Assignment 3 A SPI-based 1-wire Device Driver for LED Strip (150 points)

Part 1: A SPI device driver for LED strip (100 points)

In this assignment, you are required to implement a SPI device driver which can accept pixel information from users and light up 16 RGB LEDs of a LED ring. The LEDs of the ring are connected serially and use a 1-wire communication protocol to receive pixel data for full color display (8 bits for each R, G, and B color).

The 16 bit LED ring module is integrated with WS2812 drivers. The 16 WS2812 drivers are in cascade connection to receive and latch 24 bits color data and to drive LED display. Using different pulse durations of a 1-wire serial protocol, a controller can reset the WS2812 drivers and send data bits (0 or 1) to the drivers. The data transmission speed is around 800Kbps. The datasheet of WS2812 can be found at https://cdn-shop.adafruit.com/datasheets/WS2812.pdf.

To connect the LED ring to Galileo Gen 2 board, you need to solder 3 jump wires to GND, Vcc (3.3V), and Data_in. The Data_in should be connected to SPI_MOSI of spidev1.0 via IO11 of Arduino connector. The ring should be registered as a SPI device of Galileo Gen 2 board, i.e., by defining its *spi_board_info* and invoking *spi_register_board_info()* function. It should be initialized by the probe function of the WS2812 driver once a match of device and driver is found.

The WS2812 driver should be implemented as a spi driver and enable the following device file operations:

- *open:* to open a device (the device is "/dev/*WS2812*").
- *write:* The write call sends *n* pixel data to light up the *n* LEDs of the ring, where $1 \le n \le 16$. Each pixel data consists of 3 bytes for green, red, and blue colors. The call initiates an asynchronous SPI transfer that leads to a proper 1-wire data transmission on SPI-MOSI. As a consequence, the WS2812 can receive the display data and drive the LED display accurately.
- *Ioctl:* to include one command "RESET" to reset the SPI to a suitable operation mode. Note that the SPI mode must be defined properly to make correct 1-wire data transmission. The IO pin multiplexing should also be reset to enable the connection from SPI_MOSI to Data_in of the ring.
- *release:* to close the descriptor of an opened device file.

You also need to implement a user-level testing program which uses the WS2812 driver to show a circular display pattern continuously. To design the WS2812 driver, you may want to refer to the following existing programs in Linux source code for Galileo Gen 2:

1. intel_qrk_plat_galileo_gen2.c
2. adc1x8s102.c
3. spidev.c

Part 2: Accurate Delays in Linux (50 points)

Rather than the spi-based approach to drive 1-wire communication with WS2812, an intuitive approach is something similar to so called "*bit banging",* i.e., using software and delay loops to control a gpio output signal to generate any waveform. For instance, to generate a pulse of *5µsec*, a program can

write 1 to a gpio pin, wait for *5µsec*, and finally write 0 to the gpio pin. This approach is highly dependent of the accuracy of delay functions and any possible preemptions.

In this part of assignment, please consider two delay mechanisms, ndelay() and hrtimer, for a possible implementation of bit-banging approach to drive WS2812-based LED strip. You may need to do few experiments to verify that the timing requirement of WS2812 can be met by any of these two approaches. If yes, you should develop an alternate WS2812 driver that uses gpio12 (IO1 of Arduino connector) to connect with Data_in pin of the LED ring.

The submission of part 2 of the assignment should include:

1. Program(s) to verify the timing property using ndelay() and hrtimer.
2. A pdf report and experiment results to discuss whether the timing of WS2812 can be met.
3. An alternate WS2812 driver if bit-banging approach is feasible.

Due Date

The due date for both parts is 11:59pm, Nov. 16. Please note that this is a firm deadline. The assignment 4 will be announced immediately after the due date.

What to Turn in for Grading

- Create working directories, named "EOSI-teamX-assgn03_partN", for the assignment to include your source files (.c and .h), makefile(s), readme, and report (for part 2). Compress the directories into a zip archive file named EOSI-teamX-assgn03.zip. Note that any object code or temporary build files should not be included in the submission. Submit the zip archive to Blackboard by the due date and time.
- Please make sure that you comment the source files properly and the readme file includes a description about how to make and use your software. A sample result from your test run can be included in readme file. Don't forget to add each team member's name and ASU id in the readme file.
- There will be 20 points penalty per day if the submission is late. Note that submissions are time stamped by Blackboard. If you have multiple submissions, only the newest one will be graded. If needed, you can send an email to the instructor and TA to drop a submission.
- Your team must work on the assignment without any help from other teams and is responsible to the submission in Blackboard. No collaboration between teams is allowed, except the open discussion in the forum on Blackboard.
- Failure to follow these instructions may cause deduction of points.
- Here are few general rule for deductions:
    - No make file or compilation error -- 0 point for the part of the assignment.
    - Must have "–Wall" flag for compilation -- 5-point deduction for each warning.
    - 10-point deduction if there is no instruction on compilation or execution in README file.
    - Source programs are not commented properly -- 10-20-point deduction.
- ASU Academic Integrity Policy (http://provost.asu.edu/academicintegrity), and FSE Honor Code (http://engineering.asu.edu/integrity) are strictly enforced and followed.