GE420 Laboratory Assignment 2
Introduction to DSP Programming with Texas Instruments Code Composer Studio

Goals for this Lab Assignment:

1. Introduce the "SYS/BIOS" kernel and its Configuration Manager.
2. Use the Clock scheduling item to periodically call a function.
3. What to do with a compiler error.
4. Debugging your source code with Breakpoints and the Watch Window.

SYS/BIOS Objects Used:

- **Clock** object

Library Functions Used:

- (none)

Matlab Functions Used:

- (none)

Prelab:

Complete: http://coecsl.ece.illinois.edu/ge420/GE420Lab2ExtraPrelab.pdf

Laboratory Exercises

Exercise 1: The Clock Object

Now let's create a new application that uses the Clock object in SYS/BIOS. First use GE420_F28377S_ProjectCreator.exe to generate a new project in your repository's "trunk" directory. Give it a descriptive name so that you can find it easily. I recommend something like c:\<username>\GE420\trunk\lab2switchchecker. *Remember to backup these directories to your U:\ when you are finished working for the day.*

1. Open your project inside CCS and find the configuration (*.cfg) file for your **lab2switchchecker** project. Double click on the CFG file and the TI-RTOS window should open along with its Outline view. The Outline view shows the different parts of the SYS/BIOS kernel that are being used for your project. Modify your SYS/BIOS configuration to add a Clock object by right clicking on the Clock and selecting New Clock. Click on the new Clock which was probably given the default name clock0 and the Clock Instance tab will be selected. In the Instance tab change the Handle of this Clock object to CLK_OneSecond (you can pick a different name if you would like). Then in Function item enter the name of the function you are going to create in your C code that will be called by SYS/BIOS every one second. A possible name would be DoEverySecond, but you can create any function name that you would like. Finally we need to tell SYS/BIOS to call our function every second. The unit of time for the default Clock item is one millisecond, also called one tick. To make our function be called every second enter 1000 in both the "Initial timeout" item and the "Period" item. Also check the "Start at boot time when instance is created." Save the .cfg file.
2. Now write the function that the Clock object will call every second. Use the function name you chose above. For example if you used the suggested name above, "DoEverySecond", you would create a

function in your user_<projectname>.c file “void DoEverySecond(void)”. For now simply call the UART_printfLine(1, "WhatEver"); to print the same text to the LCD every second. Not very interesting yet but this will allow us to compile and initially test the program. Debug your code and make sure your text prints to the LCD.

3. Write two worker functions “void SetLEDsOnOff(int leds)” and “int ReadSwitches(void)”.
   i. void SetLEDsOnOff(int leds) takes an integer as a parameter. The four least significant bits of this integer determine if the four LEDs are on or off. Bit 0 determines LED 1’s state. Bit 1 determines LED 2’s state. Bit 2 determines LED 3’s state. Bit 3 determines LED 4’s state. So for example if 6 (which is binary 0110) is passed to your function then LED1 should be off, LED2 should be on, LED3 should be on and LED4 should be off. Use four if statements inside your function to check, using the bitwise AND, &, operator, if the integer passed to your function has the least significant four bits either individually set or cleared. If set, turn ON the corresponding LED. If cleared, turn OFF the LED. Remember from Lab1, the instructions to control the GPIOs connected to the LEDs:
      GpioDataRegs.GPCDAT.bit.GPIO64 = 0 or 1;
      GpioDataRegs.GPCDAT.bit.GPIO91 = 0 or 1;
      GpioDataRegs.GPCDAT.bit.GPIO92 = 0 or 1;
      GpioDataRegs.GPDDAT.bit.GPIO99 = 0 or 1;
   ii. int ReadSwitches(void) returns an integer that the least significant four bits indicates the state of the four switches. (Note that when each of the switches are not pressed the GPIO pin reads a 1 or high voltage. When pressed the GPIO pin read a 0 or ground. This is because the IO pin is using an internal pullup resistor.) This function should have four if statements and use the bitwise OR, |, operator to create this return value. So start the return value at zero. Then if switch 1 is pressed OR 0x1 with value. If switch 2 is pressed OR 0x2 with value. If switch 3 is pressed OR ??? with value. If switch 4 is pressed OR ??? with value. Finally return value. Instructions for checking the state of the four GPIOs (set as inputs) connected to the switches
      Switch1State = GpioDataRegs.GPBDAT.bit.GPIO41;
      Switch2State = GpioDataRegs.GPCDAT.bit.GPIO66;
      Switch3State = GpioDataRegs.GPCDAT.bit.GPIO73;
      Switch4State = GpioDataRegs.GPCDAT.bit.GPIO78;
4. Now that you have these worker functions make your program a bit more interesting. Add code in your Clock function so that you print the value returned from your ReadSwitches() function. (Note to print an integer in a printf statement example: UART_printfLine(1, "WhatEver %d",DansInt);) Also echo whatever is returned from your ReadSwitches() function to the SetLEDsOnOff(value);. This way the LEDs will indicate the state of the switches.
   Show this working to your TA.

If you are working with a partner on this exercise, trade places and give the other person a chance to do the coding for the next several steps.

4. To get some more practice with the project creator, create **another** new project with GE420_F28377S_ProjectCreator. Again add a Clock object to your SYS/BIOS configuration, and as above, have it call a function, you are going to write, every **second**. Also copy from your previous project the two worker functions you created. Do not modify these worker functions. Instead use them in step 5 below.
5. Write your new Clock function to increment a global integer variable by one each time it is called. Pass this count variable to the SetLEDsOnOff(value) function to display the least significant 4 bits of the count to the four LEDS. Compile, download to the DSP and verify that indeed the LEDS are counting in binary. Add one more item to this Clock function as an exercise to see the use of bitwise operators in C. Using the ReadSwitches() function, an "if" statement and one or more of the bitwise C operators, check if switch 2 is in the down position. If switch 2 is in the down position, stop the incrementing of the global count integer. Again compile and download to the DSP, and when finished demonstrate your application to your TA.

Exercise 2: Breakpoints and Watch Windows

Starting with the code you just finished, we want to experiment with adding breakpoints to your code and using the "Expressions window" to edit the values of your variables.

1. In your previous code (with the DSP halted), put your cursor over the integer variable that you are incrementing. You should see that the value of the variable appears. Run your code, halt it again, and again put your cursor over the variable to confirm that it changes.
2. An easier method than using the cursor repeatedly is to add the variable to the Expressions window. When the DSP is halted, the Expressions window displays the current value of each variable in the Expressions window. To add your counting integer variable to the Expressions window, highlight the variable and then right-click, then select **Add Watch Expression…**. The variable will appear in the Expressions window with the current value of the variable. The Expressions window dialog is also found under the View menu.
3. Next play a bit with adding breakpoints and single stepping through a section of code. The code you have written to this point is very small. Add the following nonsense code to allow for easy use of breakpoints and code stepping. At the top of your C-file, but below the #includes, add the following global variables:
```
float x1= 6.0;
float x2= 2.3;
float x3= 7.3;
float x4= 7.1;
```
Then inside your period function add this nonsense code:
```
x4 = x3 + 2.0;
 x3 = x4 + 1.3;
 x1 = 9*x2;
 x2 = 34*x3;
```

Build and load your code. Add a breakpoint to your code by double clicking on the left gray margin of your source file. A breakpoint is a location where the program will literally halt during execution. This allows you to check the values of your variables during operation. After a breakpoint, you can single step through your code (F5) and watch the variables update as different calculations are performed. You remove breakpoints by again clicking in the left gray margin. Since much of your program in located in the Flash memory of the processor most of the time you can only set one breakpoint at a time. If you really need to set multiple breakpoints at one time it is possible to move the functions you want to debug into the processor's RAM. We probably will not need to do that in this class.

4. If you happened not to receive any compiler errors during any of the above exercises, you should intentionally add some errors to your code so that you will see how CCS will alert you during the build process. Try double clicking on the error message. The editor will then take you to the line of code that has the error.

Last Exercise

Write a program that spells out your name in Morse code using all 4 LEDS as one light source. See http://en.wikipedia.org/wiki/Morse_code for specifics on Morse code. Use an array to store the Morse code data. Each Clock call (say about .25 seconds) set the LEDS to the value in the array (either all LEDS off (0) or all LEDS on (1)).

Lab Check Off:

1. Demonstrate your first application, the one that continually checks the status of the four dip-switches and displays their current state on the four LEDs.
2. Demonstrate your second application, the one that updates a counter every second and outputs the least significant 4 bits of the count to the four LEDs. The count should stop if switch 2 is in the down position and resume when it is in the up position.
3. Demonstrate that you know how to use Breakpoints and the Watch Window to debug your source code.
4. Demonstrate your Morse code application.