Assignment 2 Device Drivers for HC-SR04 (200 points)

Exercise Objectives

1. To learn the basic programming technique in Linux gpio kernel modules, gpio pin multiplexing, and TSC clock.
2. To learn the software structures of kernel timer and gpio interrupt
3. To learn the development skill for Linux sysfs interface and platform device initiation.

Project Assignment

In this assignment, you are required to work on HC-SR04 ultrasonic distance sensors. After sending a "ping" signal on its trigger pin, the HC-SR04 sensor sends 8 40khz square wave pulses and automatically detect whether receive any echo from a distance object. So the pulse width on the echo pin tells the time of sound traveled from the sensor to measured distance and back. The distance to the object is computed as

*distance = (pulse width * ultrasonic spreading velocity in air) / 2*

To connect each HC-SR04 sensor to Galileo Gen2 board, two digital IO pins of Arduino connector will be used for trigger and echo signals. They are configured as Linux GPIO pins. The edges on echo pin should trigger interrupts and the differences of TSC's (time-stamp counter) of x86 processor at the positive and negative edges can give a precise measure of pulse width.

Part 1: A Linux kernel module to enable user-space device interface for HC-SR04 (100 points)

The first task of the assignment is to develop a loadable kernel module which initiates instances of HC-SR04 sensors (named as HCSR_n where n=0, 1, 2,…) and allows the sensors being accessed as device files in user space. The operations to configure these devices include:

1. Configuration of trigger and echo pins of HC-SR04 to the digital IO pins of Arduino connector. The pin configuration operation must include the setting of pin multiplexing on Galileo Gen 2 board. This can be done based on the pinmux table of Galileo Gen2 board.
2. Configuration of HC-SR04 operation parameters (*m, delta*): Once a trigger for a measurement is presented, *m+2* distance samples should be collected every *delta* milliseconds. The resultant distance measurement is the average of *m* samples with the two outlier samples excluded.

The number of devices, *n,* is a command line argument passed to your module when the module is installed. To manage the devices, you will need to implement a Linux *miscellaneous character driver* in the module which provides the following file operations:

- *open:* to open a device (the device is "*HCSR_n*").
- *read:* to retrieve a distance measure (an integer in centimeter) and a timestamp saved in the per-device FIFO buffer. The timestamp is a 64-bits integer of TSC clock indicating the time that the measurement is done. If the buffer is empty, the read call is blocked until a new measurement is collected from an on-going sampling operation, or from a new measure triggered by the call.
- *write:* The write call triggers a new measurement if there is no on-going measurement operation. The argument of the write call is an integer. The existing content of the per-device buffer should be cleared if the input integer has a non-zero value, and has no effect if the value is 0. EINVAL should be returned if there is an on-going measurement.

- *Ioctl:* to include two commands "CONFIG_PINS" and "SET_PARAMETERS" for configuring the pins and setting up the operation parameters of sensor device. For "CONFIG_PINS", the arguments include two integers to specify digital IO pins of Arduino connector for the sensor's trigger and echo pins. For "SET_PARAMETERS", the arguments consist of two integers for the number of samples per measurement and the sampling period. If any of the input arguments is invalid, e.g., Linux gpio numbers is a negative number, EINVAL is returned. Note that the echo pin of HC-SR04 should be able to trigger edge interrupts (such as gpio 8 -15). It should not be configured to any non-interrupt-capable pins.
- *release:* to close the descriptor of an opened device file.

You may assume the per-device FIFO buffer can keep the most-recent 5 measures. In addition to the kernel module, you need to develop a test program for your driver. Various scenarios should be exercised, including distance measurement from multiple HC-SR04 devices concurrently.

A good reference on Linux miscellaneous character driver can be found in http://www.embeddedlinux.org.cn/essentiallinuxdevicedrivers/final/ch05lev1sec7.html. An example of misc driver can be found in https://gist.github.com/17twenty/6313566.

Part 2: Platform device driver and sysfs interface for HC-SR04

In part 1, you loadable module enables driver operations for multiple HC-SR04 sensors. In this part, your task is to develop a platform driver/platform device infrastructure for HC-SR04 sensors. The goals are:

1. Any HC-SR04 devices defined as platform devices can be instantiated and bound with a platform driver, named as "HCSR_of_driver".
2. Sysfs interface is enabled for any platform HC-SR04 devices.

Your development can be based on the Galileo Linux kernel 3.19.8 (i.e. the version that is available in the dropbox). Your implementation should end up with two loadable modules. One is to instantiate *n* HC-SR04 devices based on platform device definitions, where *n* is passed to the module as a command line argument. The other one is the driver for HC-SR04 devices where the sysfs interface are realized. The sysfs interface is defined as follows:

/sys/class/HCSR/HCSR_n/
/trigger ... the digital IO pin of Arduino Connector to HC-SR04 trigger pin
/echo ... the digital IO pin of Arduino Connector to HC-SR04 echo pin
/number_samples ... number of samples per measurement
/sampling_period … sampling period
/enable … storing 1 to start a measurement, 0 to disable measurement
/distance … the most recent distance measure

Note that the modules are loadable (they can be loaded in any order). To test the sysfs interface, you will need to develop a Bash script file to access HC-SR04 devices.

Due Date

The due date is 11:59pm, March 20.

What to Turn in for Grading

What to Turn in for Grading

- Create a working directory, named "EOSI-LastName-FirstInitial-assgn02", that consists of two sub-directories, "part1" and "part2", to include your source files (.c and .h), makefile(s), and readme.
- Compress the directory into a zip archive file named EOSI-LastName-FirstInitial-assgn02.zip and submit the zip archive to Canvas by the due date. Note that any object code or temporary build files should not be included in the submission.
- 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 your 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 Canvas. 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.
- The assignment must be done individually. No collaboration is allowed, except the open discussion in the forum on Canvas. The instructor reserves the right to ask any student to explain the work and adjust the grade accordingly.
- 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 no compilation or execution instruction 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.