Week 3 Laboratory – PWM, buttons and display

1. *Outcomes:*

- To learn how to control two pulse width modulated (PWM) outputs.
- 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 helicopter 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. *Laboratory books:*

Remember that your lab book (one per student) must be maintained over Weeks 2 to 5.

4. *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**

There are also two programs which demonstrate how the button interface module can be used: **butsTest4.c** and **pwmGen.c**.

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`**.

Example PWM program

pwmGen.c is an example program which generates a single PWM output with duty cycle fixed and the frequency controlled by UP and DOWN buttons (the two pushbuttons on the Orbit BoosterPack) in the range 50 Hz to 400 Hz.

Demonstration OLED display program

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

5. Specification for Week 3:

5.1 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_7_4_tutorial_Week_2_2018.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.

5.2 Compile, link and load pwmGen.c (needs access to buttons4.c and buttons4.h)
(At this stage you need to exclude butsTest4.c and OLEDTest.c from the build.) Run the program and monitor the PWM output signal with the oscilloscope. Remember that you need to connect both the probe tip to the PWM signal pin (J4 pin 5) and the probe earth to a Gnd pin (e.g. J2 pin 1). Confirm that the program behaves as the description in the file header states. The output corresponds to the signal controlling the MAIN helicopter motor.

5.3 **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 -	R 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 **Finish** The OrbitOLED folder will appear in the project with a link icon alongside. Add **P:\Courses\ENCE361\labs** (i.e., your workspace) 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.

5.4 **Compile and link butsTest4.c** (needs access to the OLED support module, and to the ustdlib module)
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.

5.5 **Prepare a new version, say 'pwmGen2.c'**, which allows the user to control both the frequency and the duty cycle of the PWM output: each push of the LEFT button should reduce the duty cycle by 5%, with a minimum of 5%; each push of the RIGHT button should increase the duty cycle by 5%, with a maximum of 95%; the UP and DOWN buttons should still operate as for pwmGen.c. Thoroughly test the new program.

5.6 Prepare a third version, say 'pwmGen3.c', which displays the current PWM output frequency and duty cycle (with units) on the Orbit display. Thoroughly test the program.

5.7 Prepare a fourth version, say 'pwmGen4.c', which configures and starts a second PWM output according to the data provided in the table below (extracted from the reference manuals). This output corresponds to the TAIL helicopter motor. Set the PWM frequency to 200 Hz (fixed) and the duty cycle to 10% (fixed). The operation and control of the MAIN PWM output should be identical to 5.6. Test the program.

Details of the TAIL PWM output	
PWM Module	1
Generator	2
Output number	5
Shorthand output description	M1PWM5
GPIO Port	F
GPIO pin	1
Shorthand GPIO pin description	PF1
Tiva board connection	J3-10

6. *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.

----:----