ADUX1020 DRIVER INTEGRATION GUIDE

ANALOG DEVICES, INC.

www.analog.com

REV 0.1,NOV 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 ADUX Driver.

1.1 Scope

The document is intended to assist software developers integrating the ADUX Driver for Cortex-M processors to their application. The document helps the user to bring up the ADUX sensor driver..

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
ADUX	**A**nalog **D**evices **U**ser **E**xpirence Sensor
HAL	**H**ardware **A**bstraction **L**ayer
ISR	**I**nterrupt **S**ervice **R**outine

1.4 References

1. ADUX1020 Data Sheet

2 Specifications

2.1 Version Information

This document uses release 0.1 of the ADUX1020 Driver.

2.2 Features

- Configuring the ADUX1020 device and reading data from the device registers and FIFO

2.3 Deliverables

- ADUX1020 Driver modules with a C-callable API (Application Programming Interface)
- Sample application code as reference for integrating the driver
- ADUX1020 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.

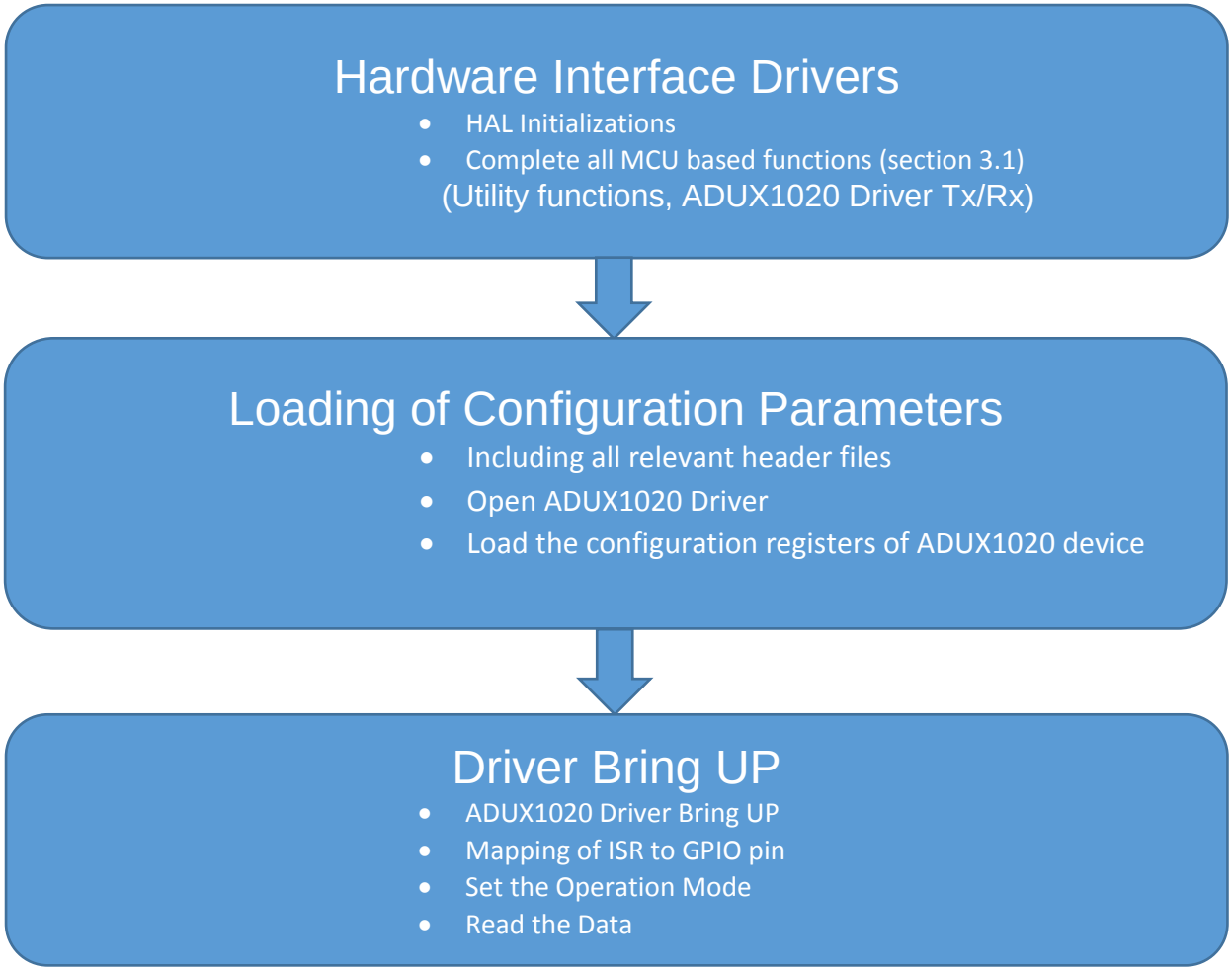

Figure 1: Integration Flow

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 ADUX communication. Configure the voltage regulators.

4. Complete the following functions to support the driver:

    a. *adi_printf()* – To print the ADUX1020 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

    ADUX 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 ADUX1020, 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 ADUX1020 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 ADUX1020 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 ADUX1020 I2C address is used while implementing this function.

3.2 Loading of configuration parameters

1. Include *Adux1020Drv.h* file. This has declarations of all the APIs supported by the driver for ADUX1020 sensor.
2. Register a data ready callback routine through *Adux1020DrvDataReadyCallback()* function. A sample of the data ready callback routine is shown below.

```
void AduxFifoCallBack(void) {
   gnAduxDataReady = 1;
   gnAduxTimeCurVal = MCU_HAL_GetTick();
}
```

3. The ADUX1020 device is soft reset by calling the function *Adux1020DrvSoftReset()*.
4. The ADUX1020 driver is initialized using a call to *Adux1020DrvOpenDriver()*.
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 0x0 to register 0x45.
    b. When using FIFO mode(which is recommended), write 0xF4F to register 0x32, followed by writing 0x00FF to register 0x00 to clear all interrupts. The FIFO contents has to be cleared by writing 0x80FF to register 0x49.
    c. Write the configuration registers through dcfg [] array while device is in this *program* mode. For the default configuration settings of the device in use, refer to the dcfg [] 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

```
uint8_t LoadDefaultConfig(uint32_t *pnCfg)
{
  uint8_t nRegAddr, nIdx, nRet = -1;
  uint16_t nRegData;

  if (pnCfg == 0)
    return nRet;
  nIdx = 0;

  while (1) {
    /* Read the address and data from the config */
    nRegAddr = (uint8_t)(pnCfg[nIdx] >> 16);
    nRegData = (uint16_t)(pnCfg[nIdx]);
    nIdx++;

    if (nRegAddr == 0xFF)
      break;
    /* Load the data into the ADUX1020 registers */
    if (Adux1020DrvRegWrite(nRegAddr, nRegData) != ADUXDrv_SUCCESS) {
      debug(":FAIL. Set Reg%d=%d\r\n", (int)nRegAddr, (int)nRegData);
      return nRet;
    }
    nRegData = 0;
    Adux1020DrvRegRead(nRegAddr, &nRegData);
    debug("Config:, 0x%X ,  0x%X \r\n", nRegAddr, nRegData);
  }
  nRet = 0;
  debug(":PASS. Load Default Configure pass\r\n");
  return nRet;
}

void VerifyDefaultConfig(uint32_t *cfg)
{
  uint16_t def_val;
  uint8_t  nIdx;
  uint8_t  regAddr;
  uint16_t regData;
  if (cfg == 0) {
    return;
  }
  nIdx = 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 ADUX registers and verify */
  while (regAddr != 0xFF) {
    if (Adux1020DrvRegRead(regAddr, &regData) != ADUXDrv_SUCCESS) {
      debug("DCFG: Read Error reg(%0.2x)\r\n", regAddr);
      return;
    } else if (regData != def_val) {
      debug("DCFG: Read mismatch reg(%0.2x) (%0.2x != %0.2x)\r\n",
            regAddr, def_val, regData);
      return;
    }
```

```
    nIdx ++;
        regAddr = (uint8_t)(cfg[nIdx] >> 16);
        def_val = (uint16_t)(cfg[nIdx]);
    }
}
```

3.3 Driver Bring Up

This section and the two sub-sections explains the steps to bring up the devices such as ADUX1020 sensor. 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 ADUX1020 is mapped, where ADUX1020_INT_PIN are assigned to GPIO pins. For eg:-

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

#define ADUX1020_INT_PIN GPIO_PIN_13

```
if (GPIO_Pin == ADUX1020_INT_PIN){
   Adux1020DrvISR(GPIO_Pin);
}
```

3.3.1 ADUX1020 Driver Bring Up

The ADUX1020 driver bring up code is available in the example code in Section 4. The following are the steps.

1. Put the device into *sample* mode, by writing 0x148 to register 0x45 using the function *Adux1020DrvSetOperationMode(ADUXDrv_MODE_SAMPLE)*.
2. 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 eight 8-bit words.

```
 /* Set the fifo interrupt */
      Adux1020DrvSetInterrupt(FIFO_INT_ENABLE);
      /* Set Fifo level to trigger the interrupt */
      Adux1020DrvSetFifoLevel();
      /* Set the device operation to sample mode. The data can be
collected now */
      Adux1020DrvSetOperationMode(ADUXDrv_MODE_SAMPLE);

      while (1) {
        /* Check if the data is ready */
        if(gnAduxDataReady)  {
          gnAduxDataReady = 0;
          /* Read the size of the data available in the FIFO */
          Adux1020DrvGetParameter(FIFO_LEVEL, &nAduxFifoLevelSize);
          Adux1020DrvGetParameter(INT_STATUS, &nIntStatus);

          /* Enable the 32MHz Clock */
          if(nAduxFifoLevelSize >= nAduxDataSetSize) {
            Adux1020DrvEnable32MHzClock();
            getFifoData = 1;
          }
          /* Read the data from the FIFO and print them */
          while (nAduxFifoLevelSize >= nAduxDataSetSize) {
            nRetValue = Adux1020DrvReadFifoData(&value[0],
nAduxDataSetSize);
            if (nRetValue == ADUXDrv_SUCCESS) {
              for (LoopCnt = 0; LoopCnt < 8; LoopCnt += 2)
                /* Byte swapping is needed to print Adux data in proper
format */
                debug("%u ", (value[LoopCnt] << 8) | value[LoopCnt + 1]);
              debug("%u\r\n", gnAduxTimeCurVal);
              //debug("Interrupt Status %04X \r\n", tmp);
              nAduxFifoLevelSize = nAduxFifoLevelSize - nAduxDataSetSize;
            }
          }
          if ((nIntStatus & IRQ_MASK_SAMPLE) != 0U) {
            Adux1020DrvRegWrite(REG_INT_STATUS,
nIntStatus&IRQ_MASK_SAMPLE);
          }
          /* Disable the 32MHz Clock */
          if(getFifoData) {
            Adux1020DrvDisable32MHzClock();
            getFifoData = 0;
          }

        }
      }
```

3. The data from the device can be captured using tera term and saved as a .csv file.

```
For Sample Mode:
1533 2705 3869 3321
1540 2726 3843 3289
1556 2704 3830 3332

For Proximity Mode:
1201
1175
1200
```

4 Sample Application code

```
/**
  ******************************************************************************
   @file       example1020.c
   @author     ADI
   @version    V0.1
   @date       8-November-2016
   @brief      Sample application to use ADI ADUX1020 driver.
*/
/*******************************************************************************
*                                                                              *
  Clear BSD license
*                                                                              *
  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 ADUX1020 and derivative parts
   only
*                                                                   *
*********************************************************************************/
Includes ---------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>
#include "Adux1020Drv.h"

/* Enum -------------------------------------------------------------------*/
typedef enum {
  SAMPLE_MODE = 0,
  PROXIMITY_MODE
} Adux_Operation_Mode;

/* Macros -----------------------------------------------------------------*/

#define BUF_SIZE (256U)
#define debug(M, ...)  {_SBZ[0] = 0; \
                    snprintf(_SBZ, BUF_SIZE, "" M "", ##__VA_ARGS__); \
                      adi_printf("%s", _SBZ);}

char _SBZ[BUF_SIZE]; // used by 'debug'
```

```
/* Function Prototype */
static void HWGlobalInit();
void Adux1020FifoCallBack(void);
uint8_t LoadDefaultConfig(uint32_t *pnCfg);
void VerifyDefaultConfig(uint32_t *cfg);
void AduxDriverBringUp(Adux_Operation_Mode);

/* Private variables ---------------------------------------------------------*/
uint8_t gnAduxDataReady = 0U;
uint32_t gnAduxTimeCurVal = 0;

/* Configuration file buffer */
uint32_t dcfg[] = {
  0x000c000f,
  0x00101010,
  0x0011004c,
  0x00125f0c,
  0x0013ada5,
  0x00140080,
  0x00150000,
  0x00160600,
  0x00170000,
  0x00182693,
  0x00190004,
  0x001a4280,
  0x001b0060,
  0x001c2094,
  0x001d0020,
  0x001e0001,
  0x001f0000,
  0x00200320,
  0x00210A13,
  0x00220320,
  0x00230113,
  0x00240000,
  0x00252412,
  0x00262412,
  0x00270022,
  0x00280000,
  0x00290300,
  0x002a0700,
  0x002b0600,
  0x002c6000,
  0x002d4000,
```

```
  0x002e0000,
  0x002f0000,
  0x00300000,
  0x00310000,
  0x00320040,
  0x00330008,
  0x0034E400,
  0x00388080,
  0x00398080,
  0x003a2000,
  0x003b1f00,
  0x003c2000,
  0x003d2000,
  0x003e0000,
  0x00408069,
  0x00411f2f,
  0x00424000,
  0x00430000,
  0x00440008,
  0x00460000,
  0x004800ef,
  0x00490000,
  0x00450000,
  0xFFFFFFFF
};

/**
*  @brief    Callback function.
*  @param    None
*  @retval   None
*/
void Adux1020FifoCallBack(void)
{
        /* Set gnAduxDataReady to 1 to indicate that the data and timestamp is ready */
        gnAduxDataReady = 1;
        /* Read the timestamp when the interrupt comes */
        gnAduxTimeCurVal = McuHalGetTick();
}

/**
  * Flow diagram of the code *
     ----------------------------
     | Hardware initializations |
     ----------------------------
```

```
    |
    |
    ---------------------------
    |  Data ready callback    |
    ---------------------------
    |
    |
    ---------------------------
    |Initialize the ADUX driver|
    ---------------------------
    |
    |
    ---------------------------
    | Load the default config  |
    | and verify it           |
    ---------------------------
    |
    |
    ---------------------------
    | Write standard value of  |
    | clock registers         |
    ---------------------------
    |
    |
    ---------------------------
 --->|    Driver bring up      |
 |   ---------------------------
 |            |
 |            |
 |------------|
 */

void main(void)
{
        /* Hardware initializations */
        HWGlobalInit();
        /* Register data ready callback */
        Adux1020DrvDataReadyCallback(Adux1020FifoCallBack);
        /* Initialize the ADUX1020 driver*/
        Adux1020DrvOpenDriver();
        /* Load default configuration parameters */
        LoadDefaultConfig(dcfg);
        /* Read default configuration parameters from the device registers and verify */
        VerifyDefaultConfig(dcfg);
```

```
        /**
        Driver bring up, pass the parameter from Adux_Operation_Mode Enum
         -> SAMPLE_MODE for Sample Mode
         -> PROXIMITY_MODE for Proximity Mode
       */
        AduxDriverBringUp(SAMPLE_MODE);
}

/* Private functions ---------------------------------------------------------*/
/**
* @brief    Software Initialization.
* @retval   None
*/
static void SWGlobalInit()
{

}

/**
* @brief    Hardware Initialization.
* @retval   None
*/
static void HWGlobalInit()
{
        /**
        * HAL initializations such as enabling system tick and low level
        * hardware initialization.
        */
        HAL_Init();
        /* Configure the system clock TO 16 MHz*/
        SystemClockConfig(168U);
        /* Initialize the GPIO. Should be called before I2C_Init() */
        GPIO_Init();
        /* Initialize the UART */
        UartInit();
        /* Initialize the I2C. Should be called after GPIO_Init() */
        I2CInit();
        /* Configure the voltage regulators in proper mode */
        I2C1Init();       //For communicating with ADP5258 and ADP5061
}

/**
* @brief    Load Adux1020 default configuration
* @note     This function will load the default configuration to the device.
```

```
* @param    pnCfg - Pointer to the configuration array
* @retval   None
*/
uint8_t LoadDefaultConfig(uint32_t *pnCfg)
{
        uint8_t nRegAddr, nIdx, nRet = -1;
        uint16_t nRegData;

        if (pnCfg == 0)
                return nRet;
        nIdx = 0;

        while (1) {
                /* Read the address and data from the config */
                nRegAddr = (uint8_t)(pnCfg[nIdx] >> 16);
                nRegData = (uint16_t)(pnCfg[nIdx]);
                nIdx++;

                if (nRegAddr == 0xFF)
                        break;
                /* Load the data into the ADUX1020 registers */
                if (Adux1020DrvRegWrite(nRegAddr, nRegData) != ADUXDrv_SUCCESS) {
                        debug(":FAIL. Set Reg%d=%d\r\n", (int)nRegAddr, (int)nRegData);
                        return nRet;
                }
                nRegData = 0;
                Adux1020DrvRegRead(nRegAddr, &nRegData);
                debug("Config:, 0x%X ,  0x%X \r\n", nRegAddr, nRegData);
        }
        nRet = 0;
        debug(":PASS. Load Default Configure pass\r\n");
        return nRet;
}

/**
*  @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 ADUX registers and verify */
        while (regAddr != 0xFF) {
                if (Adux1020DrvRegRead(regAddr, &regData) != ADUXDrv_SUCCESS) {
                        debug("DCFG: Read Error reg(%0.2x)\r\n", regAddr);
                        return;
                } else if (regData != def_val) {
                        debug("DCFG: Read mismatch reg(%0.2x) (%0.2x != %0.2x)\r\n",
                            regAddr, def_val, regData);
                        return;
                }
                i++;
                regAddr = (uint8_t)(cfg[i] >> 16);
                def_val = (uint16_t)(cfg[i]);
        }
}

/**
 *  @brief  Adux1020 Driver bring up.
 *  @param  eModeOperation - Input to the function, specify which operation
 *                    mode need to be set.
 *
 *  @retval None
*/
void AduxDriverBringUp(Adux_Operation_Mode eModeOperation)
{
  uint32_t LoopCnt;
  uint16_t nRetValue = 0;
  uint16_t nAduxFifoLevelSize = 0, nAduxDataSetSize = 8U, nIntStatus;
  uint8_t value[8] = {0U}, getFifoData = 0U;

  switch(eModeOperation) {
  case SAMPLE_MODE:
     /* Set the fifo interrupt */
     Adux1020DrvSetInterrupt(FIFO_INT_ENABLE);
     /* Set Fifo level to trigger the interrupt */
     Adux1020DrvSetFifoLevel();
```

```
    /* Set the device operation to sample mode. The data can be collected now */
    Adux1020DrvSetOperationMode(ADUXDrv_MODE_SAMPLE);

    while (1) {
      /* Check if the data is ready */
      if(gnAduxDataReady)  {
        gnAduxDataReady = 0;
        /* Read the size of the data available in the FIFO */
        Adux1020DrvGetParameter(FIFO_LEVEL, &nAduxFifoLevelSize);
        /* Read the interrupt status */
        Adux1020DrvGetParameter(INT_STATUS, &nIntStatus);

        /* Enable the 32MHz Clock */
        if(nAduxFifoLevelSize >= nAduxDataSetSize) {
          Adux1020DrvEnable32MHzClock();
          getFifoData = 1;
        }
        /* Read the data from the FIFO and print them */
        while (nAduxFifoLevelSize >= nAduxDataSetSize) {
          nRetValue = Adux1020DrvReadFifoData(&value[0], nAduxDataSetSize);
          if (nRetValue == ADUXDrv_SUCCESS) {
            for (LoopCnt = 0; LoopCnt < 8; LoopCnt += 2)
              /* Byte swapping is needed to print Adux1020 data in proper format */
              debug("%u ", (value[LoopCnt] << 8) | value[LoopCnt + 1]);
            debug("%u\r\n", gnAduxTimeCurVal);
            //debug("Interrupt Status %04X \r\n", tmp);
            nAduxFifoLevelSize = nAduxFifoLevelSize - nAduxDataSetSize;
          }
        }
        /* Clear the interrupt status, inorder to occure next interrupt */
        if ((nIntStatus & IRQ_MASK_SAMPLE) != 0U) {
          Adux1020DrvRegWrite(REG_INT_STATUS, nIntStatus&IRQ_MASK_SAMPLE);
        }
        /* Disable the 32MHz Clock */
        if(getFifoData) {
          Adux1020DrvDisable32MHzClock();
          getFifoData = 0;
        }
              }
    }
    break;
  case PROXIMITY_MODE:
    /* Set the proximity interrupt */
    Adux1020DrvSetInterrupt(PROXIMITY_INT_ENABLE);
```

```
    /* Set the device operation to sample mode. The data can be collected now */
    Adux1020DrvSetOperationMode(ADUXDrv_MODE_PROXIMITY);
    while (1) {
     /* Check if the data is ready */
     if(gnAduxDataReady) {
      gnAduxDataReady = 0;
      /* Read the size of the data available in the FIFO */
      Adux1020DrvGetParameter(FIFO_LEVEL, &nAduxFifoLevelSize);
      /* Read the interrupt status */
      Adux1020DrvGetParameter(INT_STATUS, &nIntStatus);

      /* Enable the 32MHz Clock */
      if(nAduxFifoLevelSize >= nAduxDataSetSize) {
       Adux1020DrvEnable32MHzClock();
       getFifoData = 1;
      }
      /* Read the data from the FIFO and print them */
      while (nAduxFifoLevelSize >= nAduxDataSetSize) {
       nRetValue = Adux1020DrvReadFifoData(&value[0], nAduxDataSetSize);
       if (nRetValue == ADUXDrv_SUCCESS) {
        LoopCnt = 0;
        debug("%u ", (value[LoopCnt] << 8) | value[LoopCnt + 1]);
        debug("%u\r\n", gnAduxTimeCurVal);
        debug("Interrupt Status %04X \r\n", nIntStatus);
        nAduxFifoLevelSize = nAduxFifoLevelSize - nAduxDataSetSize;
       }
      }
      /* Clear the interrupt status, inorder to occure next interrupt */
      if ((nIntStatus & IRQ_MASK_PROXIMITY) != 0U) {
       Adux1020DrvRegWrite(REG_INT_STATUS,
nIntStatus&IRQ_MASK_PROXIMITY);
      }
      /* Disable the 32MHz Clock */
      if(getFifoData) {
       Adux1020DrvDisable32MHzClock();
       getFifoData = 0;
      }
     }
    }
    break;
  default:
    break;
  }
}
```