```
 1
 2 /*
 3  * File:   adc_driver.c
 4  * Author: M.Ricchieri
 5  *
 6  * Created on 31. mai 2023, 08:55
 7  *
 8  * Inspired by the "Mc32DriverAdc.c" file
 9  */
10
11 //-----------------------------------------------------------------------------// Includes
12 #include "app.h"
13 #include "I2C_ICM42670P_Functions.h"
14 #include "imu/inv_imu_driver.h"
15 #include "imu/inv_imu_transport.h"
16 #include "Invn/EmbUtils/RingBuffer.h"
17
18
19 //-----------------------------------------------------------------------------// Constants
20 #if !USE_FIFO
21 /*
22  * Buffer to keep track of the timestamp when IMU data ready interrupt fires.
23  * The buffer can contain up to 64 items in order to store one timestamp
24      * for each packet in FIFO.
25  */
26 RINGBUFFER(timestamp_buffer, 64, uint64_t);
27 #endif
28
29
30 //-----------------------------------------------------------------------------// get_imu_data
31  /** @brief this function provides a way to retrieve IMU data either from a FIFO
32  *          buffer if USE_FIFO is defined, or directly from the registers if
33  *          USE_FIFO is not defined.
34  *
35  *   @param[in]     -               -
36  *   @return        data from "inv_imu_get_data_from_registers" function or
37  *                  data from "inv_imu_get_data_from_fifo"
38  */
39 int get_imu_data(void)
40 {
41 #if USE_FIFO
42 return inv_imu_get_data_from_fifo(&myImuDevice);
43 #else
44 return inv_imu_get_data_from_registers(&myImuDevice);
45 #endif
46 }
47
48

49 //-----------------------------------------------------------------------------// configureImuDevice
50  /** @brief This function configures the device in order to output gyro and
51  *          accelerometer.
52  *
53  *   @return        rc              InvError structure parameter, 0 on success
54  */
55 int configureImuDevice(void)
56 {
57 int rc = 0;
58
59 if (!USE_FIFO)
60 rc |= inv_imu_configure_fifo(&myImuDevice, INV_IMU_FIFO_DISABLED);
61
62 if (USE_HIGH_RES_MODE) {
63 rc |= inv_imu_enable_high_resolution_fifo(&myImuDevice);
64 } else {
65 rc |= inv_imu_set_accel_fsr(&myImuDevice, ACCEL_CONFIG0_FS_SEL_4g);
66 rc |= inv_imu_set_gyro_fsr(&myImuDevice, GYRO_CONFIG0_FS_SEL_250dps);
67 }
68
69 if (USE_LOW_NOISE_MODE) {
70 rc |= inv_imu_set_accel_frequency(&myImuDevice, ACCEL_CONFIG0_ODR_50_HZ);
71 rc |= inv_imu_set_gyro_frequency(&myImuDevice, GYRO_CONFIG0_ODR_200_HZ);
72 rc |= inv_imu_enable_accel_low_noise_mode(&myImuDevice);
73 } else {
74 rc |= inv_imu_set_accel_frequency(&myImuDevice, ACCEL_CONFIG0_ODR_50_HZ);
```

```
 75 rc |= inv_imu_set_gyro_frequency(&myImuDevice, GYRO_CONFIG0_ODR_200_HZ);
 76 rc |= inv_imu_enable_accel_low_power_mode(&myImuDevice);
 77 }
 78 //    rc |= inv_imu_set_accel_lp_avg(&myImuDevice, ACCEL_CONFIG1_ACCEL_FILT_AVG_8);
 79
 80 rc |= inv_imu_enable_gyro_low_noise_mode(&myImuDevice);
 81
 82 if (!USE_FIFO)
 83 inv_imu_sleep_us(GYR_STARTUP_TIME_US);
 84
 85 return rc;
 86 }
 87
 88
 89
 90 //-------------------------------------------------------------------------------// setupImuDevice
 91  /** @brief This function is in charge of reseting and initializing IMU device.
 92  *          It should be successfully executed before any access to IMU device.
 93  *
 94  *   @param[in]     icm_serif       pointer to the serial interface structure
 95  *   @return        rc              InvError structure parameter, 0 on success
 96  */

 97 int setupImuDevice(struct inv_imu_serif *icm_serif){
 98
 99 int rc = 0;
100 uint8_t who_am_i;
101
102 // Initialization of the device
103 rc = inv_imu_init(&myImuDevice, icm_serif, imu_callback);
104 if (rc != INV_ERROR_SUCCESS){
105
106 return rc;
107 }
108
109 // Check WHOAMI
110 rc = inv_imu_get_who_am_i(&myImuDevice, &who_am_i);
111 if (rc != INV_ERROR_SUCCESS){
112
113 return rc;
114 }
115
116 if (who_am_i != ICM_WHOAMI){
117
118 return INV_ERROR;
119 }
120
121 #if !USE_FIFO
122 RINGBUFFER_CLEAR(&timestamp_buffer);
123 #endif
124
125 return rc;
126 }
127
128
129 //-------------------------------------------------------------------------------// inv_imu_sleep_us
130  /** @brief This function is in charge of delaying the program for a certain
131  *          time.
132  *
133  *   @param[in]     us              time in microsecond
134  *   @return        -               -
135  */
136 void inv_imu_sleep_us(uint32_t us){
137
138     uint16_t finalValue;
139
140     // Prepares the Timer3 and the counting variable
141     DRV_TMR3_CounterClear();
142 //    appData.usCounter32 = 0;
143     // Starts Timer3
144     DRV_TMR3_Start();

145
146 //    finalValue = (us * 14.745600) + 8;  //+8 pour arrondir à la 1/2 us supérieure lors du passage float -> int
147     finalValue = us * 15;
148
```

```
149     // Wait until the while loop is not true anymore
150 //    while(appData.usCounter32 < us){}
151     while (DRV_TMR3_CounterValueGet() < finalValue);
152     // Stops Timer3
153     DRV_TMR3_Stop();
154 }
155 
156 
157 //------------------------------------------------------------------------------// inv_imu_sleep_ms
158  /** @brief This function is in charge of delaying the program for a certain
159  *          time.
160  *
161  *   @param[in]     ms              time in millisecond
162  *   @return        -               -
163  */
164 void inv_imu_sleep_ms(uint32_t ms){
165 
166     uint32_t i;
167     for (i = 0; i < ms; i++) {
168 
169         inv_imu_sleep_us(1000);
170     }
171 }
172 
173 
174 //------------------------------------------------------------------------------// inv_imu_get_time_us
175  /** @brief This function is in charge of TIMOUT uses
176  *
177  *   @param[in]     -               -
178  *   @return        xxx
179  */
180 uint64_t inv_imu_get_time_us(void){
181 
182     // Not in int64 but normal
183     return DRV_TMR2_CounterValueGet() / 15; // + appData.usCounter64;
184 }
```