Flashing LEDs Overview and Reference Guide

August 2008

The purpose of the Flashing LEDs project is to be a simple example project for first time users and to serve as a template framework for creating software that can be used on multiple F28x devices. This project's purpose and implementation is simple and blinks an LED on the controlCARD at a given frequency, however the framework can be seen as large and obtrusive to a user not familiar with its features. This guide is meant to explain some of the features found in the project FlashingLEDs.pjt and give information on how to customize the FlashingLEDs project in order to make a new project.

FlashingLEDs Project

The FlashingLEDs project main task is to blink LD3 on a F28x controlCARD based on a blink period that is controlled by the variable Gui_LEDPrd. This Gui_LEDPrd is a 16-bit integer given in ms with a Q0 format. A Q format is a term that specifies where a fictitious decimal point lies in fixed point math. For example, in Q0 this decimal point will lie 0 bit spots from the right side of 16-bit data type, giving this value a range from -32768 to 32767. By contrast a 16-bit integer written in Q15 format would have a range from -1 to almost 1.

This period can be edited beforehand by initializing it to a different value, during runtime within Code Composer Studio by changing its value in a Watch Window, or by editing it externally. A simple GUI has been created to simply edit and extract variables from the F28x target externally via a serial port. More information on this GUI can be found in the **General Purpose GUI** guide.

Procedure

Open a CCS Project

1. Connect a JTAG emulator to the motherboard.

2. Turn on the power (SW1 for the Experimenter's Kit/Docking Station) to any docking station or motherboard. Open Code Composer Studio and maximize it to fill your screen.

3. Connect to the board by clicking:

   `Debug → Connect`

   If the board fails to connect you may need to check your emulator configuration. Close Code Composer Studio then edit the configuration for your emulator in the program "Setup Code Composer vX.X". This program is installed with Code Composer Studio.

4. Open the project `FlashingLEDs.pjt` by clicking:

   `Project → Open…`

   and look in `C:\TI_F28xxx_SysSW\FlashingLEDs\.`

5. This project can be configured to create code for multiple target controlCARDs and can be ran in either flash or RAM. In the project window, right-click on `FlashingLEDs.pjt` and select "`Configurations…`". In the new window select the target F28x controller that is applicable to use in RAM, click "`Set Active`" and then click "`Done`".

6. Code Composer Studio can automatically load the output file after a successful build. On the menu bar click: `Option` → `Customize…` and select the "`Program/Project/CIO`" tab, then check "`Load Program After Build`".

   Also, Code Composer Studio can automatically connect to the target when started. Select the "`Debug Properties`" tab, check "`Connect to the target at startup`", then click OK.

Device Initialization, Main, and ISR Files

7. Open and inspect `FlashingLEDs-Main.c`. The framework contains two main sections, the initialization and a continually running state machine. The initialization handles setting up the device (through the function DeviceInit found in the FlashingLEDs-DevInit file), configuring the CPUTimers to run at a certain frequency, as well as configuring the ADCs so that they will update and read continually.

8. Open and inspect `FlashingLEDs-DevInit.c` by double clicking on the filename in the project window. Confirm that GPIO34, the GPIO responsible for LD3, is configured to be a GPIO output. Close this file.

9. In the `FlashingLEDs-Main.c` file notice the infinite loop that is started at line 333. This loop begins a sequence of three state machines each run by their own CPUTimer. State Machine A contains two tasks and task A2 controls the blinking frequency of the LED. By default each task will run once every two milliseconds.

   The portion of the software responsible for blinking the LEDs is shown below for convenience. Note that Gui_LEDPrd is the value the user will edit.

```
//-----------------------------------------------------------------
void A2(void) //** Toggle GPIO-34 (LD3)
//-----------------------------------------------------------------
{
      // task runs every 2ms = 1ms (CpuTimer0 period) * 2 (# of A Tasks)
      if(Gui_LEDPrd < temp_LEDDelay)
      {
            //LED blink period = Gui_LEDPrd (Q0 in ms)
            GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1;

            temp_LEDDelay = 0;
      }
      //LED frq is 4ms => task frq (2ms) * two toggles in a period (2) = 4ms
      else temp_LEDDelay = temp_LEDDelay + 4;

      //-------------------
      //the next time CpuTimer0 'counter' reaches Period value go to A1
      A_Task_Ptr = &A1;
      //-------------------
}
```

Build and Load the Project

10. Click the "`Rebuild All`" button and watch the tools run in the build window. The output file should automatically load.

11. Under Debug on the menu bar click "`Reset CPU`", "`Restart`", and then "`Go Main`". You should now be at the start of Main().

Setup Watch Window

12. Open the *watch window* to view the variables used in the project.

    Click: `View → Watch Window` on the menu bar.

    Click the "Watch 1" tab at the bottom of the watch window. In the empty box in the "Name" column, type the the variable "Gui_LEDPrd". This value should be 1000 ms by default.

13. Change `Gui_LEDPrd` to 250. This will decrease the blink period to 250 ms.

14. Change `Gui_LEDPrd` to 2000. This will increase the blink period to 2000 ms.

Further information on the framework can be found in the **System Framework Overview** guide.

Using FlashingLEDs as a Template

As mentioned one of the primary purposes of the FlashingLEDs project is to serve as a template for future design projects. Following are the steps necessary to create your own project.

1. **Rename the project:**

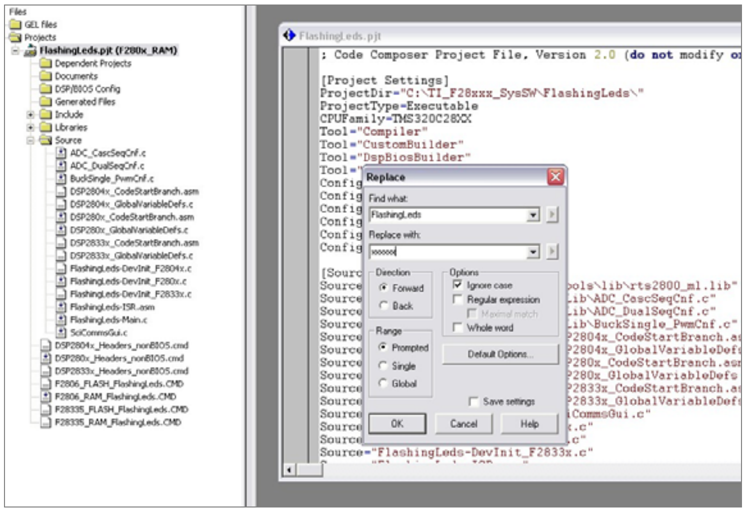
   a. Copy the files to a new directory – create a copy of the FlashingLEDs directory located within `C:\TI_28xxx_SysSW\`. This new directory should be located within `C:\TI_28xxx_SysSW\` as well. Rename this directory as desired.

   b. Inside this new directory (the one just created) rename the files via Windows Explorer. In renaming these files change only the word FlashingLeds to the desired name. For the purposes of these instructions I will call this name xxxxxx . Keep the rest of the formatting as it is and use this same name on all files.

   (ie rename `FlashingLeds-Main.c` to `xxxxxx-Main.c` *and* `FlashingLeds-DevInit-F280x` to `xxxxxx-DevInit-F280x`)

   c. Open up CCS and load the new project. For my case it would be `xxxxxx.pjt`. Opening this project should give you errors, click "Ignore All".

   d. In the project directory of CCS you should see `xxxxxx.pjt`. Open this file by right clicking on it and choosing "Open for Editing".



```
Files
  GEL files
  Projects
    FlashingLeds.pjt (F280x_RAM)
      Dependent Projects
      Documents
      DSP/BIOS Config
      Generated Files
      Include
      Libraries
      Source
        ADC_CascSeqCnf.c
        ADC_DualSeqCnf.c
        BuckSingle_PwmCnf.c
        DSP2804x_CodeStartBranch.asm
        DSP2804x_GlobalVariableDefs.c
        DSP280x_CodeStartBranch.asm
        DSP280x_GlobalVariableDefs.c
        DSP2833x_CodeStartBranch.asm
        DSP2833x_GlobalVariableDefs.c
        FlashingLeds-DevInit_F2804x.c
        FlashingLeds-DevInit_F280x.c
        FlashingLeds-DevInit_F2833x.c
        FlashingLeds-ISR.asm
        FlashingLeds-Main.c
        SciCommsGui.c
      DSP2804x_Headers_nonBIOS.cmd
      DSP280x_Headers_nonBIOS.cmd
      DSP2833x_Headers_nonBIOS.cmd
      F2806_FLASH_FlashingLeds.CMD
      F2806_RAM_FlashingLeds.CMD
      F28335_FLASH_FlashingLeds.CMD
      F28335_RAM_FlashingLeds.CMD
```

FlashingLeds.pjt

```
; Code Composer Project File, Version 2.0 (do not modify o

[Project Settings]
ProjectDir="C:\TI_F28xxx_SysSW\FlashingLeds\"
ProjectType=Executable
CPUFamily=TMS320C28XX
Tool="Compiler"
Tool="CustomBuilder"
Tool="DspBiosBuilder"
Tool="
Config
Config
Config
Config
Config
Config

[Sourc
Source                                   ools\lib\rts2800_ml.lib"
Source                                   Lib\ADC_CascSeqCnf.c"
Source                                   Lib\ADC_DualSeqCnf.c"
Source                                   Lib\BuckSingle_PwmCnf.c"
Source                                   P2804x_CodeStartBranch.a
Source                                   P2804x_GlobalVariableDef
Source                                   P280x_CodeStartBranch.as
Source                                   P280x_GlobalVariableDefs
Source                                   P2833x_CodeStartBranch.a
Source                                   P2833x_GlobalVariableDef
Source                                   iCommsGui.c"
Source                                   x.c"
Source                                   .c"
Source="FlashingLeds-DevInit_F2833x.c"
```

Replace

Find what: FlashingLeds

Replace with: xxxxxx

Direction: [✓ Forward] Back

Range: [✓ Prompted] Single Global

Options: [✓ Ignore case] Regular expression Maximal match Whole word

Save settings

e. Once this file opens, click Edit->Find/Replace in CCS. Change the "Find What" field to *FlashingLeds* and the "Replace With" field to *xxxxxx*. Note that xxxxxx should instead be the project name you chose.

f. Click OK. CCS will now attempt to rename several words in this file. Click "Yes" to each of the rename instances.

g. Save this file. In a few moments CCS will prompt you to reopen the file. Click "Yes". CCS should now not complain when loading.

h. Next open up `xxxxxx-Main.c` and click Edit->Find/Replace in CCS. The "Find What" and "Replace With" field should still show *FlashingLeds* and *xxxxxx* respectively. Click "OK". Click "Yes" to each of the rename instances.

i. Repeat with `xxxxxx-ISR.c`.

2. **Customize your project** –

Listed below are several commonly used modules and methods required for many applications. Each will show how the module is currently used and where to begin customizing it. Further edits will be necessary based on the specifications for the specific project.

The EPWM modules are currently set to do nothing although a configuration function is commented within the file. Uncommenting this function sets up ePWM1 to run at 400kHz with the ePWM1 module's CMPA value editing the duty cycle.

```
// ePWM1, Period=prd, Master, Phase=Don't Care
//BuckSingle_CNF(1, 250, 1, 0);
```

This function is specified for use with a buck converter although many other functions and modules created for the EPWM modules appear in the libraries that are provided with this project. The library files are found within `C:\TI_F28xxx_SysSW\~SupportFiles\lib\`

References

For more information please see the following guides:

- **System Framework Overview Guide** – presents more information on the system framework found in F28xxx EVM projects.

  C:\TI_28xxx_SysSW\~Docs\SystemFrameworkOverview.pdf

- **QSG-GeneralPurposeGUI Quick Start Guide** – gives an overview on how to connect your project to an external GUI via the SCI peripheral.

  C:\TI_28xxx_SysSW\~GeneralPurposeGUI\~Docs\QSG-GeneralPurposeGUI.pdf

- **F28xxx User's Guides** –

  http://www.ti.com/f28xuserguides