Week 3 Laboratory – Accelerometer, buttons and display

1. Outcomes:

- To learn how to read data from the Orbit BoosterPack's accelerometer.
- To generate a program which responds reliably to manual button pushes.
- To generate a program which utilises SysTick timer interrupts.
- To continue development of useful C code functions for the fitness monitor project.

2. Groups:

It is important that by the time of your Week 3 lab session you are working with the group that you will stay in for the project. Each group must have at least one Tiva kit. It is however highly recommended that each student purchases a kit; having a kit allows each student to perform code testing independently.

3. Source files:

You have been supplied with a number of source files for this laboratory.

All of the files can be found in a zip file on Learn: **Section 1 | Laboratory source code | Week-3**

Button support code

The button support module comprises the files:

buttons4.c, **buttons4.h**

butsTest4.c is a program which demonstrates how the button interface module can be used.

Care must be taken with manually controlled switches, since "switch bounce" can occur. The philosophy behind the **buttons4** module and its design will be explained in detail in Lecture 10 (Week 4). Until then, the following brief explanation should suffice:

> All the buttons supported are polled regularly, using the **`updateButtons()`** function, to see if their state has altered. Any change of state is stored in variables which are module globals. The **`checkButton()`** function can be called at any time to see if a particular button, identified as **`UP`**, **`DOWN`**, **`LEFT`** or **`RIGHT`**, has been **`PUSHED`** or **`RELEASED`**. The **`checkButton()`** function only returns the altered state once per action of the physical pushbutton, otherwise it returns **`NO_CHANGE`**.

Demonstration OLED display program

OLEDTest.c is an example program which demonstrates how to use the display on the Orbit BoosterPack.

Example accelerometer program

readAcc.c is an example program which reads data from the Orbit BoosterPack's onboard accelerometer. A datasheet for the accelerometer (Analog Devices ADXL345) is available on Learn.

4. *Specification for Week 3:*

4.1 Set up a new project for the Week-3 Lab (call it "Week-3 Lab")

While this is not strictly necessary, it will help to keep your work in sections. Follow a similar procedure to that you used last week (see CCS Tutorial.pdf). Unzip the source files from the zip file downloaded from Learn into your new project directory. Then, move the OrbitOLED directory and its contents to your workspace directory, i.e. to **P:\Courses\ENCE361\labs\**
This is so that you can include this code in your programs without duplicating it in every project.

> Remember that you should only compile the .c files that are part of each program – use right click and "Exclude from Build" as required.

4.2 Compile, link and load OLEDTest.c

The build of this program needs access to the ustdlib module and to the OLED support module.

ustdlib.c:	Right-click on the week3Lab project, **Add Files** browse to **C:\ti\TivaWare_C_Series-2.1.4.178\utils\ustdlib.c** **Open**, then choose **Link** ustdlib.c will appear in the project with a link icon alongside.
OrbitOLED:	**File | New | Folder | Advanced >> | Link to alternate location** Browse to the OrbitOLED folder in your workspace, click **Finish** The OrbitOLED folder will appear in the project with a link icon alongside. Add **P:\Courses\ENCE361\labs** (i.e., your workspace containing the OrbitOLED folder) to the compiler include paths, using **Properties | ARM Compiler | Include Options** and **+**

> Remember: use "Exclude from Build" to only compile the .c files that are part of each program.

This very straightforward program puts 4 lines of characters on the display, with the third row updating the display of 0 to 255 at about 2 Hz. It does not use the buttons. It illustrates how to initialise and use the Orbit OLED display for text.

4.3 Compile and link butsTest4.c

NB: This file needs access to the OLED support module and to the ustdlib module. Remember to exclude OLEDTest.c from your build at this stage.
Confirm that each button push is detected by the program as a single "PUSH", followed by a "RELS" (release). This may seem trivial, but it is not, because most simple switches "bounce", as described in section 4 above.

4.4 Compile, link and load readAcc.c

NB: This file needs access to the OLED support module. You also need to exclude butsTest4.c from your build.
Run the program and watch the values on the OLED display update as you hold your Tiva board in different orientations. Confirm that the program behaves as the description in the file header states.

4.5 Prepare a new version, say 'readAcc2.c', which displays acceleration in m/s^2 on the Orbit display.
Change the accelerometer's settings to give the best possible sensitivity (m/s^2/bit).
Hint: Consult the ADXL345 datasheet to find out how to change the accelerometer's *resolution* and *range*.

4.6 Prepare a third version, say ‘readAcc3.c’, which causes the Tiva’s red and green LEDs to light up when the device is held approximately level in the *x* and *y* directions, respectively.

4.7 Prepare a fourth version, say ‘readAcc4.c’, which calculates the orientation (pitch and roll) of the Tiva board based on the acceleration vector. Display the pitch and roll as percentages, where 0% corresponds to a completely level orientation and 100% corresponds to a 90° rotation in the relevant direction.

5. Guidance:

- Before you start modifying a source file that has been provided to you, **always** use *SaveAs* to first copy it to a new file and a new name. Remember to exclude the original file from the next build.

- Prepare the new programs one step-at-a-time.

- Some code that you may link to requires the use of the “heap”. This requires that the heap size is increased from its default of 0 bytes. Change this to (say) 512 via the Project Properties | ARM Linker | Basic Options window. The stack size may need to be increased also in some cases, as your programs become more extensive.

- Make new code as modular as possible, using appropriately designed and named functions. This will pay off when you come to build a much bigger program to control the helicopter. If an existing and already tested function does what you want, leave it alone.

- Test, test, and test once more. Make sure your tests cover the specification fully. A lot of problems with software occur because the developer has not been thorough in testing their code.

----:----