Lab Assignment 4
Controlling the Wiimote with Object-Oriented Programming

Lab 4.0 Introduction

In this lab, you will use the Wiimote as an input device for the ZedBoard, obtaining information related with pressed buttons or device motion. Then, these values are used to display acceleration information on the LEDs of the ZedBoard, using an object-oriented approach. The development of these assignments is aided by the implementation of short shell scripts.

Lab 4.1 Shell scripts

System administrators and users often find themselves running an identical sequence of commands repeatedly to perform a common set of actions. For example, you might be interested in listing the content of the current directory with detailed information (command `ls -l`) and then printing the current date and time (command `date`). Instead of typing these two commands every time, it is more convenient to create a shell script containing them. We could name such script `list-and-date.sh`, and populate it with the following content:

File `list-and-date.sh`

```
ls -l
date
```

After file `list-and-date.sh` is created, it needs to be assigned execution permissions with the following command:

```
$ chmod +x list-and-date.sh
```

This command needs to be run only once after the file is created. The execution permissions are kept as part of the file properties, even if you later edit the content of the file. You can run the shell script in the same way that you would run an executable binary file:

```
$ ./list-and-date.sh
```

Pre-lab assignment

The following reading list will help you to complete the pre-lab assignment. The readings will also help you in subsequent lab assignments. Please complete the readings that are marked ***[Required]*** before attempting the pre-lab assignments.

[1] Materials on the Bash Tutorial ***[Required]***

- http://www.ansatt.hig.no/erikh/tutorial-bash/bash-notes.pdf

[2] Materials on the Bluetooth communication protocol

- Lecture 13 – Bluetooth Communication on Blackboard ***[Required]***
- http://en.wikipedia.org/wiki/Bluetooth

Complete the following assignment on the COE Linux machines using the Bash Tutorial linked above, and any other online resources on simple features of shell scripting. **Include all the code and screen captures from the following questions in one PDF file that you will submit in Blackboard.**

a) Find information on how to pass arguments to a shell script. Write a shell script named `CreateDir.sh` that takes a directory name as an argument, and creates it within the current directory. The shell script should use command `mkdir` internally.

b) Find information on how to use command echo to print messages to the terminal. Write a shell script named ShowDate.sh that outputs a message like this, using commands echo and date:

```
The current date is Sun Feb  7 12:37:22 EST 2016
```

Use the manual pages to find out more about all options available in command echo, including the one used to avoid printing a newline character at the end of a message.

c) Complete the assignment (started in class) of converting "ZedBoardLeds.cpp" into ZedBoard.h ZedBoard.cpp, zedMain.cpp, and a Makefile. Copy/paste your files in the submission file.

d) Study the code in section "Lab 4.4 Reading button events in C++" below and convert it into a C++ class WiimoteBtns.cpp as outlined in section "Lab 4.5". It does **not** have to compile.

e) Study the code in section "Lab 4.6 Reading acceleration events in C++" below and convert it into a C++ class WiimoteAccel as outlined in section "Lab 4.7". It does **not** have to compile.

Lab 4.2 Instructions to set up the Wiimote

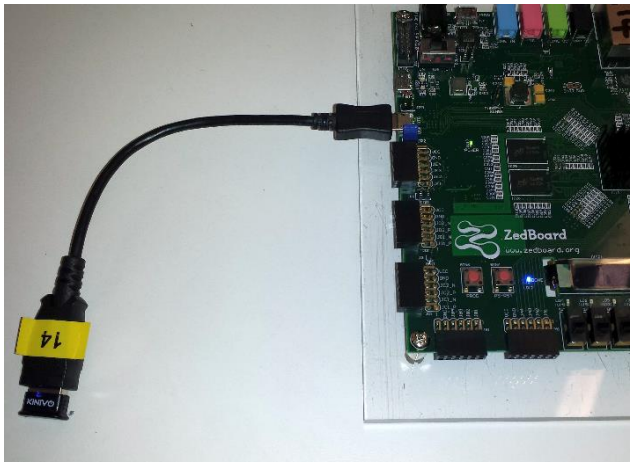
The bulk of this lab session focuses on the introduction of a new hardware device: the Wii remote, or Wiimote. You will use a USB Bluetooth dongle to establish a Bluetooth connection between the Wiimote and the ZedBoard. The Linux on the ZedBoard uses the BlueZ stack (http://ww.bluez.org), which is an implementation of the Bluetooth standard

The following steps guide you through the configuration process.

1. Making the USB Bluetooth Dongle Operational

- Connect the USB Bluetooth dongle to the standard USB-to-microUSB adapter. Then connect the

USB-to-microUSB adapter to the ZedBoard on the port labelled “USB OTG”, as shown in the figure.



- Run command `hciconfig -a` on the ZedBoard to see the status of the Bluetooth adapter on the platform, and try to interpret the information reported.
- If the device shows up in state DOWN (it should be UP by default), run:

  ```
  sudo hciconfig hci0 up
  ```

  to turn it on. Run:

  ```
  hciconfig -a
  ```

  again to make sure the device is up. Record the MAC address of the Bluetooth adapter connected to the Zedboard. This is an identifier composed of 6 pairs of hexadecimal digits separated by : (for example, 00:02:72:CA:F2:44), labeled BD Address.
  - Write down this MAC address associated with your Bluetooth adapter.

2. Pair the Bluetooth adapter on the board with the Wiimote to create a Bluetooth wireless connection.
   - First, set the Wiimote in pairing mode, so that it can be found by the Bluetooth adapter on the ZedBoard. Follow these steps:
     - Remove the battery cover and push the red sync button.
     - The four blue LEDs on the Wiimote should be blinking, indicating the device is in pairing mode.
     - Pairing mode is active for about 20 seconds. If you miss this window, press the red button again.
   - Run command

     ```
     hcitool scan
     ```

     The Wiimote should show up as Nintendo RVL-CNT-01, together with a MAC address of type XX:XX:XX:XX:XX:XX. Note that you will see any Wiimote that is trying to sync at the same time. Make sure you identify yours by checking the MAC address on the Wiimote’s box.

3. To make your first connection:
   - Put the Wiimote into pairing mode (see Step 2).
   - Run command

     ```
     sudo bluez-simple-agent hci0 XX:XX:XX:XX:XX:XX
     ```

     where XX:XX:XX:XX:XX:XX is the Wiimote MAC address written on the Wiimote’s box. Run this command while the Wiimote is in paring mode. Make sure the four blue LEDs on the Wiimote are blinking when you run it. If not, push the red sync button and run the command again.
   - Write down this MAC address, associated with your Wiimote. Now you have recorded two MAC addresses: one for your Bluetooth adapter, and another one for the Wiimote.

- If you complete the previous step correctly, you may be asked for a PIN Code. Press *Enter* without entering any PIN code to cancel pairing. This will generate a message reporting an authentication failure. This is expected; the next steps will deal with the creation of a trusted connection without authentication.

4. Some tricks to avoid needing to input the PIN each time.

- Directory `/var/lib/bluetooth/` lists all Bluetooth adapters known to the operating system. Change the directory (using cd) to `/var/lib/bluetooth/<BT_Adapter_MAC>`. Replace <BT_Adapter_MAC> with the MAC address of your Bluetooth adapter. List the directory contents. The directory contains files regarding Bluetooth devices that are or have been connected to this adapter.
- Run command

```
cd /var/lib/bluetooth/XX:XX:XX:XX:XX:XX
```

  where XX:XX:XX:XX:XX:XX is the MAC address of your Bluetooth Adapter.

- Modify the contents of the file did, to be able to connect to the Wiimote without entering a PIN code. Open the did file in an editor, e.g. sudo vi did. Find a line that looks like this:

```
XX:XX:XX:XX:XX:XX FFFF 0000 0000 0000
```

  where XX:XX:XX:XX:XX:XX is the Wiimote MAC address. Change the last 16 characters, starting with FFFF in that line with 0002 057E 0306 8500. For example, if your Wiimote MAC address is 8C:56:C5:40:00:D0, the line would be

```
8C:56:C5:40:00:D0 0002 057E 0306 8500.
```

  Save the file and exit. Go to your home directory by typing `cd`. Notice that if another group of students has been using the same pair of Wiimote and ZedBoard, you might find these steps to have been completed already.

5. Now that a trusted device has been set up, pair and create the connection again.

- Put the Wiimote in pairing mode (i.e., push the red button).
- Run command

```
sudo bluez-simple-agent hci0 XX:XX:XX:XX:XX:XX
```

  where XX:XX:XX:XX:XX:XX is the Wiimote MAC address. This should pair the Bluetooth devices without entering a PIN code.

- Run command

```
sudo bluez-test-device trusted XX:XX:XX:XX:XX:XX yes
```

  to place the Wiimote on the trusted devices list. Again, XX:XX:XX:XX:XX:XX is the Wiimote MAC address.

- Put the Wiimote into pairing mode one last time. To establish the connection, run

```
sudo bluez-test-input connect XX:XX:XX:XX:XX:XX
```

  using the Wiimote MAC address. The Wiimote should now have one of the blue lights on all the time, indicating that it is connected to the Zedboard.

6. Create a shell script to automate this process.

- We will be using the Wiimote in other labs as well. In order to avoid following all these steps each time, we will write a shell script that does this for us. Go to your home folder and create a file named `WiimoteConnect.sh` using *vi*, or any other text editor of your choice. Place the following content in the file:

```
bluez-simple-agent hci0 XX:XX:XX:XX:XX:XX
bluez-test-device trusted XX:XX:XX:XX:XX:XX yes
bluez-test-input connect XX:XX:XX:XX:XX:XX
```

  where XX:XX:XX:XX:XX:XX refers to the MAC address of the Wiimote. Save the file and exit.

- Assign execution permissions to the shell script with the following command:

```
$ chmod +x WiimoteConnect.sh
```

- Validate that the script works. First, break the active connection between the Wiimote and the Zedboard by pressing the red sync button on the Wiimote, and waiting until the pairing time window expires. Now, the Wiimote is disconnected from the ZedBoard.
- Then, press the red sync button again, and before the pairing time window expires, call the script to connect again. You should invoke the script using the sudo command, as follows:

```
$ sudo ./WiimoteConnect.sh
```

Lab 4.3 Inspecting the Bluetooth messages

When the Bluetooth connection is created, Linux creates a virtual device file in directory `/dev/input`. By reading the content of these files with shell commands, the user can inspect the information packets received from the Wiimote in the ZedBoard.

- Read the content of file `/dev/input/event2` with command

```
hexdump /dev/input/event2
```

  Tool hexdump displays the content of a file in hexadecimal format. Notice that /dev/input/event2 is a special file virtually with no end, where new bytes are added every time new information is received from the Wiimote. Try pressing different buttons on the Wiimote (except for the power button), and observe the codes received on the ZedBoard. You can return to the shell anytime by pressing *Control+C*.

- Now read the content of file `/dev/input/event0` with a similar command.

```
hexdump /dev/input/event0
```

  This file represents the information received from the accelerometer embedded in the Wiimote. The sensitivity of this device will cause new acceleration messages to be sent continually, unless the device is completely still. In order to inspect the value of the packets, you can lay the Wiimote face down on the table. In the packets, codes 03, 04, and 05 represent the X, Y, and Z dimensions. The bytes after these codes represent angles.

Lab 4.4 Reading button events in C++

As shown above, file /dev/input/event2 can be read in order to obtain information from the Wiimote related with push buttons. The following simple C++ program does the work. It opens the file using function open(), and then uses the returned file descriptor in a read() function, which will obtain a 32-byte packet from the file if it's ready, or will suspend the program until one becomes ready. For each packet, the program extracts the information in bytes 10 (code) and 12 (value).

```
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <iostream>

int main()
{
        // Open Wiimote event file
        int fd;
        fd = open("/dev/input/event2", O_RDONLY);
        if (fd == -1)
        {
                std::cerr << "Error: Could not open event file - forgot sudo?\n";
                exit(1);
        }

        while (true)
        {
                // Read a packet of 32 bytes from Wiimote
                char buffer[32];
                read(fd, buffer, 32);

                // Extract code (byte 10) and value (byte 12) from packet
                int code = buffer[10];
                int value = buffer[12];

                // Print them
                std::cout << "Code = " << code << ", value = " << value << '\n';

        }

        // Close Wiimote event file
        close(fd);
        return 0;
}
```

Assignment 1

Compile and run this program. Write down the codes associated to each button (except the power button), and show them on a table in your report. What does variable `value` represent?

Lab 4.5 Using object-oriented programming to read button events

In this section, we will convert the code presented above into an equivalent program using object-oriented programming. For this purpose, we will create a class called WiimoteBtns with the following fields and functions:

- A class constructor opens /dev/input/event2 and checks for errors.
- A class destructor closes the file.
- Private field fd is used to store the file descriptor associated with file /dev/input/event2, as returned by the open() call in the constructor.
- A public function called Listen() enters an infinite loop where a new event is read from the virtual file. When an event is ready, its associated code and value fields are passed in an invocation to a new function called ButtonEvent().
- A public function called ButtonEvent() takes a code and value as integer arguments, and displays a message on the screen reporting these two values.

With the help of this class, your main program can be written in just two lines of code (plus the return statement). The first line is a static instantiation of an object of type WiimoteBtns, and the second line is a call to function Listen().

Assignment 2

Implement your program in three different files: WiimoteBtns.h, WiimoteBtns.cpp, and main.cpp. The first one contains the class declaration, the second one the class function definitions, and the third one the main program.

Create a Makefile for your program, using different rules for the compilation of WiimoteBtns.cpp, the compilation of main.cpp, and the linking of executable file main. Add an additional rule called clean that removes all intermediate files generated by g++.

Download files WiimoteBtns.h, WiimoteBtns.cpp, main.cpp, and Makefile into your local Windows machine, and create a ZIP file named Lab4a.zip. Attach this file to the submission of the lab report.

Lab 4.6 Reading acceleration events in C++

The following program has a very similar structure to the program shown earlier, except here we're reading acceleration events from virtual file /dev/input/event0. From the packet received in this case, we care about two fields. The first is code, a value specifying the dimension in which the acceleration happens, which can be extracted from byte 10. The second is acceleration, an integer value taking positive or negative values, representing the magnitude of the acceleration and its direction (positive means right, negative means left). This value is a 16-bit number located in bytes 12 and 13 of the packet. Notice the strategy to extract it: we obtain the address of byte 12, cast this address to a pointer to a short (16-bit integer), and dereference it.

```
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <iostream>

int main()
{
        // Open Wiimote event file
        int fd;
        fd = open("/dev/input/event0", O_RDONLY);
        if (fd == -1)
        {
                std::cerr << "Error: Could not open event file - forgot sudo?\n";
                exit(1);
        }

        while (true)
        {
                // Read a packet of 16 bytes from Wiimote
                char buffer[16];
                read(fd, buffer, 16);

                // Extract code (byte 10) and value (byte 12) from packet
                int code = buffer[10];
                short acceleration = * (short *) (buffer + 12);

                // Print them
                std::cout << "Code = " << code <<
                                ", acceleration = " << acceleration << '\n';

        }

        // Close Wiimote event file
        close(fd);
        return 0;
}
```

Lab 4.7 Using object-oriented programming to read acceleration events

Create a new version of class WiimoteAccel, with the following functionality:

- A class constructor opens /dev/input/event0 (instead of /dev/input/event2 as before) and checks for errors.
- A class destructor closes the file.
- Private field fd is used as before.
- A public function called Listen() enters an infinite loop where a new acceleration event is read from the virtual file. When found, the associated code and acceleration values are passed to a new public function called AccelerationEvent().
- A public function called AccelerationEvent() takes a code and acceleration as integer arguments, and displays a message on the screen reporting these two values. This should be a virtual function, which will later allow you to inherit class Wiimote and override this function, redefining the action to be carried out when a new acceleration value is read.

Again, the main program can be written in two lines of code, which instantiate an object of type WiimoteAccel statically, and invoke function Listen().

Assignment 3

Implement the new program with the same structure as before, based on three different files: WiimoteAccel.h, WiimoteAccel.cpp, and main.cpp. Create a Makefile for your program with the same structure as before, including a clean rule accessible through command make clean in order to get rid of all generated binary files.

Download files WiimoteAccel.h, WiimoteAccel.cpp, main.cpp, and Makefile into your local Windows machine, and create a ZIP file named Lab4b.zip. Attach this file to the submission of the lab report.

Lab 4.8 Displaying acceleration values on the LEDs

The objective of this last experiment is having the LEDs on the ZedBoard serve as an indicator panel for Wiimote motions. For higher accelerations, all LEDs should light up. When the Wiimote stands still or moves at a constant speed, no LED should light up. And only a subset of the LEDs should light up for any other intermediate acceleration value.

For this experiment, you need to reuse the code for class ZedBoard implemented in the previous lab assignment. Write the ZedBoard class declaration in a file named ZedBoard.h, and its definition in a file named ZedBoard.cc. The functions we'll need for class ZedBoard are the following:

- Constructor initializing the memory-mapped I/O.
- Destructor finalizing the memory-mapped I/O.
- Function RegisterWrite(offset, value), writing a value into a register given its offset.
- Function RegisterRead(offset), returning the value read from a register given its offset.
- Function setLed(led, value), which turns on (value = 1) or off (value = 0) the LED with the given (led is a number between 0 and 7).

Your program will also need class WiimoteAccel from the previous assignment, with support for reading acceleration values. Right in file main.cpp, define a new class called WiimoteToLed as a child of class WiimoteAccel. The child class has the following fields and functions:

- A private field of type ZedBoard * stores a pointer to the ZedBoard object instantiated in the main program, which should be passed to the class constructor, as shown in the code below. This is useful in order to access the ZedBoard functions from your new class.
- A constructor, whose only task is receiving a pointer to the ZedBoard object as an argument, and storing it in the private field.
- Function AccelerationEvent(code, acceleration) overrides the behavior of the parent virtual function of the same name. Remember that the parent function only printed the acceleration values on the screen. Now, this behavior will be redefined such that:
  - Return right away if the detected acceleration is not in the X axis. This condition is detected by a value of code equal to 3.
  - Truncate the acceleration in the range [-100, 100]. Most of the read acceleration values will be smaller anyway, but this will provide a nicer output on the LEDs.
  - Convert the absolute value of the acceleration into a value between 0 and 8, indicating the number of LEDs that should light up, regardless of the direction of the Wiimote.
  - Light up the suitable number of LEDs, by repeatedly invoking function setLed() on the ZedBoard object, saved as a private member in the class.

As in the previous examples, write a Makefile that automatically builds your code, selectively recompiling a subset of the files when needed. As before, add a rule clean to the Makefile that removes all object and executable files.

To help you with the development, this should be the exact content of function main():

```
int main()
{
        // Instantiate ZedBoard object statically
        ZedBoard zed_board;

        // Instantiate WiimoteToLed object statically, passing a pointer to the
        // recently created ZedBoard object.
        WiimoteToLed wiimote_to_led(&zed_board);

        // Enter infinite loop listening to events. The overridden function
        // WiimoteToLed::AccelerationEvent() will be invoked when the user moves
        // the Wiimote.
        wiimote_to_led.Listen();

        // Unreachable code, previous function has an infinite loop
        return 0;
}
```

Assignment 4

Run the new program and make sure that Wiimote accelerations are correctly represented by the LED states.

a) Take a short video demonstrating the behavior of the system, where both members of the team appear shortly, and with a duration not exceeding 10 seconds. Attach a file named q4.mov to the submission of your lab report—the file name extension may vary depending on the device you use to record the video.

b) Download files main.cpp, ZedBoard.cpp, ZedBoard.h, WiimoteAccel.cpp, WiimoteAccel.h, and Makefile into your local Windows machine, and create a ZIP file named q4.zip. Attach this file in the submission of your lab report.

Lab 4.9 Displaying 3D position values

In this optional assignment, you are encouraged to infer the current 3D coordinates of the Wiimote from the transmitted acceleration values. Every time your program reads a new acceleration value from the device, it should infer the speed and position in each coordinate. For this calculation, you need to take into account the time between the current and the previous acceleration sample. You can use function gettimeofday to obtain the current system time (see man pages). Your calculations should use the appropriate coefficient values to provide the final X, Y, and Z position values in inches. The position should be given relative to the position of the Wiimote when the program started to run.

When new acceleration values are read from the Wiimote, your program should print a line of output indicating the 3D acceleration, 3D speed, and 3D position. To avoid flooding the terminal with too many lines of output, limit the frequency of the output messages to one per second. You can do this by recording when the last message was printed, and avoiding a new message until at least one second has elapsed.

Extra credit

Write your program in C++, reusing all or part of the code implemented in previous assignments.

a) Take a short video demonstrating the behavior of the program, with a maximum duration of 20 seconds. Attach a file named extra.mov to the submission of your lab report—the file name extension may vary depending on the device you use to record the video.

b) Download all files in the project, including a Makefile, into your local Windows machines, and create a ZIP file named extra.zip. The Teaching Assistants should be able to compile the project without errors by just running make. Attach this file in the submission of your lab report.