```
/*******************************************************************************
  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
```

```
/*******************************************************************************
  MPLAB Harmony Application Header File

  Company:
    Microchip Technology Inc.

  File Name:
    app.h

  Summary:
    This header file provides prototypes and definitions for the application.

  Description:
    This header file provides function prototypes and data type definitions for
    the application.  Some of these are required by the system (such as the
    "APP_Initialize" and "APP_Tasks" prototypes) and some of them are only used
    internally by the application (such as the "APP_STATES" definition).  Both
    are defined here for convenience.
*******************************************************************************/

//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

#ifndef _APP_H
#define _APP_H

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

#include "stepperDriver.h"
#include "lights.h"
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include "system_config.h"
#include "system_definitions.h"

// DOM-IGNORE-BEGIN
#ifdef __cplusplus  // Provide C++ Compatibility

extern "C" {

```

```
#endif
// DOM-IGNORE-END

// *****************************************************************************
// *****************************************************************************
// Section: Type Definitions
// *****************************************************************************
// *****************************************************************************

#define SYS_CLK 40000000

#define PWM_A_CMD_CH MCPWM_CHANNEL1
#define PWM_B_CMD_CH MCPWM_CHANNEL1
#define PWM_C_CMD_CH MCPWM_CHANNEL2
#define PWM_D_CMD_CH MCPWM_CHANNEL2
#define PWM_BL_CH MCPWM_CHANNEL3
#define PWM_BUZZER_CH MCPWM_CHANNEL4
#define PWM_DIM_CH MCPWM_CHANNEL6

#define MARGIN_LED_DELAY 50

/* Intensity in percent */
#define BACKLIGHT_INTENSITY_MIN 0
#define BACKLIGHT_INTENSITY_MAX 100


/* Value used to check if the EEPROM is already writent by this code */
#define CONTROL_VALUE 0x11223344

// *****************************************************************************
/* Application states

  Summary:
    Application states enumeration

  Description:
    This enumeration defines the valid application states.  These states
    determine the behavior of the application at various times.
*/

typedef enum
{
    /* Application's state machine's initial state. */
    APP_STATE_INIT=0,
    APP_STATE_SERVICE_TASKS,
    APP_STATE_SERVICE_CAPTURE,
    APP_STATE_WAIT,
} APP_STATES;

typedef enum{

    SYS_STATE_MENU = 0,
    SYS_STATE_MANUAL,
    SYS_STATE_AUTO
} SYSTEM_STATES;



// *****************************************************************************
/* Application Data

  Summary:
    Holds application data

  Description:
    This structure holds the application's data.

  Remarks:
    Application strings and buffers are be defined outside this structure.
```

```
 */

typedef enum{

    ALL_LED_DISABLE = 0,
    PWR_LED1,
    PWR_LED2,
    PWR_LED3,
    PWR_LED4,
    PWR_LED5,

}LED_ID;


typedef struct
{
    /* The application's current state */
    APP_STATES appState;
    SYSTEM_STATES systemState;
    LED_ID ledId;
    uint32_t msCounter;

    /* LED config */
    uint16_t lightIntensity;
    uint16_t timeBetweenPictures;
    uint16_t exposureDuration;

    /* Auto mode param */
    uint8_t angleBwEachSeq;

    uint32_t seqClock1_ms;
    uint32_t seqClock2_ms;
    bool isFiveShotsSeqEnable;
    bool isFullImaginSeqEnable;
    bool isFirstPass;
    uint16_t nbrOfShotsPerformed;
    uint8_t valSeq;

    uint16_t backLightIntensitiy;

    uint16_t buzzerIntensity;

} APP_DATA;

typedef struct
{
    bool state[4];
    bool isPressed;

} SW;

typedef struct{

    /* Motor data */
    int16_t     stepPerSec;
    uint16_t    stepPerTurn;
    uint16_t    gearValue;
    float       anglePerStep;

    /* LEDs data */
    uint16_t lightIntensity;
    uint16_t timeBetweenPictures;
    uint16_t exposureDuration;

    uint16_t backLightIntensitiy;

    /* Security value */
    uint32_t controlValue;

```

```
208       } DATA_IN_EEPROM;
209   // *****************************************************************************
210   // *****************************************************************************
211   // Section: Application Callback Routines
212   // *****************************************************************************
213   // *****************************************************************************
214   /* These routines are called by drivers when certain events occur.
215   */
216
217   // *****************************************************************************
218   // *****************************************************************************
219   // Section: Application Initialization and State Machine Functions
220   // *****************************************************************************
221   // *****************************************************************************
222
223   /*******************************************************************************
224     Function:
225       void APP_Initialize ( void )
226
227     Summary:
228        MPLAB Harmony application initialization routine.
229
230     Description:
231       This function initializes the Harmony application.  It places the
232       application in its initial state and prepares it to run so that its
233       APP_Tasks function can be called.
234
235     Precondition:
236       All other system initialization routines should be called before calling
237       this routine (in "SYS_Initialize").
238
239     Parameters:
240       None.
241
242     Returns:
243       None.
244
245     Example:
246       <code>
247       APP_Initialize();
248       </code>
249
250     Remarks:
251       This routine must be called from the SYS_Initialize function.
252   */
253
254   void APP_Initialize ( void );
255
256
257   /*******************************************************************************
258     Function:
259       void APP_Tasks ( void )
260
261     Summary:
262       MPLAB Harmony Demo application tasks function
263
264     Description:
265       This routine is the Harmony Demo application's tasks function.  It
266       defines the application's state machine and core logic.
267
268     Precondition:
269       The system and application initialization ("SYS_Initialize") should be
270       called before calling this.
271
272     Parameters:
273       None.
274
275     Returns:
276       None.
```

```
277
278     Example:
279       <code>
280       APP_Tasks();
281       </code>
282
283     Remarks:
284       This routine must be called from SYS_Tasks() routine.
285     */
286
287       void APP_Tasks( void );
288       void APP_Delay_ms(uint32_t ms);
289
290       void setBlIntensity(int32_t *backLightIntensitiy);
291       int32_t getBlIntensity(void);
292
293
294       void scanSwitch(void);
295       bool getSwitchEvent(void);
296
297       void initLcd(void);
298
299       void updateMcpwmDuty(void);
300
301
302
303   #endif /* _APP_H */
304
305   //DOM-IGNORE-BEGIN
306   #ifdef __cplusplus
307   }
308   #endif
309   //DOM-IGNORE-END
310
311   /*******************************************************************************
312    End of File
313    */
314
315
```

```c
/*
 * File:   stepperDriver.c
 * Author: ricch
 *
 * Created on August 30, 2023, 10:39 PM
 *
 * DRV8432 driver 2H bridge
 */


#include <stepperDriver.h>

static STEPPER_DATA stepperData;
extern APP_DATA appData;


//------------------------------------------------------------------------------//
initStepperData
void initStepperParam(void){

    stepperData.isAtHomeInCW     = false;
    stepperData.isAtHomeInCCW    = false;
    stepperData.isIndexed        = false;
    stepperData.isInAutoHomeSeq  = false;

    stepperData.performedSteps   = 0;
    stepperData.stepToReach      = 0;

    stepperData.stepPerSec       = 1000;

    stepperData.stepPerTurn      = 200;
    stepperData.gearValue        = 200;

    stepperData.anglePerStep     = 1.8;

    stepperData.dutyCycleStepper = 30;
}

void initStepperMotor(){

    //setStepperPower(&stepperData, &stepperData.dutyCycleStepper);

    /* Disable RESET on both H bridge */
    RESET_AB_CMDOn();
    RESET_CD_CMDOn();
}

//------------------------------------------------------------------------------//
turnOffStepperPwms
/* Disable all PWMs for motor control */
void turnOffStepperPwms(void){

    /* A */
    PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
    /* B */
    PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
    /* A_ */
    PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
    /* B_ */
    PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
}

//------------------------------------------------------------------------------//
changeSpeed
void changeSpeed(STEPPER_DATA *pStepperData){

    uint16_t tmrPerdiod = 0;
    uint16_t frequency = 0;
    //uint16_t presc = 0;
```

```
67
68      frequency = pStepperData->stepPerSec;
69      //presc = TMR_PrescaleGet_Default(TMR_ID_3);
70      tmrPerdiod = SYS_CLK / (frequency * 16) - 1;
71      PLIB_TMR_Counter16BitClear(TMR_ID_3);
72      PLIB_TMR_Period16BitSet(TMR_ID_3, tmrPerdiod);
73  }
74
75  //------------------------------------------------------------------------//
    processStepper
76  void processStepper(STEPPER_DATA *pStepperData){
77
78      static uint8_t step = 0;
79      //---------------------------// Counter clockwise CCW
80      if(pStepperData->performedSteps > pStepperData->stepToReach){
81          if(pStepperData->isAtHomeInCCW == false){
82              switch(step){
83                  /* Sequence of 4 steps for CCW rotation */
84                  case 1:
85                      /* A */
86                      PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
87                      /* B */
88                      PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
89                      /* A_ */
90                      PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
91                      /* B_ */
92                      PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
93                      break;
94
95                  case 2:
96                      PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
97                      PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
98                      PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
99                      PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
100                     break;
101
102                 case 3:
103                     PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
104                     PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
105                     PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
106                     PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
107                     break;
108
109                 case 0:
110                     PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
111                     PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
112                     PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
113                     PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
114                     break;
115             }
116             step++;
117         }
118         /* Four steps performed in CCW */
119         if(step == 4){
120
121             step = 0;
122             pStepperData->performedSteps -= 4;
123         }
124         /* Index is reach in CCW */
125         if(INDEXStateGet() && pStepperData->isAtHomeInCW == false){
126
127             pStepperData->isAtHomeInCCW = true;
128 //            pStepperData->stepToDoReach = pStepperData->performedStep;
129
130             if(pStepperData->isInAutoHomeSeq == true){
131
132                 pStepperData->stepToReach = 0;
133                 pStepperData->performedSteps = 0;
134                 pStepperData->isIndexed = true;
```

```
                    pStepperData->isInAutoHomeSeq = false;
                }
            }
            else pStepperData->isAtHomeInCCW = false;
        }
        //---------------------------// Clockwise CW
        else if(pStepperData->performedSteps < pStepperData->stepToReach){
            if(pStepperData->isAtHomeInCW == false){
                switch(step){
                    /* Sequence of 4 steps for CW rotation */
                    case 1:
                        /* A */
                        PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        /* B */
                        PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        /* A_ */
                        PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        /* B_ */
                        PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        break;

                    case 0:
                        PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        break;

                    case 3:
                        PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        PLIB_MCPWM_ChannelPWMxHEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        PLIB_MCPWM_ChannelPWMxLDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        break;

                    case 2:
                        PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        PLIB_MCPWM_ChannelPWMxHDisable(MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL1);
                        PLIB_MCPWM_ChannelPWMxLEnable (MCPWM_ID_0 ,MCPWM_CHANNEL2);
                        break;
                }
                step++;
            }
            /* Four steps performed in CW */
            if(step == 4){

                step = 0;
                pStepperData->performedSteps += 4;
            }
            /* Index is reach in CW */
            if(INDEXStateGet() && pStepperData->isAtHomeInCCW == false){

                pStepperData->isAtHomeInCW = true;
                /* Stop the automatic sequence */
                appData.isFullImaginSeqEnable = false;
                /* Stop the motor */
                pStepperData->stepToReach = pStepperData->performedSteps;
            }
            else pStepperData->isAtHomeInCW = false;
        }


        // The motor reach its desired position
    //    if(pStepperData->performedSteps == pStepperData->stepToReach){
    ////        turnOffStepperPwms();
    //    } else {
    //
    ////        PLIB_MCPWM_Enable(MCPWM_ID_0);
```

```
204   //    }
205   }
206
207
208
209   //-------------------------------------------------------------------------// setSpeed
210   void setSpeed(STEPPER_DATA *pStepperData, uint32_t *pStepPerSec){
211
212       // Limit values to avoid problems
213       if(*pStepPerSec < STEP_PER_SEC_MIN) *pStepPerSec = STEP_PER_SEC_MIN;
214       if(*pStepPerSec > STEP_PER_SEC_MAX) *pStepPerSec = STEP_PER_SEC_MAX;
215
216       // Save data
217       pStepperData->stepPerSec = *pStepPerSec;
218   }
219
220   int32_t getSpeed(STEPPER_DATA *pStepperData){
221
222       return pStepperData->stepPerSec;
223   }
224
225   //-------------------------------------------------------------------------//
      setGearReduction
226   void setGearReduction(STEPPER_DATA *pStepperData, uint32_t *pGearValue){
227
228       // Limit values to avoid problems
229       if(*pGearValue < GEAR_VALUE_MIN) *pGearValue = GEAR_VALUE_MIN;
230       if(*pGearValue > GEAR_VALUE_MAX) *pGearValue = GEAR_VALUE_MAX;
231
232       // Save data
233       pStepperData->gearValue = *pGearValue;
234   }
235   //-------------------------------------------------------------------------//
      getGearReduction
236   uint32_t getGearReduction(STEPPER_DATA *pStepperData){
237
238       return pStepperData->gearValue;
239   }
240
241   //-------------------------------------------------------------------------//
      setAnglePerStep
242   void setAnglePerStep(STEPPER_DATA *pStepperData, uint32_t *pAnglePerStep){
243
244       float temp = (*pAnglePerStep / 10.0);
245
246       // Limit values to avoid problems
247       if(temp < ANGLE_PER_STEP_MIN) temp = (ANGLE_PER_STEP_MIN);
248       if(temp > ANGLE_PER_STEP_MAX) temp = (ANGLE_PER_STEP_MAX);
249       *pAnglePerStep = temp * 10;
250
251       // Save data
252       pStepperData->anglePerStep = temp;
253   }
254   //-------------------------------------------------------------------------//
      getAnglePerStep
255   uint32_t getAnglePerStep(STEPPER_DATA *pStepperData){
256
257       // x10 ???
258       return pStepperData->anglePerStep * 10;
259   }
260
261   //-------------------------------------------------------------------------//
      getPerformedSteps
262   int32_t getPerformedSteps(STEPPER_DATA *pStepperData){
263
264       return pStepperData->performedSteps / pStepperData->stepPerTurn;
265   }
266
267
```

```
268 //------------------------------------------------------------------------//
    setRotationToDo
269 void setRotationToDo(STEPPER_DATA *pStepperData, int32_t *pRotationToDo){
270
271     // Limit values to avoid problems
272     if(*pRotationToDo < ROTATION_TO_DO_MIN) *pRotationToDo = ROTATION_TO_DO_MIN;
273     if(*pRotationToDo > ROTATION_TO_DO_MAX) *pRotationToDo = ROTATION_TO_DO_MAX;
274
275     // Save data
276     pStepperData->stepToReach = *pRotationToDo * pStepperData->stepPerTurn;
277 }
278 //------------------------------------------------------------------------//
    getRotationTodo
279 int32_t getRotationToDo(STEPPER_DATA *pStepperData){
280
281     return pStepperData->stepToReach / pStepperData->stepPerTurn;
282 }
283
284 //------------------------------------------------------------------------// autoHome
285 void startAutoHome(STEPPER_DATA *pStepperData){
286
287     pStepperData->isInAutoHomeSeq = true;
288     // Check if the arm is not at home
289     if(pStepperData->isAtHomeInCCW == false){
290         // Put steps to do for returning home in CCW
291         pStepperData->stepToReach = -50000; // DEFINE? STEP_TO_DO_MAX
292     }
293 }
294
295 //------------------------------------------------------------------------//
    setStepperPower
296 void setStepperPower(STEPPER_DATA *pStepperData, uint16_t *pDutyCycleStepper){
297
298     uint16_t dutyValCh1 = 0;
299
300     // Limit values to avoid problems
301     if(*pDutyCycleStepper < MCPWM_DUTYCYCLE_MIN) *pDutyCycleStepper
302             = MCPWM_DUTYCYCLE_MIN;
303     if(*pDutyCycleStepper > MCPWM_DUTYCYCLE_MAX) *pDutyCycleStepper
304             = MCPWM_DUTYCYCLE_MAX;
305
306     /* Save configuration in the structure */
307     pStepperData->dutyCycleStepper = *pDutyCycleStepper;
308
309     /* Must be the inverse of the CHANNEL 1 */
310     dutyValCh1 = MCPWM_PRIMARY_PERIOD - *pDutyCycleStepper;
311
312     PLIB_MCPWM_ChannelPrimaryDutyCycleSet(MCPWM_ID_0 ,MCPWM_CHANNEL1,
313             dutyValCh1);
314     PLIB_MCPWM_ChannelPrimaryDutyCycleSet(MCPWM_ID_0 ,MCPWM_CHANNEL2,
315             *pDutyCycleStepper);
316 }
317
318 int16_t getStepperPower(STEPPER_DATA *pStepperData){
319
320     return pStepperData->dutyCycleStepper;
321 }
322
323
324 //------------------------------------------------------------------------//
    getStepperStruct
325 STEPPER_DATA* getMyStepperStruct(void){
326
327     /* Return the address of the structure */
328     return &stepperData;
329 }
```

```c
/*
 * File:   stepperDriver.h
 * Author: ricch
 *
 * Created on August 30, 2023, 10:39 PM
 */

#ifndef STEPPERDRIVER_H
#define STEPPERDRIVER_H

#ifdef __cplusplus
extern "C" {
#endif

    #include <stdint.h>
    #include "system_definitions.h"

    // Defines
    #define STEP_PER_SEC_MIN 40
    #define STEP_PER_SEC_MAX 1000

    #define GEAR_VALUE_MIN 1
    #define GEAR_VALUE_MAX 1000

    #define STEP_PER_TURN_MIN 4
    #define STEP_PER_TURN_MAX 400

    #define ANGLE_PER_STEP_MIN 0.1
    #define ANGLE_PER_STEP_MAX 10.0

    #define ROTATION_TO_DO_MIN -50000
    #define ROTATION_TO_DO_MAX 50000

    /* Period for 50kHz PWMs */
    #define MCPWM_PRIMARY_PERIOD 199
    #define MCPWM_DUTYCYCLE_MIN 9
    #define MCPWM_DUTYCYCLE_MAX 189

    // Structures
    typedef struct{

        /* Motion motor data */
        bool        isAtHomeInCW;
        bool        isAtHomeInCCW;
        bool        isIndexed;
        bool        isInAutoHomeSeq;

        int32_t     performedSteps;
        int32_t     stepToReach;

        /* Motor characteristics */
        int16_t     stepPerSec;

        uint16_t    stepPerTurn;
        uint16_t    gearValue;

        float       anglePerStep;

        uint16_t    dutyCycleStepper;

    } STEPPER_DATA;



    // Prototypes
    void initStepperParam(void);
    void turnOffStepperPwms(void);
    void changeSpeed(STEPPER_DATA *pStepperData);
    void processStepper(STEPPER_DATA *pStepperData);
```

```
70
71
72      void setSpeed(STEPPER_DATA *pStepperData, uint32_t *pStepPerSec);
73      int32_t getSpeed(STEPPER_DATA *pStepperData);
74      void setGearReduction(STEPPER_DATA *pStepperData, uint32_t *pGearValue);
75      uint32_t getGearReduction(STEPPER_DATA *pStepperData);
76      void setAnglePerStep(STEPPER_DATA *pStepperData, uint32_t *pAnglePerStep);
77      uint32_t getAnglePerStep(STEPPER_DATA *pStepperData);
78      int32_t getPerformedSteps(STEPPER_DATA *pStepperData);
79      void setRotationToDo(STEPPER_DATA *pStepperData, int32_t *pRotationToDo);
80      int32_t getRotationToDo(STEPPER_DATA *pStepperData);
81      void startAutoHome(STEPPER_DATA *pStepperData);
82
83      void setStepperPower(STEPPER_DATA *pStepperData, uint16_t *pDutyCycleStepper);
84      int16_t getStepperPower(STEPPER_DATA *pStepperData);
85
86      STEPPER_DATA* getMyStepperStruct(void);
87
88  #ifdef  __cplusplus
89  }
90  #endif
91
92  #endif  /* STEPPERDRIVER_H */
93
94
```

```c
/*
 * File:   menu.c
 * Author: ricch
 *
 * Created on August 30, 2023, 10:15 AM
 */

#include "menu.h"
#include "lcd_spi.h"
#include "stdio.h"
#include "pec12.h"
#include "stepperDriver.h"


MENU menu;
extern APP_DATA appData;
bool isInModifMode = 0;
bool isFirstDataProcessPass = false;


void initMenuParam(){

    menu.menuPage = 0;
    menu.menuSize = 2;
    menu.menuState = MAIN_MENU;
}


//_____________________________________________________________________________//
menuManagementProcess
void menuManagementProcess(void){

    static int32_t pec12RotationValue = 0;

    /* Get PEC12 increments or decrements if there are */
    int incrOrDecr = getPec12IncrOrDecr();
    //-------------------------------------------------------------------------//
    isInModifMode == true
    if(isInModifMode){

        pec12RotationValue += incrOrDecr;
        menuDataProcess(&pec12RotationValue, getMyStepperStruct());
        menuPrintProcess(getMyStepperStruct());

        /* PEC12 switch pressed */
        if(getPec12SwitchEvent()){

            /* Leave modification mode */
            isInModifMode = false;
            /* Put the cursor on the first line */
            pec12RotationValue = 0;
        }
    }
    //-------------------------------------------------------------------------//
    isInModifMode == false
    else if(isInModifMode == false){

        pec12RotationValue += incrOrDecr;

        if(pec12RotationValue > menu.menuSize) pec12RotationValue =
                menu.menuSize;
        else if(pec12RotationValue < 0) pec12RotationValue = 0;

        if(pec12RotationValue <= 3){

            menu.menuPage = 0;
            menuPrintProcess(getMyStepperStruct());
            printCursor(pec12RotationValue);
        }
```

```
67          else{
68
69              menu.menuPage = 1;
70              menuPrintProcess(getMyStepperStruct());
71              printCursor(pec12RotationValue - 4);
72          }
73
74          /* PEC12 switch pressed */
75          if(getPec12SwitchEvent()){
76
77              menuActionProcess(pec12RotationValue);
78              menuDataProcess(&pec12RotationValue, getMyStepperStruct());
79              menuPrintProcess(getMyStepperStruct());
80
81              /* Put the cursor on the first line */
82  //            pec12RotationValue = 0;
83          }
84      }
85      if(getSwitchEvent()){
86
87          //startFiveShotsSequence();
88  //        startFullImagingSequence();
89
90          switch (menu.menuState){
91
92              case MANUAL_MODE_MENU:
93                  /* Start a sequence of 5 pictures */
94                  //startFiveShotsSequence();
95                  startFiveShotsSeqProcess();
96                  break;
97
98              default:
99                  break;
100         }
101     }
102 }
103
104
105 //_____________________________________________________________________//
106 void menuActionProcess(int32_t pec12RotationValue){
107
108     /* Menu action switch */
109     if(isInModifMode == false){
110         switch(menu.menuState){
111
112             //-------------------------------------------------------------// Main
                menu
113             case MAIN_MENU:
114
115                 switch(pec12RotationValue){
116
117                     case CAPTURE_MODE_SEL:
118                         menu.menuState = CAPTURE_MODE_MENU;
119                         menu.menuSize = 2;
120                         break;
121
122                     case SETTINGS_SEL:
123                         menu.menuState = SETTINGS_MENU;
124                         menu.menuSize = 5;
125                         break;
126
127                     case ABOUT_SEL:
128                         menu.menuState = ABOUT_MENU;
129                         menu.menuSize = 0;
130                         break;
131                 }
132                 break;
133
134             //-------------------------------------------------------------// Main
```

```
                menu -> Choice menu
135             case CAPTURE_MODE_MENU:
136
137                 switch(pec12RotationValue){
138
139                     case RETURN_SEL:
140                         menu.menuState = MAIN_MENU;
141                         menu.menuSize = 2;
142                         break;
143
144                     case MANUAL_MODE_SEL:
145                         menu.menuState = MANUAL_MODE_MENU;
146                         menu.menuSize = 2;
147                         break;
148
149                     case AUTOMATIC_MODE_SEL:
150                         menu.menuState = AUTOMATIC_MODE_MENU;
151                         menu.menuSize = 1;
152                         break;
153                 }
154                 break;
155
156             //-----------------------------------------------------------------// Main
                menu -> Choice menu -> Manual Mode
157             case MANUAL_MODE_MENU:
158                 switch(pec12RotationValue){
159
160                     case RETURN_SEL:
161                         menu.menuState = CAPTURE_MODE_MENU;
162                         menu.menuSize = 2;
163                         break;
164
165                     case AUTO_HOME_SEL:
166                         menu.menuState = AUTO_HOME_MENU;
167                         menu.menuSize = 1;
168                         break;
169
170                     case ANGLE_SEL:
171                         menu.modifState = ANGLE_MODIF;
172                         isInModifMode = true;
173                         isFirstDataProcessPass = true;
174                         break;
175                 }
176                 break;
177
178             //-----------------------------------------------------------------// Main
                menu -> Auto menu
179             case AUTOMATIC_MODE_MENU:
180                 switch(pec12RotationValue){
181
182                     case RETURN_SEL:
183                         menu.menuState = CAPTURE_MODE_MENU;
184                         menu.menuSize = 2;
185                         break;
186
187                     case AUTOMATIC_MODE_START_SEL:
188                         menu.modifState = AUTOMATIC_MODE_START;
189                         isInModifMode = true;
190                         isFirstDataProcessPass = true;
191                         break;
192                 }
193                 break;
194
195             //-----------------------------------------------------------------// Main
                menu -> Manual menu -> Auto home
196             case AUTO_HOME_MENU:
197                     switch(pec12RotationValue){
198
199                     case RETURN_SEL:
```

```
                        menu.menuState = MANUAL_MODE_MENU;
                        menu.menuSize = 2;
                        break;

                    case AUTO_HOME_START_SEL:
                        menu.modifState = AUTO_HOME_START;
                        isInModifMode = true;
                        isFirstDataProcessPass = true;
                        break;
                    }
                break;

                //returnToHome(); PEUT ETRE METTRE AILLEUR
                break;

            //-----------------------------------------------------------------// Main menu -> Settings menu
            case SETTINGS_MENU:

                switch(pec12RotationValue){

                    case RETURN_SEL:
                        menu.menuState = MAIN_MENU;
                        menu.menuSize = 2;
                        break;

                    case MOTOR_SEL:
                        menu.menuState = MOTOR_MENU;
                        menu.menuSize = 4;
                        break;

                    case LEDS_SEL:
                        menu.menuState = LIGHT_MENU;
                        menu.menuSize = 1;
                        break;

                    case BACKLIGHT_SEL:
                        menu.menuState = BACKLIGHT_MENU;
                        menu.menuSize = 1;
                        break;

                    case CAMERA_SEL:
                        menu.menuState = CAMERA_MENU;
                        menu.menuSize = 3;
                        break;

                    case SAVE_DATA_SEL:
                        menu.menuState = SAVE_DATA_MENU;
                        menu.menuSize = 1;
                        break;

                }
                break;

            //-----------------------------------------------------------------// Main menu -> Settings menu -> Motor menu
            case MOTOR_MENU:
                switch(pec12RotationValue){

                    case RETURN_SEL:
                        menu.menuState = SETTINGS_MENU;
                        menu.menuSize = 5;
                        break;

                    case SPEED_SEL:
                        menu.modifState = SPEED_MODIF;
                        isInModifMode = true;
                        isFirstDataProcessPass = true;
                        break;
```

```
267
268                     case GEAR_SEL:
269                         menu.modifState = GEAR_MODIF;
270                         isInModifMode = true;
271                         isFirstDataProcessPass = true;
272                         break;
273
274                     case STEP_PER_TURN_SEL:
275                         menu.modifState = STEP_PER_TURN_MODIF;
276                         isInModifMode = true;
277                         isFirstDataProcessPass = true;
278                         break;
279
280                     case POWER_SEL:
281                         menu.modifState = POWER_MODIF;
282                         isInModifMode = true;
283                         isFirstDataProcessPass = true;
284                         break;
285                 }
286                 break;
287
288             //-------------------------------------------------------------// Main menu -> Settings menu -> Light menu
289             case LIGHT_MENU:
290                 switch(pec12RotationValue){
291
292                     case RETURN_SEL:
293                         menu.menuState = SETTINGS_MENU;
294                         menu.menuSize = 5;
295                         break;
296
297                     case LIGHT_INTENSITY_SEL:
298                         menu.modifState = LIGHT_INTENSITY_MODIF;
299                         isInModifMode = true;
300                         isFirstDataProcessPass = true;
301                         break;
302
303 //                    case LIGHT_TIME_SEL: // <--- in camera param
304 //                        menu.modifState = LIGHT_TIME_MODIF;
305 //                        isInModifMode = true;
306 //                        isFirstDataProcessPass = true;
307 //                        break;
308                 }
309                 break;
310
311             //-------------------------------------------------------------// Main menu -> Settings menu -> Back-light menu
312             case BACKLIGHT_MENU:
313                 switch(pec12RotationValue){
314
315                     case RETURN_SEL:
316                         menu.menuState = SETTINGS_MENU;
317                         menu.menuSize = 5;
318                         break;
319
320                     case LIGHT_INTENSITY_SEL:
321                         menu.modifState = BL_INTENSITY_MODIF;
322                         isInModifMode = true;
323                         isFirstDataProcessPass = true;
324                         break;
325                 }
326                 break;
327
328             //-------------------------------------------------------------// Main menu -> Settings menu -> Camera
329             case CAMERA_MENU:
330                 switch(pec12RotationValue){
331
332                     case RETURN_SEL:
```

```
                        menu.menuState = SETTINGS_MENU;
                        menu.menuSize = 5;
                        break;

                    case EXPOSURE_TIME_SEL:
                        menu.modifState = EXPOSURE_TIME_MODIF;
                        isInModifMode = true;
                        break;

                    case TIME_BW_PICTURES_SEL:
                        menu.modifState = TIME_BW_PICTURES_MODIF;
                        isInModifMode = true;
                        break;
                }
                isFirstDataProcessPass = true;
                break;

            //-----------------------------------------------------------------// Main menu -> Settings menu -> Save data
            case SAVE_DATA_MENU:
                switch(pec12RotationValue){

                    case RETURN_SEL:
                        menu.menuState = SETTINGS_MENU;
                        menu.menuSize = 5;
                        break;

                    case SAVE_DATA_SEL - 4:
                        menu.modifState = SAVE_DATA_START;
                        isInModifMode = true;
                        break;
                }
                isFirstDataProcessPass = true;
                break;

            //-----------------------------------------------------------------// Main menu -> About menu
            case ABOUT_MENU:

                switch(pec12RotationValue){

                    case RETURN_SEL:
                        menu.menuState = MAIN_MENU;
                        menu.menuSize = 2;
                        break;
                }
                break;

            default:
                break;
        }
    }
}

//_____________________________________________________________________//
void menuDataProcess(int32_t *pec12RotationValue, STEPPER_DATA *pStepperData){

    /* Data action switch */
    if(isInModifMode){
        switch(menu.modifState){

            //-----------------------------------------------------------//
            ANGLE_MODIF
            case ANGLE_MODIF:
                if(isFirstDataProcessPass){

                    isFirstDataProcessPass = false;
                    /* A TESTER ET VALIDER, PERTE DE PAS POSSIBLE */
                    *pec12RotationValue = getRotationToDo(pStepperData);
```

```
399                 }
400                 setRotationToDo(pStepperData, pec12RotationValue);
401                 break;
402 
403             //-----------------------------------------------------------------//
                SPEED_MODIF
404             case SPEED_MODIF:
405                 if(isFirstDataProcessPass){
406 
407                     isFirstDataProcessPass = false;
408                     *pec12RotationValue = getSpeed(pStepperData);
409                 }
410                 setSpeed(pStepperData, pec12RotationValue);
411                 break;
412 
413             //-----------------------------------------------------------------//
                GEAR_MODIF
414             case GEAR_MODIF:
415                 if(isFirstDataProcessPass){
416 
417                     isFirstDataProcessPass = false;
418                     *pec12RotationValue = getGearReduction(pStepperData);
419                 }
420                 setGearReduction(pStepperData, pec12RotationValue);
421                 break;
422 
423             //-----------------------------------------------------------------//
                STEP_PER_TURN_MODIF
424             case STEP_PER_TURN_MODIF :
425                 if(isFirstDataProcessPass){
426 
427                     isFirstDataProcessPass = false;
428                     *pec12RotationValue = getAnglePerStep(pStepperData);
429                 }
430                 setAnglePerStep(pStepperData, pec12RotationValue);
431                 break;
432 
433             //-----------------------------------------------------------------//
                POWER_MODIF
434             case POWER_MODIF:
435                 if(isFirstDataProcessPass){
436 
437                     isFirstDataProcessPass = false;
438                     *pec12RotationValue = getStepperPower(pStepperData);
439                 }
440                 setStepperPower(pStepperData, (uint16_t*)pec12RotationValue); ///
                    ????_____Dwaf-ad-***
441                 break;
442 
443             //-----------------------------------------------------------------//
                BL_INTENSITY_MODIF
444             case BL_INTENSITY_MODIF :
445                 if(isFirstDataProcessPass){
446 
447                     isFirstDataProcessPass = false;
448                     *pec12RotationValue = getBlIntensity();
449                 }
450                 setBlIntensity(pec12RotationValue);
451                 break;
452 
453             //-----------------------------------------------------------------//
                LIGHT_INTENSITY_MODIF
454             case LIGHT_INTENSITY_MODIF:
455                 if(isFirstDataProcessPass){
456 
457                     isFirstDataProcessPass = false;
458                     *pec12RotationValue = getLightIntensity();
459                 }
460                 setLightIntensity(pec12RotationValue);
```

```
461                 break;
462
463             //-----------------------------------------------------------------//
                EXPOSURE_TIME_MODIF
464             case EXPOSURE_TIME_MODIF:
465                 if(isFirstDataProcessPass){
466
467                     isFirstDataProcessPass = false;
468                     *pec12RotationValue = getExposureTime();
469                 }
470                 setExposureTime(pec12RotationValue);
471                 break;
472
473             //-----------------------------------------------------------------//
                TIME_BW_PICTURES_MODIF
474             case TIME_BW_PICTURES_MODIF:
475                 if(isFirstDataProcessPass){
476
477                     isFirstDataProcessPass = false;
478                     *pec12RotationValue = getTimeBwPictures();
479                 }
480                 setTimeBwPictures(pec12RotationValue);
481                 break;
482
483             //-----------------------------------------------------------------//
                SAVE_DATA_START
484             case SAVE_DATA_START:
485                 if(isFirstDataProcessPass){
486
487                     isFirstDataProcessPass = false;
488 //                    isInModifMode = false; // AFFICHER .. ECRAN
489                     saveDataInEeprom(pStepperData);
490                     /* Once the data are saved, back to previous menu */
491                     isInModifMode = false;
492                     menu.menuState = SETTINGS_MENU;
493                     menu.menuSize = 5;
494                 }
495                 break;
496
497             //-----------------------------------------------------------------//
                AUTO_HOME_START
498             case AUTO_HOME_START:
499                 if(isFirstDataProcessPass){
500
501                     isFirstDataProcessPass = false;
502                     /* Start the auto home seq. */
503                     startAutoHome(pStepperData);
504                     /* Once auto home seq. is started, back to previous menu */
505                     isInModifMode = false;
506                     menu.menuState = MANUAL_MODE_MENU;
507                     menu.menuSize = 2;
508                 }
509                 break;
510
511             //-----------------------------------------------------------------//
                AUTOMATIC_MODE_START
512             case AUTOMATIC_MODE_START:
513                 if(isFirstDataProcessPass){
514
515                     isFirstDataProcessPass = false;
516                     /* Start the auto home seq. */
517                     startFullImagingSequence();
518                     /* Once auto home seq. is started, back to previous menu */
519                     isInModifMode = false;
520                     menu.menuState = AUTOMATIC_MODE_MENU;
521                     menu.menuSize = 1;
522                 }
523                 break;
524         }
```

```
    }
}



//_____________________________________________________________________________//
void menuPrintProcess(STEPPER_DATA *pStepperData){
    /* Print switch */
    switch(menu.menuState){

        case MAIN_MENU:
            printMainMenu();
            break;

        case SETTINGS_MENU:
            switch (menu.menuPage){
                case 0: printParameterMenuPage0();
                    break;
                case 1: printParameterMenuPage1();
                    break;
            }
            break;

        case MOTOR_MENU:
            switch (menu.menuPage){
                case 0: printMotorMenu0(pStepperData);
                    break;
                case 1: printMotorMenu1(pStepperData);
                    break;
            }
            break;

        case LIGHT_MENU:
            printLedsMenu();
            break;

        case BACKLIGHT_MENU:
            printBackLightMenu();
            break;

        case CAMERA_MENU:
            printCameraMenu();
            break;

        case SAVE_DATA_MENU:
            printSaveDataMenu();
            break;

        case CAPTURE_MODE_MENU:
            printChoiceSeqMenu();
            break;

        case MANUAL_MODE_MENU:
            printManualModeMenu(pStepperData);
            break;

        case AUTOMATIC_MODE_MENU:
            printAutoModeMenu(pStepperData);
            break;

        case ABOUT_MENU:
            printAboutMenu();
            break;

        case AUTO_HOME_MENU:
            printAutoHomeMenu();
            break;

```

```
            default:
                break;
        }
    }
    
    
    
    
    
    void printLcdInit(void){
    
        char str[2];
        ClrDisplay();
        DisplayOnOff(DISPLAY_ON); //Disable cursor
        SetPostion(LINE1);
        WriteString("Auto RTI Capt System");
        SetPostion(LINE2);
        WriteString("08-09 2023");
        SetPostion(LINE3);
        WriteString("Meven Ricchieri");
        SetPostion(LINE4);
    
        int i;
        for (i = 0; i < 20; i++){
    
            APP_Delay_ms(75);
            SetPostion(LINE4 + i);
            sprintf(str, "%c", 0xD0);
            WriteString(str);
        }
        APP_Delay_ms(150);
    }
    
    void printMainMenu(void){
    
        ClrDisplay();
        SetPostion(LINE1);
        WriteString("  Capture mode");
        SetPostion(LINE2);
        WriteString("  Settings");
        SetPostion(LINE3);
        WriteString("  About");
        SetPostion(LINE4);
        WriteString("  ");
    }
    
    void printParameterMenuPage0(void){
    
        ClrDisplay();
        SetPostion(LINE1);
        WriteString("  Return");
        SetPostion(LINE2);
        WriteString("  Motor");
        SetPostion(LINE3);
        WriteString("  Power light");
        SetPostion(LINE4);
        WriteString("  Back-light");
    }
    
    void printParameterMenuPage1(void){
    
        ClrDisplay();
        SetPostion(LINE1);
        WriteString("  Camera");
        SetPostion(LINE2);
        WriteString("  Save data");
    }
```

```
663  void printMotorMenu0(STEPPER_DATA *pStepperData){
664
665      char str[21];
666      ClrDisplay();
667      SetPostion(LINE1);
668      WriteString("  Return");
669      SetPostion(LINE2);
670      sprintf(str, "  Speed: %4dsteps/s", pStepperData->stepPerSec);
671      WriteString(str);
672      SetPostion(LINE3);
673      sprintf(str, "  Gear:        1:%3d", pStepperData->gearValue);
674      WriteString(str);
675      SetPostion(LINE4);
676      sprintf(str, "  Step angle: %1.2f%c", pStepperData->anglePerStep, 0x01);
677      WriteString(str);
678  }
679
680  void printMotorMenu1(STEPPER_DATA *pStepperData){
681
682      char str[21];
683      ClrDisplay();
684      SetPostion(LINE1);
685      /* A changer, en Duty pure, ensuite en %%% */
686      sprintf(str, "  Power : %03d", pStepperData->dutyCycleStepper);
687      WriteString(str);
688  }
689
690  void printLedsMenu(void){
691
692      char str[21];
693      ClrDisplay();
694      SetPostion(LINE1);
695      WriteString("  Return");
696      SetPostion(LINE2);
697      /* 0.04 = 100 / 2500 */
698      sprintf(str, "  Intensity : %03.0f%%", ((float)appData.lightIntensity * 0.04));
699      WriteString(str);
700  //    SetPostion(LINE3);
701  //    sprintf(str, "  Light time: %03dms", appData.lightTime);
702  //    WriteString(str);
703  }
704
705  void printChoiceSeqMenu(void){
706
707      ClrDisplay();
708      SetPostion(LINE1);
709      WriteString("  Return");
710      SetPostion(LINE2);
711      WriteString("  Manual mode");
712      SetPostion(LINE3);
713      WriteString("  Auto mode");
714      SetPostion(LINE4);
715      WriteString("  ");
716  }
717
718  void printAboutMenu(void){
719
720      ClrDisplay();
721      SetPostion(LINE1);
722      WriteString("  Return");
723      SetPostion(LINE2);
724      WriteString("  Version 1.0.0");
725      SetPostion(LINE3);
726      WriteString("  Meven Ricchieri");
727      SetPostion(LINE4);
728      WriteString("  08-09 2023");
729  }
730
731  void printManualModeMenu(STEPPER_DATA *pStepperData){
```

```
732
733         char str[21];
734         ClrDisplay();
735         SetPostion(LINE1);
736         WriteString("  Return");
737         SetPostion(LINE2);
738         if(pStepperData->isIndexed == true){
739             sprintf(str, "  Auto home    :%s", "DONE");
740         } else {
741             sprintf(str, "  Auto home     :%s", "NOK");
742         }
743         WriteString(str);
744         SetPostion(LINE3);
745         sprintf(str, "  Des. angle :%03.1f%c", (((float)pStepperData->stepToReach * 1.8)
746                 / pStepperData->gearValue), 0x01);
747     //    sprintf(str, "  Steps      : %05d", stepperData.stepToDoReach);
748         WriteString(str);
749         SetPostion(LINE4);
750         sprintf(str, "  Real angle :%03.1f%c", (((float)pStepperData->performedSteps * 1.8)
751                 / pStepperData->gearValue), 0x01);
752     //    sprintf(str, "  Steps       :%05d", pStepperData->performedStep);
753         WriteString(str);
754     }
755
756     void printAutoModeMenu(STEPPER_DATA *pStepperData){
757
758         char str[21];
759         ClrDisplay();
760         SetPostion(LINE1);
761         WriteString("  Return");
762         SetPostion(LINE2);
763         if(appData.isFullImaginSeqEnable == false) {
764             sprintf(str, "  Start sequence");
765         } else {
766             sprintf(str, "  Sequence is ON");
767         }
768         WriteString(str);
769         SetPostion(LINE3);
770         sprintf(str, "  Pictures:      %03d", appData.nbrOfShotsPerformed);
771         WriteString(str);
772         SetPostion(LINE4);
773         sprintf(str, "  Real angle :%03.1f%c", (((float)pStepperData->performedSteps * 1.8)
774                 / pStepperData->gearValue), 0x01);
775         WriteString(str);
776     }
777
778     void printAutoHomeMenu(void){
779
780         ClrDisplay();
781         SetPostion(LINE1);
782         WriteString("  Return");
783         SetPostion(LINE2);
784         WriteString("  Press to index");
785         SetPostion(LINE3);
786         WriteString("  ");
787         SetPostion(LINE4);
788         WriteString("  ");
789     }
790
791     void printBackLightMenu(void){
792
793         char str[21];
794         ClrDisplay();
795         SetPostion(LINE1);
796         WriteString("  Return");
797         SetPostion(LINE2);
798         /* 0.04 = 100 / 2500 */
799         sprintf(str, "  Intensity : %03.0f%%", ((float)appData.backLightIntensitiy * 0.04));
800         WriteString(str);
```

```
801 }
802
803 void printCameraMenu(void){
804
805     char str[21];
806     ClrDisplay();
807     SetPostion(LINE1);
808     WriteString("  Return");
809     SetPostion(LINE2);
810     sprintf(str, "  Expos time: %04dms", appData.exposureDuration);
811     WriteString(str);
812     SetPostion(LINE3);
813     sprintf(str, "  Time bw pic:%04dms", appData.timeBetweenPictures);
814     WriteString(str);
815     SetPostion(LINE4);
816     WriteString("  Trigger : cable"); // <-- or IR but not ready
817 }
818
819 void printSaveDataMenu(){
820
821     ClrDisplay();
822     SetPostion(LINE1);
823     WriteString("  Return");
824     SetPostion(LINE2);
825     WriteString("  Confirm to save");
826     SetPostion(LINE3);
827     WriteString("  ! Old values will ");
828     SetPostion(LINE4);
829     WriteString("  be overwritten ! ");
830 }
831
832
833
834
835 /* Clear the first row all 4 lines */
836 void clearFirstRow(void){
837
838     SetPostion(LINE1);
839     WriteString(" ");
840     SetPostion(LINE2);
841     WriteString(" ");
842     SetPostion(LINE3);
843     WriteString(" ");
844     SetPostion(LINE4);
845     WriteString(" ");
846 }
847
848 /* Print cursor */
849 void printCursor(int32_t cursor){
850
851     char str[2];
852     clearFirstRow();
853     SetPostion(cursor * 0x20);
854     sprintf(str, "%c", RIGHT_ARROW);
855     WriteString(str);
856 }
857
858
859
860 //-------------------------------------------------------------------------//
    saveDataInEeprom
861 bool saveDataInEeprom(STEPPER_DATA *pStepperData){
862
863     DATA_IN_EEPROM dataToSaveInEeprom;
864
865     /* Set the structure value for saving in EEPROM */
866     dataToSaveInEeprom.stepPerSec   = pStepperData->stepPerSec;
867     dataToSaveInEeprom.stepPerTurn  = pStepperData->stepPerTurn;
868     dataToSaveInEeprom.gearValue    = pStepperData->gearValue;
```

```
869       dataToSaveInEeprom.anglePerStep = pStepperData->anglePerStep;
870
871       dataToSaveInEeprom.lightIntensity       = appData.lightIntensity;
872       dataToSaveInEeprom.timeBetweenPictures  = appData.timeBetweenPictures;
873       dataToSaveInEeprom.exposureDuration     = appData.exposureDuration;
874
875       dataToSaveInEeprom.backLightIntensitiy  = appData.backLightIntensitiy;
876
877       dataToSaveInEeprom.controlValue = CONTROL_VALUE;
878
879       Init_DataBuff();
880       /* Write in the EEPROM */
881       NVM_WriteBlock((uint32_t*)&dataToSaveInEeprom, sizeof(dataToSaveInEeprom));
882
883       return 0;
884   }
885
886   //-------------------------------------------------------------------------//
      readDataFromEeprom
887   /* Read the parameters from the EEPROM */
888   bool readDataFromEeprom(STEPPER_DATA *pStepperData){
889
890       DATA_IN_EEPROM dataReadFromEeprom;
891
892       Init_DataBuff();
893       /* Read in the EEPROM */
894       NVM_ReadBlock((uint32_t*)&dataReadFromEeprom, sizeof(dataReadFromEeprom));
895
896       /* Check if the control value is already inside the EEPROM */
897       if(dataReadFromEeprom.controlValue == CONTROL_VALUE){
898
899           /* Save data from EEPROM */
900           pStepperData->stepPerSec    = dataReadFromEeprom.stepPerSec;
901           pStepperData->stepPerTurn   = dataReadFromEeprom.stepPerTurn;
902           pStepperData->gearValue     = dataReadFromEeprom.gearValue;
903           pStepperData->anglePerStep  = dataReadFromEeprom.anglePerStep;
904
905           appData.lightIntensity      = dataReadFromEeprom.lightIntensity;
906           appData.timeBetweenPictures = dataReadFromEeprom.timeBetweenPictures;
907           appData.exposureDuration    = dataReadFromEeprom.exposureDuration;
908
909           appData.backLightIntensitiy = dataReadFromEeprom.backLightIntensitiy;
910
911       } else {
912
913           // SAVE INIT VAL
914           saveDataInEeprom(pStepperData);
915       }
916   }
```

```c
/*
 * File:   menu.h
 * Author: ricch
 *
 * Created on August 30, 2023, 10:17 AM
 */

#ifndef MENU_H
#define MENU_H

#ifdef __cplusplus
extern "C" {
#endif

    #include <stdbool.h>
    #include <stdint.h>
    #include <stepperDriver.h>
    #include "Mc32NVMUtil.h"

    #define RIGHT_ARROW 0x10

    // Enumerations

    /* All menus */
    typedef enum
    {
        MAIN_MENU = 0,
        CAPTURE_MODE_MENU,
        SETTINGS_MENU,
        ABOUT_MENU,
        MOTOR_MENU,
        MANUAL_MODE_MENU,
        LIGHT_MENU,
        BACKLIGHT_MENU,
        CAMERA_MENU,
        SAVE_DATA_MENU,
        AUTO_HOME_MENU,
        AUTOMATIC_MODE_MENU,

    } MENU_STATE;



    typedef enum{

        RETURN_SEL = 0,

    } COMMON;

    typedef enum{

        CAPTURE_MODE_SEL = 0,
        SETTINGS_SEL,
        ABOUT_SEL,

    } MAIN_MENU_LIST;

    typedef enum{

        LIGHT_INTENSITY_SEL = 1,
        LIGHT_TIME_SEL,
        TIME_BW_PICTURES,

    }LEDS_MENU_LIST;

    typedef enum{

        AUTO_HOME_START_SEL = 1,
```

```
70
71      } AUTO_HOME_MENU_LIST;
72
73      typedef enum{
74
75          MANUAL_MODE_SEL = 1,
76          AUTOMATIC_MODE_SEL,
77
78      } CHOICE_SEQ_MENU_LIST;
79
80      typedef enum{
81
82          AUTO_HOME_SEL = 1,
83          ANGLE_SEL,
84
85      } MANUAL_MODE_MENU_LIST;
86
87      typedef enum{
88
89          AUTOMATIC_MODE_START_SEL = 1,
90
91      } AUTO_MODE_MENU_LIST;
92
93      typedef enum{
94
95          MOTOR_SEL = 1,
96          LEDS_SEL,
97          BACKLIGHT_SEL,
98          CAMERA_SEL,
99          SAVE_DATA_SEL,
100
101     } SETTINGS_MENU_LIST;
102
103     typedef enum{
104
105         SPEED_SEL = 1,
106         GEAR_SEL,
107         STEP_PER_TURN_SEL,
108         POWER_SEL,
109
110     } MOTOR_MENU_LIST;
111
112     typedef enum{
113
114         BACKLIGHT_INTENSITY_SEL = 1,
115
116     } BACKLIGHT_MENU_LIST;
117
118     typedef enum{
119
120         EXPOSURE_TIME_SEL = 1,
121         TIME_BW_PICTURES_SEL,
122
123     } CAMERA_MENU_LIST;
124
125
126
127     typedef enum{
128
129         ANGLE_MODIF = 0,
130         SPEED_MODIF,
131         GEAR_MODIF,
132         STEP_PER_TURN_MODIF,
133         POWER_MODIF,
134         BL_INTENSITY_MODIF,
135         LIGHT_INTENSITY_MODIF,
136         LIGHT_TIME_MODIF,
137         EXPOSURE_TIME_MODIF,
138         TIME_BW_PICTURES_MODIF,
```

```
139
140             SAVE_DATA_START,
141             AUTO_HOME_START,// INTERACT
142             AUTOMATIC_MODE_START,
143
144         } MODIF_LIST;
145
146
147
148         // Structures
149         typedef struct{
150
151             uint8_t menuPage;
152             uint8_t menuSize;
153             MENU_STATE menuState;
154             MODIF_LIST modifState;
155
156         } MENU;
157
158
159
160
161
162         // Prototypes
163         void printLcdInit(void);
164         void printMainMenu(void);
165         void printParameterMenuPage0(void);
166         void printParameterMenuPage1();
167         void printMotorMenu0(STEPPER_DATA *pStepperData);
168         void printMotorMenu1(STEPPER_DATA *pStepperData);
169         void printLedsMenu(void);
170         void printChoiceSeqMenu(void);
171         void printAboutMenu(void);
172         void printManualModeMenu(STEPPER_DATA *pStepperData);
173         void printAutoModeMenu(STEPPER_DATA *pStepperData);
174         void printAutoHomeMenu(void);
175         void printBackLightMenu(void);
176         void printCameraMenu(void);
177         void printSaveDataMenu(void);
178
179         void menuManagementProcess(void);
180         void menuActionProcess(int32_t pec12RotationValue);
181         void menuDataProcess(int32_t *pec12RotationValue, STEPPER_DATA *pStepperData);
182         void menuPrintProcess(STEPPER_DATA *pStepperData);
183
184         void clearFirstRow(void);
185         void printCursor(int32_t cursor);
186
187         bool saveDataInEeprom(STEPPER_DATA *pStepperData);
188         bool readDataFromEeprom(STEPPER_DATA *pStepperData);
189
190
191     #ifdef  __cplusplus
192     }
193     #endif
194
195     #endif  /* MENU_H */
196
197
```

```c
/*
 * File:   lights.c
 * Author: ricch
 *
 * Created on September 5, 2023, 7:15 PM
 */

#include "app.h"

extern APP_DATA appData;

//------------------------------------------------------------------------------//
lightManagementProcess
void sequenceManagementProcess(void){

    static int32_t order = 5; //= angleDesired / gear;

    if(appData.isFiveShotsSeqEnable){

        /* Sequence of 5 pictures is enable */
        //fiveShotsSeqProcess();
        //startFiveShotsSeqProcess();
    }
    if(appData.isFullImaginSeqEnable){

        /* Full sequence is enable */
        switch (appData.valSeq){

            case 0:
//               appData.valSeq += fiveShotsSeqProcess();
                break;
            case 1:
                setRotationToDo(getMyStepperStruct(), &order);
                if(getPerformedSteps(getMyStepperStruct()) == order){
                    order += 5; // appData.angleBwEachSeq;
                    appData.valSeq = 0;
                    appData.seqClock1_ms = 0;
                    appData.seqClock2_ms = 0;
                    startFiveShotsSeqProcess();
                }
                break;
        }
    }
}


//------------------------------------------------------------------------------//
turnOffAllPwrLeds
void turnOffAllPwrLeds(void){

    /* Turn off all power LED */
    LED1_CMDOff();
    LED2_CMDOff();
    LED3_CMDOff();
    LED4_CMDOff();
    LED5_CMDOff();
}

//------------------------------------------------------------------------------//
startFiveShotsSequence
/* Start a sequence for 5 shots */
void startFiveShotsSequence(void){

    appData.seqClock1_ms = 0;
    appData.seqClock2_ms = 0;
    appData.isFiveShotsSeqEnable = true;
}

//------------------------------------------------------------------------------//
```

```
    startFullImagingSequence
67  void startFullImagingSequence(void){
68
69      appData.seqClock1_ms = 0;
70      appData.seqClock2_ms = 0;
71      appData.isFullImaginSeqEnable = true;
72      appData.valSeq = 0;
73      appData.nbrOfShotsPerformed = 0;
74      startFiveShotsSeqProcess();
75  }
76
77  //------------------------------------------------------------------------//
    simpleShotProcess
78  void startSimpleShotProcess(void){
79
80      appData.seqClock2_ms = 0;
81      DRV_TMR4_Start();
82  }
83
84  void startFiveShotsSeqProcess(void){
85
86      appData.seqClock1_ms = 0;
87      DRV_TMR0_Start();
88  }
89
90  //------------------------------------------------------------------------//
    imagingSeqProcess
91  /* This function takes 5 pictures with 5 different LEDs */
92  bool fiveShotsSeqProcess(void){
93
94  //    if(appData.seqClock1_ms == 0){
95  //        appData.ledId = PWR_LED1;
96  //        startSimpleShotProcess();
97  //
98  //    } else if(appData.seqClock1_ms == 1 * appData.timeBetweenPictures){
99  //        appData.ledId = PWR_LED2;
100 //        startSimpleShotProcess();
101 //
102 //    } else if(appData.seqClock1_ms == 2 * appData.timeBetweenPictures){
103 //        appData.ledId = PWR_LED3;
104 //        startSimpleShotProcess();
105 //
106 //    } else if(appData.seqClock1_ms == 3 * appData.timeBetweenPictures){
107 //        appData.ledId = PWR_LED4;
108 //        startSimpleShotProcess();
109 //
110 //    } else if(appData.seqClock1_ms == 4 * appData.timeBetweenPictures){
111 //        appData.ledId = PWR_LED5;
112 //        startSimpleShotProcess();
113 //    }
114 //    if(appData.seqClock1_ms >= 5 * appData.timeBetweenPictures){
115 //
116 //        appData.seqClock1_ms = 0;
117 //        appData.seqClock2_ms = 0;
118 //        appData.isFiveShotsSeqEnable = false;
119 //        return 1;
120 //    }
121 //    return 0;
122 }
123
124
125 //------------------------------------------------------------------------//
    setLighIntensity
126 void setLightIntensity(int32_t *lightIntensity){
127
128     // Limit values to avoid problems
129     if(*lightIntensity < LIGHT_INTENSITY_MIN) *lightIntensity
130             = LIGHT_INTENSITY_MIN;
131     if(*lightIntensity > LIGHT_INTENSITY_MAX) *lightIntensity
```

```
132             = LIGHT_INTENSITY_MAX;
133
134     /* 25 = 2500 / 100 */
135     appData.lightIntensity = *lightIntensity * 25;
136     PLIB_MCPWM_ChannelPrimaryDutyCycleSet(MCPWM_ID_0, PWM_DIM_CH, appData.lightIntensity
        );
137 }
138 int32_t getLightIntensity(void){
139
140     return appData.lightIntensity / 25;
141 }
142
143 //------------------------------------------------------------------------------//
    setTimeBwPictures
144 void setTimeBwPictures(int32_t *timeBwPictures){
145
146     int32_t time_bw_pictures_min = appData.exposureDuration +
147             3 * MARGIN_LED_DELAY;
148     // Limit values to avoid problems
149     if(*timeBwPictures < time_bw_pictures_min) *timeBwPictures
150             = time_bw_pictures_min;
151     if(*timeBwPictures > TIME_BW_PICTURES_MAX) *timeBwPictures
152             = TIME_BW_PICTURES_MAX;
153
154     appData.timeBetweenPictures = *timeBwPictures;
155 }
156 int32_t getTimeBwPictures(void){
157
158     return appData.timeBetweenPictures;
159 }
160
161 //------------------------------------------------------------------------------//
    setExposureTime
162 void setExposureTime(int32_t *exposureTime){
163
164     // Limit values to avoid problems
165     if(*exposureTime < EXPOSURE_TIME_MIN) *exposureTime = EXPOSURE_TIME_MIN;
166     if(*exposureTime > EXPOSURE_TIME_MAX) *exposureTime = EXPOSURE_TIME_MAX;
167
168     appData.exposureDuration = *exposureTime;
169 }
170 int32_t getExposureTime(void){
171
172     return appData.exposureDuration;
173 }
```

```
/*
 * File:   lights.h
 * Author: ricch
 *
 * Created on September 5, 2023, 7:16 PM
 */
```

```c
/*
 * File:   pec12.c
 * Author: ricch
 *
 * Created on August 30, 2023, 9:15 AM
 */

#include "app.h"
#include "pec12.h"
#include "lcd_spi.h"

PEC12 pec12;

void scanPec12(void){

    // Save old states for debounce
    pec12.chA.state[3] = pec12.chA.state[2];
    pec12.chA.state[2] = pec12.chA.state[1];
    pec12.chA.state[1] = pec12.chA.state[0];
    pec12.chA.state[0] = CHANNEL_AStateGet();

    pec12.chB.state[1] = pec12.chB.state[0];
    pec12.chB.state[0] = CHANNEL_BStateGet();

    pec12.chC.state[3] = pec12.chC.state[2];
    pec12.chC.state[2] = pec12.chC.state[1];
    pec12.chC.state[1] = pec12.chC.state[0];
    pec12.chC.state[0] = PEC12R_SWStateGet();

    // Check if PEC12 is in rotation
    if(pec12.chA.state[0] == 0 && pec12.chA.state[1] == 0
            && pec12.chA.state[2] == 1 && pec12.chA.state[3] == 1){

        // Check direction of rotation
        if(pec12.chB.state[0] == 1 && pec12.chB.state[1] == 1){

            // CW
            pec12.incrOrDecr++;
//            SetPostion(LINE3);
//            sprintf(a_toPrint, "counter = %d",counter);
//            WriteString(a_toPrint);
        }
        else{

            //CCW
            pec12.incrOrDecr--;
//            SetPostion(LINE3);
//            sprintf(a_toPrint, "counter = %d",counter);
//            WriteString(a_toPrint);
        }
    }
    // Check if PEC12 switch is pressed
    if(pec12.chC.state[0] == 0 && pec12.chC.state[1] == 0
            && pec12.chC.state[2] == 1 && pec12.chC.state[3] == 1){

        pec12.isPressed = true;
    }
}

int8_t getPec12IncrOrDecr(void){

    int8_t incrOrDecr = pec12.incrOrDecr;
    pec12.incrOrDecr = 0;

    return incrOrDecr;
}

int8_t getPec12SwitchEvent(void){

```

```
70      int8_t isPressed = pec12.isPressed;
71      pec12.isPressed = 0;
72
73      return isPressed;
74  }
```

```c
/*
 * File:   pec12.h
 * Author: ricch
 *
 * Created on August 30, 2023, 9:15 AM
 */

#ifndef PEC12_H
#define PEC12_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <stdbool.h>

typedef struct
{
    bool state[4];

} CHANNEL;

typedef struct
{
    CHANNEL chA;
    CHANNEL chB;
    CHANNEL chC;
    int8_t incrOrDecr;
    bool isPressed;

} PEC12;




void scanPec12(void);
int8_t getPec12IncrOrDecr(void);



#ifdef  __cplusplus
}
#endif

#endif  /* PEC12_H */

```

```
/*******************************************************************************
 System Interrupts File

  File Name:
    system_interrupt.c

  Summary:
    Raw ISR definitions.

  Description:
    This file contains a definitions of the raw ISRs required to support the
    interrupt sub-system.

  Summary:
    This file contains source code for the interrupt vector functions in the
    system.

  Description:
    This file contains source code for the interrupt vector functions in the
    system.  It implements the system and part specific vector "stub" functions
    from which the individual "Tasks" functions are called for any modules
    executing interrupt-driven in the MPLAB Harmony system.

  Remarks:
    This file requires access to the systemObjects global data structure that
    contains the object handles to all MPLAB Harmony module objects executing
    interrupt-driven in the system.  These handles are passed into the individual
    module "Tasks" functions to identify the instance of the module to maintain.
 *******************************************************************************/

// DOM-IGNORE-BEGIN
/*******************************************************************************
Copyright (c) 2011-2014 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 "system/common/sys_common.h"
#include "app.h"
#include "system_definitions.h"
#include "lcd_spi.h"
#include "pec12.h"

// *****************************************************************************
// *****************************************************************************
```

```
70  // Section: System Interrupt Vector Functions
71  // *****************************************************************************
72  // *****************************************************************************
73
74  extern APP_DATA appData;
75  extern STEPPER_DATA stepperData;
76
77  //-------------------------------------------------------------------------// TMR ID 1
78  /* This timer clocks the capture sequence */
79  /* Frequency = 1000Hz */
80  void __ISR(_TIMER_1_VECTOR, ipl1AUTO) IntHandlerDrvTmrInstance0(void){
81
82      PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_1);
83
84      /* Control sequence of the 5 LEDs */
85      if(appData.seqClock1_ms == 0){
86          appData.ledId = PWR_LED1;
87          startSimpleShotProcess();
88
89      } else if(appData.seqClock1_ms == 1 * appData.timeBetweenPictures){
90          appData.ledId = PWR_LED2;
91          startSimpleShotProcess();
92
93      } else if(appData.seqClock1_ms == 2 * appData.timeBetweenPictures){
94          appData.ledId = PWR_LED3;
95          startSimpleShotProcess();
96
97      } else if(appData.seqClock1_ms == 3 * appData.timeBetweenPictures){
98          appData.ledId = PWR_LED4;
99          startSimpleShotProcess();
100
101     } else if(appData.seqClock1_ms == 4 * appData.timeBetweenPictures){
102         appData.ledId = PWR_LED5;
103         startSimpleShotProcess();
104     }
105     if(appData.seqClock1_ms >= 5 * appData.timeBetweenPictures){
106
107         DRV_TMR0_Stop();
108         appData.seqClock1_ms = 0;
109         appData.isFiveShotsSeqEnable = false;
110         appData.valSeq = 1;
111         //return 1;
112     } else {
113     appData.seqClock1_ms++;
114     }
115 }
116
117 //-------------------------------------------------------------------------// TMR ID 2
118 /* This timer clocks the main state machine */
119 /* Frequency = 10000Hz */
120 void __ISR(_TIMER_2_VECTOR, ipl1AUTO) IntHandlerDrvTmrInstance1(void){
121
122     PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_2);
123
124     /* States machines update */
125     APP_UpdateAppState(APP_STATE_SERVICE_TASKS);
126 }
127
128 //-------------------------------------------------------------------------// TMR ID 3
129 /* This timer clocks the stepper sequence */
130 /* Variable frequency */
131 void __ISR(_TIMER_3_VECTOR, ipl4AUTO) IntHandlerDrvTmrInstance2(void){
132
133 //    SIGN_LED_CMDOff();
134     PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_3);
135
136     changeSpeed(getMyStepperStruct());
137     processStepper(getMyStepperStruct());
138 //    SIGN_LED_CMDOn();
```

```
139 }
140
141 //------------------------------------------------------------------------// TMR ID 4
142 /* This timer is used for the APP_Delay_ms() function */
143 /* Frequency = 1000Hz */
144 void __ISR(_TIMER_4_VECTOR, ipl1AUTO) IntHandlerDrvTmrInstance3(void){
145
146     PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_4);
147
148     /* Timer for ms delay */
149     appData.msCounter++;
150 }
151
152 //------------------------------------------------------------------------// TMR ID 5
153 /* Frequency = 1000Hz */
154 void __ISR(_TIMER_5_VECTOR, ipl3AUTO) IntHandlerDrvTmrInstance4(void)
155 {
156     PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_5);
157
158     //------------------------------------------------------------------------// Start
        of sequence
159     if(appData.seqClock2_ms == 0){
160
161         switch (appData.ledId){
162             /* Turn on LED */
163             case PWR_LED1:
164                 turnOffAllPwrLeds();
165                 LED1_CMDOn();
166                 break;
167
168             case PWR_LED2:
169                 turnOffAllPwrLeds();
170                 LED2_CMDOn();
171                 break;
172
173             case PWR_LED3:
174                 turnOffAllPwrLeds();
175                 LED3_CMDOn();
176                 break;
177
178             case PWR_LED4:
179                 turnOffAllPwrLeds();
180                 LED4_CMDOn();
181                 break;
182
183             case PWR_LED5:
184                 turnOffAllPwrLeds();
185                 LED5_CMDOn();
186                 break;
187         }
188     }
189     if(appData.seqClock2_ms == MARGIN_LED_DELAY){
190
191         /* Capture the target */
192         FOCUS_CMDOn();
193         TRIGGER_CMDOn();
194         appData.nbrOfShotsPerformed++;
195 //        SIGN_LED_CMDOn();
196     }
197     if(appData.seqClock2_ms == appData.exposureDuration + MARGIN_LED_DELAY){
198
199         TRIGGER_CMDOff();
200         FOCUS_CMDOff();
201     }
202     //------------------------------------------------------------------------// End of
        sequence
203     if(appData.seqClock2_ms >= appData.exposureDuration + (2 * MARGIN_LED_DELAY)){
204
205         /* Turn off TMR4 */
```

```
206             DRV_TMR4_Stop();
207             turnOffAllPwrLeds();
208             appData.seqClock2_ms = 0;
209             appData.ledId = ALL_LED_DISABLE;
210         } else {
211             appData.seqClock2_ms++;
212         }
213     }
214     /*******************************************************************************
215      End of File
216     */
217
```