```
/*
 Controller:    nRF52832
 Hardware:      BT_RFIDREADER_REV02
 Folder/SDK:    C:\Nordic Semi\nRF5_SDK_16.0.0\examples\ble_peripheral\experimental\ble_app_cgms
 Application:   BLE RFID Reader
 IDE:           Segger Embedded Studio
 Complier:      Theo Segger Embedded Studio
 Programer: Segger J-Link (nRF52832 Starter Kit)(nRFgo Studio + SoftDevice S132)
 Company:       OSSTech Co.,Ltd
 Writer:        Vo Duc Minh
 Date:          2019.10.29, 2019.11.13, 2019.12.02, 2019.12.23,
                2020.01.10, 2020.01.15, 2020.02.28, 2020.03.05, 2020.07.31

 * Cac file da sua:

 * Cac file lay tu thu vien-chuyen sang cung thu muc voi project, va doi ten.
 Ta khong can tac dong den cac file nay nua.
 cgms_db.c -> rfid_db.c
 cgms_db.h -> rfid_db.h
 cgms_meas.c -> rfid_meas.c
 cgms_meas.h -> rfid_meas.h
 cgms_racp.c -> rfid_racp.c
 cgms_racp.h -> rfid_racp.h
 cgms_socp.c -> rfid_socp.c
 cgms_socp.h -> rfid_socp.h
 cgms_sst.c -> rfid_sst.c
 cgms_sst.h -> rfid_sst.h
 nrf_ble_cgms.c -> rfid_nrf_ble.c
 nrf_ble_cgms.h -> rfid_nrf_ble.h

 * Ngay 2020.01.15:
 Sua trong file "rfid_meas.c": tang kieu du lieu cho cgms tu 16bit len 40bit. Nhu vay moi du cho du lieu rfid.

 * Ngay 2020.02.20:
 Sua trong file "rfid_socp.c": viet them code trong ham "on_socp_value_write" de nhan du lieu tu app. Luu flash.
 Sua trong file "sdk_config.h": khai bao cac dia chi flash.
 + Phan tich goi tin nhan duoc tu characteristic "CGM Specific Ops Control Point" va luu vao flash.
 De lay ten thiet bi, thi "socp_request.p_operand" chi cho phep max 20 kytu thoi.
 VD: "TenTB=voducminh12345" ok
     "TenTB=voducminh123456" thi kg dc vi 21 kytu.

 * Ngay 2020.03.05:
 Sua trong project: them thu vien "nRF_Drivers" >> "nrf_nvmc.c"
 Sua trong file "rfid_socp.c": them thu vien #include "nrf_nvmc.h"

 */

#include <stdint.h>
#include <string.h>
#include "nordic_common.h"
#include "nrf.h"
#include "app_error.h"
#include "ble.h"
#include "ble_srv_common.h"
#include "ble_advertising.h"
#include "ble_conn_params.h"
#include "sensorsim.h"
#include "nrf_sdh.h"
#include "nrf_sdh_soc.h"
#include "nrf_sdh_ble.h"
#include "app_timer.h"
#include "bsp_btn_ble.h"
#include "peer_manager.h"
#include "peer_manager_handler.h"
#include "fds.h"
```

```
66  #include "ble_conn_state.h"
67  #include "nrf_ble_qwr.h"
68  #include "nrf_ble_gatt.h"
69  #include "ble_bas.h"
70  #include "nrf_ble_bms.h"
71  #include "rfid_nrf_ble.h"
72  #include "ble_dis.h"
73  #include "ble_racp.h"
74  #include "nrf_pwr_mgmt.h"
75
76  // Minh them vao
77  #include <stdbool.h>
78  #include "nrf_delay.h"
79  #include "nrf52.h"
80
81  #define MANUFACTURER_NAME               "OSSTECH Co.,Ltd"
    /**< Manufacturer. Will be passed to Device Information Service. */
82
83  #define USE_AUTHORIZATION_CODE          1
84
85  #define APP_BLE_OBSERVER_PRIO           3
    /**< Application's BLE observer priority. You shouldn't need to modify this value. */
86  #define APP_BLE_CONN_CFG_TAG            1
    /**< A tag identifying the SoftDevice BLE configuration. */
87
88  #define GLUCOSE_MEAS_INTERVAL           1
    /**< Fastest possible communication interval*/
89
90  #define SECOND_10_MS_UNITS              100
    /**< Definition of 1 second, when 1 unit is 10 ms. */
91  #define MIN_CONN_INTERVAL               7
    /**< Minimum acceptable connection interval (0.25 seconds), Connection interval uses
    1.25 ms units. */
92  #define MAX_CONN_INTERVAL               400
    /**< Maximum acceptable connection interval (0.5 second), Connection interval uses 1.25
    ms units. */
93  #define SLAVE_LATENCY                   0
    /**< Slave latency. */
94  #define CONN_SUP_TIMEOUT                (4 * SECOND_10_MS_UNITS)
    /**< Connection supervisory timeout (4 seconds), Supervision Timeout uses 10 ms units. */
95
96  #define FIRST_CONN_PARAMS_UPDATE_DELAY  APP_TIMER_TICKS(15000)
    /**< Time from initiating event (connect or start of notification) to first time
    sd_ble_gap_conn_param_update is called (5 seconds). */
97  #define NEXT_CONN_PARAMS_UPDATE_DELAY   APP_TIMER_TICKS(5000)
    /**< Time between each call to sd_ble_gap_conn_param_update after the first call (30
    seconds). */
98  #define MAX_CONN_PARAMS_UPDATE_COUNT    3
    /**< Number of attempts before giving up the connection parameter negotiation. */
99
100 #define APP_ADV_FAST_INTERVAL           80   /**< Fast advertising interval (in units
    of 0.625 ms. This value corresponds to 50ms.) */
101 #define APP_ADV_SLOW_INTERVAL           1600 /**< Slow advertising interval (in units
    of 0.625 ms. This value corresponds to 1s). */
102
103 #define APP_ADV_FAST_DURATION           (2*6000) /**< 2 phut The advertising duration
    of fast advertising in units of 10 milliseconds. */
104 #define APP_ADV_SLOW_DURATION           (5*6000) /**< 5 phut The advertising duration
    of slow advertising in units of 10 milliseconds. */
105
106 #define MEM_BUFF_SIZE                   512
107
108 #define DEAD_BEEF                       0xDEADBEEF
    /**< Value used as error code on stack dump, can be used to identify stack location on
    stack unwind. */
109
110 APP_TIMER_DEF(m_battery_timer_id);
    /**< Battery timer. */
111 BLE_BAS_DEF(m_bas);
```

```
    /**< Battery service instance. */
112 NRF_BLE_CGMS_DEF(m_cgms);
    /**< CGMS instance. */
113 NRF_BLE_BMS_DEF(m_bms);
    /**< Bond Management service instance. */
114 NRF_BLE_GATT_DEF(m_gatt);
    /**< GATT module instance. */
115 NRF_BLE_QWR_DEF(m_qwr);
    /**< Context for the Queued Write module. */
116 BLE_ADVERTISING_DEF(m_advertising);
    /**< Advertising module instance. */
117 NRF_BLE_GQ_DEF(m_ble_gatt_queue,
    /**< BLE GATT Queue instance. */
118                NRF_SDH_BLE_PERIPHERAL_LINK_COUNT,
119                NRF_BLE_GQ_QUEUE_SIZE);
120
121 static uint16_t m_conn_handle = BLE_CONN_HANDLE_INVALID,
    /**< Handle of the current connection. */
122                 m_current_offset;
123
124 static ble_conn_state_user_flag_id_t m_bms_bonds_to_delete;
    /**< Flags used to identify bonds that should be deleted. */
125 static bool delete_all_pending = false;
    /**< Flag to show that advertising should not be started after a single peer is deleted
    (@ref PM_EVT_PEER_DELETE_SUCCEEDED) because we are waiting for ALL peers to be deleted
    (@ref PM_EVT_PEERS_DELETE_SUCCEEDED). */
126
127 static sensorsim_cfg_t   m_battery_sim_cfg;
    /**< Battery Level sensor simulator configuration. */
128 static sensorsim_state_t m_battery_sim_state;
    /**< Battery Level sensor simulator state. */
129
130 static ble_uuid_t m_adv_uuids[] = {{BLE_UUID_CGM_SERVICE,  BLE_UUID_TYPE_BLE},};
    /**< Universally unique service identifiers. */
131
132 #if USE_AUTHORIZATION_CODE
133 static uint8_t m_auth_code[]            = {'A', 'B', 'C', 'D'}; //0x41, 0x42, 0x43, 0x44
134 static int m_auth_code_len              = sizeof(m_auth_code);
135 #endif
136
137 #define BTN1 (1<<13)
138 #define BTN2 (1<<14)
139 #define BTN3 (1<<15)
140 #define BTN4 (1<<16)
141
142 #define LED1 (1<<17)
143 #define LED2 (1<<18)
144 #define LED3 (1<<19)
145 #define LED4 (1<<20)
146
147 #define BUZZER 5
148 #define TEST_POINT_P26 26
149 #define TEST_POINT_P28 28
150 #define TEST_POINT_P29 29
151 #define TEST_POINT_P20 20
152 #define TEST_POINT_P22 22
153 #define TEST_POINT_P23 23
154 #define TEST_POINT_P24 24
155 #define TEST_POINT_P25 25
156 #define TEST_POINT_P13 13
157 #define TEST_POINT_P14 14
158 #define TEST_POINT_P8  8
159 #define TEST_POINT_P9  9
160 #define TEST_POINT_P10 10
161 #define TEST_POINT_P11 11
162 #define TEST_POINT_P6  6
163 #define BT_HANDLE_PIN           12  // Noi voi P0.16 de kich hoat P0.16.
164 #define ACTIVE_EM_PIN           27
165 #define KENH_ANA_BATT           7   // la AIN7
```

```
166 #define PinDieukhien5V          30
167 #define RFID_DTA_PIN1           3   // De doc muc logic DTA
168 #define RFID_DTA_PIN2           4   // De bat xung len
169 #define RFID_CLK_PIN            7   // De dem xung clock
170 #define RFID_DTA_SAMPLING       ((NRF_P0->IN & (1<<RFID_DTA_PIN1))&&1)
171
172 #define SO_BYTES_TRONG_MASOTHE  5
173 #define XUNG_TPHONG             14  // Thong so nay co the chinh tuy y trong khoang 5
    den 16,
174                                     // cpu chay nhanh thi tiem can 16, neu cpu chay
                                        cham thi tiem can 5.
175
176 #define CANHLEN_RFID_DTA        (NRF_GPIOTE->EVENTS_IN[RFID_DTA_PIN2])
177 #define DEM_RFID_CLK            (NRF_TIMER2->CC[0])
178 #define XOA_DEM_RFID_CLK        (NRF_TIMER3->TASKS_START=1)   // Tu dong STOP qua PPI
179
180 APP_TIMER_DEF(batt_level_timer_id);
181 #define DINH_THOI_KTRA_PIN            APP_TIMER_TICKS(59000)  // 59s de kg bi trung voi
    2 phut
182 #define BATTERY_LEVEL_MEAS_INTERVAL   APP_TIMER_TICKS(120000) // 2 phut gui % pin len
    app 1 lan
183
184 APP_TIMER_DEF(em4095_timer_id);
185 #define TGIAN_TAT_EM4095   APP_TIMER_TICKS(100)  // 100ms
186 #define TGIAN_BAT_EM4095   APP_TIMER_TICKS(200)  // 200ms
187
188 #define TGIAN_GIAIMA      94    // 94ms
189 #define SL_ADC_TRUNGBINH  21    // So luong nay tuy y nguoi su dung
190 #define PWM_CNT_TOP       46    // =(PRESCALER 125000 Hz / tan so mong muon).
191 #define TGIAN_COIKEU      74    // 74ms
192 #define TGIAN_CHONGLAPTHE 800   // 800ms
193
194 static uint8_t m_mem[MEM_BUFF_SIZE], battLevel=100, knoiApp=0, bienHang, bienCot,
195                maTran[11][5], em4102[SO_BYTES_TRONG_MASOTHE],
                   em4102Sub[SO_BYTES_TRONG_MASOTHE],
196                manchesterBit, bienDem110, em4102Memory[120],
197                bienLatGiaima=1;
198 static uint16_t battLevelAvg[SL_ADC_TRUNGBINH],
199                 pwmLduty[4]={23, 0, 0, 0};  // pwmLduty la L-duty trong chu ky; nen
                    tang pwmLduty la giam H-duty.
200                                             // Duty 50% nen bang 23 = (PWM_CNT_TOP / 2).
201                                             // Chi dat gia tri tai pwmLduty[0] vi chi
                                                can phat xung ra PSEL.OUT[0] thoi.
202 static uint32_t tGiaima, bienChay, bienChung, kquaAdcBatt[1], tCoikeu=0,
    tChongLapThe=0; // 32 bit
203 static volatile uint32_t tMs=0; // Vi bien xuat hien trong ham ngat nen phai khai bao
    la volatile
204 static uint64_t dtaRfid;
205
206 typedef enum
207 {
208     DO_RONG_XUNG_DTA=0,
209     DO_RONG_XUNG_DTA_CANH2,
210     BATDAU_90MS,
211     CHO_90MS_TIM9BIT1,
212     KTRA_PARITY
213 } _CHUTRINH_GIAIMA;
214 _CHUTRINH_GIAIMA chutrinhGiaima= DO_RONG_XUNG_DTA;
215
216 //*****************************************************************************
    *******
217
218 // Cac ham ngat dat static thi lam cho cong suat kg giam duoc.
219 void TIMER1_IRQHandler (void)
220 {
221     // Giua moi lan co EVENTS_COMPARE[0] bat len la 1ms.
222     if(NRF_TIMER1->EVENTS_COMPARE[0])     // Bat EVENTS_COMPARE[0] vi lien quan CC[0]
223     {
224         NRF_TIMER1->EVENTS_COMPARE[0]=0;  // Kg co thi kg chay
```

```
225             ++tMs;
226         }
227     }
228 
229     //*************************************************************************************
    *******
230 
231     static void batt_level_timeout_handler(void * p_context)
232     {
233         uint8_t bienTam, bienTam2;
234         uint32_t bienTam1;
235         UNUSED_PARAMETER(p_context);
236 
237         // Cai dat SAADC
238         NRF_SAADC->CH[KENH_ANA_BATT].PSELP=8;       // AIN7 - P0.31
239         NRF_SAADC->CH[KENH_ANA_BATT].PSELN=0;       // Not connect
240         NRF_SAADC->CH[KENH_ANA_BATT].CONFIG=(0      // Bypass RESP
241                                             |0<<4  // Bypass RESN
242                                             |5<<8  // Gain=1
243                                             |0<<12 // Internal Ref 0.6V
244                                             |2<<16 // TACQ 10us
245                                             |0<<20 // Mode SingleEnd, PSELN will be
                                                    ignored, negative input to ADC shorted to GND
246                                             );
247         // Tu dong nhan biet so luong kenh
248         bienTam2=0;
249         for (bienTam=0; bienTam<8; bienTam++) // 8 la kenh analog 1->8 cua nRF52832
250         {
251             if (NRF_SAADC->CH[bienTam].PSELP)
252                 ++bienTam2;
253         }
254         NRF_SAADC->RESOLUTION=1;                // 1 = 10-bit
255         NRF_SAADC->SAMPLERATE=0;                // Rate is controlled from SAMPLE task
256         NRF_SAADC->RESULT.PTR = (uint32_t)&kquaAdcBatt;
257         NRF_SAADC->RESULT.MAXCNT=bienTam2;
258         NRF_SAADC->ENABLE=1;
259 
260         // Xoa cac co bao truoc khi cho ADC chay tiep
261         NRF_SAADC->EVENTS_STARTED=0;
262         NRF_SAADC->EVENTS_RESULTDONE=0;
263 
264         // Bat dau convert lai 1 lan nua
265         NRF_SAADC->TASKS_START = 1;   // Ngay sau lenh nay, TASKS_START tu xoa thanh 0
266         while (!NRF_SAADC->EVENTS_STARTED);
267         NRF_SAADC->TASKS_SAMPLE = 1;  // Ngay sau lenh nay, TASKS_SAMPLE tu xoa thanh 0,
268                                       // Tuc la, TASKS_SAMPLE=1 de cho phep 1 lan convert,
                                          va ADC chi convert 1 lan roi dung lai.
269         while (!NRF_SAADC->EVENTS_RESULTDONE);  // Co ve on dinh hon EVENTS_DONE
270                                                 // Sau lenh nay, EVENTS_RESULTDONE van = 1
271         while (NRF_SAADC->STATUS);  // Nen kiem tra thanh ghi nay, vi:
272                                     // EVENTS_RESULTDONE may occur before the actual values
                                        transferred into RAM by EasyDMA.
273         NRF_SAADC->TASKS_STOP=1;
274         while(!NRF_SAADC->EVENTS_STOPPED);
275         // Tat ca cac while tren chay qua rat nhanh, khoang it hon 18.6us
276 
277         /*
278         battLevel la ket qua cuoi cung de gui len app.
279         battLevelAvg:     [0]...[SL_ADC_TRUNGBINH-2]            [SL_ADC_TRUNGBINH-1]
280                               lich su kqua adc                    so luong hien tai
281 
282         785 tuong ung muc ap minnimum 0.46V tren thang do, 2 don vi tuong ung voi 1%
283         Tom tat:    V batt    V thang dien tro    Phan tram     Gia tri ADC
284                      4.2V          0.6V              100%           1023
285                      ---           ---               ---            ---
286                     3.227V        0.461V             1%             787
287                      3.2V          0.46V             0%             785
288         */
289         battLevelAvg[battLevelAvg[SL_ADC_TRUNGBINH-1]]=((kquaAdcBatt[0]&0x3FF)-785)/2;
```

```
290
291         // Ket qua cuoi cung la trung binh cong cua 20 ket qua trung gian
292         bienTam1=0;
293         bienTam2=0;
294         for(bienTam=0; bienTam<(SL_ADC_TRUNGBINH-1); bienTam++)
295         {
296             bienTam1 += battLevelAvg[bienTam];
297             if(battLevelAvg[bienTam]!=0)
298                 ++bienTam2;
299         }
300         battLevel = (bienTam1/bienTam2);
301
302         // Luu het 20 ket qua thi quay ve ban dau de luu.
303         ++battLevelAvg[SL_ADC_TRUNGBINH-1];
304         if(battLevelAvg[SL_ADC_TRUNGBINH-1]==(SL_ADC_TRUNGBINH-1))
305             battLevelAvg[SL_ADC_TRUNGBINH-1]=0;
306     }
307
308     //*************************************************************************************
    *******
309
310     void BatPWM0(void)
311     {
312         // Cai dat PWM0
313         NRF_PWM0->DECODER           = (2 |      // Load Individual
314                                        (0<<8));  // Refresh
315         NRF_PWM0->PSEL.OUT[0]       = (BUZZER |
316                                        (0<<31)); // Internal Connect
317         NRF_PWM0->MODE              = 0;        // Up counter - edge aligned PWM duty-cycle
318         NRF_PWM0->PRESCALER         = 7;        // Divide by 128 ( 125kHz)
319         NRF_PWM0->COUNTERTOP        = PWM_CNT_TOP;
320         NRF_PWM0->SEQ[0].PTR        = (uint32_t)pwmLduty;
321         NRF_PWM0->SEQ[0].CNT        = 4; // 0, 1, 2, 3 deu kg chay; phai dat 4 moi dc
322         NRF_PWM0->SEQ[0].REFRESH    = 0; // Kg co cung dc.
323                                          // Setting the register to zero will result in a
                                             new duty cycle update every PWM period
324                                          // as long as the minimum PWM period is observed.
325         NRF_PWM0->SEQ[0].ENDDELAY   = 0; // Kg co cung dc
326         NRF_PWM0->ENABLE            = 1; // Nen dat o cuoi vi khi ENABLE=0 thi PSEL.OUT moi
            co tac dung
327         NRF_PWM0->TASKS_SEQSTART[0] = 1;
328     }
329
330     void TatPWM0(void)
331     {
332         NRF_PWM0->ENABLE      = 0;  // Kg co van chay
333         NRF_PWM0->TASKS_STOP  = 1;
334     }
335
336     //*************************************************************************************
    *******
337
338     void TatNgoaiviQuetthe(void)
339     {
340         NRF_P0->OUTCLR = (1<<ACTIVE_EM_PIN);
341         NRF_GPIOTE->CONFIG[RFID_DTA_PIN2]=0;  // Tat di vi khi active cung tieu ton max 0.5uA
342         NRF_GPIOTE->CONFIG[RFID_CLK_PIN]=0;
343         NRF_PPI->CHENCLR = 1;
344         NRF_PPI->CHENCLR = (1<<1);
345         NRF_PPI->CHENCLR = (1<<2);
346         NRF_TIMER2->TASKS_STOP = 1;
347         NRF_TIMER3->TASKS_STOP = 1;
348     }
349
350     //*************************************************************************************
    *******
351
352     static void Em4095_timeout_handler(void * p_context)
353     {
```

```
354         ret_code_t err_code;
355         while(1)
356         {
357             err_code = app_timer_stop(em4095_timer_id); // Lenh nay xu ly cuc ky nhanh
358             if (err_code == NRF_SUCCESS)
359                 break;
360             // Neu dung "APP_ERROR_CHECK(err_code)" thi hay bi loi mat ket noi voi app
361         }
362         switch(knoiApp)
363         {
364             case 1:
365             {
366                 knoiApp=0x11;
367                 while(1)
368                 {
369                     err_code = app_timer_start(em4095_timer_id, TGIAN_BAT_EM4095, NULL);
370                     if (err_code == NRF_SUCCESS)
371                         break;
372                 }
373 
374 
                    //.....................................................................
                    ...........
375 
376                 // Cho EM4095 chay
377                 NRF_P0->OUTSET = (1<<ACTIVE_EM_PIN);
378 
379 
                    //.....................................................................
                    ...........
380 
381                 // Cai dat GPIOTE de bat canh len RFID data. Kg can cai dat
                    NRF_P0->PIN_CNF[RFID_DTA_PIN2],
382                 // vi event mode la da config thanh input luon roi.
383                 NRF_GPIOTE->CONFIG[RFID_DTA_PIN2] = (1|                 // Event mode
384                                                      (RFID_DTA_PIN2<<8)| // Chon chan tin
                                                        hieu la RFID_DTA_PIN2
385                                                      (1<<16));          // Polarity=LoToHi
386 
387 
                    //.....................................................................
                    ...........
388 
389                 // Cai dat GPIOTE. Kg can cai dat NRF_P0->PIN_CNF[RFID_CLK_PIN], vi event
                    mode la da config thanh input luon roi.
390                 NRF_GPIOTE->CONFIG[RFID_CLK_PIN] = (1|                // Event mode
391                                                    (RFID_CLK_PIN<<8)|
392                                                    (1<<16));          // Polarity=LoToHi
393                 // Cai dat Counter2 va PPI[0]
394                 // Can Counter2 de dem RFID Clock, nhung lien quan GPIOTE-PPI vi nRF52 kg
                    co chan rieng biet cho chuc nang dem.
395                 NRF_PPI->CH[0].EEP    = (uint32_t)&(NRF_GPIOTE->EVENTS_IN[RFID_CLK_PIN]);
396                 NRF_PPI->CH[0].TEP    = (uint32_t)&(NRF_TIMER2->TASKS_COUNT);
397                 NRF_PPI->FORK[0].TEP  = (uint32_t)&(NRF_TIMER2->TASKS_CAPTURE[0]);
398                 NRF_PPI->CHENSET      = 1; // Kenh 0
399                 // Cai dat Counter2 de dem xung RFID_CLK
400                 NRF_TIMER2->MODE            = 2;  // Che do Low Power Counter
401                 NRF_TIMER2->BITMODE         = 1;  // 8 bit width
402                 NRF_TIMER2->TASKS_CLEAR     = 1;
403                 NRF_TIMER2->TASKS_START     = 1;
404 
405 
                    //.....................................................................
                    ...........
406 
407                 // Cai dat PPI[1], lien quan Timer3
408                 NRF_PPI->CH[1].EEP = (uint32_t)&(NRF_TIMER3->EVENTS_COMPARE[0]);
409                 NRF_PPI->CH[1].TEP = (uint32_t)&(NRF_TIMER2->TASKS_CLEAR);
410                 NRF_PPI->CHENSET = (1<<1);
```

```
411             // Cai dat Timer3 va PPI[2]
412             // Vi sao dung timer nay: xem 2 hinh "XOA_DEM_RFID_CLK chay ok.jpg",
                "XOA_DEM_RFID_CLK chay sai.jpg"
413             NRF_PPI->CH[2].EEP = (uint32_t)&(NRF_TIMER3->EVENTS_COMPARE[1]);
414             NRF_PPI->CH[2].TEP = (uint32_t)&(NRF_TIMER2->TASKS_COUNT);
415             NRF_PPI->FORK[2].TEP = (uint32_t)&(NRF_TIMER2->TASKS_CAPTURE[0]);
416             NRF_PPI->CHENSET = (1<<2);
417             // Cai dat Timer3, ho tro XOA_DEM_RFID_CLK
418             NRF_TIMER3->CC[0]       = 1;        // De cho NRF_TIMER2->TASKS_CLEAR xay
                ra truoc
419             NRF_TIMER3->CC[1]       = 2;        // De cho NRF_TIMER3->EVENTS_COMPARE[1]
                xay ra sau
420             NRF_TIMER3->SHORTS      = ((1<<1)|  // Enable shortcut between COMPARE[1]
                event and CLEAR task
421                                        (1<<9)); // Enable shortcut between
                                        COMPARE[1] event and STOP task
422             NRF_TIMER3->MODE        = 0;        // Che do Timer
423             NRF_TIMER3->BITMODE     = 1;        // 8 bit timer bit width
424             NRF_TIMER3->PRESCALER   = 0;        // Sao cho nhanh nhat co the
425
426
                //..............................................................................
                ..........
427
428             break;
429         }
430
431         case 0x11:
432         {
433             knoiApp=1;  // Vi chay den day la da co ket noi voi app roi, nen phai tra
                ve 1
434             while(1)
435             {
436                 err_code = app_timer_start(em4095_timer_id, TGIAN_TAT_EM4095, NULL);
437                 if (err_code == NRF_SUCCESS)
438                     break;
439             }
440             // Tat peripheral trc khi sleep de tiet kiem dien het muc co the
441             TatNgoaiviQuetthe();
442             break;
443         }
444     }
445 }
446
447 //*****************************************************************************
    *******
448
449 bool Tim9Bit1 (void)
450 {
451     // Vi cach giai ma cu delay rat nhieu lan 32 xung rfid nen gay tre va de sai. Bay
    gio dung cach khac, ta se cho canh
452     // len, sau do nhieu nhat la 4 lan delay lien tuc 32 xung clock, nhu vay it sai sot
    hon.
453     // Toi day kg nen "return" lien tuc nhu trong "GiaiMaEm4102(void)", se mat RFID
    clock-kg chay dc.
454
455     /*
456     Bo doan code nay van chay duoc:
457     XOA_DEM_RFID_CLK;
458     CANHLEN_RFID_DTA=0;
459     while(!CANHLEN_RFID_DTA)
460     {
461         // Cho xung dau tien lam moc, la xung len dau tien trong hinh
        "rfid_grafik.gif", thuoc phan cuoi cua StopBit.
462         if (DEM_RFID_CLK > (64+64+XUNG_TPHONG)) // Day la loai xung dai nhat
463             return 0;
464     }
465     */
466
```

```
467       // Xung dau tien xuat hien roi, ta se cho xung thu 2 xuat hien, luc cho se kiem tra
          vai dieu kien
468       // Xung truoc vua xuat hien la XOA_DEM_RFID_CLK ngay, sau do lai cho xung tiep theo
          nen viec
469       // DEM_RFID_CLK van ok - kg bi hao hut.
470       XOA_DEM_RFID_CLK;
471       CANHLEN_RFID_DTA=0;
472       bienChay=0;
473       while(!CANHLEN_RFID_DTA) // Cho xung RFID_DTA xuat hien
474       {
475           // Vi ngay sau day co nhieu xu ly lien quan DEM_RFID_CLK, nen luu 1 lan de dung
              chung, tranh sai lech khi read DEM_RFID_CLK.
476           bienHang=DEM_RFID_CLK;
477           // Kiem tra 3 vi tri dau tien trong hinh "rfid_grafik.gif"
478           if
              ((bienHang==XUNG_TPHONG)||(bienHang==(XUNG_TPHONG+32))||(bienHang==(XUNG_TPHONG+3
              2*2)))
479           {
480               if ((bienChay==0)&&(!RFID_DTA_SAMPLING))
481                   return 0;
482               else if ((bienChay==1)&&(!RFID_DTA_SAMPLING))
483                   return 0;
484               else if ((bienChay==2)&&RFID_DTA_SAMPLING)
485                   return 0;
486               ++bienChay;
487               while ((DEM_RFID_CLK-bienHang)==0); // Cho DEM_RFID_CLK tang len mot chut,
                  neu kg thi cpu se quet moc XUNG_TPHONG
488                                                   // mot lan nua - do cpu nhanh hon clock
                                                      RFID.
489           }
490           else if (bienHang > (64+32+XUNG_TPHONG))// Loai bo xung data qua dai
491               return 0;                           // Can bat duoc 01 (chinh la giao diem
                  bit stop va bit 1 dau tien)
492       }
493       if (bienChay<3)     // Neu dung thi phai bang 3; loai bo xung qua ngan, xung ngan
          thi chi bang 0-1-2 la cung
494           return 0;
495
          //.....................................................................................
          ....
496       // Kiem tra 8 bit 1 tiep theo trong so 9 bit 1, co the co data nua.
497       bienDem110=1;   // Vi sau nay co dieu kien chia het cho 2, nen dat bang 1 tu ban
          dau cho de hieu.
498       for (bienChay=0; bienChay<8; ++bienChay)
499       {
500           XOA_DEM_RFID_CLK;
501           CANHLEN_RFID_DTA=0;
502           bienCot=0;
503           while(!CANHLEN_RFID_DTA)
504           {
505               // Vi ngay sau day co nhieu xu ly lien quan DEM_RFID_CLK, nen luu 1 lan de
                  dung chung, tranh sai lech khi read DEM_RFID_CLK.
506               bienHang=DEM_RFID_CLK;
507               if ((bienHang==XUNG_TPHONG)||(bienHang==(XUNG_TPHONG+32)))
508               {
509                   ++bienCot;
510                   while ((DEM_RFID_CLK-bienHang)==0); // Cho DEM_RFID_CLK tang len mot
                      chut, neu kg thi cpu se quet moc XUNG_TPHONG
511                                                       // mot lan nua - do cpu nhanh hon
                                                          clock RFID.
512               }
513               if ((bienHang==XUNG_TPHONG)&&(!RFID_DTA_SAMPLING))
514                   return 0;
515               else if ((bienHang==XUNG_TPHONG+32)&&RFID_DTA_SAMPLING)
516                   return 0;
517               else if ((bienChay==7)&&(bienHang==(XUNG_TPHONG+32*2)))
518               {
519                   em4102Memory[bienDem110]=RFID_DTA_SAMPLING;
520                   ++bienDem110;
```

```
521                 while ((DEM_RFID_CLK-bienHang)==0);
522                 // Cho xung len chu kg break, vi con phai lay moc xung len de sau nay
                    XOA_DEM_RFID_CLK nua.
523                 // bienCot kg can xuat hien trong nay
524             }
525             else if ((bienChay==7)&&(bienHang>(64+32+XUNG_TPHONG))) // Loai bo xung
                data qua dai
526                 return 0;
527             else if ((bienChay<7)&&(bienHang>(64+XUNG_TPHONG)))     // Loai bo xung
                data qua dai
528                 return 0;
529         }
530         if (bienCot<2)     // Neu dung thi phai bang 2, loai bo xung qua ngan, xung
            ngan thi chi bang 0 hoac 1 la cung
531             return 0;
532     }
533     // Den day da tim duoc 9 bit 1 roi, loc lay cac bit manchester luon cho nhanh.
534     // Neu o tren em4102Memory[bienDem110] co du lieu thi bienDem110 phai bang 2, con
      kg thi bienDem110 van bang 1.
535     for ( ;bienDem110<=110; )
536     {
537         XOA_DEM_RFID_CLK;
538         CANHLEN_RFID_DTA=0;
539         while(!CANHLEN_RFID_DTA)
540         {
541             bienHang=DEM_RFID_CLK;
542             if
                ((bienHang==XUNG_TPHONG)||(bienHang==(XUNG_TPHONG+32))||(bienHang==(XUNG_TPHO
                NG+2*32))||(bienHang==(XUNG_TPHONG+3*32)))
543             {
544                 // em4102Memory chay tu em4102Memory[1] -> em4102Memory[110]
545                 em4102Memory[bienDem110]=RFID_DTA_SAMPLING;
546                 ++bienDem110;
547                 bienChung=1;
548                 while ((DEM_RFID_CLK-bienHang)==0);
549                 if (bienDem110>110) // Thoat ngay cho nhanh, vi co luc thay bienDem110
                    dem len den 113, co the gay loi sau nay
550                     break;
551             }
552             else
553                 bienChung=0;
554
555             // Kiem tra nua truoc va nua sau cua manchester bit, hai nua nay phai luon
                khac nhau thi moi dung.
556             // Vi kg long nhieu lenh if duoc nen phai dung den bienChung.
557             if (((bienDem110-1)%2==0) && bienChung &&
                (em4102Memory[bienDem110-1]==em4102Memory[bienDem110-2]))
558                 return 0;
559             if (bienHang>(64+64+XUNG_TPHONG))  // Loai bo xung data qua dai
560                 return 0;
561         }
562     }
563     // Them dieu kien nay nua de thoat cho nhanh, day la bit ket thuc bo nho
      em4102Memory.
564     // Toi day chac chan em4102Memory[110] phai khac em4102Memory[109] roi
565     if (em4102Memory[110]==0)
566         return 0;
567     return 1;
568 }
569
570 //*************************************************************************************
    *******
571
572 bool GiaiMaEm4102 (void)  // Ham GiaiMaEm4102 va Tim9Bit1 dat static thi kha nang kg
    quet the duoc.
573 {
574     if (chutrinhGiaima==DO_RONG_XUNG_DTA)
575     {
576         // Do do rong xung.
```

```
577             if (bienLatGiaima)
578             {
579                 bienLatGiaima=0;
580                 XOA_DEM_RFID_CLK;
581                 CANHLEN_RFID_DTA=0;
582                 return 0; // Thoat ra cho nhanh-de con quet main nua, chu kg co van dc.
583             }
584             else if (DEM_RFID_CLK > (64+64+XUNG_TPHONG)) // Day la loai xung dai nhat, loai
                bo
585                 return 0;
586             else if (!CANHLEN_RFID_DTA)    // Cho xung thu nhat
587                 return 0;
588 
589             // Khoang thoi gian giua xung thu nhat den xung thu 2 nho hon ~64 xung thi cung
                thoat-xung nay qua ngan.
590             if (DEM_RFID_CLK < (64-XUNG_TPHONG))
591                 return 0;
592 
593             chutrinhGiaima = DO_RONG_XUNG_DTA_CANH2;
594             bienLatGiaima=1;
595             return 0;
596         }
597 
            //..............................................................................
            ....
598         else if (chutrinhGiaima==DO_RONG_XUNG_DTA_CANH2)
599         {
600             // Do do rong xung.
601             if (bienLatGiaima)
602             {
603                 bienLatGiaima=0;
604                 XOA_DEM_RFID_CLK;
605                 CANHLEN_RFID_DTA=0;
606                 return 0;
607             }
608             else if (DEM_RFID_CLK > (64+64+XUNG_TPHONG)) // Day la loai xung dai nhat, loai
                bo
609             {
610                 chutrinhGiaima = DO_RONG_XUNG_DTA;
611                 bienLatGiaima=1;
612                 return 0;
613             }
614             else if (!CANHLEN_RFID_DTA)    // Cho xung thu 2
615                 return 0;
616 
617             // Khoang thoi gian giua xung thu nhat den xung thu 2 nho hon ~64 xung thi cung
                thoat-xung nay qua ngan.
618             if (DEM_RFID_CLK < (64-XUNG_TPHONG))
619             {
620                 chutrinhGiaima = DO_RONG_XUNG_DTA;
621                 bienLatGiaima=1;
622                 return 0;
623             }
624             chutrinhGiaima = BATDAU_90MS;
625             return 0;
626         }
627 
            //..............................................................................
            ....
628         else if (chutrinhGiaima==BATDAU_90MS)
629         {
630             // Xung EM4102 la 2Kbits/s thi can 31.25ms de truyen di het 64 bit trong bo nho
                cua no.
631             // Minh se doi 3 lan truyen het ROM cua EM4102.
632             tGiaima=tMs;
633             chutrinhGiaima = CHO_90MS_TIM9BIT1;
634             return 0;
635         }
636 
```

```
        //.....................................................................
        ....
637     else if (chutrinhGiaima==CHO_90MS_TIM9BIT1)
638     {
639         // Bat dau giai ma
640         if (Tim9Bit1())
641         {
642             chutrinhGiaima = KTRA_PARITY;
643             return 0;  //Khong can doi het 90ms, thoat luon.
644         }
645         /*
646         Quet trong vong ~90ms:
647         Neu khong tim duoc Tim9Bit1 + chua qua 90ms nen van tiep tuc tim Tim9Bit1.
648         Neu khong tim duoc Tim9Bit1 + hon 90ms moi thoat hoan toan.
649
650         Quet the lan 1 thi maTran[][] mang gia tri dung, lan 2 khong quet the, nhung do
            anten qua nhay/anten dat gan
651         nhau hoac dat gan bo IOServer thi xung data gan giong luc co the, nen firmware
            qua duoc giai doan kiem tra
652         do rong xung; khong tim duoc 9bit1 nen timer_out 90ms.
653         Neu khong return 0 sau timer out 90ms, thi firmware tiep tuc kiem tra parity,
            va maTran[][] con mang gia
654         tri cua lan quet 1 nen cuoi cung GiaiMaEm4102=1
655         */
656         if (tMs-tGiaima >= TGIAN_GIAIMA)
657         {
658             chutrinhGiaima = DO_RONG_XUNG_DTA;
659             bienLatGiaima=1;
660             return 0;
661         }
662     }
663
        //.....................................................................
        ....
664     else if (chutrinhGiaima==KTRA_PARITY)
665     {
666         //Tong hop em4102Memory vao matran.
667         bienDem110=1;
668         for (bienHang=0; bienHang<11; ++bienHang)   //Tao ma tran cac bit giong het rom
            cua EM4102
669         {
670             for (bienCot=0; bienCot<5; ++bienCot)
671             {
672                 manchesterBit=0;
673                 for (bienChay=0; bienChay<2; ++bienChay)
674                 {
675                     manchesterBit <<= 1 ;
676                     manchesterBit += em4102Memory[bienDem110];
677                     ++bienDem110;
678                 }
679                 maTran[bienHang][bienCot]=(manchesterBit >>= 1);
680             }
681         }
682
683         //Kiem tra parity hang
684         for (bienHang=0; bienHang<10; ++bienHang)
685         {
686             bienChay=0;
687             for (bienCot=0; bienCot<5; ++bienCot)
688             {
689                 bienChay ^= maTran[bienHang][bienCot];
690             }
691             if (bienChay)   //Bang 1 neu parity khong hop le, parity chan
692             {
693                 chutrinhGiaima = DO_RONG_XUNG_DTA;
694                 bienLatGiaima=1;
695                 return 0;
696             }
697         }
```

```
698
699         //Kiem tra parity cot
700         for (bienCot=0; bienCot<4; ++bienCot)
701         {
702             bienChay=0;
703             for (bienHang=0; bienHang<11; ++bienHang)
704             {
705                 bienChay ^= maTran[bienHang][bienCot];
706             }
707             if (bienChay)   //Bang 1 neu parity khong hop le, parity chan
708             {
709                 chutrinhGiaima = DO_RONG_XUNG_DTA;
710                 bienLatGiaima=1;
711                 return 0;
712             }
713         }
714
715         //Gan phan tu cua Matran vao em4102
716         for (bienChay=0; bienChay<5; ++bienChay)
717         {
718              for (bienHang = (bienChay*2); bienHang < (bienChay*2+2); ++bienHang)
719              {
720                  for (bienCot=0; bienCot < 4; ++bienCot)
721                  {
722                      em4102[bienChay] <<=1;
723                      em4102[bienChay] += maTran[bienHang][bienCot];
724                  }
725              }
726         }
727         chutrinhGiaima = DO_RONG_XUNG_DTA;
728         bienLatGiaima=1;
729
730         //Khi return, de tranh truong hop tat ca ma so deu bang 0, thi phai && 1
731         return ((em4102[0] || em4102[1] || em4102[2] || em4102[3] || em4102[4]) && 1);
732     }
733 }
734
735 //*****************************************************************************
    *******
736
737 /**@brief Callback function for asserts in the SoftDevice.
738  *
739  * @details This function will be called in case of an assert in the SoftDevice.
740  *
741  * @warning This handler is an example only and does not fit a final product. You need
     to analyze
742  *          how your product is supposed to react in case of Assert.
743  * @warning On assert from the SoftDevice, the system can only recover on reset.
744  *
745  * @param[in]   line_num   Line number of the failing ASSERT call.
746  * @param[in]   file_name  File name of the failing ASSERT call.
747  */
748 void assert_nrf_callback(uint16_t line_num, const uint8_t * p_file_name)
749 {
750     app_error_handler(DEAD_BEEF, line_num, p_file_name);
751 }
752
753 //*****************************************************************************
    *******
754
755 /**@brief Function for handling Service errors.
756  *
757  * @details A pointer to this function will be passed to each service which may need to
     inform the
758  *          application about an error.
759  *
760  * @param[in]   nrf_error   Error code containing information about what went wrong.
761  */
762 static void service_error_handler(uint32_t nrf_error)
```

```
763   {
764       APP_ERROR_HANDLER(nrf_error);
765   }
766
767   //**************************************************************************************
      *******
768
769   /**@brief Function for handling advertising errors.
770    *
771    * @param[in] nrf_error  Error code containing information about what went wrong.
772    */
773   static void ble_advertising_error_handler(uint32_t nrf_error)
774   {
775       APP_ERROR_HANDLER(nrf_error);
776   }
777
778   //**************************************************************************************
      *******
779
780   /**@brief Function for handling Queued Write Module errors.
781    *
782    * @details A pointer to this function will be passed to each service which may need to
       inform the
783    *          application about an error.
784    *
785    * @param[in]   nrf_error   Error code containing information about what went wrong.
786    */
787   static void nrf_qwr_error_handler(uint32_t nrf_error)
788   {
789       APP_ERROR_HANDLER(nrf_error);
790   }
791
792   //**************************************************************************************
      *******
793
794   static void conn_params_error_handler(uint32_t nrf_error)
795   {
796       APP_ERROR_HANDLER(nrf_error); // nrf_error: Error code containing information about
          what went wrong.
797   }
798
799   //**************************************************************************************
      *******
800
801   static void battery_level_update(void)
802   {
803       ret_code_t err_code;
804       err_code = ble_bas_battery_level_update(&m_bas, battLevel, BLE_CONN_HANDLE_ALL);
805       if ((err_code != NRF_SUCCESS) &&
806           (err_code != NRF_ERROR_INVALID_STATE) &&
807           (err_code != NRF_ERROR_RESOURCES) &&
808           (err_code != NRF_ERROR_BUSY) &&
809           (err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
810       )
811       {
812           APP_ERROR_HANDLER(err_code);
813       }
814   }
815
816   //**************************************************************************************
      *******
817
818   bool delete_bonds_pending(void)
819   {
820       // Function for determening if there are one or more connections with bonds that
          are flagged for deletion.
821       ble_conn_state_conn_handle_list_t conn_handle_list = ble_conn_state_conn_handles();
822
823       for (uint32_t i = 0; i < conn_handle_list.len; i++)
```

```
824         {
825             uint16_t conn_handle = conn_handle_list.conn_handles[i];
826             bool pending         = ble_conn_state_user_flag_get(conn_handle,
            m_bms_bonds_to_delete);
827 
828             if (pending == true)
829             {
830                 return pending;
831             }
832         }
833         return false;
834     }
835 
836     //*****************************************************************************
        *******
837 
838     // Function for setting filtered whitelist. @param[in] skip  Filter passed to @ref
        pm_peer_id_list.
839     static void whitelist_set(pm_peer_id_list_skip_t skip)
840     {
841         pm_peer_id_t peer_ids[BLE_GAP_WHITELIST_ADDR_MAX_COUNT];
842         uint32_t     peer_id_count = BLE_GAP_WHITELIST_ADDR_MAX_COUNT;
843 
844         ret_code_t err_code = pm_peer_id_list(peer_ids, &peer_id_count, PM_PEER_ID_INVALID,
            skip);
845         APP_ERROR_CHECK(err_code);
846 
847         err_code = pm_whitelist_set(peer_ids, peer_id_count);
848         APP_ERROR_CHECK(err_code);
849     }
850 
851     //*****************************************************************************
        *******
852 
853     // Function for setting filtered device identities. @param[in] skip  Filter passed to
        @ref pm_peer_id_list.
854     static void identities_set(pm_peer_id_list_skip_t skip)
855     {
856         pm_peer_id_t peer_ids[BLE_GAP_DEVICE_IDENTITIES_MAX_COUNT];
857         uint32_t     peer_id_count = BLE_GAP_DEVICE_IDENTITIES_MAX_COUNT;
858 
859         ret_code_t err_code = pm_peer_id_list(peer_ids, &peer_id_count, PM_PEER_ID_INVALID,
            skip);
860         APP_ERROR_CHECK(err_code);
861 
862         err_code = pm_device_identities_list_set(peer_ids, peer_id_count);
863         APP_ERROR_CHECK(err_code);
864     }
865 
866     //*****************************************************************************
        *******
867 
868     static void delete_bonds(void)
869     {
870         ret_code_t err_code;
871 
872         delete_all_pending = true;
873         err_code = pm_peers_delete();
874         APP_ERROR_CHECK(err_code);
875     }
876 
877     //*****************************************************************************
        *******
878 
879     static void advertising_start(bool erase_bonds)
880     {
881         // Khi khoi dong chua thay vao, luc bat dau bond moi thay chay vao day.
882         if (erase_bonds == true)
883         {
```

```
884             delete_bonds();
885             // Advertising is started by PM_EVT_PEERS_DELETE_SUCCEEDED event.
886         }
887         else
888         {
889             whitelist_set(PM_PEER_ID_LIST_SKIP_NO_ID_ADDR);
890             ret_code_t ret = ble_advertising_start(&m_advertising,
                BLE_ADV_MODE_DIRECTED_HIGH_DUTY);
891             APP_ERROR_CHECK(ret);
892         }
893     }
894
895     //**********************************************************************************
    *******
896
897     /**@brief Function for handling the Battery measurement timer timeout.
898      * @details This function will be called each time the battery level measurement timer
     expires.
899      * @param[in]   p_context   Pointer used for passing some arbitrary information
     (context) from the
900      *                          app_start_timer() call to the timeout handler.
901      */
902     static void battery_level_meas_timeout_handler(void * p_context)
903     {
904         UNUSED_PARAMETER(p_context);
905         battery_level_update();
906     }
907
908     //**********************************************************************************
    *******
909
910     static void read_glucose_measurement(void)
911     {
912         ble_cgms_rec_t rec;
913         ret_code_t     err_code;
914
915         memset(&rec, 0, sizeof(ble_cgms_rec_t));
916
917         dtaRfid=em4102[4];
918         dtaRfid=(dtaRfid<<8);
919         dtaRfid+=em4102[3];
920         dtaRfid=(dtaRfid<<8);
921         dtaRfid+=em4102[2];
922         dtaRfid=(dtaRfid<<8);
923         dtaRfid+=em4102[1];
924         dtaRfid=(dtaRfid<<8);
925         dtaRfid+=em4102[0];
926
927         rec.meas.glucose_concentration                = dtaRfid;
928         rec.meas.sensor_status_annunciation.warning   = 0;
929         rec.meas.sensor_status_annunciation.calib_temp = 0;
930         rec.meas.sensor_status_annunciation.status    = 0;
931         rec.meas.flags                                = 0;
932         rec.meas.time_offset                          = m_current_offset;
933         nrf_ble_cgms_meas_create(&m_cgms, &rec);
934
935         // Update status
936         m_cgms.sensor_status.time_offset = m_current_offset;
937         err_code = nrf_ble_cgms_update_status(&m_cgms, &m_cgms.sensor_status);
938         APP_ERROR_CHECK(err_code);
939     }
940
941     //**********************************************************************************
    *******
942
943     static void timers_init(void)
944     {
945         ret_code_t err_code;
946
```

```
947         // Initialize timer module, making it use the scheduler.
948         err_code = app_timer_init();
949         APP_ERROR_CHECK(err_code);
950 
951         // Create battery timers.
952         err_code = app_timer_create(&m_battery_timer_id, APP_TIMER_MODE_REPEATED,
            battery_level_meas_timeout_handler);
953         APP_ERROR_CHECK(err_code);
954 
955         // Tao ham dinh thoi do muc pin
956         err_code = app_timer_create(&batt_level_timer_id, APP_TIMER_MODE_REPEATED,
            batt_level_timeout_handler);
957         APP_ERROR_CHECK(err_code);
958 
959         // Tao ham dinh thoi tat EM4095
960         err_code = app_timer_create(&em4095_timer_id, APP_TIMER_MODE_REPEATED,
            Em4095_timeout_handler);
961         APP_ERROR_CHECK(err_code);
962     }
963 
964     //*****************************************************************************
    *******
965 
966     static void gap_params_init(void)
967     {
968         // This function sets up all the necessary GAP (Generic Access Profile) parameters
            of the
969         // device including the device name, appearance, and the preferred connection
            parameters.
970         ret_code_t              err_code;
971         ble_gap_conn_params_t   gap_conn_params;
972         ble_gap_conn_sec_mode_t sec_mode;
973 
974         BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);  // Cai dat security mode
975 
976         if (*(uint32_t*)DCHI_TENTB == 0xFFFFFFFF)
977             err_code = sd_ble_gap_device_name_set(&sec_mode,(const uint8_t *)TENTB_MACDINH,
                strlen(TENTB_MACDINH));
978         else
979             err_code = sd_ble_gap_device_name_set(&sec_mode,(const uint8_t *)DCHI_TENTB,
                strlen((uint8_t*)DCHI_TENTB));
980         APP_ERROR_CHECK(err_code);
981 
982         err_code = sd_ble_gap_appearance_set(BLE_APPEARANCE_GENERIC_GLUCOSE_METER);
983         APP_ERROR_CHECK(err_code);
984 
985         memset(&gap_conn_params, 0, sizeof(gap_conn_params));
986 
987         gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
988         gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
989         gap_conn_params.slave_latency     = SLAVE_LATENCY;
990         gap_conn_params.conn_sup_timeout  = CONN_SUP_TIMEOUT;
991 
992         err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
993         APP_ERROR_CHECK(err_code);
994     }
995 
996     //*****************************************************************************
    *******
997 
998     static void gatt_init(void)
999     {
1000        ret_code_t err_code = nrf_ble_gatt_init(&m_gatt, NULL);
1001        APP_ERROR_CHECK(err_code);
1002    }
1003 
1004    //*****************************************************************************
    *******
1005
```

```
1006  static void cgms_evt_handler(nrf_ble_cgms_t * p_cgms, nrf_ble_cgms_evt_t * p_evt)
1007  {
1008      ret_code_t err_code;
1009      switch (p_evt->evt_type)
1010      {
1011          case NRF_BLE_CGMS_EVT_NOTIFICATION_ENABLED:
1012          case NRF_BLE_CGMS_EVT_NOTIFICATION_DISABLED:
1013              break;
1014
1015          case NRF_BLE_CGMS_EVT_START_SESSION:
1016              break;
1017
1018          case NRF_BLE_CGMS_EVT_STOP_SESSION:
1019              break;
1020
1021          case NRF_BLE_CGMS_EVT_WRITE_COMM_INTERVAL:
1022              break;
1023
1024          default:
1025              break;
1026      }
1027  }
1028
1029  //*************************************************************************************
      *******
1030
1031  void bms_evt_handler(nrf_ble_bms_t * p_ess, nrf_ble_bms_evt_t * p_evt)
1032  {
1033      ret_code_t err_code;
1034      switch (p_evt->evt_type)
1035      {
1036          case NRF_BLE_BMS_EVT_AUTH:
1037          {
1038              bool is_authorized = true;
1039  #if USE_AUTHORIZATION_CODE
1040              if ((p_evt->auth_code.len != m_auth_code_len) ||
1041                  (memcmp(m_auth_code, p_evt->auth_code.code, m_auth_code_len) != 0))
1042              {
1043                  is_authorized = false;
1044              }
1045  #endif
1046              err_code = nrf_ble_bms_auth_response(&m_bms, is_authorized);
1047              APP_ERROR_CHECK(err_code);
1048          } break; //NRF_BLE_BMS_EVT_AUTH
1049
1050          default:
1051          break;
1052      }
1053  }
1054
1055  //*************************************************************************************
      *******
1056
1057  uint16_t qwr_evt_handler(nrf_ble_qwr_t * p_qwr, nrf_ble_qwr_evt_t * p_evt)
1058  {
1059      return nrf_ble_bms_on_qwr_evt(&m_bms, &m_qwr, p_evt);
1060  }
1061
1062  //*************************************************************************************
      *******
1063
1064  static void delete_disconnected_bonds(void)
1065  {
1066      ret_code_t err_code;
1067      ble_conn_state_conn_handle_list_t conn_handle_list = ble_conn_state_conn_handles();
1068
1069      for (uint32_t i = 0; i < conn_handle_list.len; i++)
1070      {
1071          pm_peer_id_t peer_id;
```

```
1072            uint16_t conn_handle = conn_handle_list.conn_handles[i];
1073            bool pending         = ble_conn_state_user_flag_get(conn_handle,
                m_bms_bonds_to_delete);
1074
1075            if (pending)
1076            {
1077                err_code = pm_peer_id_get(conn_handle, &peer_id);
1078                if (err_code == NRF_SUCCESS)
1079                {
1080                    err_code = pm_peer_delete(peer_id);
1081                    APP_ERROR_CHECK(err_code);
1082                }
1083            }
1084            // Deletion is no longer pending for the bonds of this peer. Clear the flag.
1085            ble_conn_state_user_flag_set(conn_handle, m_bms_bonds_to_delete, false);
1086        }
1087    }
1088
1089    //*************************************************************************************
    *******
1090
1091    static void delete_requesting_bond(nrf_ble_bms_t const * p_bms)
1092    {
1093        ble_conn_state_user_flag_set(p_bms->conn_handle, m_bms_bonds_to_delete, true);
1094    }
1095
1096    //*************************************************************************************
    *******
1097
1098    static void delete_all_bonds(nrf_ble_bms_t const * p_bms)
1099    {
1100        ret_code_t err_code;
1101        uint16_t conn_handle;
1102
1103        pm_peer_id_t peer_id = pm_next_peer_id_get(PM_PEER_ID_INVALID);
1104        while (peer_id != PM_PEER_ID_INVALID)
1105        {
1106            err_code = pm_conn_handle_get(peer_id, &conn_handle);
1107            APP_ERROR_CHECK(err_code);
1108
1109            if (conn_handle != BLE_CONN_HANDLE_INVALID)
1110            {
1111                /* Defer the deletion since this connection is active. */
1112                ble_conn_state_user_flag_set(conn_handle, m_bms_bonds_to_delete, true);
1113            }
1114            else
1115            {
1116                err_code = pm_peer_delete(peer_id);
1117                APP_ERROR_CHECK(err_code);
1118            }
1119
1120            peer_id = pm_next_peer_id_get(peer_id);
1121        }
1122    }
1123
1124    //*************************************************************************************
    *******
1125
1126    static void services_init(void)
1127    {
1128        ret_code_t          err_code;
1129        nrf_ble_bms_init_t  bms_init;
1130        nrf_ble_cgms_init_t cgms_init;
1131        ble_dis_init_t      dis_init;
1132        ble_bas_init_t      bas_init;
1133        nrf_ble_qwr_init_t  qwr_init;
1134
1135        // Initialize Queued Write Module
1136        memset(&qwr_init, 0, sizeof(qwr_init));
```

```
1137        qwr_init.mem_buffer.len   = MEM_BUFF_SIZE;
1138        qwr_init.mem_buffer.p_mem = m_mem;
1139        qwr_init.callback         = qwr_evt_handler;
1140        qwr_init.error_handler    = nrf_qwr_error_handler;
1141
1142        err_code = nrf_ble_qwr_init(&m_qwr, &qwr_init);
1143        APP_ERROR_CHECK(err_code);
1144
1145        // Initialize Bond Management Service
1146        memset(&bms_init, 0, sizeof(bms_init));
1147
1148        m_bms_bonds_to_delete  = ble_conn_state_user_flag_acquire();
1149
1150        bms_init.evt_handler   = bms_evt_handler;
1151        bms_init.error_handler = service_error_handler;
1152
1153    #if USE_AUTHORIZATION_CODE
1154        bms_init.feature.delete_requesting_auth         = true;
1155        bms_init.feature.delete_all_auth                = true;
1156        bms_init.feature.delete_all_but_requesting_auth = false;
1157    #else
1158        bms_init.feature.delete_requesting              = true;
1159        bms_init.feature.delete_all                     = true;
1160        bms_init.feature.delete_all_but_requesting      = false;
1161    #endif
1162
1163        bms_init.bms_feature_sec_req = SEC_JUST_WORKS;
1164        bms_init.bms_ctrlpt_sec_req  = SEC_JUST_WORKS;
1165
1166        bms_init.p_qwr                              = &m_qwr;
1167        bms_init.bond_callbacks.delete_requesting = delete_requesting_bond;
1168        bms_init.bond_callbacks.delete_all        = delete_all_bonds;
1169
1170        err_code = nrf_ble_bms_init(&m_bms, &bms_init);
1171        APP_ERROR_CHECK(err_code);
1172
1173        // Initialize Glucose Service - sample selection of feature bits
1174        m_cgms.comm_interval = GLUCOSE_MEAS_INTERVAL; // Kg co thi kg chay
1175        memset(&cgms_init, 0, sizeof(cgms_init));
1176        cgms_init.evt_handler   = cgms_evt_handler;
1177        cgms_init.error_handler = service_error_handler;
1178        cgms_init.p_gatt_queue  = &m_ble_gatt_queue;
1179        cgms_init.initial_run_time = 20;
1180        cgms_init.initial_sensor_status.time_offset    = 0x00;
1181        cgms_init.initial_sensor_status.status.status |= NRF_BLE_CGMS_STATUS_SESSION_STOPPED;
1182        err_code = nrf_ble_cgms_init(&m_cgms, &cgms_init);
1183        APP_ERROR_CHECK(err_code);
1184
1185        //add a basic measurement with only mandatory fields
1186
1187        // Initialize Battery Service
1188        memset(&bas_init, 0, sizeof(bas_init));
1189
1190        // Here the sec level for the Battery Service can be changed/increased.
1191        bas_init.bl_rd_sec        = SEC_OPEN;
1192        bas_init.bl_cccd_wr_sec   = SEC_OPEN;
1193        bas_init.bl_report_rd_sec = SEC_OPEN;
1194
1195        bas_init.evt_handler          = NULL;
1196        bas_init.support_notification = true;
1197        bas_init.p_report_ref         = NULL;
1198        bas_init.initial_batt_level   = 100;
1199
1200        err_code = ble_bas_init(&m_bas, &bas_init);
1201        APP_ERROR_CHECK(err_code);
1202
1203        // Initialize Device Information Service.
1204        memset(&dis_init, 0, sizeof(dis_init));
1205
```

```
1206        ble_srv_ascii_to_utf8(&dis_init.manufact_name_str, MANUFACTURER_NAME);
1207
1208        dis_init.dis_char_rd_sec = SEC_OPEN;
1209
1210        err_code = ble_dis_init(&dis_init);
1211        APP_ERROR_CHECK(err_code);
1212    }
1213
1214    //*************************************************************************************
    *******
1215
1216    static void application_timers_start(void)
1217    {
1218        ret_code_t err_code;
1219
1220        // Start application timers.
1221        err_code = app_timer_start(m_battery_timer_id, BATTERY_LEVEL_MEAS_INTERVAL, NULL);
1222        APP_ERROR_CHECK(err_code);
1223
1224        // Bat dau chay thoi gian dinh thoi kiem tra muc pin
1225        err_code = app_timer_start(batt_level_timer_id, DINH_THOI_KTRA_PIN, NULL);
1226        APP_ERROR_CHECK(err_code);
1227    }
1228
1229    //*************************************************************************************
    *******
1230
1231    /**@brief Function for handling the Connection Parameter events.
1232     *
1233     * @details This function will be called for all events in the Connection Parameters
     Module which
1234     *          are passed to the application.
1235     *          @note All this function does is to disconnect. This could have been done by
     simply
1236     *                setting the disconnect_on_fail configuration parameter, but instead
     we use the
1237     *                event handler mechanism to demonstrate its use.
1238     *
1239     * @param[in]   p_evt   Event received from the Connection Parameters Module.
1240     */
1241    static void on_conn_params_evt(ble_conn_params_evt_t * p_evt)
1242    {
1243        ret_code_t err_code;
1244
1245        if (p_evt->evt_type == BLE_CONN_PARAMS_EVT_FAILED)
1246        {
1247            err_code = sd_ble_gap_disconnect(m_conn_handle,
            BLE_HCI_CONN_INTERVAL_UNACCEPTABLE);
1248            APP_ERROR_CHECK(err_code);
1249        }
1250    }
1251
1252    //*************************************************************************************
    *******
1253
1254    static void conn_params_init(void)
1255    {
1256        ret_code_t             err_code;
1257        ble_conn_params_init_t cp_init;
1258        memset(&cp_init, 0, sizeof(cp_init));
1259        cp_init.p_conn_params                  = NULL;
1260        cp_init.first_conn_params_update_delay = FIRST_CONN_PARAMS_UPDATE_DELAY;
1261        cp_init.next_conn_params_update_delay  = NEXT_CONN_PARAMS_UPDATE_DELAY;
1262        cp_init.max_conn_params_update_count   = MAX_CONN_PARAMS_UPDATE_COUNT;
1263        cp_init.start_on_notify_cccd_handle    = BLE_GATT_HANDLE_INVALID;
1264        cp_init.disconnect_on_fail             = false;
1265        cp_init.evt_handler                    = on_conn_params_evt;
1266        cp_init.error_handler                  = conn_params_error_handler;
1267        err_code = ble_conn_params_init(&cp_init);  // Sets the connection parameters based
```

```
        on the initialized cp_init variable.
1268                                                    // These are used by the Peripheral to
                                                        request an update to the Connection
                                                        Parameters from the Central
1269        APP_ERROR_CHECK(err_code);
1270    }
1271
1272    //*****************************************************************************
        *******
1273
1274    static void on_adv_evt(ble_adv_evt_t ble_adv_evt)
1275    {
1276        ret_code_t err_code;
1277        switch (ble_adv_evt)
1278        {
1279            case BLE_ADV_EVT_DIRECTED_HIGH_DUTY:
1280                err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING_DIRECTED);
1281                APP_ERROR_CHECK(err_code);
1282                break;
1283
1284            case BLE_ADV_EVT_FAST:
1285                err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING);
1286                APP_ERROR_CHECK(err_code);
1287                break;
1288
1289            case BLE_ADV_EVT_SLOW:
1290                err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING_SLOW);
1291                APP_ERROR_CHECK(err_code);
1292                break;
1293
1294            case BLE_ADV_EVT_FAST_WHITELIST:
1295                err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING_WHITELIST);
1296                APP_ERROR_CHECK(err_code);
1297                break;
1298
1299            case BLE_ADV_EVT_SLOW_WHITELIST:
1300                err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING_WHITELIST);
1301                APP_ERROR_CHECK(err_code);
1302                break;
1303
1304            case BLE_ADV_EVT_IDLE:
1305                break;
1306
1307            case BLE_ADV_EVT_WHITELIST_REQUEST:
1308             {
1309                ble_gap_addr_t whitelist_addrs[BLE_GAP_WHITELIST_ADDR_MAX_COUNT];
1310                ble_gap_irk_t  whitelist_irks[BLE_GAP_WHITELIST_ADDR_MAX_COUNT];
1311                uint32_t addr_cnt = BLE_GAP_WHITELIST_ADDR_MAX_COUNT;
1312                uint32_t irk_cnt  = BLE_GAP_WHITELIST_ADDR_MAX_COUNT;
1313
1314                err_code = pm_whitelist_get(whitelist_addrs, &addr_cnt, whitelist_irks,
                    &irk_cnt);
1315                APP_ERROR_CHECK(err_code);
1316
1317                // Set the correct identities list (no excluding peers with no Central
                    Address Resolution).
1318                identities_set(PM_PEER_ID_LIST_SKIP_NO_IRK);
1319
1320                // Apply the whitelist.
1321                err_code = ble_advertising_whitelist_reply(&m_advertising, whitelist_addrs,
                    addr_cnt, whitelist_irks, irk_cnt);
1322                APP_ERROR_CHECK(err_code);
1323            } break;
1324
1325            case BLE_ADV_EVT_PEER_ADDR_REQUEST:
1326            {
1327                pm_peer_data_bonding_t peer_bonding_data;
1328
1329                // Only Give peer address if we have a handle to the bonded peer.
```

```
1330                pm_peer_id_t peer_id = PM_PEER_ID_INVALID;
1331                err_code = pm_peer_id_get(m_conn_handle , &peer_id);
1332                APP_ERROR_CHECK(err_code);
1333
1334                if (peer_id != PM_PEER_ID_INVALID)
1335                {
1336                    err_code = pm_peer_data_bonding_load(peer_id, &peer_bonding_data);
1337                    if (err_code != NRF_ERROR_NOT_FOUND)
1338                    {
1339                        APP_ERROR_CHECK(err_code);
1340
1341                        // Manipulate identities to exclude peers with no Central Address
                            Resolution.
1342                        identities_set(PM_PEER_ID_LIST_SKIP_ALL);
1343
1344                        ble_gap_addr_t * p_peer_addr =
                            &(peer_bonding_data.peer_ble_id.id_addr_info);
1345                        err_code = ble_advertising_peer_addr_reply(&m_advertising,
                            p_peer_addr);
1346                        APP_ERROR_CHECK(err_code);
1347                    }
1348                }
1349            } break; //BLE_ADV_EVT_PEER_ADDR_REQUEST
1350
1351            default:
1352                break;
1353        }
1354    }
1355
1356    //*************************************************************************************
    *******
1357
1358    static void ble_evt_handler(ble_evt_t const * p_ble_evt, void * p_context)
1359    {
1360        ret_code_t err_code = NRF_SUCCESS;
1361
1362        switch (p_ble_evt->header.evt_id)
1363        {
1364            case BLE_GAP_EVT_CONNECTED:
1365                knoiApp=1;  // Da ket noi duoc voi app
1366                while(1)
1367                {
1368                    err_code = app_timer_start(em4095_timer_id, TGIAN_TAT_EM4095, NULL); //
                        Bat dau chay t.gian Stop/Run Em4095
1369                    if (err_code == NRF_SUCCESS)
1370                        break;
1371                }
1372                NRF_P0->OUTSET = (1<<PinDieukhien5V); // Bat nguon 5V
1373
1374                err_code = bsp_indication_set(BSP_INDICATE_CONNECTED);
1375                APP_ERROR_CHECK(err_code);
1376                m_conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
1377                err_code = nrf_ble_bms_set_conn_handle(&m_bms, m_conn_handle);
1378                APP_ERROR_CHECK(err_code);
1379                err_code = nrf_ble_qwr_conn_handle_assign(&m_qwr, m_conn_handle);
1380                APP_ERROR_CHECK(err_code);
1381                err_code = nrf_ble_cgms_conn_handle_assign(&m_cgms, m_conn_handle);
1382                APP_ERROR_CHECK(err_code);
1383                break;
1384
1385            case BLE_GAP_EVT_DISCONNECTED:
1386                knoiApp=0;  // Da ngat ket noi voi app
1387                while(1)
1388                {
1389                    err_code = app_timer_stop(em4095_timer_id);  // Ngung chay EM4095
1390                    if (err_code == NRF_SUCCESS)
1391                        break;
1392                }
1393                TatNgoaiviQuetthe();
```

```
1394                TatPWM0();
1395                NRF_P0->OUTCLR = (1<<PinDieukhien5V); // Tat nguon 5V
1396
1397                if (delete_bonds_pending())
1398                {
1399                    // Advertising is started by PM_EVT_PEERS_DELETE_SUCCEEDED or
                        PM_EVT_PEERS_DELETE_SUCCEEDED event.
1400                    delete_disconnected_bonds();
1401                }
1402                else
1403                {
1404                    advertising_start(false);
1405                }
1406                m_conn_handle = BLE_CONN_HANDLE_INVALID;
1407                break;
1408
1409            case BLE_GAP_EVT_PHY_UPDATE_REQUEST:
1410            {
1411                ble_gap_phys_t const phys =
1412                {
1413                    .rx_phys = BLE_GAP_PHY_AUTO,
1414                    .tx_phys = BLE_GAP_PHY_AUTO,
1415                };
1416                err_code = sd_ble_gap_phy_update(p_ble_evt->evt.gap_evt.conn_handle, &phys);
1417                APP_ERROR_CHECK(err_code);
1418            } break;
1419
1420            case BLE_GATTC_EVT_TIMEOUT:
1421                // Disconnect on GATT Client timeout event.
1422                err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gattc_evt.conn_handle,
1423                                                 BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
1424                APP_ERROR_CHECK(err_code);
1425                break;
1426
1427            case BLE_GATTS_EVT_TIMEOUT:
1428                // Disconnect on GATT Server timeout event.
1429                err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gatts_evt.conn_handle,
1430                                                 BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
1431                APP_ERROR_CHECK(err_code);
1432                break;
1433
1434            default:
1435                // No implementation needed.
1436                break;
1437        }
1438    }
1439
1440    //*************************************************************************************
        *******
1441
1442    static void ble_stack_init(void)
1443    {
1444        ret_code_t err_code;
1445
1446        err_code = nrf_sdh_enable_request();
1447        APP_ERROR_CHECK(err_code);
1448
1449        // Configure the BLE stack using the default settings.
1450        // Fetch the start address of the application RAM.
1451        uint32_t ram_start = 0;
1452        err_code = nrf_sdh_ble_default_cfg_set(APP_BLE_CONN_CFG_TAG, &ram_start);
1453        APP_ERROR_CHECK(err_code);
1454
1455        // Enable BLE stack.
1456        err_code = nrf_sdh_ble_enable(&ram_start);
1457        APP_ERROR_CHECK(err_code);
1458
1459        // Register a handler for BLE events.
1460        NRF_SDH_BLE_OBSERVER(m_ble_observer, APP_BLE_OBSERVER_PRIO, ble_evt_handler, NULL);
```

```
1461    }
1462
1463    //**************************************************************************************
        *******
1464
1465    static void bsp_event_handler(bsp_event_t event)
1466    {
1467        ret_code_t err_code;
1468        switch (event)
1469        {
1470            case BSP_EVENT_SLEEP:
1471                break;
1472
1473            case BSP_EVENT_DISCONNECT:
1474                err_code = sd_ble_gap_disconnect(m_conn_handle,
                    BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
1475                if (err_code != NRF_ERROR_INVALID_STATE)
1476                {
1477                    APP_ERROR_CHECK(err_code);
1478                }
1479                break;
1480
1481            case BSP_EVENT_WHITELIST_OFF:
1482                if (m_conn_handle == BLE_CONN_HANDLE_INVALID)
1483                {
1484                    err_code = ble_advertising_restart_without_whitelist(&m_advertising);
1485                    if (err_code != NRF_ERROR_INVALID_STATE)
1486                    {
1487                        APP_ERROR_CHECK(err_code);
1488                    }
1489                }
1490                break;
1491
1492            case BSP_EVENT_KEY_2: // Nut nhan 3
1493                // Luc binh thuong nhan button 3 de xoa bond
1494                delete_bonds();
1495                break;
1496
1497            case BSP_EVENT_KEY_3: // Nut nhan 4
1498                // Bat dau gui du lieu len app
1499                m_current_offset++; // La thong so status
1500                if(m_current_offset>0xFFFE)
1501                    m_current_offset=0;
1502
1503                read_glucose_measurement();
1504                NRF_P0->OUTSET = (1<<BT_HANDLE_PIN);  // BT_HANDLE_PIN kich hoat xong thi
                    tro ve trang thai ban dau
1505                NRF_P0->OUTSET = LED3;  // Huy kich hoat
1506                TatPWM0();
1507                break;
1508
1509            default:
1510                break;
1511        }
1512    }
1513
1514    //**************************************************************************************
        *******
1515
1516    static void pm_evt_handler(pm_evt_t const * p_evt)
1517    {
1518        pm_handler_on_pm_evt(p_evt);
1519        pm_handler_disconnect_on_sec_failure(p_evt);
1520        pm_handler_flash_clean(p_evt);
1521
1522        switch (p_evt->evt_id)
1523        {
1524            case PM_EVT_PEER_DELETE_SUCCEEDED:
1525                if (!delete_bonds_pending() && !delete_all_pending)
```

```
1526                {
1527                    // No more peers are flagged for deletion and we are not going to
                        delete all peers.
1528                    advertising_start(false);
1529                }
1530                break;
1531
1532            case PM_EVT_PEERS_DELETE_SUCCEEDED:
1533                delete_all_pending = false;
1534                advertising_start(false);
1535                break;
1536
1537            case PM_EVT_PEER_DATA_UPDATE_SUCCEEDED:
1538                if (     p_evt->params.peer_data_update_succeeded.flash_changed
1539                     && (p_evt->params.peer_data_update_succeeded.data_id ==
                         PM_PEER_DATA_ID_BONDING))
1540                {
1541                    // Note: You should check on what kind of white list policy your
                        application should use.
1542
1543                    whitelist_set(PM_PEER_ID_LIST_SKIP_NO_ID_ADDR);
1544                }
1545                break;
1546
1547            default:
1548                break;
1549        }
1550    }
1551
1552    //*****************************************************************************
    *******
1553
1554    static void peer_manager_init(void)
1555    {
1556        ble_gap_sec_params_t sec_param;
1557        ret_code_t err_code;
1558
1559        err_code = pm_init(); // initializes the Peer Manager module.
1560        APP_ERROR_CHECK(err_code);
1561
1562        memset(&sec_param, 0, sizeof(ble_gap_sec_params_t));
1563
1564        // Security parameters to be used for all security procedures.
1565        sec_param.bond           = 1;   // Perform bonding
1566        sec_param.mitm           = 0;   // Man In The Middle protection not required
1567        sec_param.lesc           = 0;   // LE Secure Connections not enabled
1568        sec_param.keypress       = 0;   // Keypress notifications not enabled
1569        sec_param.io_caps        = BLE_GAP_IO_CAPS_NONE; // No I/O capabilities
1570        sec_param.oob            = 0;   // Out Of Band data not available
1571        sec_param.min_key_size   = 7;   // Minimum encryption key size
1572        sec_param.max_key_size   = 16;  // Maximum encryption key size
1573        sec_param.kdist_own.enc  = 1;
1574        sec_param.kdist_own.id   = 1;
1575        sec_param.kdist_peer.enc = 1;
1576        sec_param.kdist_peer.id  = 1;
1577
1578        err_code = pm_sec_params_set(&sec_param); // sets the Security Requirements for the
        connection.
1579        APP_ERROR_CHECK(err_code);
1580
1581        err_code = pm_register(pm_evt_handler);
1582        APP_ERROR_CHECK(err_code);
1583    }
1584
1585    //*****************************************************************************
    *******
1586
1587    static void advertising_init(void)
1588    {
```

```
1589        ret_code_t             err_code;
1590        uint8_t                adv_flags;
1591        ble_advertising_init_t init;
1592
1593        memset(&init, 0, sizeof(init));
1594
1595        adv_flags                                           =
            BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;
1596        init.advdata.name_type                              = BLE_ADVDATA_FULL_NAME;
1597        init.advdata.include_appearance                     = true;
1598        init.advdata.flags                                  = adv_flags;
1599        init.advdata.uuids_complete.uuid_cnt                = sizeof(m_adv_uuids) /
            sizeof(m_adv_uuids[0]);
1600        init.advdata.uuids_complete.p_uuids                 = m_adv_uuids;
1601
1602        init.config.ble_adv_on_disconnect_disabled     = true;
1603        init.config.ble_adv_whitelist_enabled          = true;
1604        init.config.ble_adv_directed_high_duty_enabled = true;
1605        init.config.ble_adv_directed_enabled           = false;
1606        init.config.ble_adv_directed_interval          = 0;
1607        init.config.ble_adv_directed_timeout           = 0;
1608        init.config.ble_adv_fast_enabled               = true;
1609        init.config.ble_adv_fast_interval              = APP_ADV_FAST_INTERVAL;
1610        init.config.ble_adv_fast_timeout               = APP_ADV_FAST_DURATION;
1611        init.config.ble_adv_slow_enabled               = true;
1612        init.config.ble_adv_slow_interval              = APP_ADV_SLOW_INTERVAL;
1613        init.config.ble_adv_slow_timeout               = APP_ADV_SLOW_DURATION;
1614
1615        init.evt_handler   = on_adv_evt;
1616        init.error_handler = ble_advertising_error_handler;
1617
1618        err_code = ble_advertising_init(&m_advertising, &init);
1619        APP_ERROR_CHECK(err_code);
1620
1621        ble_advertising_conn_cfg_tag_set(&m_advertising, APP_BLE_CONN_CFG_TAG);
1622    }
1623
1624    //**************************************************************************************
    *******
1625
1626    static void buttons_leds_init(bool * p_erase_bonds)
1627    {
1628        ret_code_t err_code;
1629        bsp_event_t startup_event;
1630
1631        err_code = bsp_init(BSP_INIT_LEDS | BSP_INIT_BUTTONS, bsp_event_handler);
1632        APP_ERROR_CHECK(err_code);
1633
1634        err_code = bsp_btn_ble_init(NULL, &startup_event);
1635        APP_ERROR_CHECK(err_code);
1636
1637        *p_erase_bonds = (startup_event == BSP_EVENT_CLEAR_BONDING_DATA);
1638    }
1639
1640    //**************************************************************************************
    *******
1641
1642    static void power_management_init(void)
1643    {
1644        ret_code_t err_code;
1645        err_code = nrf_pwr_mgmt_init();
1646        APP_ERROR_CHECK(err_code);
1647    }
1648
1649    //**************************************************************************************
    *******
1650
1651    int main(void)
1652    {
```

```
1653        bool erase_bonds;
1654
1655        // Initialize.
1656        timers_init();  // Dang ky cac ham tao timer
1657        buttons_leds_init(&erase_bonds);  // Dang ky cho "bsp_event_handler"
1658        power_management_init();
1659        ble_stack_init(); // Dang ky cho "ble_evt_handler"
1660        gap_params_init();
1661        gatt_init();
1662        advertising_init();
1663        services_init();
1664        sensorsim_init(&m_battery_sim_state, &m_battery_sim_cfg);
1665        conn_params_init();
1666        peer_manager_init();
1667        application_timers_start(); // Bat dau chay cac ham timer
1668        advertising_start(erase_bonds);
1669
1670
            //.....................................................................
            ...
1671
1672        // Cai dat Timer1, dung de dem thoi gian muc dich chung
1673        NRF_TIMER1->TASKS_STOP      = 1;
1674        NRF_TIMER1->CC[0]       = 500;      // Moi lan tran la 1ms
1675        NRF_TIMER1->SHORTS      = 1;        // Enable shortcut between COMPARE[0] event and
            CLEAR task
1676        NRF_TIMER1->MODE        = 0;        // Che do Timer
1677        NRF_TIMER1->BITMODE     = 3;        // 32 bit timer bit width
1678        NRF_TIMER1->PRESCALER       = 5;        // Tu 0 den 9
1679        NRF_TIMER1->TASKS_CLEAR     = 1;
1680        NRF_TIMER1->INTENSET        = (1<<16);  // Enable interrupt for COMPARE[0] event
1681        NVIC_SetPriority(TIMER1_IRQn, 6);
1682        NVIC_ClearPendingIRQ(TIMER1_IRQn);
1683        NVIC_EnableIRQ(TIMER1_IRQn);            // Kg co lenh nay thi kg chay
1684        NRF_TIMER1->TASKS_START     = 1;
1685        /* Dung loai ham nay khong chinh xac 1ms, ma con lam cho giai ma chay kg tot
1686        static void tMs_timeout_handler(void * p_context)
1687        {
1688            ++tMs;
1689        } */
1690
1691
            //.....................................................................
            ...
1692
1693        // Cai dat chan dieu khien kich hoat gui du lieu qua BLE
1694        NRF_P0->PIN_CNF[BT_HANDLE_PIN] = (1    | // Output
1695                                          (0<<2)| // No pull
1696                                          (0<<1)| // Connect input buffer
1697                                          (3<<8));// High drive '0', high drive '1'
1698        NRF_P0->OUTSET = (1<<BT_HANDLE_PIN);     // Binh thuong BT_HANDLE_PIN phai o muc 1,
            no kich hoat muc 0.
1699
1700
            //.....................................................................
            ...
1701
1702        // Cai dat chan dieu khien EM4095
1703        NRF_P0->PIN_CNF[ACTIVE_EM_PIN] = (1    | // Output
1704                                          (0<<2)| // No pull
1705                                          (0<<1)| // Connect input buffer
1706                                          (3<<8));// High drive '0', high drive '1'
1707        NRF_P0->OUTCLR = (1<<ACTIVE_EM_PIN);     // Binh thuong ACTIVE_EM_PIN phai o muc
            0-de EM4095 kg chay.
1708
1709
            //.....................................................................
            ...
1710
```

```
1711        // Cai dat chan dieu khien buzzer
1712        NRF_P0->PIN_CNF[BUZZER] = (1    | // Output
1713                                   (0<<2)| // No pull
1714                                   (0<<1)| // Connect input buffer
1715                                   (3<<8));// High drive '0', high drive '1'
1716        NRF_P0->OUTCLR = (1<<BUZZER);     // Cung la trang thai cua chan khi tat PWM
1717
1718
            //.....................................................................
            ...
1719
1720        // Cai dat chan dieu khien 5V
1721        NRF_P0->PIN_CNF[PinDieukhien5V] = (1    | // Output
1722                                           (0<<2)| // No pull
1723                                           (0<<1)| // Connect input buffer
1724                                           (3<<8));// High drive '0', high drive '1'
1725        NRF_P0->OUTCLR = (1<<PinDieukhien5V);
1726
1727
            //.....................................................................
            ...
1728
1729        // Cai dat chan lay mau tin hieu RFID data
1730        NRF_P0->PIN_CNF[RFID_DTA_PIN1] = ((0<<2)|   // No pull
1731                                          (0<<1)|   // Connect input buffer
1732                                          (3<<8));  // High drive '0', high drive '1'
1733
            //.....................................................................
            ...
1734
1735        // Cai dat chan TestPoint_24 de debug
1736        NRF_P0->PIN_CNF[TEST_POINT_P24] = (1    | // Output
1737                                           (0<<2)| // No pull
1738                                           (0<<1)| // Connect input buffer
1739                                           (3<<8));// High drive '0', high drive '1'
1740        NRF_P0->OUTCLR = (1<<TEST_POINT_P24);
1741
1742
            //.....................................................................
            ...
1743
1744        // Cai dat chan TestPoint_26 de debug
1745        NRF_P0->PIN_CNF[TEST_POINT_P26] = (1    | // Output
1746                                           (0<<2)| // No pull
1747                                           (0<<1)| // Connect input buffer
1748                                           (3<<8));// High drive '0', high drive '1'
1749        NRF_P0->OUTCLR = (1<<TEST_POINT_P26);
1750
1751
            //.....................................................................
            ...
1752
1753        // Keu coi bao hieu khoi dong
1754        BatPWM0();
1755
1756
            //.....................................................................
            ...
1757
1758        for (;;)
1759        {
1760            if(knoiApp==0)
1761            {
1762                // Tu dong System On Sleep
1763                // Ham nay xu ly mat 1ms, nen quet the kg nhay
1764                nrf_pwr_mgmt_run();
1765            }
1766            else if((knoiApp==0x11) && GiaiMaEm4102()) // Kg co ketnoi app thi bo qua cho
                khoi ton pin.
```

```
1767            {
1768                if (tChongLapThe==0 || em4102Sub[0]!=em4102[0] || em4102Sub[1]!=em4102[1]
1769                || em4102Sub[2]!=em4102[2] || em4102Sub[3]!=em4102[3] ||
                    em4102Sub[4]!=em4102[4])
1770                {
1771                    BatPWM0();  // Coi keu bao hieu da quet duoc the
1772                    NRF_P0->OUTCLR = (1<<BT_HANDLE_PIN);  // Kich hoat gui ma the len app.
1773                    NRF_P0->OUTCLR = LED3;  // Nhay den LED3 de bao hieu.
1774                                            // Sau khi gui xong thi ham con khac se huy
                                                kich hoat LED3.
1775                                            // Do bang Oscilo thay thong thuong kich hoat
                                                trong 74ms (cung kha lau).
1776
1777                    //Dat lai bien em4102Sub
1778                    for (bienChay=0; bienChay<SO_BYTES_TRONG_MASOTHE; bienChay++)
1779                    {
1780                        em4102Sub[bienChay]=em4102[bienChay];
1781                    }
1782                    tChongLapThe=tMs;
1783                }
1784            }
1785            // Het thoi gian chong lap the, cho quet the tro lai
1786            if (tChongLapThe && (tMs-tChongLapThe >= TGIAN_CHONGLAPTHE))
1787            {
1788                for (bienChay=0; bienChay<SO_BYTES_TRONG_MASOTHE; bienChay++)
1789                {
1790                    em4102Sub[bienChay]=0;  //Reset lai gia tri cho bien em4102Sub.
1791                }
1792                tChongLapThe=0;
1793            }
1794            // Tat coi bao hieu khoi dong
1795            if ((!tCoikeu) && (tMs-tCoikeu >= TGIAN_COIKEU)) // Chi cho chay vao day 1 lan
            duy nhat khi khoi dong
1796            {
1797                tCoikeu=1;  // Mien khac 0 la duoc
1798                TatPWM0();
1799            }
1800        }
1801    }
1802
```