**What headers are included at the top of the code and what are they used for?**

The headers included at the top of the code are as follows:

1. xdc/std.h : This header is used to instantiate function declarations and variable types used in xdc/std. This allows the code in Lab2Idle\_main.c to use standard variable types such as Bool and UInt.
2. xdc/runtime/System.h : This header is used to instantiate functions declarations for system functions. This allows the code to use the System\_printf function.
3. ti/sysbios/BIOS.h : This header is used to instantiate function declarations for BIOS functions, as well as related structure definitions. This allows the code to call BIOS\_start() to jump to the RTOS.
4. Headers/F2837xD\_device.h : This header includes all the peripheral headers for the F2837xD device, allowing the code to interface with the GPIO for toggling the LED.

**How can you see the non-graphical, XDC-script-based version of the .cfg file?**

The non-graphical, XDC-script-based version of the .cfg file can be accessed by opening the file with the XDCscript Editor instead of XGCONF.

**Find where in the graphical .cfg you can set the size of the buffer. Try changing it and observing the OutputBuffer.**

The buffer size of SysMin can be adjusted by left-clicking on SysMin under the Outline section when using the graphical .cfg. The default is set to 128 chars but adjusting it, for example increasing it to 256, adjusts the amount of data stored in the buffer before it circles back. Increasing the buffer size, increases the amount of data able to be buffered before it is overwritten.

**Graphical user interface, text, application, email

Description automatically generated**

Figure : SysMin buffer size adjustment from 128 to 256 in SYS/BIOS.

Table

Description automatically generated

Figure : OutputBuffer with size of 256 chars.

Graphical user interface, text, application

Description automatically generated

Figure : OutputBuffer with size of 64 chars. Visibly smaller when compared to buffer size 256.

**Research the benefits of using a PLL to produce the CPU clock and describe two of them.**

1. A PLL can generate a much higher frequency than the reference frequency, this requires a lower frequency crystal. This is evident in the LaunchPad schematic, which has a 10MHz crystal but a max clock rate of 200MHz.
2. The frequency of the CPU can be adjusted through software-based configuration. This allows the designer to change the speed of the clock to best suit the application.

**Are the settings we have chosen guaranteed to always generate a 200MHz CPU clock and never exceed it? Explain your answer.**

Given that a PLL is a control loop, it’s possible that there will be overshoot as the clock approaches the set point. This would result in a higher CPU clock frequency than 200MHz. However, once the loop reaches steady state it will have much less deviation from the set point. That is to say, it is still possible for the clock to exceed 200MHz with the settings we have chosen.

**Snapshot of the RTOS Object View Display and User Defined Idle Functions**

The following images are the bock diagram of the SYS/BIOS graphical interface and the User Defined Idle Functions section where myIdleFxn2 is defined, respectively.

Graphical user interface, diagram

Description automatically generated

Figure : RTOS Object View

Graphical user interface, text, application, email

Description automatically generated

Figure : RTOS Idle Functions