# System Framework Overview Guide and Instructions on How to Use the Template Projects

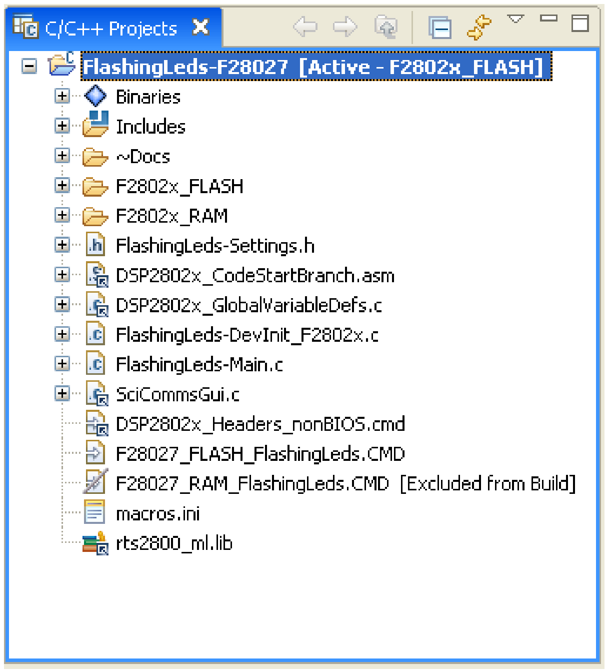
Brett Larimore and Manish Bhardwaj
*C2000 Systems and Applications Team*

*Version 2.0 – May 2008 – Revised September 2010*

The Texas Instruments' F28xxx System Framework is designed to be an intuitive baseline set of code that allows each EVM software package to be flexible, powerful, and easy to use. The framework features:

- An organized program structure which allows the user can focus on creating application code and not just device initialization.
- Configurabilty to edit whether the TMS320F28x will create an .out file that is ran from RAM during debug or production with FLASH with minimal code changes.
- FRAMEWORK and USER sections so that it is obvious which code can be changed as desired and which areas should only be edited minimally.
- Built-in state machines that cause a task will run continually at a specific frequency, but are designed to be interruptable if needed.
- Explicit GPIO mapping located in the project's DevInit file makes it clear how each GPIO is configured
- Ability to connect to an external GUI via UART and the F28x SCI peripheral

The FlashingLeds and DplibTemplate projects have been created as templates to decrease the amount of time it takes to begin new software projects.



FlashingLeds-F28027 [Active - F2802x_FLASH]
Binaries
Includes
~Docs
F2802x_FLASH
F2802x_RAM
FlashingLeds-Settings.h
DSP2802x_CodeStartBranch.asm
DSP2802x_GlobalVariableDefs.c
FlashingLeds-DevInit_F2802x.c
FlashingLeds-Main.c
SciCommsGui.c
DSP2802x_Headers_nonBIOS.cmd
F28027_FLASH_FlashingLeds.CMD
F28027_RAM_FlashingLeds.CMD [Excluded from Build]
macros.ini
rts2800_ml.lib

## FlashingLeds Projects

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 for the C2000 MCU family. This project's purpose and implementation is simple and blinks an LED on the controlCARD at a given frequency.

The FlashingLEDs project main task is to blink LD3 on an F28x controlCARD based on a blink period that is controlled by the variable Gui_LedPrd_ms. This Gui_LedPrd_ms variable is a 16-bit integer given that represents the period in ms of the LED in Q0 format.

## DplibTemplate Projects

The Dplib template projects are meant to provide a good structure on which to build a project that uses Texas Instruments' Digital Power Library. These projects each configure two PWMs to run in up-count mode at 200kHz, setup an interrupt to run synchronized with the PWM timer, and update the PWM duty cycle each interrupt cycle. This is all done by various functions and modules that are located in the Digital Power Library. Please see the Digital Power library documentation for more information and details on the various modules.

This can be found at:
\controlSUITE\libs\app_libs\digital_power\f2802x_vX.X\Doc\DPLib_vX.pdf

The structure of the Dplib Template projects is very similar to the FlashingLeds projects so moving from one to the other should be relatively easy.

## Configuring a Project

The framework was developed to create a project that can be configured to generate an output file that will run from RAM or from FLASH. The RAM configuration is usually used early in the software development process for convenience. Once code development becomes more stable or more RAM memory is needed than the current MCU has, the FLASH configuration may be used. The FLASH configuration stores the entire program in non-volatile FLASH memory and then, early in the program, the section "ramfuncs" is copied into RAM for quicker execution.

To change the configuration mode follow these steps:

1. Right-click on the project name in the Project Window and then highlighting "Active Build Configuration" in the context menu.
2. Select the correct configuration based on whether the code will be run in FLASH or RAM.

In the Project Window some files will have a slash and some will have an arrow on the image that precedes the filename. A slash shows that the file is not active in the current build configuration. Only active files are compiled and put into an output file. An arrow shows that the file has been linked into the project and resides in a path other than the project directory.

## Running a Project

**NOTE:** The following instructions and pictures show how to connect up to the FlashingLeds project with a F28035 controlCARD. Please edit the names slightly for the specific controlCARD and project you wish to use.

### Installing Code Composer and controlSUITE

1) If not already installed, please install Code Composer v4.x from the DVD included with the kit.

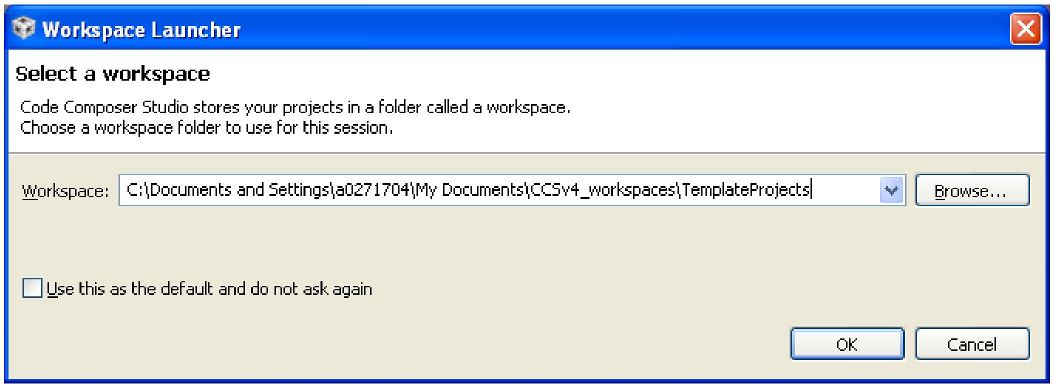
2) Go to http://www.ti.com/controlsuite and run the controlSUITE installer. In the installer, select and install the "Experimenter Kit" software.

### Setup Code Composer Studio to Work with a Template Project

3) Open "Code Composer Studio v4".

4) Once Code Composer Studio opens, the workspace launcher may appear that would ask to select a workspace location,: (please note workspace is a location on the hard drive where all the user settings for the IDE i.e. which projects are open, what configuration is selected etc. are saved, this can be anywhere on the disk, the location mentioned below is just for reference. Also note that if this is not your first-time running Code Composer this dialog may not appear)

   - Click the "Browse…" button
   - Create the path below by making new folders as necessary.
   - "C:\Documents and Settings\My Documents\ CCSv4_workspaces\TemplateProjects"
   - Uncheck the box that says "Use this as the default and do not ask again".
   - Click "OK"



Workspace Launcher

Select a workspace

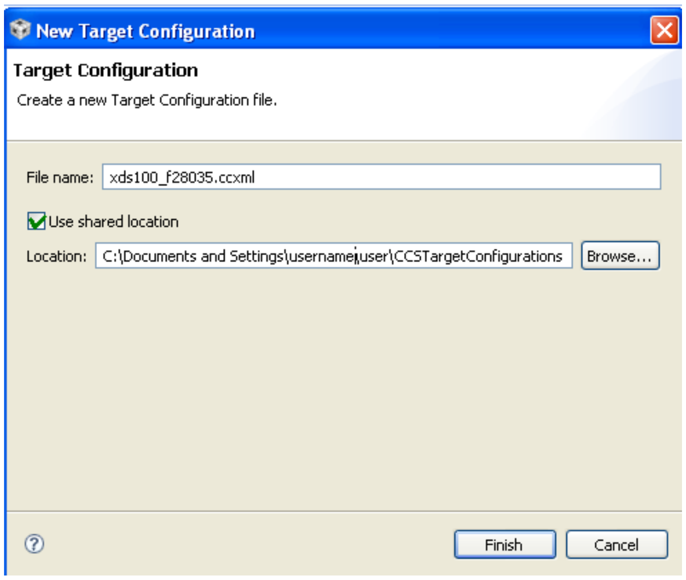
Code Composer Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Documents and Settings\a0271704\My Documents\CCSv4_workspaces\TemplateProjects

Use this as the default and do not ask again

**Figure 1: Workspace Launcher**

5) Next we will configure Code Composer to know which MCU it will be connecting to. Click "Target -> New Target Configuration…". Name the new configuration xds100v1-f28xxx where xxx is the name of the device located on the controlCARD. For example, if using a F28035 controlCARD, name the new configuration xds100v1-f28035.ccxml. Make sure that the "Use shared location" checkbox is checked and then click Finish.



File name: xds100_f28035.ccxml

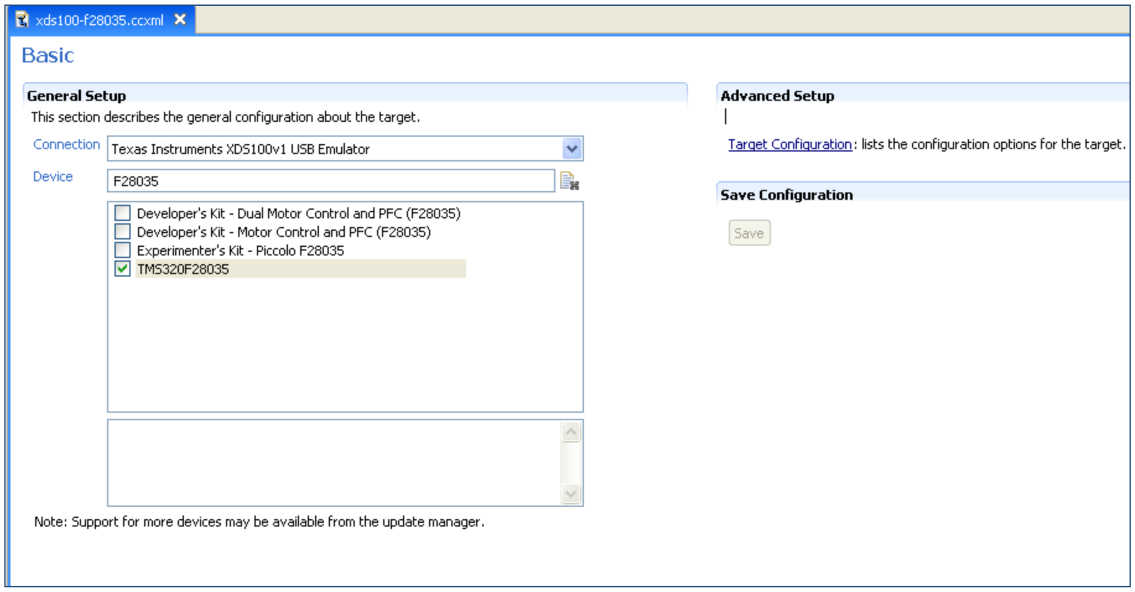
☑ Use shared location

Location: C:\Documents and Settings\username\user\CCSTargetConfigurations

Figure 2: Creating a Target Configuration

6) This should open up a new tab as seen in Figure 2. Select and enter the options as shown:

- Connection – Texas Instruments XDS100v1 USB Emulator
- Device – Select the device that is placed on the controlCARD.
- Click Save
- Close the xds100-f28xxx.ccxml tab



This section describes the general configuration about the target.

Connection: Texas Instruments XDS100v1 USB Emulator

Device: F28035

☐ Developer's Kit - Dual Motor Control and PFC (F28035)
☐ Developer's Kit - Motor Control and PFC (F28035)
☐ Experimenter's Kit - Piccolo F28035
☑ TMS320F28035

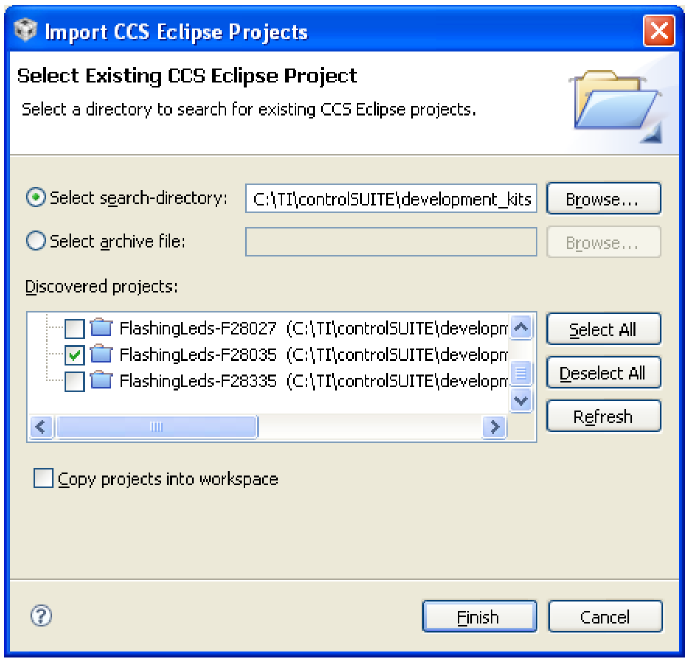
Note: Support for more devices may be available from the update manager.

Figure 3: Configuring a New Target

7) Assuming this is your first time using Code Composer, the xds100-F28xxx configuration is now set as the default target configuration for Code Composer. Please check this by going to “View->Target Configurations”. In the “User Defined” section, right-click on the xds100-F28xxx.ccxml file and select “Set as Default”. This tab also allows you to reuse existing target configurations and link them to specific projects.

8) Add the project into your current workspace by clicking “Project->Import Existing CCS/CCE Eclipse Project”.

- Select the root directory of the template projects. This will be: “C:\TI\controlSUITE\development_kits\TemplateProjects\


Import CCS Eclipse Projects
Select Existing CCS Eclipse Project
Select a directory to search for existing CCS Eclipse projects.
Browse...
Browse...
Select All
Deselect All
Refresh
Finish
Cancel

Select search-directory: C:\TI\controlSUITE\development_kits

Select archive file:

Discovered projects:

☐ FlashingLeds-F28027 (C:\TI\controlSUITE\developm
☑ FlashingLeds-F28035 (C:\TI\controlSUITE\developm
☐ FlashingLeds-F28335 (C:\TI\controlSUITE\developm

☐ Copy projects into workspace

Figure 4: Adding a Project to your Workspace

- Uncheck projects that do not pertain to your device or projects that you are not interested in
- Click Finish. This will copy the selected projects into the workspace.

## Configuring a Project

9) Set the project that you are interested in running as the active project. Right-click on this project’s name and click “Set as Active Project”. Expand the file structure of the project.

10) Each project can be configured to create code and run in either flash or RAM. You may select either of the two, however for lab experiments we will use RAM configuration to simplify the debug process and then move to the FLASH configuration for production. As shown in Figure 4, right-click on an individual project and select Active Build Configuration-> `F28xxx_RAM` configuration.

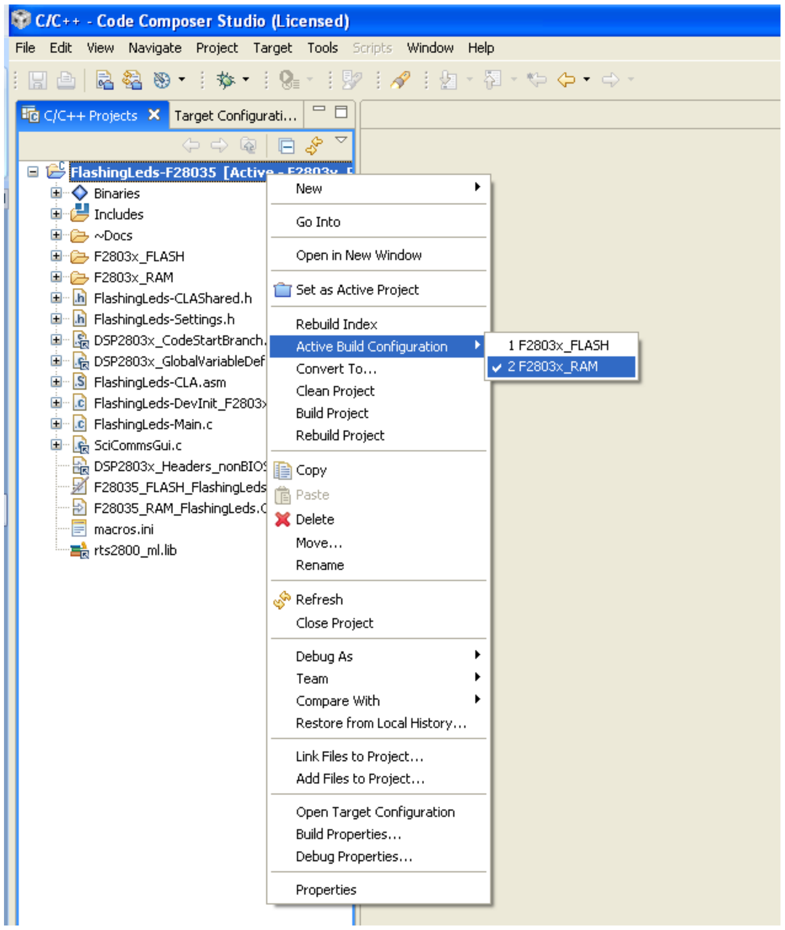


**Figure 5: Selecting the F2803x_RAM configuration**

## Build and Load the Project

11) Right Click on the Project Name and click on “Rebuild Project” and watch the Console window. Any errors in the project will be displayed in the Console window.

12) On successful completion of the build click the “Debug” button, located in the top-left side of the screen. The IDE will now automatically connect to the target, load the output file into the device and change to the Debug perspective.

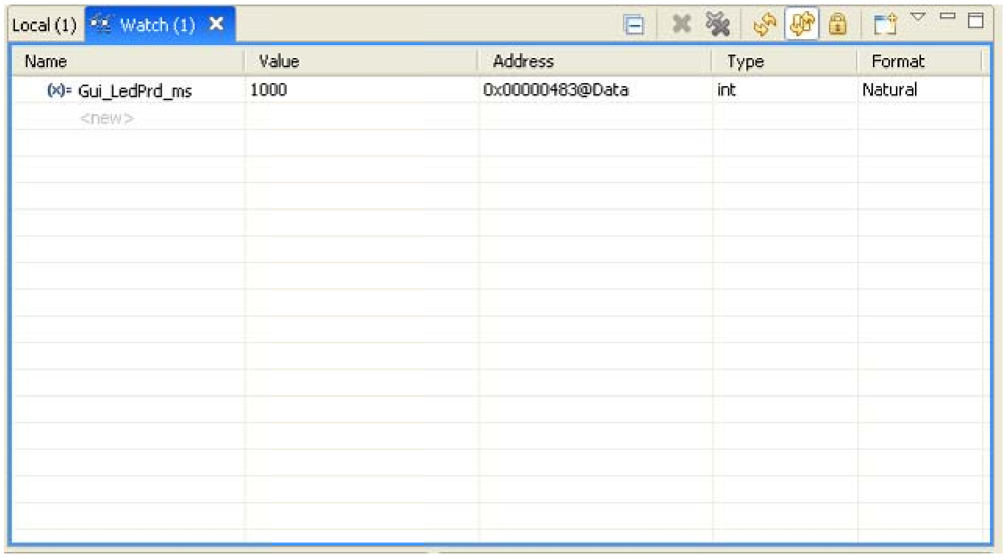
13) Click “Tools->Debugger Options->Generic Debugger Options”. You can enable the debugger to reset the processor each time it reloads program by checking “Reset the target on program load or restart” and click “Remember My Settings” to make this setting permanent. Close this window.

## Setup Watch Window & Graphs

14) Click: View → Watch on the menu bar to open a *watch window* to view the variables being used in the project. Next to each project below are the variables that need to be added to the respective project's watch window.

    FlashingLeds projects – Gui_LedPrd_ms
    DplibTemplate projects – LedBlinkCnt

    Editing these variables while the project is running will edit the speed at which the LED blinks. Additional variables can also be added as necessary.



| Name | Value | Address | Type | Format |
|---|---|---|---|---|
| (x)= Gui_LedPrd_ms | 1000 | 0x00000483@Data | int | Natural |
| <new> | | | | |

**Figure 6: Configuring Watch Window 1**

15) Click on the Continuous Refresh button in this watch window. This enables the window to run with real-time mode. By clicking the down arrow in any watch window, you may select "Customize Continuous Refresh Interval" and edit the refresh rate for the watch windows. Note that choosing too fast an interval may affect performance.

## Run the Code

16) Now click the real-time mode button that says "Enable silicon real-time mode". This will allow the user to edit and view variables in real-time without halting the program.

17) After this button is pressed, a message box *may* appear. If so, select YES to enable debug events. This will set bit 1 (DGBM bit) of status register 1 (ST1) to a "0". The DGBM is the debug enable mask bit. When the DGBM bit is set to "0", memory and register values can be passed to the host processor for updating the debugger windows.

18) Run the code by pressing Run Button in the Debug Tab.

19) The project should now run, and the LED on the controlCARD should be blinking. You may want to resize or reorganize the windows according to your preference. This can be done easily by dragging and docking the various windows.

20) Once complete, halt the device and end the debug session by clicking (Target->Terminate All). This will halt the program and disconnect Code Composer from the MCU.

## Creating a New Project from the Templates

**NOTE:** The following instructions and pictures show how to connect up to the FlashingLeds project with a F28335 controlCARD. Please edit the names slightly for the specific controlCARD and template project you wish to start with.

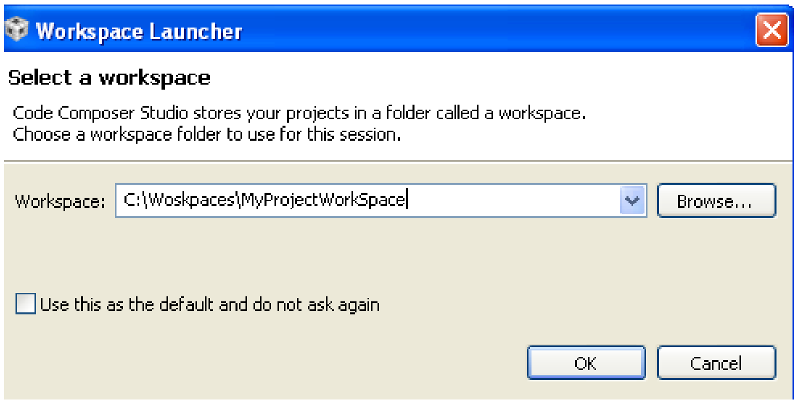
One of the main purposes of the FlashingLeds and DplibTemplate projects is to allow the user a good place to begin their own project. The following section shows the steps necessary to create a new project from the template projects.

1) Create the folder where the new project needs to reside inside controlSUITE directory, for this example we will create a project "MyProject" inside New Project folder at the location

   \controlSUITE\development_kits\NewProject\MyProject\

Note two folders of file depth are needed so as to exactly match the relative paths with the blank template files. This is necessary to link properly with other resources that exist inside the controlSUITE directory.

2) Open CCSv4, the Workspace launcher will pop up. Select a folder where your new workspace will be created (if no workspace exists then specifying the workspace path will create the workspace folder.) Note that the workspace folder does not need to be inside the controlSUITE directory.


Workspace Launcher

Select a workspace

Code Composer Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Woskpaces\MyProjectWorkSpace

☐ Use this as the default and do not ask again

3) Once CCS is launched, go to Project - > Import Existing CCS/CCE Eclipse Project and browse to the template project the new project is desired and click "ok".

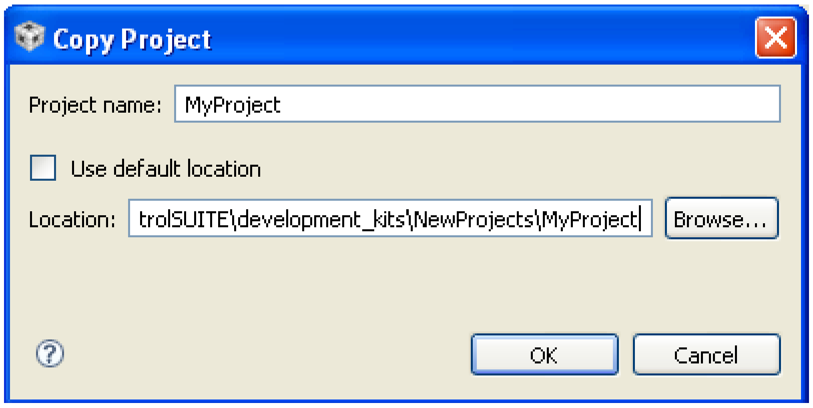
   \controlSUITE\development_kits\TemplateProjects\FlashingLeds-F28335_vX.X

4) The project will now get imported into CCS and can be seen under the C/C++ Project Window within CCS.

5) Right Click on the project name and click "Copy"

6) Now right-click on any place inside the C/C++ Project Window in CCS and click "Paste"

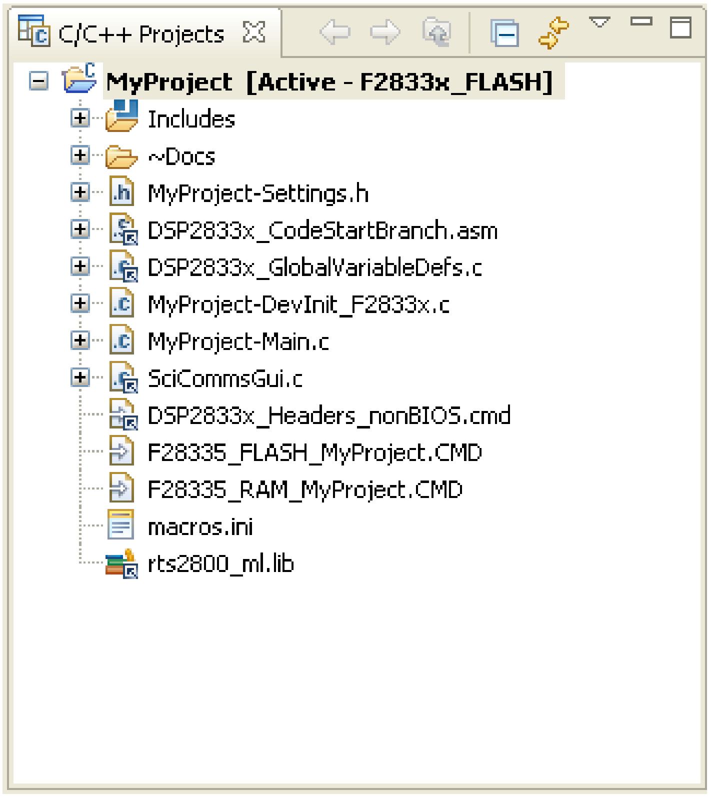
7) A Copy Project dialog box will now pop up. Specify the new project's name and point the location to the MyProject folder that was created inside controlSUITE in step 1. Click "OK" once done. This step copies all the project files inside the template project directory into the project folder specified with the given project name.



8) You should now see two projects inside the C/C++ Project window in CCS. Right click on the original template project name and click “Close Project” and once closed right click again and click “Delete”. A message asking to delete the contents of the folder would pop-up. Select “Do Not Delete Contents” and Click “Ok”.

9) Now only one project that was created by copying will appear in the C/C++ Dialog Box.

10) Expand the project to view all the files inside the project. Note that some of the files are named starting with “ProjectName”. These files are project specific and should be renamed. Right click on the file name inside the C/C++ Projects window and click “Rename” and replace “FlashingLeds” or “ProjectName” with “MyProject” in the file name. The C/C++Project view should look roughly like this now.


C/C++ Projects
MyProject [Active - F2833x_FLASH]
Includes
~Docs
MyProject-Settings.h
DSP2833x_CodeStartBranch.asm
DSP2833x_GlobalVariableDefs.c
MyProject-DevInit_F2833x.c
MyProject-Main.c
SciCommsGui.c
DSP2833x_Headers_nonBIOS.cmd
F28335_FLASH_MyProject.CMD
F28335_RAM_MyProject.CMD
macros.ini
rts2800_ml.lib

11) Also any reference to the FlashingLeds or {ProjectName} header files inside the Main.c, ISR.asm and similar files should be replaced with MyProject.

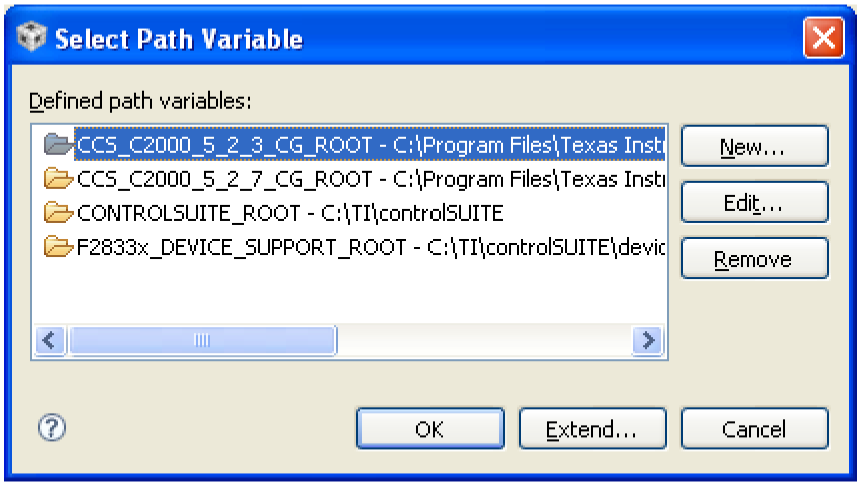
12) At this point, assuming the RAM Configuration is selected, Right-Click on the F28335_FLASH-MyProject.CMD and click on “exclude from build”. A diagonal dash should appear on the file icon to symbolize that the file is being excluded from the build. The configuration being used can be changed by right clicking on the project name -> Active Build Configuration -> and selecting the desired configuration.

    Right click on the project name and compile the project. The project should compile with no errors.

13) **“Using Peripherals that are not used in the Template Files”** To allow more peripherals to be used a few steps should be taken, 1) make sure the peripherals clock is turned on; 2) edit the GPIO mux in DeviceInit to allow it to output/input the peripheral signals; 3) create functions that initialize and run the peripheral

14) **“Adding Linked Resources”** A few of the files displayed inside the C/C++ Projects window have a small arrow displayed next to the file name. This indicates that these files are linked resources i.e. that they do not reside inside the project specific folder. Device support files such as (DevName-GlobalVariableDefc.c) and library files are typically included into the project as linked resources. These linked resources are resolved using the CCS Macro Variable Names which are defined in the macros.ini file.

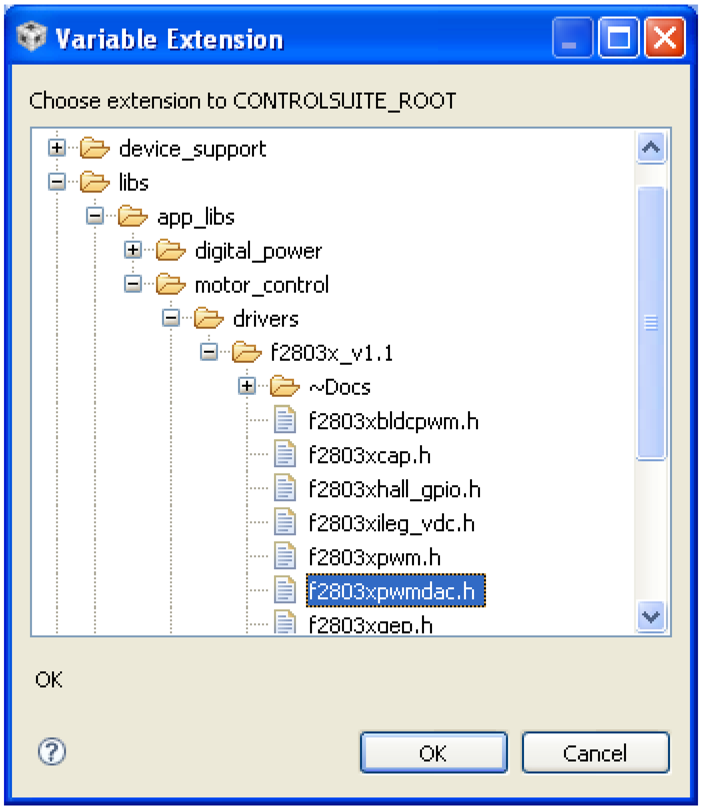
15) Now, if a linked resource needs to be added, Right Click on the Project Name -> New -> File -> Advanced -> Check Link to File in the file system -> Variables. The following window will pop up



CCS_C2000_5_2_3_CG_ROOT - C:\Program Files\Texas Inst
CCS_C2000_5_2_7_CG_ROOT - C:\Program Files\Texas Insti
CONTROLSUITE_ROOT - C:\TI\controlSUITE
F2833x_DEVICE_SUPPORT_ROOT - C:\TI\controlSUITE\devic

16) Observe that different path variable names have been defined for each resource, i.e. a Path Variable for controlSUITE is separate from the Path Variable for the device support resources. This is done to enable configurability when one of the resources is revised by TI. If this occurs, only this Path Variable would need to be changed.

17) Select the Path Variable which corresponds to the resource that needs to be added to the project and click Extend. The image below shows the extension of the CONTROLSUITE_ROOT Path Variable to f2803xpwmdac.h in the motor control library. Select an additional resource if one is needed by your application and click ok. The resource will now be added to the project and will display a small arrow next to its file name indicating that it is a linked resource.

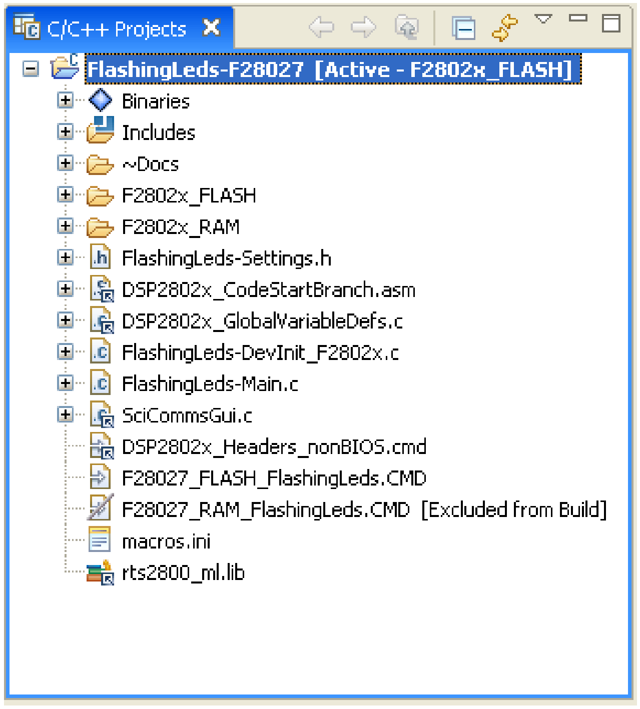


18) **"Adding/Modifying Path Variables"**, The path variables are defined in the file macros.ini which resides inside the project folder. Open the macros.ini file and edit it to add/modify the path variables as desired.

19) For the new Path Variables to get recognized the macros.ini file needs to be re-imported. To do this click File -> Import -> CCS -> Managed Build Macros and click Next. Once the 'Import Managed Build Macros" dialog box pops up, browse to the modified macros.ini file inside the MyProject folder. Check overwrite existing values box and click Finish. For the new macro names to come into affect Close and Open Project by right clicking on the project name in the C/C++ Project window.

Note: Path variables are defined on a per workspace basis. All of the linked path variables that are included in a workspace can be found by going to: Window -> Preferences… and then in the Preferences window by going to: General -> Workspace -> Linked Resources

## More Information on the Framework's Files


C/C++ Projects

```
FlashingLeds-F28027 [Active - F2802x_FLASH]
    Binaries
    Includes
    ~Docs
    F2802x_FLASH
    F2802x_RAM
    FlashingLeds-Settings.h
    DSP2802x_CodeStartBranch.asm
    DSP2802x_GlobalVariableDefs.c
    FlashingLeds-DevInit_F2802x.c
    FlashingLeds-Main.c
    SciCommsGui.c
    DSP2802x_Headers_nonBIOS.cmd
    F28027_FLASH_FlashingLeds.CMD
    F28027_RAM_FlashingLeds.CMD [Excluded from Build]
    macros.ini
    rts2800_ml.lib
```

**[Project_Name]:** the project folder which contains the various settings for the project.
The following settings can be editted by right-clicking on the project name:

- The "Build Properties" selection brings the user to a dialog box where the following are editable on a per configuration basis:
    - Compiler Settings
        - Predefined Symbols – A Flash project defines the variable FLASH, that is used in the software to initialize the flash and copy memory from FLASH to RAM if needed.
        - Include Options – show the directories in which the compiler will search for headers.
        - Runtime Model Options – enables/disables floating point and/or CLA support
        - Optimizations –
    - Linker Settings
        - The path where the output file will be generated
        - The path where the map file will be generated. The map file shows how the linker placed the sections into memory and is very useful when debugging software.
- Changing the build configuration
- Adding or Linking files to the project
- Others…

**[Project_Name]-Settings.h:** Defines global settings for the project. (an incremental build option for example)
The settings found here are in the form of

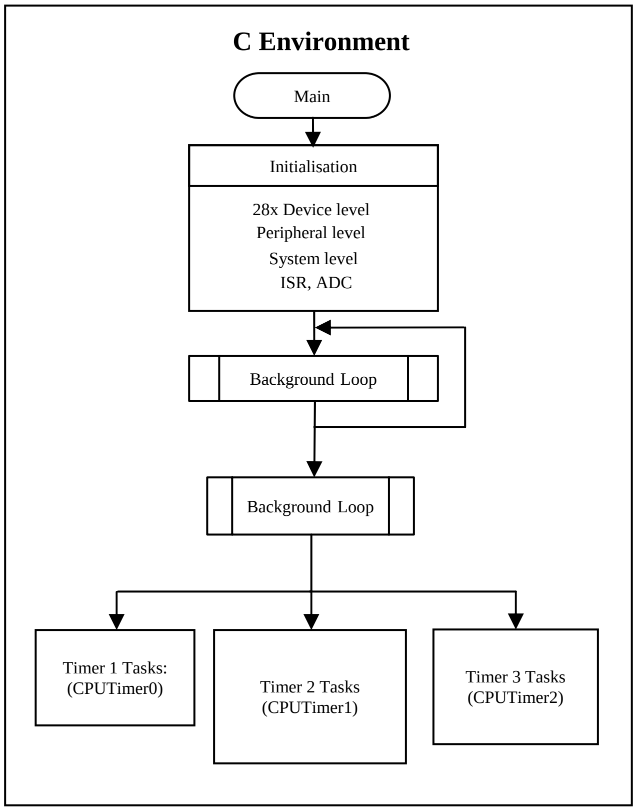
#define INCR_BUILD 1

This specific line of code defines the variable INCR_BUILD and replaces it in code with the number 1 at compile-time. This allows for conditional compilations and easier to read code.

Also note that this file is linked to both the main and ISR files, and the whole project will need to be rebuilt when this file is edited. This is done by selecting `Project -> "Rebuild All"`.

**[Project_Name]-Main.c:** Performs application management for the program and declares project variables and prototypes. The main file defines the main function, the application start point. The general flow for the main function is as follows:

- Run the DeviceInit function (located in **[Project_Name]-DevInit.c**).
- Define system variables
- Configure F28xxx peripherals and create connections to assembly blocks (if necessary)
- Initialize the system ISR
- Run the background loop. The background loop runs continually with three separate state machines (A, B, and C) each running off a separate CPU Timer. CpuTimer0 controls the frequency at which the A state machine runs. CpuTimer1 controls the frequency at which the B state machine runs, and CpuTimer2 controls the frequency of the C state machine. Each of these CpuTimer's periods are configured early in the main function and can be edited independently. Note that the actual frequency at which any A, B, or C *task* runs will be the frequency at which the CPU timer runs divided by the number of that state's tasks. These state tasks may be used to do many different things, for example:
    - Communications – A program may need to transmit data externally via a peripheral such as the CAN, SPI, or SCI (UART)
    - Variable Conversion/Scaling – Many variables used in a system are not completely obvious in what they translate to in real world units. This may happen for many reasons including the 0-3V range of the ADC as well as the way data is stored in various peripheral registers. The user may instead want to view these variables in real world units so that they can be sent externally to a data logger or so the program is more easy to use. In the F28xxx System Framework the prefix Gui_ denotes this type of variable
    - Project specific events such as updating coefficients or enabling/disabling an output.



The main file is separated into different sections based on the function of the section (ie "FUNCTION PROTOTYPES", "VARIABLE DECLARATIONS", etc.). Each of these sections is split into a FRAMEWORK and USER area. FRAMEWORK areas denote the template infrastructure code and generally will only need minor edits for each different application. USER areas are generally filled with functional example code that can be modified to fit an individual application.

**[Project_Name]-ISR.asm/[Project_Name]-ISR.c:** These files define each interrupt service routine's initialization and runtime actions. The project may contain an ISR file in assembly and/or in C based on the frequency of the ISRs, the computation being performed in each ISR, and the application.

- An assembly ISR file is usually required for applications such as Digital Power where the C-compiler induced overhead is prohibitive for the time critical tasks. Inside controlSUITE, several assembly-based libraries can be used to help create an ISR. For example, if installed, the digital power library can be found within controlSUITE at:

  \controlSUITE\libs\app_libs\digital_power

  Also note that any assembly ISR should push any registers the ISR will use to the stack prior to using any registers and then pop the registers back to their previous state when the ISR is complete.

  For more information on C28x assembly see the TMS320C28x DSP CPU and Instruction Set Reference Guide (spru430).

- A C ISR file can be used when the interrupts used will not run at an extremely high frequency (in motor control for example). Inside controlSUITE, several libraries can be used to help create a C ISR from TI-made blocks. One of these, the motor control library, can be found within controlSUITE at:

  \controlSUITE\libs\app_libs\motor_control

Note: An ISR routine should clear the interrupt flag used and acknowledge the interrupt in the PIE module before exiting so that the interrupt can run again.

**[Project_Name]-DevInit_28xxx:** Initializes the device and the device's functional pinout
The DevInit file disables the watchdog timer, sets the device's clock/PLL, initializes the PIE, enables/disables individual peripheral clocks, and then configures each GPIO to its own pin function. The DevInit file is target specific and is different based on the pin-out of the specific device.

Note the following CLA related files are valid for devices with CLA (such as the F28035), please refer to the device datasheet to indentify if the CLA is present on the device.

**[Project_Name]-CLA.asm:**
Initializes and defines the tasks that are run on a CLA (Control Law Accelerator). The CLA is designed to be module specialized for running a software loop quickly. As such, the PWM and ADC peripheral interrupts can be configured to start one of eight different CLA tasks or be manually started by the main CPU via software. This file defines what will happen in each of these tasks.

- CLA Init: Allocates memory and initializes CPU-to-CLA RAM and the various CLA library blocks.
- Tasks 1-7: Assembly code placed between _Cla**X**Task**Y** _CLA**X**T**Y**End specifies the software to be run during Task **Y**. Task **Y** can be started via ePWM**Y**, an ADC interrupt or by a CPU software force.
- Task 8: In the template projects, this task is used to initialize CLA-to-CPU message RAM and CLA data RAM (if allocated) memories to a known state.

The CLAmath, digital power library, and motor control libraries have several blocks built in for the CLA to use. Please see the documentation for each library for more information.

**[Project_Name]-CLAShared.h:**
This header file is included in Main.c and CLA.asm file and should be used to declare the shared variables and constants between the Main Core C28x and the CLA.

Note that the following files relate to the mechanism that allows the C28x registers to be defined as structs in the C language. This is done to improve the readability of code and makes writing C code more obvious.

**PeripheralHeaderIncludes.h:**
PeripheralHeaderIncludes.h is a target specific header file. This file is found within \development_kits\~SupportFiles and this header file is responsible for adding the set of F28xxx header files needed by the target and defining several target specific constants.

**DSP28xxx Header Files (ie DSP2803x_Adc.h):**
A device specific set of include files that define each peripheral's bits and registers as a C struct data structure. These registers are then grouped together by peripheral so that changing bits in a register is simpler and more readable.

These header files are found in:
\controlSUITE\device_support\

**DSP28xxx_GlobalVariableDefs.c**
Defines the registers used by the F28xxx Header File set and places them into a specifically named section of memory.

**DSP28xxx_Headers_nonBIOS.cmd**
This file defines the memory sections used by the DSP28xxx_GlobalVariableDefs.c and the DSP28xxx Header File set. The placement of peripherals in memory is fixed so this file should not be edited.

**[Device_Name]_[Config_Name]_[Project_Name].CMD -** Defines how the project will place different parts of code into memory.

Firstly, named parts of memory on the C2000 MCU are partitioned according to the specific device's datasheet. Because of the Harvard Bus architecture of the device, note that some memory is assigned to Page 0 (program bus sections) and other parts are assigned to Page 1 (data bus sections). Also note that RAM, especially of L0, L1, … can be combined or split up however necessary for a program. This means that part of L0 could be used as program RAM (perhaps a time-critical ISR) and another part as data RAM (used to store variables). For Flash and Ram the memory configurations of a project would be different and therefore a different file is used between flash and ram configurations.

Next, different sections of code are allocated to the various memory regions that were just defined. These sections are allocated in this file and defined within the source files of the project. In C, individual functions can be allocated to a section by doing:
#pragma CODE_SECTION(FunctionName, “SectionName”);

In assembly, lines of code are allocated to a specific section by placing these lines between .sect “SectionName” and .end.

```
.sect “SectionName”
MOVL XAR0, #0
.end
```

See the TMS320C28x Assembly Language Tools User's Guide (SPRU513) for more information on this file.

**DSP28xxx_CodeStartBranch.asm**
The code's starting point for C28x code. Allows the watchdog timer disable/enable to occur based on whether WD_DISABLE is defined. After this file is run, the program then goes to the main function.

**SciCommsGui.c:**

This file allows for the project to connect to an external GUI or any program on a host computer via the SCI peripheral and a RS-232 serialport/USB-serialport adapter.

**macros.ini**

This file defines path variables that can be used to make a project use more relative pathing as opposed to absolute pathing. This therefore allows the project to be more portable. See the section, "Creating a New Project from the Templates" for more information.

## References

For more information please see the following guides:

- **controlSUITE** – provides a single source for all C2000 application and product software

  http://www.ti.com/controlSUITE

- **TMS320C28x DSP CPU and Instruction Set Reference Guide** – gives a list of the assembly instructions available on the C2000's C28x device family.

  http://www.ti.com/lit/spru430

- **TMS320C28x Assembly Language Tools User's Guide** – Provides information on how the assembler and compiler work with the C2000 device family

  http://www.ti.com/lit/spru513