```
/*******************************************************************************
  MPLAB Harmony Application Source File

  Company:
    Microchip Technology Inc.

  File Name:
    app.c

  Summary:
    This file contains the source code for the MPLAB Harmony application.

  Description:
    This file contains the source code for the MPLAB Harmony application.  It
    implements the logic of the application's state machine and it may call
    API routines of other MPLAB Harmony modules in the system, such as drivers,
    system services, and middleware.  However, it does not call any of the
    system interfaces (such as the "Initialize" and "Tasks" functions) of any of
    the modules in the system or make any assumptions about when those functions
    are called.  That is the responsibility of the configuration-specific system
    files.
 *******************************************************************************/

// DOM-IGNORE-BEGIN
/*******************************************************************************
Copyright (c) 2013-2014 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
 *******************************************************************************/
// DOM-IGNORE-END


// *****************************************************************************
// *****************************************************************************
// Section: Included Files
// *****************************************************************************
// *****************************************************************************

#include "app.h"
#include "Mc32_spi_sm.h"
#include "lcd_spi.h"
#include "system/devcon/src/sys_devcon_local.h"
#include "pec12.h"
#include "menu.h"



// *****************************************************************************
// *****************************************************************************
// Section: Global Data Definitions
// *****************************************************************************
// *****************************************************************************
```

```
70
71  // *****************************************************************************
72  /* Application Data
73
74    Summary:
75      Holds application data
76
77    Description:
78      This structure holds the application's data.
79
80    Remarks:
81      This structure should be initialized by the APP_Initialize function.
82
83      Application strings and buffers are be defined outside this structure.
84  */
85
86  APP_DATA appData;
87  SW S1;
88
89  // *****************************************************************************
90  // *****************************************************************************
91  // Section: Application Callback Functions
92  // *****************************************************************************
93  // *****************************************************************************
94
95  /* TODO:  Add any necessary callback functions.
96  */
97
98  // *****************************************************************************
99  // *****************************************************************************
100 // Section: Application Local Functions
101 // *****************************************************************************
102 // *****************************************************************************
103
104
105 /* TODO:  Add any necessary local functions.
106 */
107
108
109 // *****************************************************************************
110 // *****************************************************************************
111 // Section: Application Initialization and State Machine Functions
112 // *****************************************************************************
113 // *****************************************************************************
114
115
116 //------------------------------------------------------------------------------//
    APP_UpdateAppState
117 void APP_UpdateAppState(APP_STATES newState){
118
119     appData.appState = newState;
120 }
121
122 //------------------------------------------------------------------------------//
    APP_Initialize
123 void APP_Initialize ( void )
124 {
125     SPI_Init();
126     /* Place the App state machine in its initial state. */
127     appData.appState = APP_STATE_INIT;
128     appData.msCounter = 0;
129     appData.backLightIntensitiy = 2500; /* 100% */
130     appData.lightIntensity = 2500; /* 100% */
131     appData.exposureDuration = 100;
132     appData.timeBetweenPictures = 1000;
133     appData.isFiveShotsSeqEnable = false;
134     appData.seqClock1_ms = 0;
135     appData.angleBwEachSeq = 10;
136     appData.nbrOfShotsPerformed = 0;
```

```
137         appData.buzzerIntensity = 2500;
138         appData.valSeq = 0;
139 
140         initMenuParam();
141         initStepperParam();
142     }
143 
144 
145 
146     //------------------------------------------------------------------------//
        APP_Tasks
147     void APP_Tasks ( void ){
148 
149         static uint16_t counter1 = 0;
150         static uint16_t counter2 = 0;
151 
152         /* Main state machine */
153         switch(appData.appState){
154             //------------------------------------------------------------------------//
                APP_STATE_INIT
155             case APP_STATE_INIT:
156                 /* Read data from EEPROM to restore presets */
157                 readDataFromEeprom(getMyStepperStruct());
158                 /* Initialization of the motor */
159                 initStepperMotor();
160                 /* Update MCPWM Duty-cycle of other PWM with EEPROM data */
161                 updateMcpwmDuty();
162                 /* Turn on MCPWM */
163                 PLIB_MCPWM_Enable(MCPWM_ID_0);
164                 /* Initialization sequence */
165                 initLcd();
166                 /* Print initialization menu */
167                 printLcdInit();
168                 /* Start useful Timers */
169                 DRV_TMR1_Start();
170                 DRV_TMR2_Start();
171                 /* Print main menu once all peripherals are configured */
172                 printMainMenu();
173                 /* States machines update */
174                 APP_UpdateAppState(APP_STATE_WAIT);
175                 break;
176 
177             //------------------------------------------------------------------------//
                APP_STATE_SERVICE_TASKS
178             /* Frequency = 10'000Hz */
179             case APP_STATE_SERVICE_TASKS:
180 
181                 /* Process who is responsible of the sequence, motor orders and
182                  * lights orders. */
183 
184     //            SIGN_LED_CMDToggle();
185                 sequenceManagementProcess();
186 
187                 if(counter2 >= 10){
188                     /* Frequency = 1'000Hz */
189                     counter2 = 0;
190                     /* Scan the activity of the rotary encoder */
191                     scanPec12();
192                     /* Scan the activity of the switch S1 */
193                     scanSwitch();
194                 }
195                 if(counter1 >= 1000){
196                     /* Frequency = 10Hz */
197                     counter1 = 0;
198                     menuManagementProcess();
199                 }
200                 counter1++;
201                 counter2++;
202 
```

```
203
204             // Calls the SPI do task state machine
205             SPI_DoTasks();
206
207             /* States machines update */
208             APP_UpdateAppState(APP_STATE_WAIT);
209             break;
210         //-------------------------------------------------------------------//
        APP_STATE_WAIT
211         case APP_STATE_WAIT:
212             /* Nothing is supposed to happen here */
213             break;
214
215         //-------------------------------------------------------------------// default
216         default:
217             break;
218     }
219 }
220
221
222 //---------------------------------------------------------------------------//
    APP_Delay_ms
223 void APP_Delay_ms(uint32_t ms){
224
225     DRV_TMR3_Start();
226 //    SIGN_LED_CMDToggle();
227     while(appData.msCounter < ms){
228
229     }
230 //    SIGN_LED_CMDToggle();
231
232     DRV_TMR3_Stop();
233     appData.msCounter = 0;
234 }
235
236 //---------------------------------------------------------------------------//
    setBlIntensity
237 void setBlIntensity(int32_t *backLightIntensitiy){
238
239     // Limit values to avoid problems
240     if(*backLightIntensitiy < BACKLIGHT_INTENSITY_MIN) *backLightIntensitiy
241             = BACKLIGHT_INTENSITY_MIN;
242     if(*backLightIntensitiy > BACKLIGHT_INTENSITY_MAX) *backLightIntensitiy
243             = BACKLIGHT_INTENSITY_MAX;
244
245     /* 25 = 2500 / 100 */
246     appData.backLightIntensitiy = *backLightIntensitiy * 25;
247     PLIB_MCPWM_ChannelPrimaryDutyCycleSet(MCPWM_ID_0, PWM_BL_CH,
248             appData.backLightIntensitiy);
249 }
250 int32_t getBlIntensity(void){
251
252     return appData.backLightIntensitiy / 25;
253 }
254
255
256 //---------------------------------------------------------------------------//
    scanSwitch
257 void scanSwitch(void){
258
259     // Save old states for debounce
260     S1.state[3] = S1.state[2];
261     S1.state[2] = S1.state[1];
262     S1.state[1] = S1.state[0];
263     S1.state[0] = SWITCHStateGet();
264
265     // Check if switch is pressed
266     if(S1.state[0] == 1 && S1.state[1] == 1
267             && S1.state[2] == 0 && S1.state[3] == 0){
```

```
268
269             S1.isPressed = true;
270         }
271     }
272     //------------------------------------------------------------------------//
    getSwitchEvent
273     bool getSwitchEvent(void){
274
275         bool isPressed = S1.isPressed;
276         S1.isPressed = 0;
277
278         return isPressed;
279     }
280
281     //------------------------------------------------------------------------//
    initLcdSeq
282     void initLcd(void){
283
284         RESET_LCD_CMDOff();
285         APP_Delay_ms(1);
286         RESET_LCD_CMDOn();
287         APP_Delay_ms(10);
288         initDispl();
289         /* Create degree symbol for LCD uses */
290         CreateLcdDegreeSymbol(0x01);
291     }
292
293     void updateMcpwmDuty(void){
294
295         /* Update PWMs DutyCycle with data from EEPROM */
296         PLIB_MCPWM_ChannelPrimaryDutyCycleSet(MCPWM_ID_0, PWM_BL_CH,
297                 appData.backLightIntensitiy);
298         PLIB_MCPWM_ChannelPrimaryDutyCycleSet(MCPWM_ID_0, PWM_BUZZER_CH,
299                 appData.buzzerIntensity);
300         PLIB_MCPWM_ChannelPrimaryDutyCycleSet(MCPWM_ID_0, PWM_DIM_CH,
301                 appData.lightIntensity);
302     }
303
304
305
306
307
308     /*******************************************************************************
309      End of File
310      */
311
```