ANALOG DEVICES

ADI STUDY WATCH DCB DESIGN

ANALOG DEVICES, INC.

www.analog.com

REV 1.0.0,MAR 2021

Table of Contents

List of Figures

List of Tables

No table of figures entries found.

Copyright, Disclaimer & Trademark Statements

Copyright Information

Disclaimer

Trademark and Service Mark Notice

1 Device Configuration Block (DCB)

Device configuration block (DCB) in study watch is based on the flash data storage(FDS) library provided by nRF5 SDK. FDS provides the record oriented file system for on-chip NOR flash where files are stored as a collection of records of variable length (for more details on FDS refer nRF FDS).

In the study watch, we have specified 5 Virtual pages each of 1024*4 bytes size for FDS in "sdk_config.h". Out of 5 pages, DCB and RTC uses 1 page each and for garbage collection 1 page is reserved.

We have one DCB file (ADI_DCB_FILE) with following file ID-(0xADCB) defined in "adi_dcb_config.h" occupying 1 virtual page of 1024*4 bytes size. The DCB Block for each of the sensors and bio-medical applications are basically the FDS records of different sizes present in the ADI_DCB_FILE. We can access each of these DCB blocks/records by specifying its corresponding RECORD_KEY.

In "dcb_interface.h" file, a structure(M2M2_DCB_CONFIG_BLOCK_INDEX_t) is maintained that stores all the DCB Block Index (record keys).

Each of the DCB blocks can be used for storing any sensor's device configuration (DCFG) or for any bio-medical applications, to store its library configuration(LCFG). For example - ADXL DCB block is 1 record with its corresponding record_key = ADI_DCB_ADXL362_BLOCK_IDX present in ADI_DCB_FILE.

Below are the four wrapper functions to the FDS driver layer functions that are present in "adi_dcb_config.c" file. To read/write/delete a particular DCB block, a wrapper function has to be called to below mentioned functions by specifying the corresponding DCB block index (record key). To check if data is present in DCB block or not, the API adi_dcb_check_fds_entry(record_key) should be called.

uint8_t adi_dcb_write_to_fds(const uint16_t ***rcrd_key****, uint32_t *wd_dcb_data, uint16_t len_DWORD);*

uint8_t adi_dcb_read_from_fds(const uint16_t ***rcrd_key****, uint32_t *rd_dcb_data, uint16_t *len_DWORD);*

uint8_t adi_dcb_delete_fds_settings(uint16_t ***dcb_rec_key****);*

uint8_t adi_dcb_check_fds_entry(uint16_t ***dcb_rec_key****);*

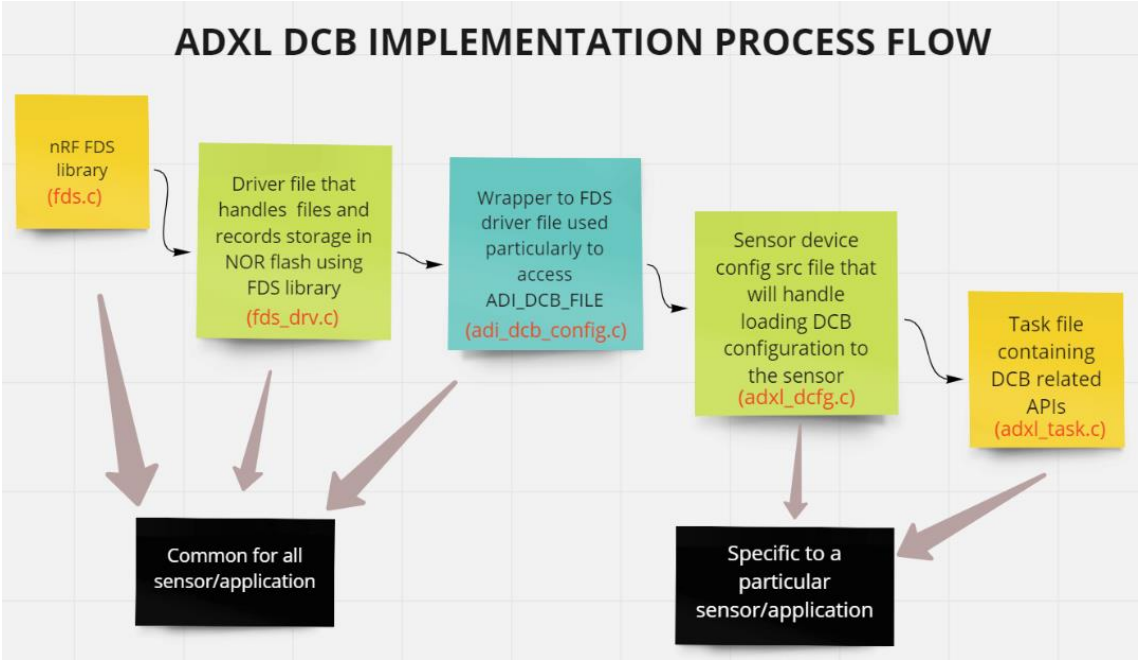

Figure 1: ADXL DCB Flow

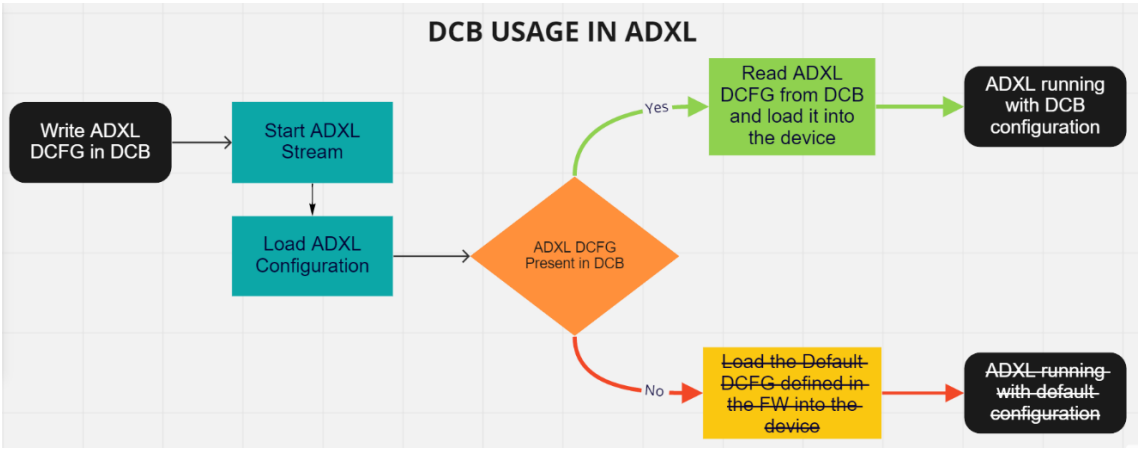

Figure 2: DCB usage in ADXL

2 Adding new DCB

The following steps are to be used to add DCB for a new sensor or application.

1. Add BLOCK_IDX(record key) for the new sensor/application in "dcb_interface.h" file.

```
typedef enum M2M2_DCB_CONFIG_BLOCK_INDEX_t
{
   ADI_DCB_GENERAL_BLOCK_IDX = 0,
   ADI_DCB_AD5940_BLOCK_IDX,
   ADI_DCB_ADPD4000_BLOCK_IDX,
   ADI_DCB_ADXL362_BLOCK_IDX,
   ADI_DCB_PPG_BLOCK_IDX,
.................
   ADI_DCB_USER3_BLOCK_IDX,
   ADI_DCB_BCM_BLOCK_IDX,
   ADI_DCB_NEW_SENSOR_BLOCK_IDX,
   ADI_DCB_MAX_BLOCK_IDX
} M2M2_DCB_CONFIG_BLOCK_INDEX_t;
```

2. In the new sensor device configuration src file(generally the file that contains functions for loading the sensor's DCFG for ex- "adxl_dcfg.c" file in case of adxl), Add wrapper functions for below mentioned functions present in "adi_dcb_config.c" file.

   - *uint8_t adi_dcb_write_to_fds(ADI_DCB_NEW_SENSOR_BLOCK_IDX, uint32_t *wd_dcb_data, uint16_t len_DWORD);*
   - *uint8_t adi_dcb_read_from_fds(ADI_DCB_NEW_SENSOR_BLOCK_IDX, uint32_t *rd_dcb_data, uint16_t *len_DWORD);*
   - *uint8_t adi_dcb_delete_fds_settings(ADI_DCB_NEW_SENSOR_BLOCK_IDX);*
   - *uint8_t adi_dcb_check_fds_entry(ADI_DCB_NEW_SENSOR_BLOCK_IDX);*

   As can be seen above, in all these functions, the corresponding block index has to be specified for that new sensor/application that is defined in Step-1.

   Note: In case of any bio-medical application, a wrapper functions to it in the application's middleware file can be defined (for ex- for PPG, the wrapper is "mw_ppg.c" file).

   For example- In "adxl_dcfg.c" file, following are the wrapper functions that are added to read/write/delete/get_dcb_present status for adxl DCB.

```
315 ADXL_DCB_STATUS_t read_adxl_dcb(uint32_t *adxl_dcb_data, uint16_t* read_size)
316 {
317     ADXL_DCB_STATUS_t dcb_status = ADXL_DCB_STATUS_ERR;
318 
319     if(adi_dcb_read_from_fds(ADI_DCB_ADXL362_BLOCK_IDX, adxl_dcb_data, read_size) == DEF_OK)
320     {
321         dcb_status = ADXL_DCB_STATUS_OK;
322     }
323     return dcb_status;
324 }
```

```
331 ADXL_DCB_STATUS_t write_adxl_dcb(uint32_t *adxl_dcb_data, uint16_t write_Size)
332 {
333     ADXL_DCB_STATUS_t dcb_status = ADXL_DCB_STATUS_ERR;
334 
335     if(adi_dcb_write_to_fds(ADI_DCB_ADXL362_BLOCK_IDX, adxl_dcb_data, write_Size) == DEF_OK)
336     {
337         dcb_status = ADXL_DCB_STATUS_OK;
338     }
339 
340     return dcb_status;
341 }
```

```
348 ADXL_DCB_STATUS_t delete_adxl_dcb(void)
349 {
350     ADXL_DCB_STATUS_t dcb_status = ADXL_DCB_STATUS_ERR;
351 
352     if(adi_dcb_delete_fds_settings(ADI_DCB_ADXL362_BLOCK_IDX) == DEF_OK)
353     {
354         dcb_status = ADXL_DCB_STATUS_OK;
355     }
356 
357     return dcb_status;
358 }
```

```
365 void adxl_set_dcb_present_flag(bool set_flag)
366 {
367     g_adxl_dcb_Present = set_flag;
368     NRF_LOG_INFO("Setting..ADXL DCB present: %s",(g_adxl_dcb_Present == true ? "TRUE" : "FALSE"));
369 }
370 
371 bool adxl_get_dcb_present_flag(void)
372 {
373     NRF_LOG_INFO("ADXL DCB present: %s", (g_adxl_dcb_Present == true ? "TRUE" : "FALSE"));
374     return g_adxl_dcb_Present;
375 }
```

```
377 void adxl_update_dcb_present_flag(void)
378 {
379     g_adxl_dcb_Present = adi_dcb_check_fds_entry(ADI_DCB_ADXL362_BLOCK_IDX);
380     NRF_LOG_INFO("Updated. ADXL DCB present: %s", (g_adxl_dcb_Present == true ? "TRUE" : "FALSE"));
381 }
```

3. Configuration stored in DCB must be given preference over the default configuration. So changes have to be added in the function that loads the device configuration (DCFG) or library configuration (LCFG) such that whenever the configuration has to be loaded, first it should check if DCB config is present or not. If present, then read the configuration from DCB and load it, otherwise load the default configuration.

   For example - In case of ADXL, the "load_adxl_cfg()" function is present in "adxl_dcfg.c" file. As seen at line#100, a check is done for whether adxl dcb is present or not. If present, then "load_adxl_dcb()" function is called, which will read the config the from adxl dcb and load it to the device, otherwise "load_adxl_dcfg" is called, that will load the default adxl config.

```
93 ADXL_DCB_STATUS_t load_adxl_cfg(uint16_t device_id)
94 {
95     ADXL_DCB_STATUS_t adxl_cfg_status = ADXL_DCB_STATUS_ERR;
96     ADXL_DCFG_STATUS_t ret;
97 #ifdef DCB
98     bool dcb_cfg = false;
99 
100    dcb_cfg = adxl_get_dcb_present_flag();
101    if(dcb_cfg == true)
102    {
103        //Load dcb Settings
104        adxl_cfg_status = load_adxl_dcb(device_id);
105        if(adxl_cfg_status != ADXL_DCB_STATUS_OK)
106        {
107            NRF_LOG_INFO("Failed in Loading adxl DCB cfg");
108        }
109        NRF_LOG_INFO("Load adxl DCB cfg");
110    }
111    else
112    {
113 #endif
114        //Load dcfg Settings
115        ret = load_adxl_dcfg(device_id);
116        adxl_cfg_status= (!ret) ? ADXL_DCB_STATUS_OK : ADXL_DCB_STATUS_ERR;
117        if(adxl_cfg_status != ADXL_DCB_STATUS_OK)
118        {
119            NRF_LOG_INFO("Failed in Loading adxl Default f/w cfg");
120        }
121        NRF_LOG_INFO("Load adxl Default f/w cfg");
122 
123 #ifdef DCB
124    }
125 #endif
126    return adxl_cfg_status;
127 }
```

4. Next step is to add the m2m handler function for DCB in the sensor/app task file. Externally from CLI or from application wavetool, there are options to read, write or delete the DCB for that sensor/app. These m2m2 handler functions have to be added corresponding to that. These m2m2 handler will call the wrapper functions that we defined in step-2.

   For example - following are the m2m2 handler functions that we added for adxl in adxl_task.c -

   - *static m2m2_hdr_t *adxl_dcb_command_read_config(m2m2_hdr_t *p_pkt)*
   - *static m2m2_hdr_t *adxl_dcb_command_write_config(m2m2_hdr_t *p_pkt);*
   - *static m2m2_hdr_t *adxl_dcb_command_delete_config(m2m2_hdr_t *p_pkt);*

5. Whenever adding new m2m2 handler function in any application task file, it has to be mentioned in the routing table also, which will be present in the same task file. This enables the post office to route this m2m2 pkt.

   For example - This is how it is done for adxl-

```
153 app_routing_table_entry_t adxl_app_routing_table[] = {
154     {M2M2_APP_COMMON_CMD_STREAM_START_REQ, adxl_app_stream_config},
155     {M2M2_APP_COMMON_CMD_STREAM_STOP_REQ, adxl_app_stream_config},
156     {M2M2_APP_COMMON_CMD_STREAM_SUBSCRIBE_REQ, adxl_app_stream_config},
157     {M2M2_APP_COMMON_CMD_STREAM_UNSUBSCRIBE_REQ, adxl_app_stream_config},
158     {M2M2_APP_COMMON_CMD_SENSOR_STATUS_QUERY_REQ, adxl_app_status},
159     {M2M2_SENSOR_COMMON_CMD_GET_DCFG_REQ, adxl_app_get_dcfg},
160     {M2M2_SENSOR_ADXL_COMMAND_LOAD_CFG_REQ, adxl_app_load_cfg},
161     {M2M2_SENSOR_COMMON_CMD_READ_REG_16_REQ, adxl_app_reg_access},
162     {M2M2_SENSOR_COMMON_CMD_WRITE_REG_16_REQ, adxl_app_reg_access},
163     {M2M2_SENSOR_COMMON_CMD_GET_STREAM_DEC_FACTOR_REQ, adxl_app_decimation},
164     {M2M2_SENSOR_COMMON_CMD_SET_STREAM_DEC_FACTOR_REQ, adxl_app_decimation},
165     {M2M2_APP_COMMON_CMD_GET_VERSION_REQ, adxl_app_get_version},
166 #ifdef DCB
167     {M2M2_DCB_COMMAND_READ_CONFIG_REQ, adxl_dcb_command_read_config},
168     {M2M2_DCB_COMMAND_WRITE_CONFIG_REQ, adxl_dcb_command_write_config},
169     {M2M2_DCB_COMMAND_ERASE_CONFIG_REQ, adxl_dcb_command_delete_config},
170 #endif
171     {M2M2_SENSOR_ADXL_COMMAND_SELF_TEST_REQ, adxl_do_self_test},
172 };
```

6. After that changes have to be made in "system_task.c" file. The m2m2 function that CLI or wavetool uses, is used to check if DCB config is present or not. It is a common function for all the DCB blocks. Search for- "M2M2_DCB_COMMAND_QUERY_STATUS_REQ" in that file.

   As done for adxl below, the same steps have to be done for the new DCB that was added, and call the new_sensor_get_dcb_present_flag() function that is defined in [step-2-]

```
2444        case M2M2_DCB_COMMAND_QUERY_STATUS_REQ: {
2445
2446          response_mail = post_office_create_msg(
2447              M2M2_HEADER_SZ + sizeof(m2m2_dcb_block_status_t));
2448          if (response_mail != NULL) {
2449            m2m2_dcb_block_status_t *resp =
2450                (m2m2_dcb_block_status_t *)&response_mail->data[0];
2451
2452            memset(resp->dcb_blk_array, 0, sizeof(resp->dcb_blk_array));
2453
2454            // Query & Fill in the DCB Block status currently in the firmware
2455 #ifdef LOW_TOUCH_FEATURE
2456            resp->dcb_blk_array[ADI_DCB_GENERAL_BLOCK_IDX] =
2457                gen_blk_get_dcb_present_flag();
2458            resp->dcb_blk_array[ADI_DCB_WRIST_DETECT_BLOCK_IDX] =
2459                wrist_detect_get_dcb_present_flag();
2460 #endif
2461            resp->dcb_blk_array[ADI_DCB_ADPD4000_BLOCK_IDX] =
2462                adpd4000_get_dcb_present_flag();
2463            resp->dcb_blk_array[ADI_DCB_ADXL362_BLOCK_IDX] =
2464                adxl_get_dcb_present_flag();
2465 #ifdef ENABLE_PPG_APP
2466            resp->dcb_blk_array[ADI_DCB_PPG_BLOCK_IDX] =
2467                ppg_get_dcb_present_flag();
2468 #endif
2469 #ifdef ENABLE_ECG_APP
2470            resp->dcb_blk_array[ADI_DCB_ECG_BLOCK_IDX] =
2471                ecg_get_dcb_present_flag();
2472 #endif
2473 #ifdef ENABLE_EDA_APP
2474            resp->dcb_blk_array[ADI_DCB_EDA_BLOCK_IDX] =
2475                eda_get_dcb_present_flag();
2476 #endif
2477            resp->dcb_blk_array[ADI_DCB_AD7156_BLOCK_IDX] =
2478                ad7156_get_dcb_present_flag();
2479
2480            /*Send the response*/
2481            response_mail->dest = pkt->src;
2482            response_mail->src = pkt->dest;
```

This completes the changes required to be done in the firmware.

7. Last step is to add changes in following functions in CLI.py corresponding to the m2m2 handlers that was defined in step-3 and step-6. Refer the changes done for existing sensors/applications present in following functions and make the similar changes for the new sensor/application.

    - *def do_query_dcb_blk_status(self,arg)*
    - *def do_read_dcb_config(self,arg)*
    - *def do_write_dcb_config(self,arg)*
    - *def do_delete_dcb_config(self,arg)*

Note: read/write dcb commands reads and write to the .DCFG(in case of sensor) or .LCFG(in case of any application) file. Add corresponding file in "/cli/m2m2/tools/dcb_cfg" directory and add corresponding DCFG/LCFG file for the new sensor/application.

For ex- the "adxl_dcb.dcfg" file stored in "dcb_cfg" folder for adxl DCFG, so whenever there is need to write to adxl dcb, run "write_dcb_config adxl adxl_dcb.dcfg" cmd. This will read the config from "adxl_dcb.dcfg" file, CLI will then parse it and send m2m2 packet

to FW, where the m2m2 handler for writing to adxl dcb will be called and hence the config will written to the adxl DCB. Similarly when we read from the DCB, the adxl dcb config is read and stored in "adxl_dcb_get.dcfg" file that will be created in the same "/cli/m2m2/tools/dcb_cfg" directory.