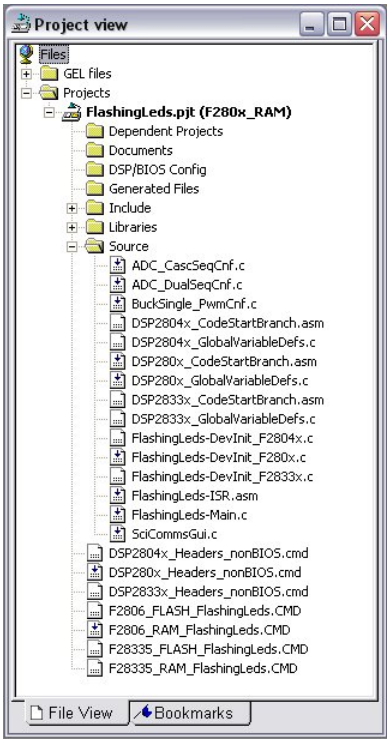
System Framework Overview

May 2008 – Revised August 2008

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:

- Configurability to allow the project to create an output file for any device within the TMS320F28x family either with RAM or production with FLASH
- FRAMEWORK and USER sections so that it is more obvious which code can be changed as desired and which areas should only be edited minimally.
- Built-in state machines that guarantee a task will run continually at a specific frequency
- 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 the SCI-peripheral

The project FlashingLeds.pjt has been created as a template to decrease the amount of time it takes to begin new software development projects.



Configurating the Project

The framework was developed in such a way as to allow any project to be developed for one target and simultaneously be used to generate code for any device in the F28xxx family. Additionally the project may be easily configured to build a program that will run in RAM for debug purposes and by a few clicks be able to generate one that will run in FLASH for production.

To change the configuration mode follow these steps:

1. Right-click on the project name in the Project Window and then selecting “Configurations…” in the context menu.
2. Select the correct configuration based on which target the code will be ran and whether the code will be ran in FLASH or RAM.

In the Project Window some files will have a blue downward-facing arrow. An arrow displayed on a file means that this file is active in the project for this configuration. Only active files are compiled and put into an output file.

Writing Code with the System Framework

Developing software with the F28x System Framework allows for project scalability throughout the F28x family of digital signal controllers. Designing within this scalability may lead to a few minor challenges for the software developer.

For instance, CodeSense (Code Composer Studio’s auto-completion tool) may have problems detecting the correct register definitions for the target controlCARD that the project is configured for. To avoid this issue you may need to open the file that corresponds to the registers you need to access:

1. Select the correct configuration for your controlCARD (see previous step)
2. Compile the project and fix any errors if necessary
3. In the project window, expand the folder marked “Include”
4. Open up the header file that contains the set of headers you need to access.

For example, if you want to edit the way a GPIO is configured while the code is running, opening the DSP28xxx_GPIO.h header file will guarantee that CodeSense will auto-complete with the correct register definitions.

Reference on Framework Files

[Project_Name].pjt: Defines many project, compiler, linker, and build settings

This file is open-able and editable by right-clicking its name on the project window then clicking “Open for Editing”. This project file contains the following:

- Project Settings – show the project directory (which changes when the project is moved) and the names of the various configurations available for this project
- Source Files – show the path of the source files in the project. Note that this path will be relative to the .pjt file.
- Compiler Settings – show the directories in which the compiler will search for headers. The –i prefix denotes include paths. Certain configurations also define the FLASH variable for the linker. The –d prefix denotes a definition.
- Linker Settings – show the path of the .out file and .map file created after linking
- Link Order
- Exclusions – changes whether certain files are active based on which configuration is chosen

This list is not a comprehensive list of the options available shown in the .pjt file. For further information please see the **Code Composer Development Tools v3.3 Getting Started Guide**.

[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

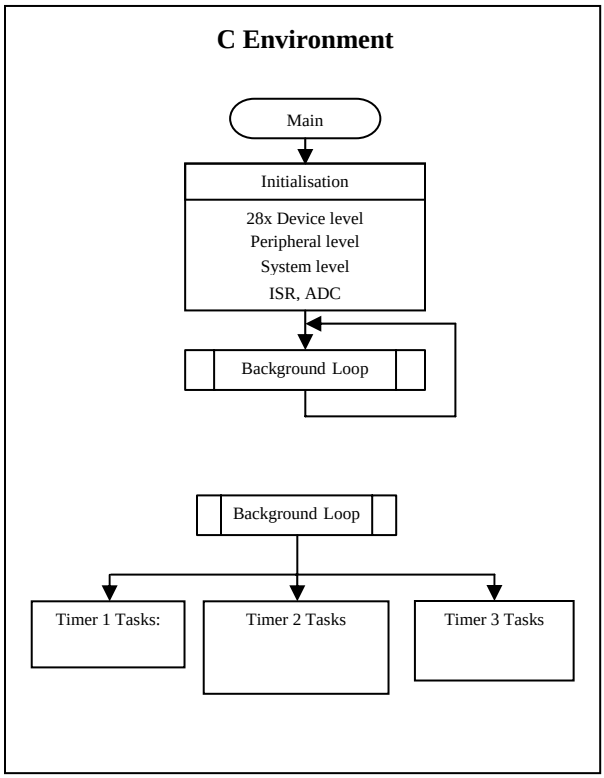
#define INCR_BUILD 1

This 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 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. 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
    - 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 routines initialization and runtime actions. The project may contain an ISR file in assembly and/or in C based on how time-critical individual ISRs in the project will be.

- An assembly ISR file is necessary for projects that require a very fast feedback loop such as those used for digital power. The ISRs located in this file may use optimized assembly module blocks found within various libraries installed by some software packages. The libraries installed will be found within:

  `C:\TI_F28xxx_SysSw\~SupportFiles\lib`

  The assembly modules essentially act as a system block diagram with certain input and output connections. These connections can be attached to other assembly blocks or a user controlled variable within the main file. The assembly modules may have an initialization macro (which should be called when the ISR is initialized) as well a runtime macro (which should run when the ISR pertaining to the assembly block is received).

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

- A C ISR file can be used when the interrupts used will not run periodically at a high frequency. Each interrupt used should be fully defined within the C ISR file and located inside this ISR file.

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

[Project_Name]-DevInit_28xxx: Initializes the device and the pinout functionality

The DevInit file disables the watchdog timer, sets the clock/PLL, initializes the PIE, 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.

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. The process of how to connect any project into an external GUI such as the GeneralPurposeGUI is detailed in **QSG-GeneralPurposeGUI**.

PeripheralHeaderIncludes.h:

PeripheralHeaderIncludes.h is a target specific header file that is chosen based on the include paths chosen in the [Project_Name].pjt file. This header file is responsible for adding the set of F28xxx header files needed by the target and defining several target specific constants. The target is known based off which configuration the project is in (see the section above title “Configuration”)

PeripheralAddress_ASM.h:

PeripheralAddress_ASM is a target specific header file that contains the addresses of the various peripherals. This file is only necessary for projects that have some assembly files that need to connect to peripherals. For files in C, the peripheral addresses are located in DSP280x_Headers_nonBIOS.cmd.

References

For more information please see the following guides:

- **QSG-GeneralPurposeGUI.pdf** – 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

- **Code Composer Development Tools v3.3 Getting Started Guide** – gives information on the various features within Code Composer.
  http://focus.ti.com/lit/ug/spru509h/spru509h.pdf

- **F28xxx User's Guides** –
  http://www.ti.com/f28xuserguides