A Round-Robin RTOS with Hardware Drivers in C-Language for dsPIC33 Micro-controllers

The Concept

Assuming we are using the microcontroller from the dsPIC33EP families. The microcontroller is connected to components providing key functions like voltage regulation, power supply decoupling, reset circuit, in-circuit serial programming port, external memory, external DAC module, precision voltage reference etc. These components together with the microcontroller are collectively called the **dsPIC33EP Core**. These circuitries are what we called **External Peripherals**. The microcontroller also has many built-in peripherals like ADC module, USB serial-interface engine, USART, SPI module, I2C module etc. We call these the **Internal Peripherals**. Sometimes we can also have external peripheral that duplicates the function of an internal peripheral, perhaps with better performance. To use the various internal and external hardware peripherals associated with the microcontroller, we write codes to setup and retrieve the states or signals from the peripherals in the form of sub-routines. Instead of repeating the same codes for every project, we store the codes in a common folder. Every project which needs to use the codes associated with a peripheral can 'link' to the codes in the project setup. We now call the codes associated with the peripheral as the **Peripheral Driver Library (PDL)**. Do note that this also mean when the codes for a peripheral driver library is modified, it will affect ALL projects! Thus we need to be careful to maintain backward compatibility to older projects.

Tools

Here we will be using MPLAB-X IDE environment for editing, compiling and linking the source codes. MPLAB XC C-compiler is used to compile and link the codes. We will be using MPLAB ICD-3 or PicKit-3 in-circuit debugger for uploading the hex data into the micro-controller.

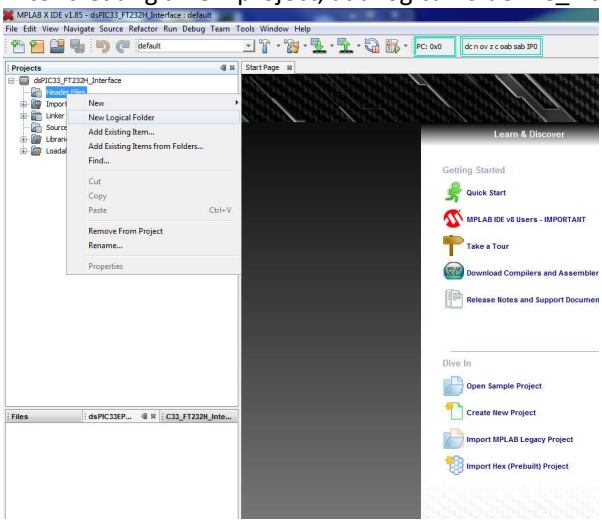
Basic Requirements to Using the Peripheral Driver Library (PDL)

To use the peripheral drivers, one needs to:

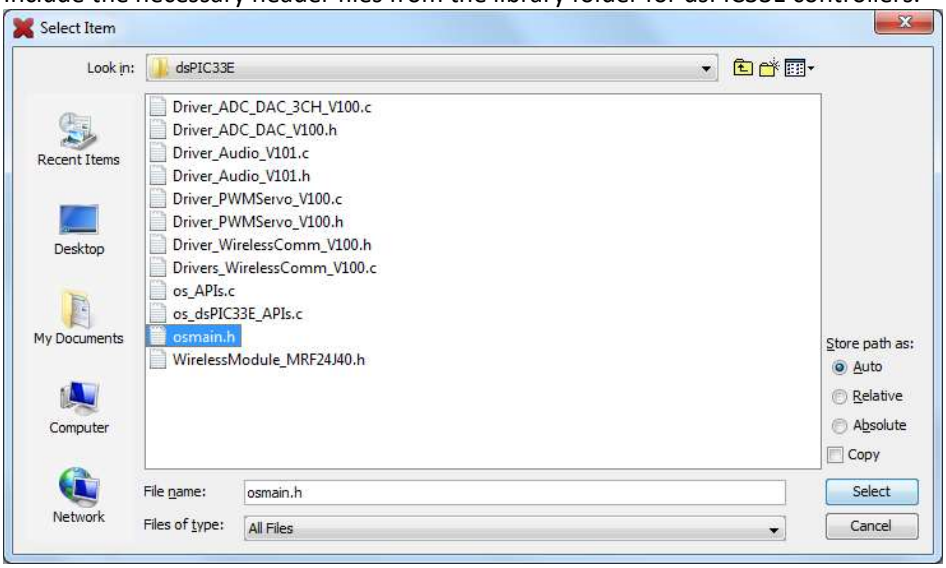
1. Map a Logical Folder to the physical folder that store that codes for the PDL.
2. Include the header file associated with the particular peripheral into the user codes. The header file contains the necessary global declaration of parameters used to access the peripheral. The procedures are shown below.

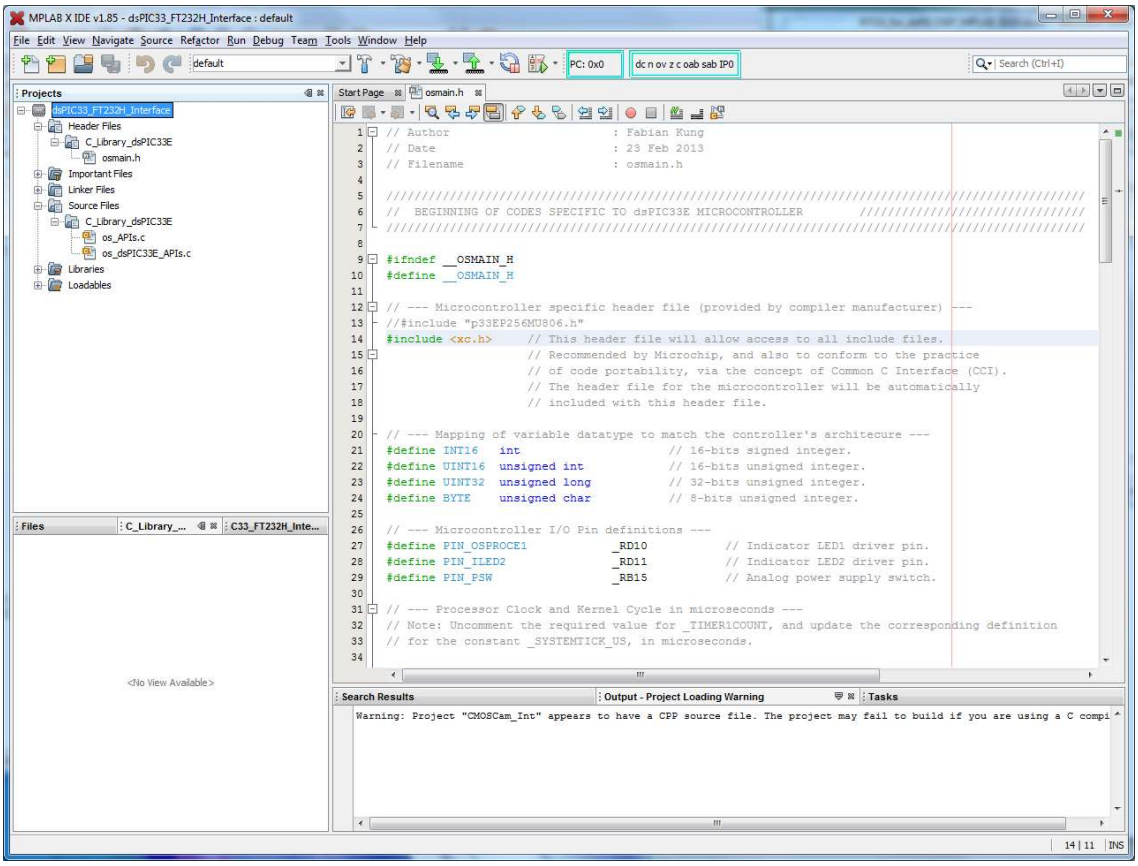
Steps

After creating a new project, add logical folder “C_Library_dsPIC33E”:

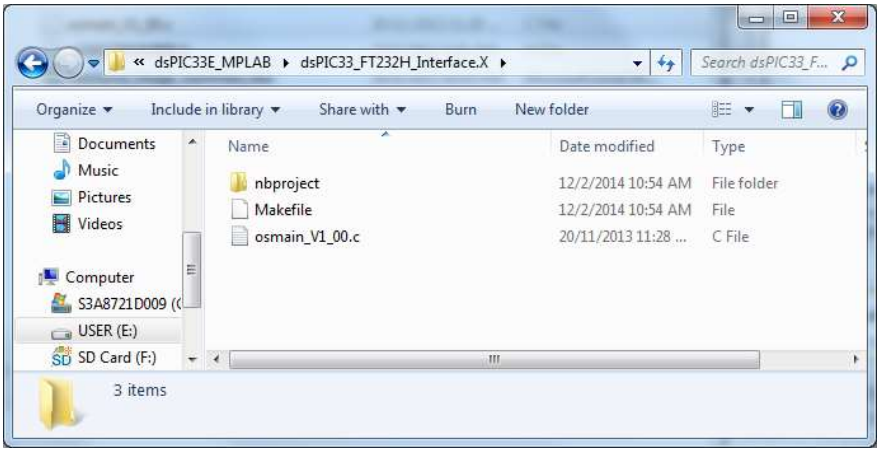


Include the necessary header files from the library folder for dsPIC33E controllers.

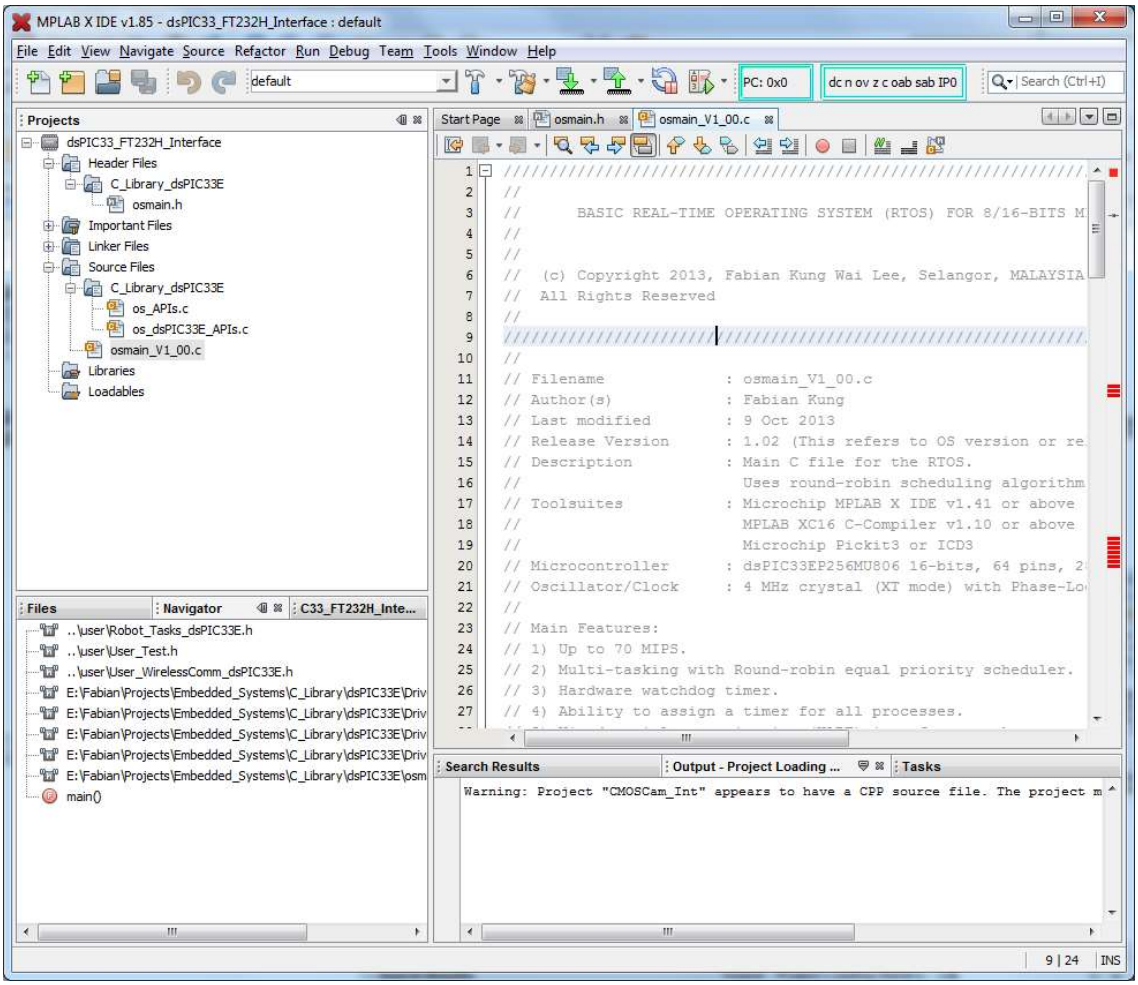




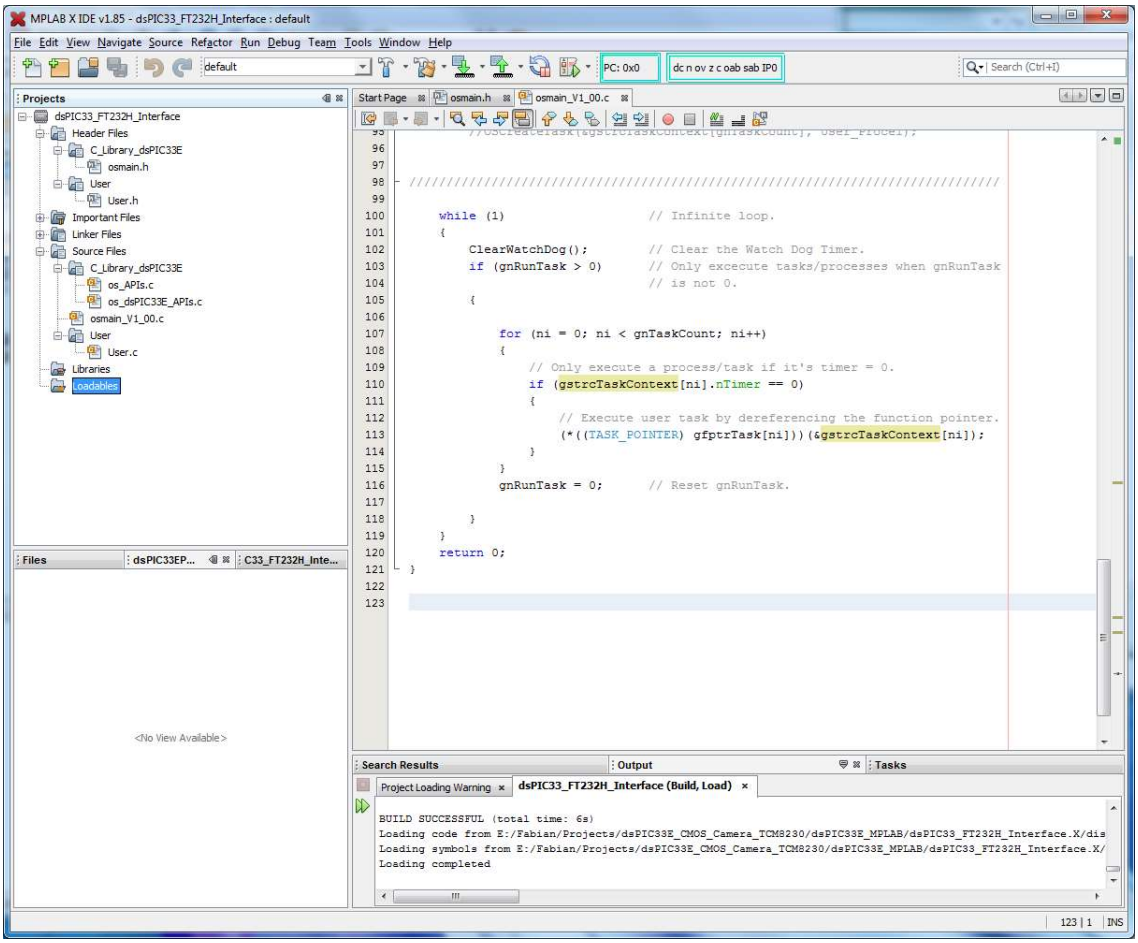
Finally copy the main C file “osmain_VXXXX.c” (XXXX being the version of the file) to the project folder.



Add the main C file to the current project (in the “Source Files” logical folder). The final logical directory and files required for a basic project is shown below. This project can be compiled and run, but there are no user files yet. There is a default task, the blink indicator LED task declared on the “os_dsPIC33E_APIs.c” file. So this project basically just blink the indicator LED on the controller board.



We can add user files to our project. Figure below shows the basic project structure while user files.



Source code listings:

File "os_main_V1_00.c":

```
////////////////////////////////////////////////////////////////////////////
//
// BASIC REAL-TIME OPERATING SYSTEM (RTOS) FOR 8/16-BITS MICROCONTROLLER
//
//
//  (c) Copyright 2013, Fabian Kung Wai Lee, Selangor, MALAYSIA
//  All Rights Reserved
//
////////////////////////////////////////////////////////////////////////////
//
// Filename                 : osmain_V1_00.c
// Author(s)                : Fabian Kung
// Last modified            : 12 Feb 2014
// Release Version          : 1.02 (This refers to OS version or release)
// Description              : Main C file for the RTOS.
//                            Uses round-robin scheduling algorithm to schedule
//                            tasks.
// Toolsuites               : Microchip MPLAB X IDE v1.81 or above
//                            MPLAB XC16 C-Compiler v1.11 or above
```

```
//                                  Microchip Pickit3 or ICD3
// Microcontroller  : dsPIC33EP256MU806 16-bits, 64 pins, 28kbyte RAM,
//                    256kbyte Flash ROM, 70 MIPS.
// Oscillator/Clock : 4 MHz crystal (XT mode) with Phase-Locked Loop
//                    multiplier to 140 MHz (70 MIPS)
//
// Main Features:
// 1) Up to 70 MIPS.
// 2) Multi-tasking with Round-robin equal priority scheduler.
// 3) Hardware watchdog timer.
// 4) Ability to assign a timer for all processes.
// 5) Wired serial communication (UART) interface stack.
// 6) Wireless RF communication stack using IEEE 802.15.4 or Bluetooth.
// 7) Include a new Resource Usage Summary, giving an overview of controller
//    and external circuits resources usage, pin usage etc.
//
// Directory structures:
// C library routines for various drivers/peripherals:
// <path for library>\C_Library\dsPIC33E
//
// <Project folder>\os\<os "main.c" codes>
// <Project folder>\user\<user codes>

////////////////////////////////////////////////////////////////////////////
// --- INCLUDE ALL DRIVER AND USER HEADER FILES --- ////////////////////////
////////////////////////////////////////////////////////////////////////////

// To include common header to all drivers and sources.  Here absolute path is used
// for some library sources.

// Main header file.  All source codes should include this.
#include "E: \Projects\Embedded_Systems\C_Library\dsPIC33E\osmain.h"

// Header file for each driver used.
#include "C: \Projects\Embedded_Systems\C_Library\dsPIC33E\Driver_ADC_DAC_V100.h"

// Header file user routines.
#include ".\User.h"
////////////////////////////////////////////////////////////////////////////

// --- MAIN LOOP ---

INT16 main(void)
{
        INT16 ni = 0;

        OSEnterCritical();          // Disable and store all processor's interrupt
                                    // setting.
        OSEnterCritical();          // Do this twice to prevent interrupt happen
                                    // midway, and the processer interrupt is
                                    // reenabled upon return from the interrupt
                                    // service routine.
        dsPIC33E_PORInit();         // Power-on Reset initialization for the
                                    // controller.
        OSInit();                   // Initialize the RTOS.
        gnTaskCount = 0;            // Initialize task counter.

                                    // Initialize core OS processes.
        OSCreateTask(&gstrcTaskContext[gnTaskCount], OSProce1); // Start the
                                    // blinking LED process.
                                    // User and other non-core OS tasks are
                                    // initialized here.
//      OSExitCritical();           // Enable all processor interrupts (optional,
                                    // some processor interrupts are turned on
                                    // within dsPIC33_PORInit routines, thus it is
                                    // not necessary for now).
```

```
////////////////////////////////////////////////////////////////////////
// --- CREATE AN INSTANCE OF DRIVER AND USER TASKS HERE --- ////////////////////
////////////////////////////////////////////////////////////////////////

        // Driver tasks:
        //OSCreateTask(&gstrcTaskContext[gnTaskCount], Proce_ADC_Driver);

        // User tasks:
        OSCreateTask(&gstrcTaskContext[gnTaskCount], User_Proce1);

////////////////////////////////////////////////////////////////////////

  while (1)                     // Infinite loop.
  {
    ClearWatchDog();            // Clear the Watch Dog Timer.
      if (gnRunTask > 0)        // Only excecute tasks/processes when gnRunTask
                  // is not 0.
      {

      for (ni = 0; ni < gnTaskCount; ni++)
      {
              // Only execute a process/task if it's timer = 0.
              if (gstrcTaskContext[ni].nTimer == 0)
              {
              // Execute user task by dereferencing the function pointer.
              (*((TASK_POINTER) gfptrTask[ni]))(&gstrcTaskContext[ni]);
              }
       }
       gnRunTask = 0;     // Reset gnRunTask.

    }
  }
  return 0;
}
```

Create the tasks for the various drivers here

Create the tasks for the user here

File "user.h":

```
// Author               : Fabian Kung
// Date                 : 12 Feb 2014
// Filename             : User.h

#ifndef _USER_H
#define _USER_H

void User_Proce1(TASK_ATTRIBUTE *);
#endif
```

File "user.c":

```
// Filename         : User.c
// Author(s)        : Fabian Kung
// Last modified    : 12 Feb 2014


// Include MPLAB XC16 standard libraries.
 #include <math.h>

// Include common header to all drivers and sources.  Here absolute path is
// used.  To edit if one changes folder.
#include "E:\ Projects\Embedded_Systems\C_Library\dsPIC33E\osmain.h"
// Include header files for driver used.  Here absolute path is used, to edit
// if one changes folder.
```

```
//
//
// --- PUBLIC VARIABLES ---
//

//
// --- PRIVATE FUNCTION PROTOTYPES ---
//



///
/// Process name            : User_Proce1
///
/// Author                  : Fabian Kung
///
/// Last modified           : 5 Aug 2013
///
/// Code Version            : 0.01
///
/// Processor               : dsPIC33EP256MU80X family.
///
/// Processor/System Resources
/// PINS                    :
///
/// MODULES                 :
///
/// RTOS                    : Ver 1 or above, round-robin scheduling.
///
/// Global variables        :

#ifdef          __OS_VER        // Check RTOS version compatibility.
        #if     __OS_VER < 1
                #error "User_Proce1: Incompatible OS version"
        #endif
#else
        #error "User_Proce1: An RTOS is required with this function"
#endif

/// Description
///

void User_Proce1(TASK_ATTRIBUTE *ptrTask)
{
   static int nCount = 0;

   if (ptrTask->nTimer == 0)
   {
        switch (ptrTask->nState)
        {
                case 0: // State 0 - Initialization.
                OSSetTaskContext(ptrTask, 1, 5000); // Next state=1, timer=5000.
                break;

                case 1: // State 1 - Turn on analog power supply.
                // Your task here.
                OSSetTaskContext(ptrTask, 2, 1); // Next state=3, timer =1
                break;

                case 2: // State 2 -
                // Your task here.
                OSSetTaskContext(ptrTask, 1, 1); // Next state=1, timer=1
                break;

                default:
                OSSetTaskContext(ptrTask, 0, 1); // Back to state=0, timer=1.
                break;
```

```
            }
        }
    }
```