Assignment 2 A Nunchuck Driver with Asynchronous SMBus Transfer (100 points) – CSE 438/598, Fall 2011

Assignment Objectives
1. To learn the basic programming technique for module and device driver in Linux kernel.
2. To learn I2C driver architecture and software
3. To apply an application framework to control graphic display with nunchuck device.

Project Assignment
“i2c-dev” kernel module can be viewed as a driver to access i2C adapters. It realizes i2c bus transfer based on requests from user space. For instance, a read function call to the driver results in a read I2c transaction with a given i2c bus address on a specific I2C bus. The module also provides the interface to perform functions of SMBus protocol.

On AIMB-212 Atom board, SMBus can be accessed via pins 8 and 11 of JFP1 and JFP2 connector. SMBus is one of the peripheral interfaces of ICH8 IO control hub. To use the SMBus adapter, we will need to install i2c-i801 module. Similar to i2c-dev, you will need to use the Linux command modprobe to install the module. The two modules are in drivers/i2c/i2c-dev.ko and drivers/i2c/busses/i2c-i801.ko of the linux source tree. Once the modules are installed, you can try the commands: “i2cdetect –l”, “ls –l /sys/class/i2c-adapter”, and “ls –l /sys/class/i2c-dev” to see what are available in the system. For user space access, /usr/include/linux/i2c-dev.h should be included in your programs. When libi2c-dev package is installed, /usr/include/linux/i2c-dev.h is available for user accesses and it contained macros for SMBus protocol.

The first task of this assignment is to develop a report about i2c-dev module. We will assume that your application program opens the SMBus adapter device and makes a call to i2c_smbus_read_block_data. This leads to a sequence of function calls in the kernel. In your report, you should:

1. Show the call sequence starting from making a call to i2c_smbus_read_block_data to the return from the call, and give a short description (2-5 lines) on the function of each routine in the sequence.
2. Describe the process of creating i2c-dev devices enabled by i2c-dev module.
3. Given that an i2c bus runs with a much slow clock, a call to i2c_i801 transfer functions must wait for the completion response from a bus transfer. Describe the waiting mechanism used in i2c-i801 driver.

The second task is to develop a user application to simulate pen movement using Wii Nunchuck. The application consists of 2 threads. The first thread invokes i2c/dev interface (Linux/Documentation/i2c/dev-interface) to read Wii nunchuk data every 100ms. It then computes the new position of the pen based on the speed of pen movement which is indicated by the X- and Y-axis values of the joystick. If there are any changes on pen positions and bottom states, a message should be sent to the second thread through a message queue. The second thread is responsible to draw the trajectory of pen movement on a 2D graphic window when a new message is received. You may use Z bottom to control the color of the trajectory (i.e., switch to the next color if pressed) and C bottom to reset the graphic windows and the simulation.

To draw the trajectory, you just need to draw lines between consecutive pen positions. You may want to consider to use Qt library for the drawing function in Linux which has been used extensively to
build GUI applications for embedded systems. The QPainter class can be used to draw geometric shapes on a widget. Since Qt uses C++, you may consider developing C++ programs for the task.

The third task is to build an I2C client driver for Nunchuck such that user programs can read 6 byte data from a Nunchuck device on i2c-5M Bus. Note that the speed of SMBus is limited to 100KHz and a transfer may take milliseconds. That is, the processing of the calling thread will continue after few milliseconds. If this delay is too long to the applications, we may want to invoke a kernel worker thread to perform the requested i2c transfer and have the calling thread returns immediately.

To provide the asynchronous processing of SMBus functions, you should develop a kernel module i2c-nunchuck. A work queue will be created for when the driver module i2c-nunchuck is installed. When a read system call is requested to a Nunchuck device, instead of calling the i2c_smbus_xfer function, the driver takes one of the following operations:

- If no nunchuck data is ready for read and no pending work request, the driver submits a transfer request to the work queue and returns immediately with -2.
- If no nunchuck data is ready for read and the work request submitted last time is still in processing, the call returns immediately with -1.
- If there is a nunchuck data ready for read, return with 0 after the data is copied to the buffer given as an argument of the read function.

Due Date
This is assignment is due on Friday, Oct. 19, at 11:59pm.

What to Turn in for Grading
- Create a working directory to include your source and object files (.o and .ko), makefiles, and test results. Your source files should include the report (.doc or .pdf) for task 1, the main program main_2.cpp and any other files for task 2, the main program main_3.cpp and the driver i2c-nunchuck.c, and any other files for task 3.
- Comment your source files properly and rewrite the readme file to describe the functions of each routine.
- Compress the directory into a zip archive file named cse438-lastname-assgn02.zip.
- Submit the zip archive to Blackboard by the due date and time.
- Failure to follow these instructions may cause an annoyed and cranky TA or instructor to deduct points while grading your assignment.

Wii Nunchuk Interface:
The Wii Nunchuk uses a proprietary connector. The connector front end looks like below. The connections are as follows:

```
1 3 5
2 4 6
```

1 = +3V (3V recommended but works at 5V) 2 = Clock
3 = N/A 4 = N/A
5 = Data 6 = Gnd

An adapter is used to connect the Wii Nunchuk to any processor. The order of signals from left to right from the image above is ( Gnd – Vcc – Data – Clock).
Frequency:

The frequency used to communicate with Nunchuk is 100KHz.

Handshaking/Initialization/Read operation:

To communicate with the Nunchuk, we must send a handshake signal. So first send 2 bytes "0x40, 0x00" to initialize the Nunchuk. Then send one byte "0x00" each time you request data from the Nunchuck. The data from the Nunchuk can be read back after a short delay (approximately 100us).

Note that SMbus doesn’t allow an i2c read of 6 bytes in one transaction. You will need to read the 6 bytes in 6 separate i2c transactions.

I2C Address:

The I2C slave address of Nunchuk is 0x52.

Default Data:

The default values (6 bytes) expected from Nunchuk are as follows. For X- and Y-axis values in the first rows, Min (Full Left) means the analog stick of the Wii Nunchuk is the to the extreme left along the axis. Medium (Center) means the stick is the default position and Max (Full right) means the stick is in the extreme right position.

<table>
<thead>
<tr>
<th>Value in example</th>
<th>Description</th>
<th>Values of sample Nunchuk</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7F</td>
<td>X-axis value of the analog stick</td>
<td>Min(Full left):0x1F / Medium(Center):0x7F / Max(Full Right):0xF1</td>
</tr>
<tr>
<td>0x7B</td>
<td>Y-axis value of the analog stick</td>
<td>Min(Full Down):0x1D / Medium(Center):0x7B / Max(Full Right):0xDF</td>
</tr>
<tr>
<td>0xAF</td>
<td>X-axis acceleration value</td>
<td>Min(at 1G):0x48 / Medium(at 1G):0x7D / Max(at 1G):0xB0</td>
</tr>
<tr>
<td>0x80</td>
<td>Y-axis acceleration value</td>
<td>Min(at 1G):0x46 / Medium(at 1G):0x7A / Max(at 1G):0xAF</td>
</tr>
<tr>
<td>0x7A</td>
<td>Z-axis acceleration value</td>
<td>Min(at 1G):0x4A / Medium(at 1G):0x7E / Max(at 1G):0xB1</td>
</tr>
<tr>
<td>0x3B</td>
<td>Button state (Bits 0/1) / acceleration LSB</td>
<td>Bit 0: &quot;Z&quot;-Button (0 = pressed, 1 = released) / Bit 1: &quot;C&quot; button (0 = pressed, 1 = released) / Bits 2-3: X acceleration LSB / Bits 4-5: Y acceleration LSB / Bits 6-7: Z acceleration</td>
</tr>
</tbody>
</table>
When you read Nunchuk data, there is a coding scheme on the data. To decode the data, please calculate the data as:

\[
\text{Exact data} = (\text{Reading data XOR 0x17}) + 0x17
\]

Also the value for pressing/releasing buttons should be

```plaintext
case 0:  // Z is pressed       00
case 1:  // C is pressed       01
case 2:  // Both C and Z      10
case 3:  // Neither           11
```

Here is a pretty good resource on how to decode. [http://www.robotshop.ca/content/PDF/inex-zx-nunchuck-datasheet.pdf](http://www.robotshop.ca/content/PDF/inex-zx-nunchuck-datasheet.pdf)