ADPD DRIVER INTEGRATION GUIDE

ANALOG DEVICES, INC.

www.analog.com

REV 0.2, DECEMBER 2016

Table of Contents

List of Figures

Copyright, Disclaimer & Trademark Statements

Copyright Information

Disclaimer

Trademark and Service Mark Notice

1 Introduction

This document describes the steps to integrate the ADPD Driver.

1.1 Scope

The document is intended to assist software developers integrating the ADPD Driver for Cortex-M processors to their application. The document helps the user to bring up the ADPD sensor driver and analyse the PPG signal quality.

1.2 Organization of this Guide

Section 1: this section contains the introduction

Section 2: lists specifications of the product

Section 3: quick start guide

Section 4: sample application code

1.3 Acronyms

ADI	**A**nalog **D**evices **I**nc.
API	**A**pplication **P**rogram **I**nterface
ADPD	**A**nalog **D**evices **P**hoto **D**iode Sensor
HAL	**H**ardware **A**bstraction **L**ayer
ISR	**I**nterrupt **S**ervice **R**outine

1.4 References

1. ADPD103 Data Sheet
2. ADPD105 Data Sheet
3. ADPD107 Data Sheet

2 Specifications

2.1 Version Information

This document uses release 2.0.0 of the ADPD Driver.

2.2 Features

- Selection of ADPD slots. Both slots can be configured independently in mixed mode
- Configuring the ADPD device and reading data from the device registers and FIFO

2.3 Deliverables

- ADPD Driver modules with a C-callable API (Application Programming Interface)
- Sample application code as reference for integrating the driver
- ADPD driver code and corresponding header file
- Documents –Integration Guide (this document), Release Notes

3 Quick Start

This section contains a step-by-step guide for integrating the driver bring up code. The Figure 1 shows the complete integration, the details of which are explained in the remaining sections.

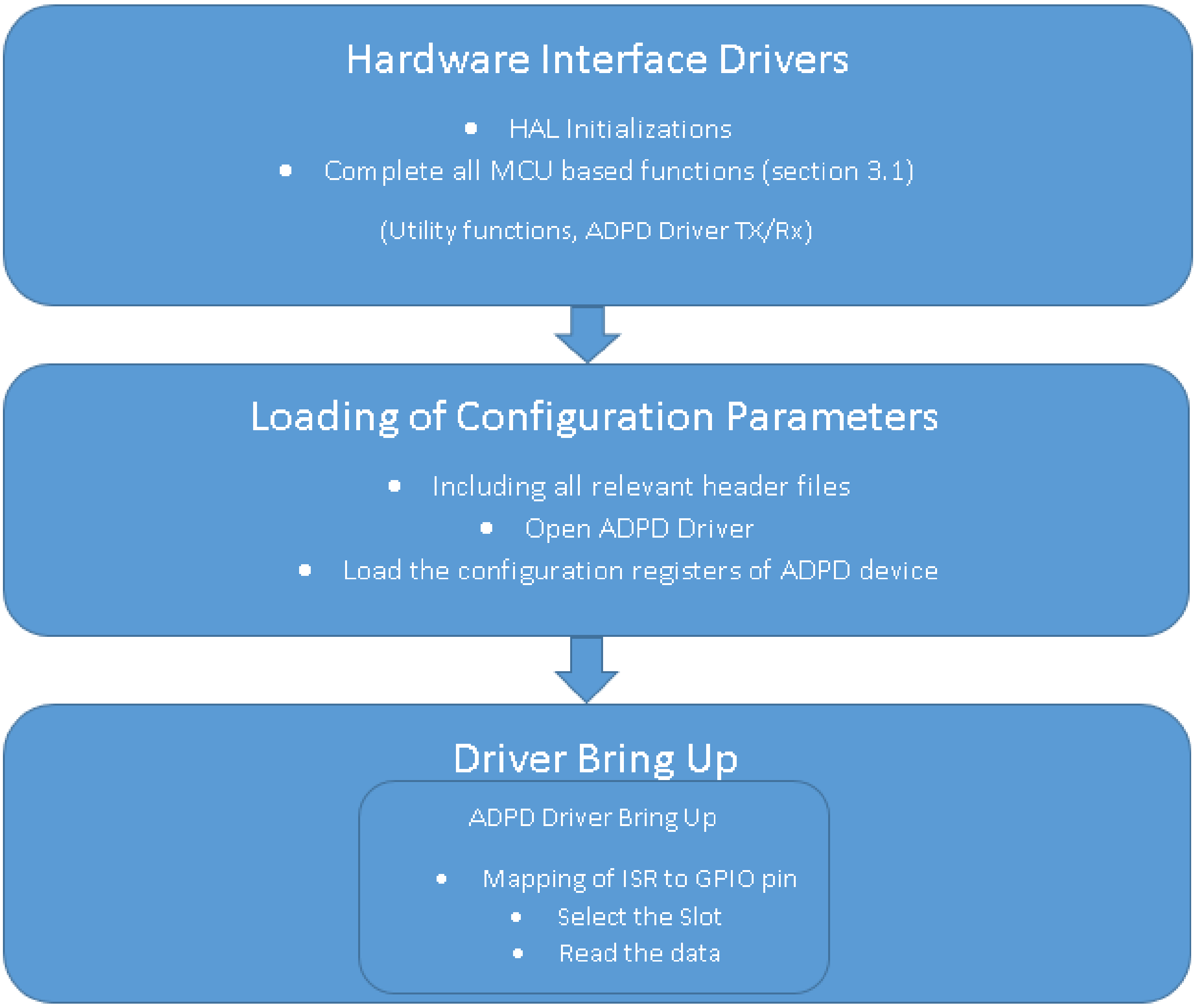

Figure 1: Integration Flow

3.1 Hardware Interface Drivers

1. HAL initializations such as enabling system tick, interrupt priority and low level hardware initialization.
2. Set the desired system frequency

3. Initialize the GPIO, UART and I2C for ADPD communication. Configure the voltage regulators.

4. Complete the following functions to support the driver:

    a. *adi_printf()* – To print the ADPD data, a function has to be written. Absence of this function will cause build error**. Note that the printf implementation should support floating point format printing**.

    b. MCU_HAL_GetTick().This function is a wrapper over the low level processor-specific implementation of get tick function. Absence of this function will cause build error.

```
uint32_t MCU_HAL_GetTick() {

   return (uint32_t)HAL_GetTick();
}
where the tick is obtained in milliseconds unit
```

5. Complete the following middleware functions before proceeding to section 3.2

    ADPD Driver transmit/receive functions:

    a. *MCU_HAL_I2C_Transmit(uint8_t *pData, uint16_t Size, uint32_t Timeout)* – This function transmits the buffer pointed to by "pData" through I2C to ADPD, where the size is specified by "size". It times out if the device does not respond within specified time "Timeout". **Note:** It is to be ensured that the correct ADPD I2C address is used while implementing this function.

    *b. MCU_HAL_I2C_TxRx(uint8_t *pTxData, uint8_t *pRxData, uint16_t RxSize, uint32_t Timeout)* – This function transmits a byte pointed to by "pTxData" and receives "RxSize" number of bytes from ADPD in buffer pointed to by "pRxData". It times out if the device does not respond within specified time "Timeout". **Note:** It is to be ensured that the correct ADPD I2C address is used while implementing this function.

3.2 Loading of configuration parameters

1. Include *AdpdDrv.h* file. This has declarations of all the APIs supported by the driver for ADPD sensor. Define the following macro in *AdpdDrv.h* file to run it on ADPD107 device. The same can be done by uncommenting line number 149 in *AdpdDrv.h* file.

    #define ADPD_SPI

2. Register a data ready callback routine through *AdpdDrvDataReadyCallback()* function. A sample of the data ready callback routine is shown below.

```
void AdpdFifoCallBack(void) {
   gnAdpdDataReady = 1;
   gnAdpdTimeCurVal = MCU_HAL_GetTick();
}
```

3. The ADPD device is soft reset by calling the function *AdpdDrvSoftReset().*
4. The ADPD driver is initialized using a call to *AdpdDrvOpenDriver()*.
5. In this step, the device configuration settings for the device has to be loaded. Before loading the settings, the recommended start sequence for the device has to be followed. This sequence is as detailed below:
   a. Put the device into *program* mode, by writing 0x01 to register 0x10.
   b. When using FIFO mode(which is recommended), set bit 0 of register 0x5F, followed by writing 0x00FF to register 0x00 to clear all interrupts. The FIFO contents has to be cleared by writing 0x80FF to register 0x00.
   c. Write the configuration registers through dcfg_org_103[] array while device is in this *program* mode. The dcfg array does segregated by two different pre-processor macros for sample and proximity modes. For the default configuration settings of the device in use, refer to the dcfg_org_103[] array in *main_external.c*
6. Once the configuration values are written, the values can be read back and compared to the values that were written, to verify the I2C communication.

Note: The function *LoadDefaultConfig() and VerifyDefaultConfig()* in the sample application code is the reference for this step. A sample of these routines are shown below

```
void LoadDefaultConfig(uint32_t *cfg) {
    uint8_t regAddr, i;
    uint16_t regData;
    if (cfg == 0) {
        return;
    }
    /* Clear the FIFO */
    AdpdDrvRegWrite(0x10, 0);
    AdpdDrvRegWrite(0x5F, 1);
    AdpdDrvRegWrite(0x00, 0x80FF);
    AdpdDrvRegWrite(0x5F, 0);
    i = 0;
    while (1) {
        /* Read the address and data from the config */
        regAddr = (uint8_t)(cfg[i] >> 16);
        regData = (uint16_t)(cfg[i]);
        i++;
        if (regAddr == 0xFF) {
            break;
        }
        /* Load the data into the ADPD registers */
        if (AdpdDrvRegWrite(regAddr, regData) != ADPDDrv_SUCCESS) {
            break;
        }
    }
}

void VerifyDefaultConfig(uint32_t *cfg) {
    uint16_t def_val;
    uint8_t  i;
    uint8_t  regAddr;
    uint16_t regData;
    if (cfg == 0) {
        return;
    }
    i = 0;
    /* Read the address and data from the config */
    regAddr = (uint8_t)(cfg[0] >> 16);
    def_val = (uint16_t)(cfg[0]);
    /* Read the data from the ADPD registers and verify */
    while (regAddr != 0xFF) {
        if (AdpdDrvRegRead(regAddr, &regData) != ADPDDrv_SUCCESS) {
            debug("DCFG: Read Error reg(%0.2x)\n", regAddr);
            return;
        } else if (regData != def_val) {
            debug("DCFG: Read mismatch reg(%0.2x) (%0.2x != %0.2x)\n",
                  regAddr, def_val, regData);
            return;
        }
        i++;
        regAddr = (uint8_t)(cfg[i] >> 16);
        def_val = (uint16_t)(cfg[i]);
    }
}
```

3.3 Driver Bring Up

This section and the two sub-sections explains the steps to bring up the devices such as ADPD sensor and accelerometer. The interrupt service routine for each of the device has to be mapped to the respective GPIO pin of the processor. The code snippet below shows how the ISR for ADPD is mapped, where ADPD_INT_PIN are assigned to GPIO pins. For eg:-

The mapping on ADI M3 reference platform is as follows:-

#define ADPD_INT_PIN GPIO_PIN_13

```
if (GPIO_Pin == ADPD_INT_PIN){
    AdpdISR(GPIO_Pin);
}
```

3.3.1 ADPD Driver Bring Up

The ADPD driver bring up code is available in the example code in Section 4. The following are the steps.

1. Once the above steps in section 3.2 are done, write register 0x4B with value 0x2695 and register 0x4D with 0x4272.
2. Optionally, the slots can be set to various modes by calling function *AdpdDrvSetSlot(nslotA, nslotB).*
3. Put the device into *sample/proximity* mode, by writing 0x02 to register 0x10 using the function *AdpdDrvSetOperationMode(ADPDDrv_MODE_SAMPLE/ ADPDDrv_MODE_PROXIMITY).*
4. The sensor device is now ready to be read. The data reading from the device using a code snippet as shown below, where ,

   value should be declared as an array of sixteen 8-bit words.

```
while (1) {
      /* Check if the data is ready */
      if(gnAdpdDataReady)  {
         gnAdpdDataReady = 0;
         /* Read the size of the data available in the FIFO */
         AdpdDrvGetParameter(ADPD_FIFOLEVEL, &nAdpdFifoLevelSize);
         /* Read the data from the FIFO and print them */
         while (nAdpdFifoLevelSize >= nAdpdDataSetSize) {
            nRetValue = AdpdDrvReadFifoData(&value[0],
                                            nAdpdDataSetSize);
            if (nRetValue == ADPDDrv_SUCCESS) {
                  for (LoopCnt = 0; LoopCnt < nLoopLim; LoopCnt += 2)
                      /* Byte swapping is needed to print ADPD data in
proper format */
                      debug("%u ", (value[LoopCnt] << 8) | value[LoopCnt +
1]);
                  debug("%u\r\n", gnAdpdTimeCurVal);
                  nAdpdFifoLevelSize = nAdpdFifoLevelSize -
nAdpdDataSetSize;
            }
         }
      }
   }
```

Note: nLoopLim variable has to be initialized with corresponding values to construct the data based on the configured mode.

5. The data from the device can be captured using tera term and saved as a .csv file. This data from the desired slot can be plotted and the quality of the obtained PPG signal can be ascertained to be clean and having good signal-to-noise ratio.

   A snap shot of the data that is logged is shown below. Figure 2 shows the PPG signal.

```
1137 1143 409 504 63493 63010 14228 14876
1031 1034 387 471 62976 62416 13832 14461
1036 1040 392 475 62965 62399 13827 14458
```

   A snap shot of the proximity data that is logged is shown below.

```
2372 4341
7404 4734
4328 5127
6876 5520
6068 5913
2554 6699
4491 7092
4429 7485
```

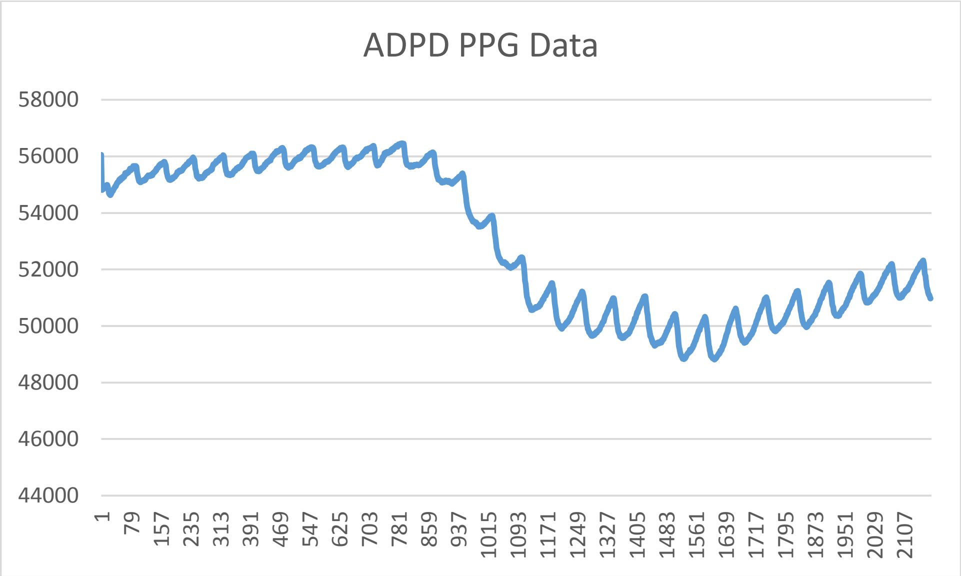

Figure 2: PPG data

4 Sample Application code

```
/**
   ******************************************************************************
     @file       example103.c
     @author      ADI
     @version     V2.0.1
     @date       19-December-2016
     @brief      Sample application to use ADI ADPD103 driver.
*/
/*******************************************************************************
*                                                                            *
   License Agreement
*                                                                            *
   Copyright (c) 2016 Analog Devices Inc.
   All rights reserved.

   Redistribution and use in source and binary forms, with or without
   modification, are permitted (subject to the limitations in the disclaimer
   below) provided that the following conditions are met:
*                                                                            *
   Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.
*                                                                            *
*                                                                            *
   Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.
*                                                                            *
*                                                                            *
   Neither the name of Analog Devices, Inc. nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.
*                                                                            *
   NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE
GRANTED BY
   THIS LICENSE.  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS
AND
   CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT
   NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR
```

```
   A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER
   OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL,
   EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO,
   PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
OR PROFITS;
   OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY,
   WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR
   OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF
   ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*                                                                             *
*                                                                             *
   This software is intended for use with the ADPD and derivative parts
   only
*                                                                             *
******************************************************************************/*
Includes ----------------------------------------------------------------*/
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <time.h>
#include "AdpdDrv.h"


/* Macros ------------------------------------------------------------------*/
#define debug(M, ...)  {_SBZ[0] = 0; \
                    snprintf(_SBZ, BUF_SIZE, "" M "", ##__VA_ARGS__); \
                    adi_printf("%s", _SBZ);}

/*    Uncomment  the following macro to set the sensor in sample mode */
//#define ADPD_SAMPLE_MODE
/*    Uncomment  the following macro to set the sensor in proximity mode */
#define ADPD_PROXIMITY_MODE

/* Private define ---------------------------------------------------------- */
uint32_t gnAdpdTimeCurVal = 0;
uint8_t gnAdpdDataReady = 0;
```

```
/* Private variables ---------------------------------------------------------*/
#define BUF_SIZE (256)
char _SBZ[BUF_SIZE]; // used by 'debug'

uint32_t dcfg_org_103[] = {
#ifdef ADPD_SAMPLE_MODE
        0x00120028,
        0x00150220,
        0x00181F00,
        0x00191F00,
        0x001A1F00,
        0x001B1F00,
        0x001E1F00,
        0x001F1F00,
        0x00201F00,
        0x00211F00,
        0x00340000,
#elif defined(ADPD_PROXIMITY_MODE)
        0x00130FA0,
        0x00150000,
        0x002A0600,
        0x002F8000,
        0x00330113,
        0x00340200,
        0x003A24D4,
#endif
        0x00060000,
        0x00113120,
        0x00140555,
        0x00223030,
        0x00233002,
        0x00243000,
        0x0025630C,
        0x00300219,
        0x00310113,
        0x00350219,
        0x00360813,
        0x003921FB,
        0x003B21FB,
        0x00421C36,
        0x00441C35,
        0x00000000,
        0xFFFFFFFF,
};
```

```
/* Private function prototypes -----------------------------------------------*/
void HW_Global_Init(void);
void LoadDefaultConfig(uint32_t *cfg);
void VerifyDefaultConfig(uint32_t *cfg);
void AdpdDriverBringUp(uint8_t nSlotA, uint8_t nSlotB);
void AdpdFifoCallBack(void);

void SystemClock_Config();
void HAL_Init();
void GPIO_Init();
void UART_Init();
void I2C_Init();
void TIM_Init();
void ADP_init();
void MCU_HAL_Delay(uint32_t);
uint32_t MCU_HAL_GetTick();

uint16_t AdpdRxBufferInsertData(uint32_t tcv);

/* Private function prototypes ---------------------------------------------- */

uint16_t AppReadAdpdDataBuffer(ADPDData_t *rxData, uint32_t *time);

/**
  *  @brief    Callback function.
  *  @param    None
  *  @retval   None
*/
void AdpdFifoCallBack(void) {
   /* Read the timestamp when the interrupt comes */
   gnAdpdTimeCurVal = MCU_HAL_GetTick();
   /* Set gnAdpdDataReady to 1 to indicate that the data and timestamp is ready */
   gnAdpdDataReady = 1;
}
```

```
/**
                  * Flow diagram of the code *

                  ----------------------------
                  | Hardware initializations |
                  ----------------------------
                              |
                              |
                  ----------------------------
                  |  Data ready callback     |
                  ----------------------------
                              |
                              |
                  ----------------------------
                  |Soft reset the ADPD device|
                  |Initialize the ADPD driver|
                  ----------------------------
                              |
                              |
                  ----------------------------
                  | Load the default config  |
                  | and verify it            |
                  ----------------------------
                              |
                              |
                  ----------------------------
                  | Write standard value of  |
                  | clock registers          |
                  ----------------------------
                              |
                              |
                  ----------------------------
               --->|    Driver bring up      |
               |  ----------------------------
               |              |
               |              |
               |--------------|

*/
```

```
/**
  * @brief    Main program.
  * @param    None
  * @retval   None
*/
void main(void) {

    /* Hardware initializations */
    HW_Global_Init();

    debug("Start");

    /* Register data ready callback */
    AdpdDrvDataReadyCallback(AdpdFifoCallBack);

    /* Soft reset the ADPD device */
    AdpdDrvSoftReset();

    /* Initialize the ADPD driver*/
    AdpdDrvOpenDriver();

    /* Load default configuration parameters */
    LoadDefaultConfig(dcfg_org_103);
    /* Read default configuration parameters from the device registers and verify */
    VerifyDefaultConfig(dcfg_org_103);

    /* Write standard value of clock registers */
    AdpdDrvRegWrite(0x004B, 0x2695);
    AdpdDrvRegWrite(0x004D, 0x4272);

#ifdef ADPD_SAMPLE_MODE
    /* Driver bring up with 16-bits output data and 8 channel mode */
    AdpdDriverBringUp(ADPDDrv_4CH_16, ADPDDrv_4CH_16);
#elif  defined(ADPD_PROXIMITY_MODE)
    /* Driver bring up with proximity mode */
    AdpdDriverBringUp(ADPDDrv_PROXIMITY, ADPDDrv_SLOT_OFF);
#elif defined(ADPD_GESTURE_MODE)
#endif
}

/**
  * @brief    Hardware Initialization.
  * @param    None
  * @retval   None
```

```
*/
void HW_Global_Init() {

   /* HAL initializations such as enabling system tick and low level hardware initialization.*/
   HAL_Init();
   /* Configure the system clock to 26 Mhz */
   SystemClock_Config();
   /* Initialize the GPIO. Should be called before I2C_Init() */
   GPIO_Init();
   /* Initialize the UART */
   UART_Init();
   /* Initialize the I2C. Should be called after GPIO_Init() */
   I2C_Init();
   /* Configure the voltage regulators in proper mode */
   ADP_init();
}

/**
  *  @brief    Load ADPD default configuration
  *  @param    uint32_t *cfg
  *  @retval   None
*/
void LoadDefaultConfig(uint32_t *cfg) {
   uint8_t regAddr, i;
   uint16_t regData;
   if (cfg == 0) {
      return;
   }
   /* Clear the FIFO */
   AdpdDrvRegWrite(0x10, 0);
   AdpdDrvRegWrite(0x5F, 1);
   AdpdDrvRegWrite(0x00, 0x80FF);
   AdpdDrvRegWrite(0x5F, 0);
   i = 0;
   while (1) {
      /* Read the address and data from the config */
      regAddr = (uint8_t)(cfg[i] >> 16);
      regData = (uint16_t)(cfg[i]);
      i++;
      if (regAddr == 0xFF) {
         break;
      }
      /* Load the data into the ADPD registers */
      if (AdpdDrvRegWrite(regAddr, regData) != ADPDDrv_SUCCESS) {
```

```
            break;
        }
    }
}

/**
  *  @brief    Read default configuration parameters to verify
  *  @param    uint32_t *cfg
  *  @retval   None
*/
void VerifyDefaultConfig(uint32_t *cfg) {
    uint16_t def_val;
    uint8_t  i;
    uint8_t  regAddr;
    uint16_t regData;
    if (cfg == 0) {
        return;
    }
    i = 0;
    /* Read the address and data from the config */
    regAddr = (uint8_t)(cfg[0] >> 16);
    def_val = (uint16_t)(cfg[0]);
    /* Read the data from the ADPD registers and verify */
    while (regAddr != 0xFF) {
        if (AdpdDrvRegRead(regAddr, &regData) != ADPDDrv_SUCCESS) {
            debug("DCFG: Read Error reg(%0.2x)\n", regAddr);
            return;
        } else if (regData != def_val) {
            debug("DCFG: Read mismatch reg(%0.2x) (%0.2x != %0.2x)\n",
                  regAddr, def_val, regData);
            return;
        }
        i++;
        regAddr = (uint8_t)(cfg[i] >> 16);
        def_val = (uint16_t)(cfg[i]);
    }
}

/**
  *  @brief  ADPD Driver bring up.
  *  @param  uint8_t nSlotA
  *  @param  uint8_t nSlotB
  *  @retval None
*/
```

```
void AdpdDriverBringUp(uint8_t nSlotA, uint8_t nSlotB) {
  uint32_t LoopCnt;
  uint16_t nRetValue = 0;
  uint16_t nAdpdFifoLevelSize = 0, nAdpdDataSetSize = 16;
  uint8_t value[16] = {0};

  /* Set the slot modes for slot A and slot B */
  AdpdDrvSetSlot(nSlotA, nSlotB);

#ifdef ADPD_SAMPLE_MODE
   /* Set the device operation to sample mode. The data can be collected now */
   AdpdDrvSetOperationMode(ADPDDrv_MODE_SAMPLE);
   nLoopLim = nAdpdDataSetSize = 16;
#elif defined(ADPD_PROXIMITY_MODE)
   /* Set the device operation to proximity mode. The data can be collected now */
   AdpdDrvSetOperationMode(ADPDDrv_MODE_PROXIMITY);
   nLoopLim = nAdpdDataSetSize = 2;
#endif
  while (1) {
    /* Check if the data is ready */
    if(gnAdpdDataReady)  {
      gnAdpdDataReady = 0;
      /* Read the size of the data available in the FIFO */
      AdpdDrvGetParameter(ADPD_FIFOLEVEL, &nAdpdFifoLevelSize);
      /* Read the data from the FIFO and print them */
      while (nAdpdFifoLevelSize >= nAdpdDataSetSize) {
        nRetValue = AdpdDrvReadFifoData(&value[0],
                                        nAdpdDataSetSize);
        if (nRetValue == ADPDDrv_SUCCESS) {
            for (LoopCnt = 0; LoopCnt < nLoopLim; LoopCnt += 2)
               /* Byte swapping is needed to print ADPD data in proper format */
               debug("%u ", (value[LoopCnt] << 8) | value[LoopCnt + 1]);
            debug("%u\r\n", gnAdpdTimeCurVal);
            nAdpdFifoLevelSize = nAdpdFifoLevelSize - nAdpdDataSetSize;
        }
      }
    }
  }
}
```