|  |
| --- |
| ADI Study Watch  DCB Design |
| REV 1.0.0,MAR 2021 |

Table of Contents

[1 Device Configuration Block (DCB) 4](#_Toc66353911)

[2 Adding new DCB 6](#_Toc66353912)

List of Figures

[Figure 1: ADXL DCB Flow 5](#_Toc66353916)

[Figure 2: DCB usage in ADXL 5](#_Toc66353917)

List of Tables

No table of figures entries found.Copyright, Disclaimer & Trademark Statements

Copyright Information

Copyright (c) 2021 Analog Devices, Inc. All Rights Reserved. This documentation is proprietary and confidential to Analog Devices, Inc. and its licensors. This document may not be reproduced in any form without prior, express consent from Analog Devices, Inc.

Disclaimer

Analog Devices, Inc. (“Analog Devices”) reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent or other rights of Analog Devices

Trademark and Service Mark Notice

Analog Devices, the Analog Devices logo, Blackfin, SHARC, TigerSHARC, CrossCore, VisualDSP, VisualDSP++, EZ-KIT Lite, EZ-Extender, SigmaStudio and Collaborative are the exclusive trademarks and/or registered trademarks of Analog Devices, Inc (“Analog Devices”).

All other brand and product names are trademarks or service marks of their respective owners.

Analog Devices’ Trademarks and Service Marks may not be used without the express written consent of Analog Devices, such consent only to be provided in a separate written agreement signed by Analog Devices. Subject to the foregoing, such Trademarks and Service Marks must be used according to Analog Devices’ Trademark Usage guidelines. Any licensee wishing to use Analog Devices’ Trademarks and Service Marks must obtain and follow these guidelines for the specific marks at issue.

# 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](https://infocenter.nordicsemi.com/topic/com.nordic.infocenter.sdk5.v11.0.0/lib_fds.html)).

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**);

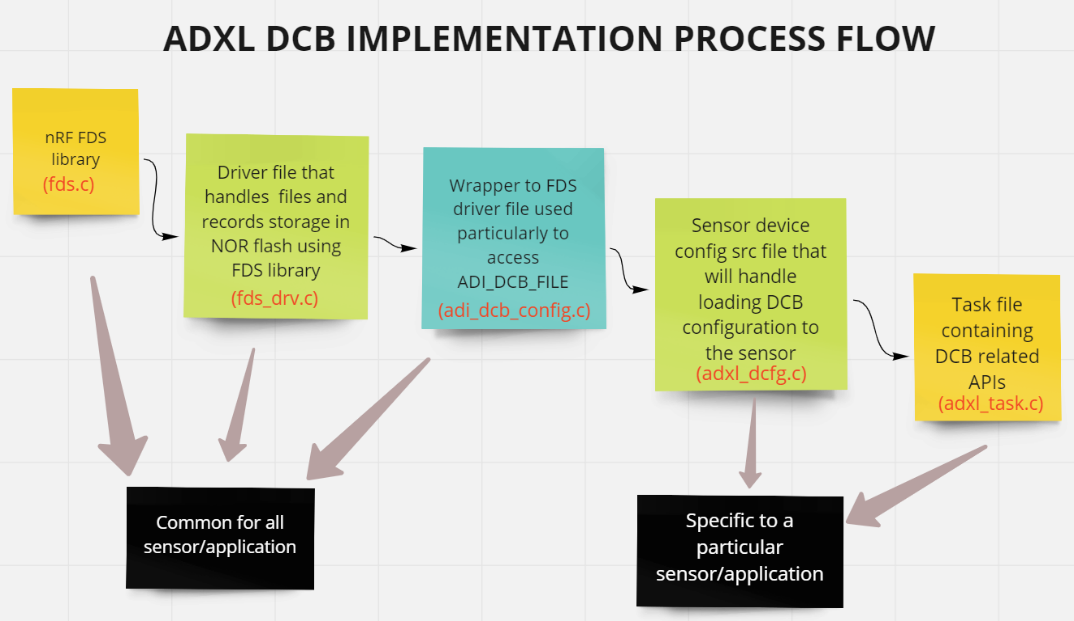


Figure 1: ADXL DCB Flow

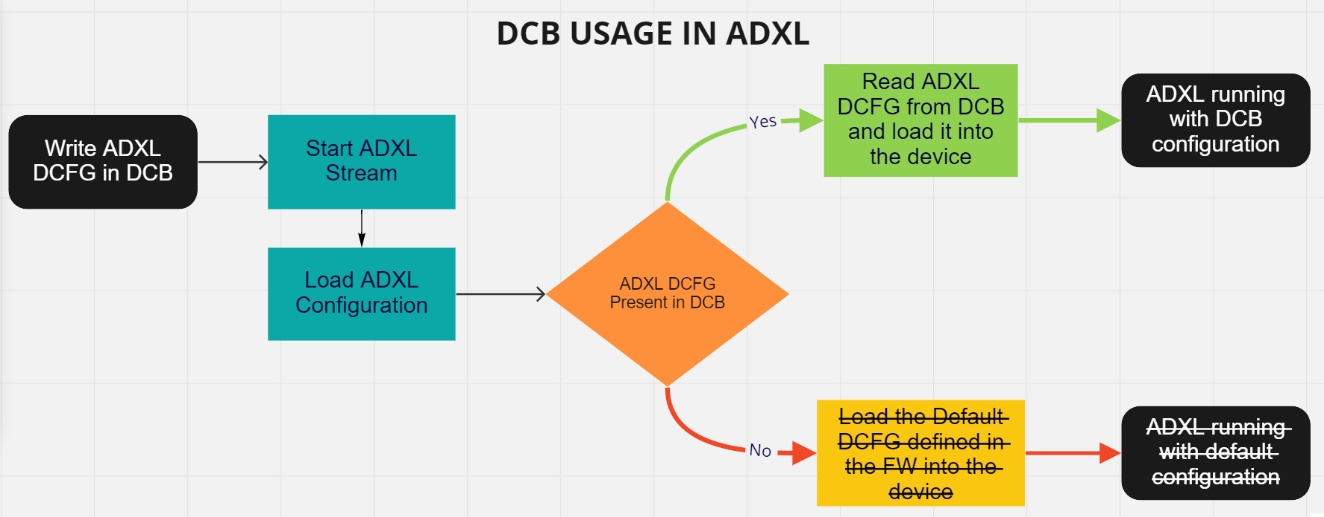


Figure 2: DCB usage in ADXL

# 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;

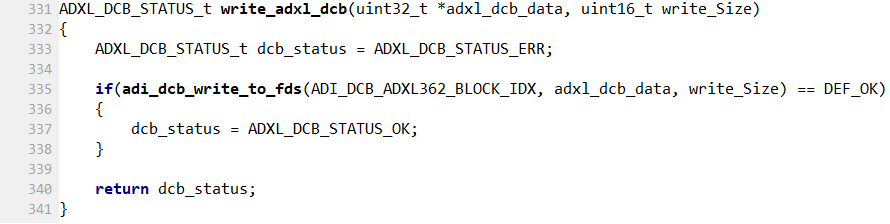
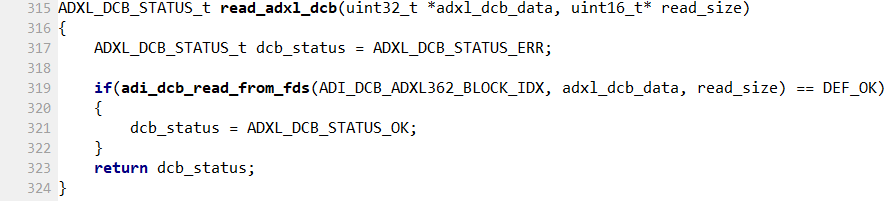
1. In the new sensor device configuration src file( generally the file that contains functions for loading the sensor's DCFG for ex- "adxl\_dcfg.c" file in case of adxl), Add wrapper functions for below mentioned functions present in "adi\_dcb\_config.c" file.

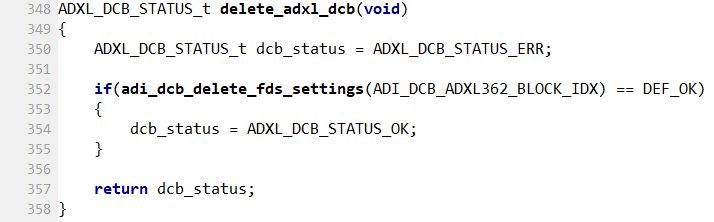
* *uint8\_t adi\_dcb\_write\_to\_fds(ADI\_DCB\_NEW\_SENSOR\_BLOCK\_IDX, uint32\_t \*wd\_dcb\_data, uint16\_t len\_DWORD);*
* *uint8\_t adi\_dcb\_read\_from\_fds(ADI\_DCB\_NEW\_SENSOR\_BLOCK\_IDX, uint32\_t \*rd\_dcb\_data, uint16\_t \*len\_DWORD);*
* *uint8\_t adi\_dcb\_delete\_fds\_settings(ADI\_DCB\_NEW\_SENSOR\_BLOCK\_IDX);*
* *uint8\_t adi\_dcb\_check\_fds\_entry(ADI\_DCB\_NEW\_SENSOR\_BLOCK\_IDX);*

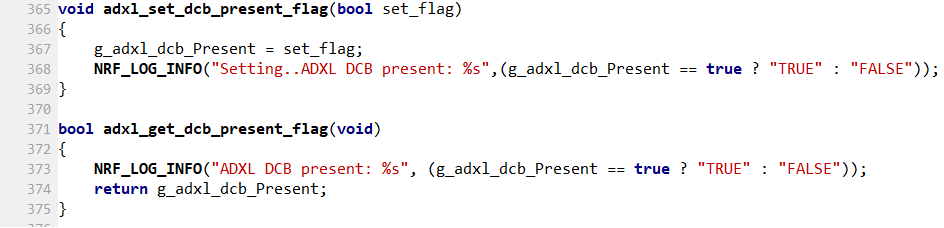
As can be seen above, in all these functions, the corresponding block index has to be specified for that new sensor/application that is defined in Step-1.

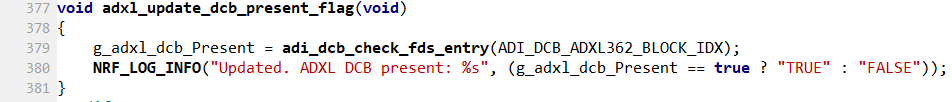
***Note: In case of any bio-medical application, a wrapper functions to it in the application’s middleware file can be defined (for ex- for PPG, the wrapper is "mw\_ppg.c" file).***

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.



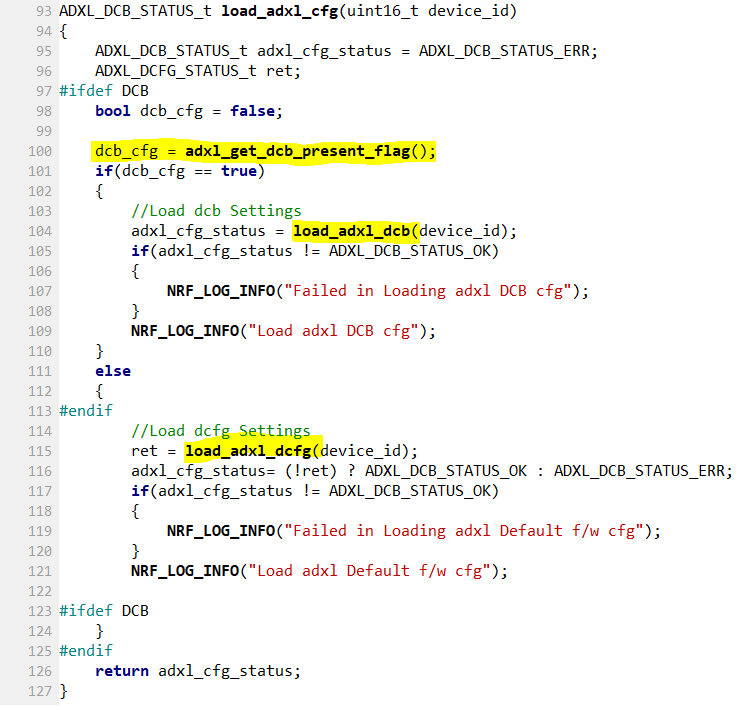






1. Configuration stored in DCB must be given preference over the default configuration. So changes have to be added in the function that loads the device configuration (DCFG) or library configuration (LCFG) such that whenever the configuration has to be loaded, first it should check if DCB config is 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.



1. 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](#step2).

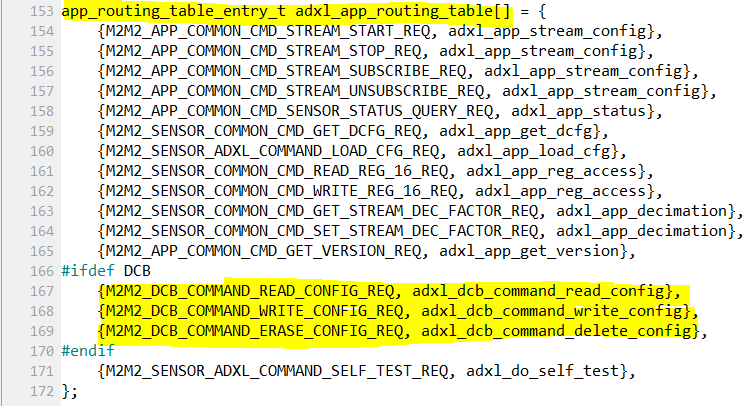
For example - following are the m2m2 handler functions that we added for adxl in

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);

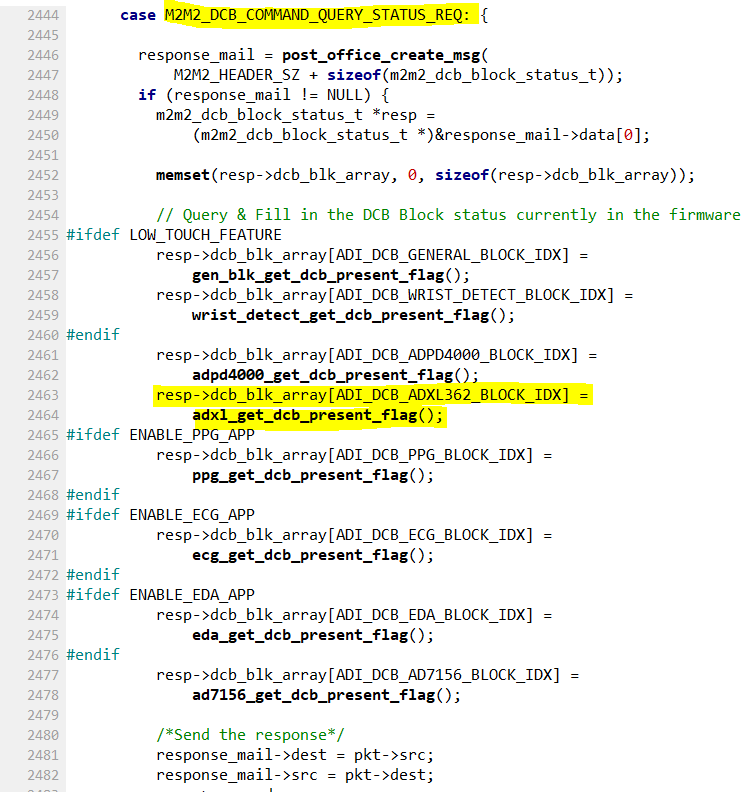
1. Whenever adding new m2m2 handler function in any application task file, it has to be mentioned in the routing table also, which will be present in the same task file. This enables the post office to route this m2m2 pkt.

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



1. 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-](#step2)



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

1. Last step is to add changes in following functions in CLI.py corresponding to the m2m2 handlers that was defined in [step-3](#step3) and [step-6](#step6). 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.