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


// Author M.Ricchieri


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

#include "app.h"
#include "Mc32_I2cUtilCCS.h"

#include "HSCMRRN001PD2A3_driver.h"

#include "RN4678_driver.h"
#include "voltageADC_driver.h"



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

```

```c
//---------------------------------------------------------------------------// Global data
APP_DATA        appData;
SENS_DATA       sensData;

struct inv_imu_device   myImuDevice;
struct inv_imu_serif    myImuSertif;



// *****************************************************************************
// *****************************************************************************
// Section: Application Callback Functions
// *****************************************************************************
// *****************************************************************************

////-------------------------------------------------------------------------// TIMER0 callback function <--- Disabled
//void TIMER0_Callback_Function(){ //156Hz
//
//    // xxxx
//}


//---------------------------------------------------------------------------// TIMER1 callback function
void TIMER1_Callback_Function(){ // 20Hz

//    if(appData.isBluetoothDiscoverable){
//
//        if(inv_imu_get_time_us() >= 120000000){ // 120 000 000 = 120 seconds
//
//            turnOffDiscoverBT();
//            appData.isBluetoothDiscoverable = false;
//        }
//    }
    // Update the main state machine
    APP_UpdateAppState(APP_STATE_SERVICE);
}


//---------------------------------------------------------------------------// USART1_Callback_Function
void USART1_Callback_Function(void){

    char a_received[50];
    char* result;

    // Gets new data from FIFO
    getUsartData(&a_received[0]);

    // If "<RFCOMM_OPEN>" is present in the array received
    result = strstr(a_received, "<RFCOMM_OPEN>");
    if(result != NULL){

        appData.isBluetoothDiscoverable = false;
        appData.isBluetoothConnected = true;
    }

    // If "<RFCOMM_CLOSE>" is present in the array received
    result = strstr(a_received, "<RFCOMM_CLOSE>");
    if(result != NULL){

        // turnOnDiscoverBT();
        appData.isBluetoothConnected = false;
    }

//    // If "<xxxxxx>" is present in the array received
//    result = strstr(a_received, "<xxxxxx>");
//    if(result != NULL){
//
//        // Does something
//    }
}


//---------------------------------------------------------------------------// IMU callback function
void imu_callback(inv_imu_sensor_event_t *event){

    // Transforms 16bits values into degrees and saves them in the sensor data
    // structure
    // 250 dps
    sensData.gyroX = (event->gyro[0])/250 + 1; // + offset
    sensData.gyroY = (event->gyro[1])/250 + 1; // + offset
    sensData.gyroZ = (event->gyro[2])/250;

    // Reads and transforms 16bits values into
    // Transforms 16bits values into g acceleration and saves them in the sensor
    // data structure (8192 bits per g)
    sensData.accelX = (float)(event->accel[0])/8192.0;
    sensData.accelY = (float)(event->accel[1])/8192.0;
    sensData.accelZ = (float)(event->accel[2])/8192.0 + 0.075; // + offset

    // Calculate the angle with the gyro values
    // 0.05 correspond to the period between each reading 1/20Hz = 0.05s
    sensData.GyrAngleX += sensData.gyroX * 0.05;
    sensData.GyrAngleY += sensData.gyroY * 0.05;
    sensData.GyrAngleZ += sensData.gyroZ * 0.05;
}


// *****************************************************************************
// *****************************************************************************
// Section: Application Local Functions
// *****************************************************************************
// *****************************************************************************

//----------------------------------------------------------------------------// APP_UpdateAppState
void APP_UpdateAppState(APP_STATES newState){

    appData.appState = newState;
}


//----------------------------------------------------------------------------// APP_UpdateServiceState
void APP_UpdateServiceState(SERVICE_STATES newState){

    appData.serviceState = newState;
}


//----------------------------------------------------------------------------// clearArray
void clearArray(size_t arraySize, char *pArrayToClear){

    int i;

    for (i = 0; i < arraySize; i++){

        pArrayToClear[i] = NULL;
    }
}


//----------------------------------------------------------------------------// frameFormatting
inline void frameFormatting(char* a_dataToSend, const SENS_DATA* sensData){

    // Saves all data into a simple frame
    // Speed in [km/h]
    // Gyros in [dps]
    // Angles in [degrees]
    // Accelerations in [g]
    // VB and VG in [V]
    sprintf(a_dataToSend, "S=%03d GX=%+.02f GY=%+.02f GZ=%+.02f GAX=%+.02f "
            "GAY=%+.02f GAZ=%+.02f AX=%+.02f AY=%+.02f AZ=%+.02f VB=%.02f "
            "VG=%.02f\n\r",
            sensData->velocity,
            sensData->gyroX, sensData->gyroY, sensData->gyroZ,
            sensData->GyrAngleX, sensData->GyrAngleY, sensData->GyrAngleZ,
            sensData->accelX, sensData->accelY, sensData->accelZ,
            sensData->batVoltage, sensData->genVoltage);
}

```

```
// *****************************************************************************
// *****************************************************************************
// Section: Application Initialization and State Machine Functions
// *****************************************************************************
// *****************************************************************************

//----------------------------------------------------------------------------// APP_Initialize
void APP_Initialize(void){

    // Initializes the appData structure

    appData.appState                = APP_STATE_INIT;
    appData.isBluethoothModuleInit  = false;
    appData.isBluetoothConnected    = false;
    appData.isBluetoothDiscoverable = false;
    sensData.GyrAngleX = 0;
    sensData.GyrAngleY = 0;
    sensData.GyrAngleZ = 0;

}


//----------------------------------------------------------------------------// initImuInterface
// Initialize serial interface between MCU and IMU
int initImuInterface(struct inv_imu_serif *icm_serif){

    // No need
icm_serif->context    = 0;
    // Points to the reading function dedicated
icm_serif->read_reg   = ICM42670P_I2C_bus_read;
    // Points to the writing function dedicated
icm_serif->write_reg  = ICM42670P_I2C_bus_write;
icm_serif->max_read   = 255; /* maximum number of bytes allowed per serial read */
icm_serif->max_write  = 255; /* maximum number of bytes allowed per serial write */
    // Set the communication interface
icm_serif->serif_type = SERIF_TYPE;

return 0;
}


//----------------------------------------------------------------------------// APP_Tasks
void APP_Tasks(void){

    RAW_ADC rawAdc;

    // Check the application's current state
    switch(appData.appState){

        // Application's initial state
        case APP_STATE_INIT:
        {
            int rc = 0;

            // Initialization of the I2C communication
            i2c_init(SLOW);

            do{
                // Initialization of the ICM42670 interface
                rc |= initImuInterface(&myImuSertif);
                // Resets and prepares the chip for the configuration
                rc |= setupImuDevice(&myImuSertif);
                // Configures ICM42670 parameters
                rc |= configureImuDevice();

            }while(rc != INV_ERROR_SUCCESS);

            // Initialization of the USART FIFOs
            initFifo(&usartFifoRx, FIFO_RX_SIZE, a_fifoRx, 0);
            initFifo(&usartFifoTx, FIFO_TX_SIZE, a_fifoTx, 0);

            do{
                // Initialization of the Bluetooth module
                appData.isBluethoothModuleInit = init_RN4678();
```

```c
            }while(appData.isBluethoothModuleInit == false);

            // Initialization of the ADC module
            initAdc();

            // Starts TIMERs
//            DRV_TMR0_Start(); <--- Disabled
            DRV_TMR1_Start();
            DRV_TMR2_Start();

            // States machines update
            APP_UpdateAppState(APP_STATE_WAIT);
            APP_UpdateServiceState(SERVICE_STATE_READ_SENSORS);
            break;
        }

        case APP_STATE_SERVICE:
        {
            int8_t a_frameToSend[130];

            switch(appData.serviceState){

                case SERVICE_STATE_READ_SENSORS:

                    // Reads voltages values
                    readRawAdc(&rawAdc);
                    convertRawToVoltage(&rawAdc, &sensData);
                    // Reads velocity value
                    convertRawToVelocity(readRawDiffPress(), &sensData);
                    // Gets new IMU data
                    get_imu_data();


                    APP_UpdateServiceState(SERVICE_STATE_PROCESS);
                    break;


                case SERVICE_STATE_PROCESS:

                    // Clears the array before saving new values
                    clearArray(sizeof(a_frameToSend), (char*)&a_frameToSend[0]);
                    // Converts float values in a char array (frame)
                    frameFormatting((char*)&a_frameToSend[0], &sensData);

                    APP_UpdateServiceState(SERVICE_STATE_SEND_DATA_BT);
                    break;

                case SERVICE_STATE_SEND_DATA_BT:

                    // Bluetooth is connected to a device
                    if(appData.isBluetoothConnected == true){

                        // Sends frame through USART
                        sendData_RN4678(&a_frameToSend[0]);
                        // Toggle signalisation LED
                        SIGN_LEDToggle();
                    }
                    else SIGN_LEDOn();

                    APP_UpdateAppState(APP_STATE_WAIT);
                    APP_UpdateServiceState(SERVICE_STATE_READ_SENSORS);
                    break;
            }
            break;
        }

        case APP_STATE_WAIT:
        {
            // Does nothing here
            break;
        }
        default:
        {
            // Does nothing here
            break;
        }
    }
}


/*******************************************************************************
 End of File
 */
```