Assignment 2 A Device Driver for HC-SR04 (200 points)

Exercise Objectives

1. To learn the basic programming technique in Linux gpio kernel modules.
2. To learn the software structures of kernel thread 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 on the echo pin width tells the time of sound traveled from the sensor to measured distance and back. The distance to the object is

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

To connect each HC-SR04 sensor to Galileo Gen2 board, two digital IO pins, configured as Linux GPIO pins, of Arduino Connector will be used for trigger and echo signals.

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 two instances of HC-SR04 sensors (named as HCSR_1 and HCSR_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 to two Linux GPIO pins.
2. Configuration of HC-SR04 operation mode: the sensor can be in one-shot mode where one distance sample is collected upon a request. The other operation mode is to sample the distance periodically (specified by a given frequency).

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_1 or HCSR_2").
- *read:* to retrieve a distance measure (an integer in centimeter) saved in the per-device FIFO buffer. If the buffer is empty, the read call is blocked until a new measure arrives. Note that, if there is no on-going sampling operation in on-shot mode, the device should be triggered to collect a new measure.
- *write:* In one-shot mode, the write call trigger a new measurement request if there is no on-going measurement operation. The argument of the write call is an integer. 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. For periodical sampling mode, the sampling operation gets started when the input integer is non-zero or stopped otherwise.
- *Ioctl:* two new commands "SETPINS" and "SETMODE" to configure the pins and the operation mode of the sensor. For "SETPINS", the argument includes two integers to specify Linux gpio numbers for the sensor's trigger and echo pins. If any of the input Linux gpio numbers are not valid, -1 is returned and errno is set to EINVAL. For "SETMODE", the argument consists of two integers. A zero value in the first integer sets the operation mode to "one-shot", while an one configures the mode to "periodic sampling". The 2nd integer gives the frequency of the sampling.

- *release:* to close the descriptor of an opened device file.

To allow interrupt from the echo pin of HC-SR04, you will need to connect the pin to Quark GIP GPIO port. 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 in which the pin multiplexing of Gen2 board is initialized and any distance measure should be printed out for various scenarios. A good reference on Linux miscellaneous character driver can be found in http://www.embeddedlinux.org.cn/essentiallinuxdevicedrivers/final/ch05lev1sec7.html.

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

In part 1, you loadable module enables driver operations for two hard-coded 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. User-space dev interface is enabled for any platform HC-SR04 devices. The interface is as same as what in part 1.
3. Sysfs interface is enabled for any platform HC-SR04 devices.

Your development should be based on a stable version of Linux for Galileo board from https://github.com/todorez/galileo-linux-stable. Your implementation should end up with two loadable modules. One is to instantiate HC-SR04 devices based on platform device definitions (including two HC-SR04 devices). The other one is the driver for HC-SR04 devices where the user-space dev interface and sysfs interface are realized. The sysfs interface is defined as follows:

/sys/class/HCSR/HCSRdeviceN/
/trigger ... the GPIO pin connected to HC-SR04 trigger
/echo ... the GPIO pin connected to HC-SR04 echo
/mode ... "0" for one-shot or "1" for periodic
/frequency … sampling frequency if mode is periodic
/enable … storing 1 to start measurement, 0 to disable measurement
/distance … the most recent distance measure

Note that the modules are loadable and there is no need to have your modules built into Linux kernel. However, you do need to build and use the Linux kernel from galileo-linux-stable version in this part of the assignment. The testing program for part 1 should work for the dev interface your modules create. To test the sysfs interface, you will need to develop a script file to access HC-SR04 devices.

Due Date

The assignment is due at 11:59pm, Oct. 20.

What to Turn in for Grading

- For each part, there will be 5 bonus points each day if you submit earlier (a maximum of 20 bonus points for the part) and 20 points penalty per day if the submission is late.
- Failure to follow these instructions may cause an annoyed and cranky TA or instructor to deduct points while grading your assignment.
- 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.