compute the same 2-byte CRC on whatever it receives. If the two CRCs match, that means that there were no bit errors in transmission, so the receiver will respond with an ACK message to sender (why both TX_ADDR & RX_ADDR_P0 must match, as it uses RX_ADDR_P0 for this ACK). If the sender gets the ACK, it will pull IRQ low and set the TX_DS bit in its status register, indicating successful transmission. If, however, the receiver doesn't receive correctly, it won't send an ACK. If the sender doesn't see an ACK before Auto Retransmit Delay (ARD), it will resend the message and wait again. It will do this Auto Retransmit Count (ARC) times. If it still doesn't get an ACK after all timeouts and restransmits have been exhausted, it will also pull IRQ low, but this time set the MAX_RT bit in the status register. All of this happens behind the scenes, so all your PSoC sees is IRQ going low. To validate, on the TX side, if the message is sent correctly, you need to read the status register, and mask off the appropriate bit. To reset IRQ high again, you need to clear those flag bits in status by writing 1s to them.

On the RX side, IRQ is also used to indicate a message has been received. If IRQ goes low in RX mode, the RX_DR bit gets set in the status register, and the message is ready to be read from the NRF24. To extract the message, the receiver needs to send the R_RX_PAYLOAD command, followed by some dummy bytes to get the message from the NRF24. After this, the RX_DR flag needs to be cleared by writing a 1 to that bit in the status register. That should pull IRQ back high and your radio can go back to waiting for a new message.

Notes

- SPI, while faster than I2C, is still much slower than the CPU clock. The SPI_SpiUartPutArray() function in the API, however, only blocks until the last byte has been put in the TX buffer. Therefore, you will still have to poll until completion to be sure SPI is finished. There IS a bit in one of the SCB registers that indicates if the SPI bus is busy, AND you can check it with the SPI_SpiIsBusBusy() function call. Apparently, however, it can take up to two SCK cycles before the busy bit is set, so if you immediately poll for busy, it might return false! Therefore, after all SPI_SpiUartPutArray() calls, you should have:

  ```
  while(SPI_SpiIsBusBusy() == 0);
  while(SPI_SpiIsBusBusy() == 1);
  ```

  The first loop waits for busy to be set in case it isn't due to the 2 SCK delay, the second loop will wait until the bus is actually finished being used, which should be when CS goes high.
- With SPI, there is a byte received for every byte sent. Even if you aren't checking a result, it would also be wise to call the SPI_SpiUartClearRxBuffer() function before sending any SPI data. This way, if you do need to read SPI data, you will know how many

SPI_SpiUartReadRxData() calls to make to get your target data, since SPI_SpiUartClearRxBuffer() starts the RX buffer over at 0.

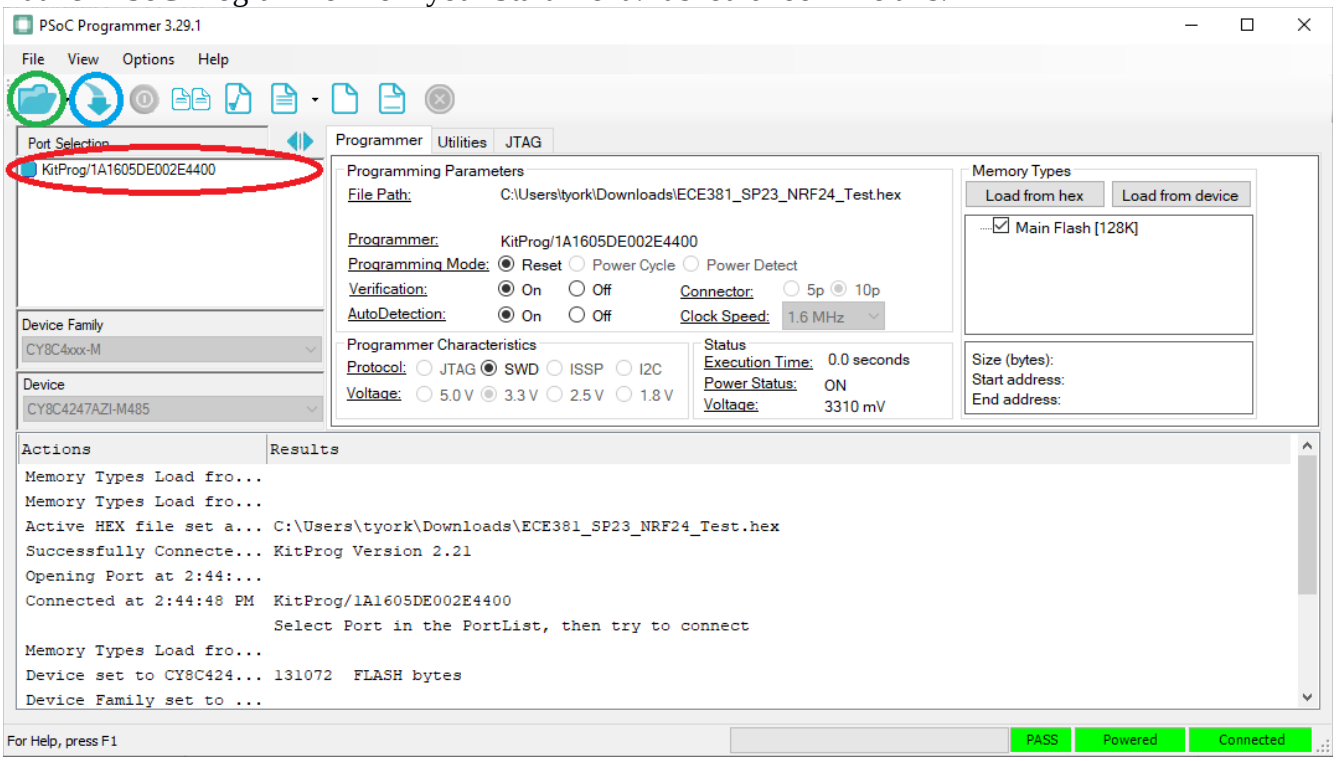
- In the same vein, the NRF24 both the TX& RX internal FIFOs should be cleared when switching modes using the radio commands 0xE1 and 0xE2 as per pg 51 of the datasheet
- Pay attention to the radio control diagram on pg. 23 in the NRF24L01+ datasheet. Changing modes requires manipulation of CE, as well as the PRIM_RX bit in CONFIG. It also specifies minimum timings that must be followed when changing modes. Build these into your functions using CyDelay() and CyDelayUs().
- You should probably break out your pins to the breadboard for potential oscilloscope debugging (especially IRQ!) instead of directly connecting them to the pin headers (I know, it's tempting…)

Requirements:

- Be able to send a command to target group using NRF24 radio to change their LED color and brightness
- Be able to receive commands using the NRF24 radio and change your own LED color from any group

PSoC Programmer

A PSoC with an NRF24 radio will be left at my station up front for testing. Additionally, a working version will be provided as a hex file for you to test your own radio and hardware. To flash your PSoC with the hex file, you use a program called "PSoC Programmer", which is separate from PSoC Creator. Launch PSoC Programmer from your Start Menu. It should look like this:



1. Click on the KitProg device first (circled in red) to select the target PSOC
2. Click on the folder (circled in green) and select the .hex file you wish to flash
3. Click on the arrow (circled in blue) to flash that to your PSoC

If the green PASS shows after flashing, your PSoC should now be running the hex file.