This version (27 Jun 2022 22:49) was ***approved*** by Brandon Bushey.
The Previously approved version (31 Jan 2022 17:39) is available.

EVAL-ADPD410X-ARDZ User Guide

EVAL-ADPD4100-ARDZ and **EVAL-ADPD4101-ARDZ** are simple, Arduino form-factor breakout boards for developing **ADPD4100** and **ADPD4101** applications, respectively. The **ADPD4101**, interfaced through I2C, and the **ADPD4100**, interfaced through SPI, are highly versatile, multimodal sensor front ends, stimulating up to eight light emitting diodes (LEDs) and measuring the return signal on up to eight separate current inputs. There are a number of other evaluation platforms for these devices, including the **EVAL-ADPD4100Z-PPG**, optimized for photoplethysmograph applications, and the reference design **CN0503**, optimized for optical liquid analysis applications such as colorimetry, turbidity, and fluorescence. The EVAL-ADPD410X-ARDZ boards come in handy for adapting these evaluation boards to meet specific application requirements, as well as for "ground up" development of new applications.

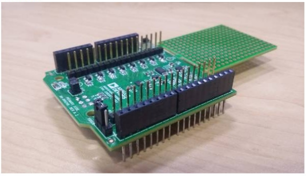


Table of Contents

- EVAL-ADPD410X-ARDZ User Guide
  - Features
  - Device Driver and Support
  - Materials Needed
  - Hardware Setup
    - Jumper Configuration
    - Prototyping Connectors
    - General Connection
  - Driver / Firmware Setup
    - Dragging and Dropping the Hex File to the Daplink Drive
    - Using CrossCore Embedded Studio
  - Software Setup
    - Installation
    - Connection
    - Reading and Configuring the Device from Command Line
    - IIO Oscilloscope
    - Python and PyADI-IIO
  - Schematic, PCB Layout, Bill of Materials
  - Additional Information and Useful Links

Features

- 8 LED excitation channels with high-voltage extension circuitry
- 8 Photodiode inputs

Device Driver and Support

A no-OS device driver and an example program are provided, targeting the EVAL-ADICUP3029 platform. The ADICUP3029 example application uses the ADPD410x no-OS driver and emulates the Linux IIO framework through the tinyiiod daemon library. The application communicates with the host computer via the serial backend over a USB-UART physical connection. This facilitates rapid application development on a host computer, independent from embedded code development.

Similarly, utility software (such as iio_info, IIO Oscilloscope, PyADI-IIO, etc.) typically associated with Linux systems can be used with this no-OS implementation.

Materials Needed

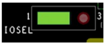
- EVAL-ADICUP3029
- EVAL-ADPD4100-ARDZ or EVAL-ADPD4101-ARDZ
- Micro USB to USB cable
- PC or laptop with a USB port

Hardware Setup

Jumper Configuration

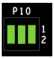
There are two shunt-configurable jumpers and three types of solder jumpers on both **EVAL-ADPD4100-ARDZ** and **EVAL-ADPD4101-ARDZ** boards.

I/O Logic Voltage (IOSEL) Shunt Positions

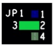
Correct Shunt Position	Layout Picture
Shorted Pin 1 and 2	

Onboard LED and Photodiode (P10) Shunt Positions

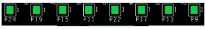
The onboard LED and photodiode are by default connected to LED1A and PD1A, respectively. **When using external LEDs and sensors on this channel, make sure to remove this connection using jumper header P10.**

Correct Shunt Position	Layout Picture
Shorted Pin 1 and 2, Shorted Pin 3 and 4, Shorted Pin 5 and 6	

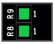
LED Supply Voltage (JP1) Solder Positions

Correct Shunt Position	Layout Picture
Shorted Pin 2 and 3	

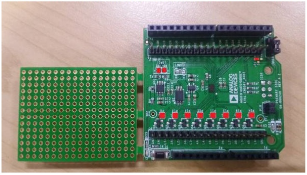
LED Driver Connection P9, P11, P13, P15, P17, P19, P22, P24

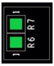
Correct Shunt Position	Layout Picture
Shorted	 P24 P19 P15 P11 P22 P17 P13 P9

SPI or I2C Interface

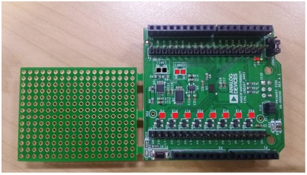
Board	Shorted Resistors	Layout Picture
EVAL-ADPD4100-ARDZ (SPI)	Shorted R8 and R9, Open R6 and R7	

Below is a photo of the EVAL-ADPD4100-ARDZ (SPI) board with all the correct shunt and solder jumper connections.



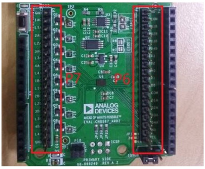
Board	Shorted Resistors	Layout Picture
EVAL-ADPD4101-ARDZ (I2C)	Shorted R6 and R7, Open R8 and R9	

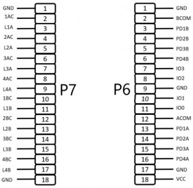
Below is a photo of the EVAL-ADPD4101-ARDZ (I2C) board with all the correct shunt and solder jumper connections.



Prototyping Connectors

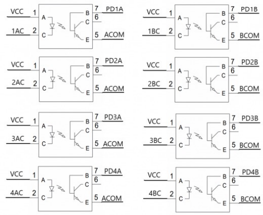
The board has two parallel 18-pin, 100-mil pitch male connectors, which give access to the LED driver channels, the photodiode inputs, custom I/O pins from the AFE, and supply voltage for the LED. The user can use this along with the break-away protoboard to implement a custom circuit for testing. The pin assignment and functions are shown below.




P7
GND 1
1AC 2
L1A 3
2AC 4
L2A 5
3AC 6
L3A 7
4AC 8
L4A 9
1BC 10
L1B 11
2BC 12
L2B 13
3BC 14
L3B 15
4BC 16
L4B 17
GND 18
P6
1 GND
2 BCOM
3 PD1B
4 PD2B
5 PD3B
6 PD4B
7 IO3
8 IO2
9 GND
10 IO1
11 IO0
12 ACOM
13 PD1A
14 PD2A
15 PD3A
16 PD4A
17 GND
18 VCC

Pins labeled **XYC** (where X refers to LED 1, 2, 3, or 4, and Y refers to channel A or B) denote connections to the LED cathode which are voltage protected via transistors to the ADPD4100/1 LED inputs (denoted by **LXY**). It is recommended to connect LED cathodes to these pins instead of connecting directly to **LXY** pins. **PDXY** (X refers to photodiode 1, 2, 3, or 4 and Y refers to channel A or B) pins denote photodiode signal inputs to the AFE. **ACOM** and **BCOM** pins refer to the common cathode bias output for photodiode sensors. These should be connected to the cathodes of photodiodes in the matching channel (for example, photodiodes connected to PD1A, PD2A, PD3A, and PD4A should have their cathodes connected to ACOM).

A simple circuit for testing LED driver outputs and photodiode current sensing using the break-away prototype board can easily be set up using optocouplers, as shown below.

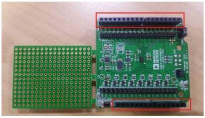

VCC 1
1AC 2
A
C
B
C
E
7 PD1A
6
5 ACOM
VCC 1
1BC 2
7 PD1B
6
5 BCOM
VCC 1
2AC 2
7 PD2A
6
5 ACOM
VCC 1
2BC 2
7 PD2B
6
5 BCOM
VCC 1
3AC 2
7 PD3A
6
5 ACOM
VCC 1
3BC 2
7 PD3B
6
5 BCOM
VCC 1
4AC 2
7 PD4A
6
5 ACOM
VCC 1
4BC 2
7 PD4B
6
5 BCOM

General Connection

- Set the following EVAL-ADICUP3029 switches according to their configuration on the table.

Switch	Configuration
UART (S2)	USB
POWER (S5)	WALL/USB

- Connect the EVAL-ADPD4100-ARDZ or EVAL-ADPD4101-ARDZ to the EVAL-ADICUP3029 using the headers shown. See correct Jumper Configuration for both evaluation boards here.

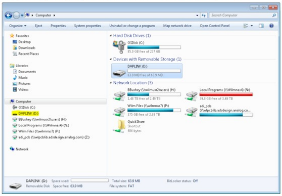


Driver / Firmware Setup

There are two basic ways to program the ▶ EVAL-ADICUP3029 with the software for both boards.

Dragging and Dropping the Hex File to the Daplink Drive

1. Ensure that the ▶ EVAL-ADICUP3029 board switches are set to the correct configuration, as detailed in General Connection.
2. Connect the ▶ EVAL-ADICUP3029 board to the PC or laptop using the Micro USB to USB cable.
3. Drag and Drop the appropriate .hex file from the list below to the Daplink Drive.


Computer
Search Computer
Organize
Eject
Properties
System properties
Uninstall or change a program
Map network drive
Open Control Panel
Favorites
Desktop
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Videos
Computer
OSDisk (C:)
DAPLINK (D:)
BBushey (\\wilmusr2\users) (H:)
Local Programs (\\Wilmnas4) (N:)
Wilm Files (\\wilmnas7) (P:)
adi_pcb (\\wlpcblib.adsdesign.analog.com) (Z:)
Network
Hard Disk Drives (1)
OSDisk (C:)
95.8 GB free of 237 GB
Devices with Removable Storage (1)
DAPLINK (D:)
63.9 MB free of 63.9 MB
Network Location (5)
BBushey (\\wilmusr2\users) (H:)
1.40 TB free of 2.49 TB
Local Programs (\\Wilmnas4) (N:)
39.8 GB free of 3.49 TB
Wilm Files (\\wilmnas7) (P:)
375 GB free of 2.69 TB
adi_pcb (\\wlpcblib.adsdesign.analog.co...
QuickShare
Shortcut
486 bytes
DAPLINK (D:) Space used:
Removable Disk Space free: 63.9 MB
Total size: 63.9 MB
File system: FAT
BitLocker status: Off

4. The drive will unmount once the .hex file is dropped and wait for it to reappear. Once it does, press the reset button of the ▶ **EVAL-ADICUP3029** to ensure that the microcontroller is updated.

> Pre-built hex files can be found here:
>
> - EVAL-ADPD4100-ARDZ .Hex File (ADuCM3029_demo_adpd410x_spi.hex)
> - EVAL-ADPD4101-ARDZ .Hex File (ADuCM3029_demo_adpd410x_i2c.hex)
>
> The latest source code can be found here:
>
> - EVAL-ADICUP3029/tree/master/projects/ADuCM3029_demo_adpd410x.

Using CrossCore Embedded Studio

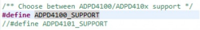
1. Acquire a copy of the project source files of the EVAL-ADPD410X-ARDZ by downloading the source files directly from the repository at EVAL-ADICUP3029/tree/master/projects/ADuCM3029_demo_adpd410x or you can clone the entire EVAL-ADICUP3029 repository and check the projects folder for ADuCM3029_demo_adpd410x.
2. Open CrossCore Embedded Studio and import the project into your workspace, as detailed in the cces_user_guide. This allows you to edit the software to fit your requirements.

   > ⚠ If this is your first time using CrossCore Embedded Studio, check the user guide to get started.

3. Once ready, you can opt to generate your own .hex file and use the first method to program the ▶ **EVAL-ADICUP3029** or you can use a debug session by following the quickstart guide.

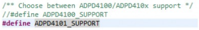
CrossCore Project Header File Configuration

- Select which board the project will support when built using app_config.h.
  - For EVAL-ADPD4100-ARDZ



```
/** Choose between ADPD4100/ADPD410x support */
#define ADPD4100_SUPPORT
//#define ADPD4101_SUPPORT
```

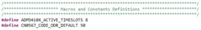
  - For EVAL-ADPD4101-ARDZ



```
/** Choose between ADPD4100/ADPD410x support */
//#define ADPD4100_SUPPORT
#define ADPD4101_SUPPORT
```

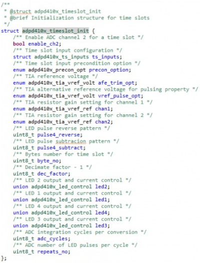
You can configure the default timeslots and other settings of the ▶ ADPD4100 or ▶ ADPD4101 using adpd410x_app_config.h.

- You can set the default number of active timeslots and output data rate.



```
/******************************************************************************/
/********************** Macros and Constants Definitions ********************/
/******************************************************************************/
#define ADPD410X_ACTIVE_TIMESLOTS 8
#define CN0567_CODE_ODR_DEFAULT 50
```

- Initial timeslot settings are configured using the **adpd410x_timeslot_init** data structure, as shown below:

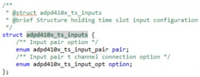


```
/**
 * @struct adpd410x_timeslot_init
 * @brief Initialization structure for time slots
 */
struct adpd410x_timeslot_init {
    /** Enable ADC channel 2 for a time slot */
    bool enable_ch2;
    /** Time slot input configuration */
    struct adpd410x_ts_inputs ts_inputs;
    /** Time slot input precondition option */
    enum adpd410x_precon_opt precon_option;
    /** TIA reference voltage */
    enum adpd410x_tia_vref_volt afe_trim_opt;
    /** TIA alternative reference voltage for pulsing property */
    enum adpd410x_tia_vref_volt vref_pulse_opt;
    /** TIA resistor gain setting for channel 1 */
    enum adpd410x_tia_vref_ref chan1;
    /** TIA resistor gain setting for channel 2 */
    enum adpd410x_tia_vref_ref chan2;
    /** LED pulse reverse pattern */
    uint8_t pulse4_reverse;
    /** LED pulse subtracion pattern */
    uint8_t pulse4_subtract;
    /** Bytes number for time slot */
    uint8_t byte_no;
    /** Decimate factor - 1 */
    uint8_t dec_factor;
    /** LED 2 output and current control */
    union adpd410x_led_control led2;
    /** LED 1 output and current control */
    union adpd410x_led_control led1;
    /** LED 4 output and current control */
    union adpd410x_led_control led4;
    /** LED 3 output and current control */
    union adpd410x_led_control led3;
    /** ADC integration cycles per conversion */
    uint8_t adc_cycles;
    /** ADC number of LED pulses per cycle */
    uint8_t repeats_no;
};
```

- **Enabling ADC channel 2**

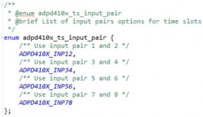
  Each timeslot can use the two ADC input channels of the ADPD4100/1. By default, the timeslot only uses channel 1. Setting this to true will enable channel 2.

- **Timeslot Inputs**



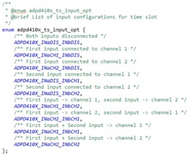
```
/**
 * @struct adpd410x_ts_inputs
 * @brief Structure holding time slot input configuration
 */
struct adpd410x_ts_inputs {
    /** Input pair option */
    enum adpd410x_ts_input_pair pair;
    /** Input pair t channel connection option */
    enum adpd410x_ts_input_opt option;
};
```

  Each timeslot input can be set to use a pair of input pins of the ADPD4100/1. Below are possible pair options:



```
/**
 * @enum adpd410x_ts_input_pair
 * @brief List of input pairs options for time slots
 */
enum adpd410x_ts_input_pair {
    /** Use input pair 1 and 2 */
    ADPD410X_INP12,
    /** Use input pair 3 and 4 */
    ADPD410X_INP34,
    /** Use input pair 5 and 6 */
    ADPD410X_INP56,
    /** Use input pair 7 and 8 */
    ADPD410X_INP78
};
```

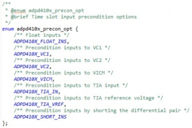
- Each of the input pins can be disabled or routed to any or both of the channels. Below are the possible configurations of the input pins.



```
/**
 * @enum adpd410x_ts_input_opt
 * @brief List of input configurations for time slot
 */
enum adpd410x_ts_input_opt {
    /** Both inputs disconnected */
    ADPD410X_INaDIS_INbDIS,
    /** First input connected to channel 1 */
    ADPD410X_INaCH1_INbDIS,
    /** First input connected to channel 2 */
    ADPD410X_INaCH2_INbDIS,
    /** Second input connected to channel 1 */
    ADPD410X_INaDIS_INbCH1,
    /** Second input connected to channel 2 */
    ADPD410X_INaDIS_INbCH2,
    /** First input -> channel 1, second input -> channel 2 */
    ADPD410X_INaCH1_INbCH2,
    /** First input -> channel 2, second input -> channel 1 */
    ADPD410X_INaCH2_INbCH1,
    /** First input + Second input -> channel 1 */
    ADPD410X_INaCH1_INbCH1,
    /** First input + Second input -> channel 2 */
    ADPD410X_INaCH2_INbCH2
};
```

- **Timeslot Input Preconditioning**

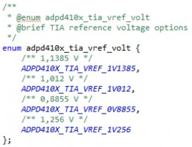
  Timeslot inputs have programmable connections, which precondition the sensor to set operating conditions before sampling. Below are the possible options:



```
/**
 * @enum adpd410x_precon_opt
 * @brief Time slot input precondition options
 */
enum adpd410x_precon_opt {
    /** Float inputs */
    ADPD410X_FLOAT_INS,
    /** Precondition inputs to VC1 */
    ADPD410X_VC1,
    /** Precondition inputs to VC2 */
    ADPD410X_VC2,
    /** Precondition inputs to VICM */
    ADPD410X_VICM,
    /** Precondition inputs to TIA input */
    ADPD410X_TIA_IN,
    /** Precondition inputs to TIA reference voltage */
    ADPD410X_TIA_VREF,
    /** Precondition inputs by shorting the differential pair */
    ADPD410X_SHORT_INS
};
```

- **TIA Reference Voltage**

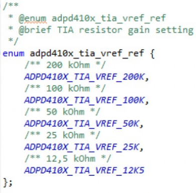
  The reference voltage of the Trans-Impedance Amplifier (TIA) is configurable. Below are the possible values:



```
/**
 * @enum adpd410x_tia_vref_volt
 * @brief TIA reference voltage options
 */
enum adpd410x_tia_vref_volt {
    /** 1,1385 V */
    ADPD410X_TIA_VREF_1V1385,
    /** 1,012 V */
    ADPD410X_TIA_VREF_1V012,
    /** 0,8855 V */
    ADPD410X_TIA_VREF_0V8855,
    /** 1,256 V */
    ADPD410X_TIA_VREF_1V256
};
```

- **TIA Gain Resistor**

  The gain resistor used by the TIA is configurable for each input channel. Below are the possible values:



```
/**
 * @enum adpd410x_tia_vref_ref
 * @brief TIA resistor gain setting
 */
enum adpd410x_tia_vref_ref {
    /** 200 kOhm */
    ADPD410X_TIA_VREF_200K,
    /** 100 kOhm */
    ADPD410X_TIA_VREF_100K,
    /** 50 kOhm */
    ADPD410X_TIA_VREF_50K,
    /** 25 kOhm */
    ADPD410X_TIA_VREF_25K,
    /** 12,5 kOhm */
    ADPD410X_TIA_VREF_12K5
};
```

- **Multiple Pulses and Integrator Chopping** The ADPD4100/1 is capable of improving SNR using multiple pulses per sample and integrator chopping, which have configurable settings.
    - **4-Pulse Reverse Pattern**

      Each pulse from the LED in a set of 4 can be configured to either have an on-off or off-on integrator chopping sequence. This is a 4-bit value, 1 bit for each pulse. Setting a bit reverses the integrator chopping sequence for that pulse.
    - **4-Pulse Subtract Pattern**

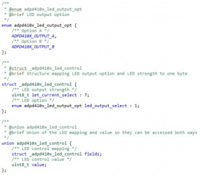
      The mathematical operation performed on a digitized ADC sample can be set to addition or subtraction for each pulse in a set of 4. This is a 4-bit value, 1 bit for each pulse. Setting a bit negates the operation for that pulse.
- **Byte Number**

  This sets the number of data bytes used in the timeslot.
- **Decimation Factor**

  The decimation factor sets the number of time slot values used in the final sample.

  ```
  output data rate = sample rate / (decimation factor - 1)
  ```
- **LED Output**

  The 4 LED outputs have configurable output current and can be set to either channel A or channel B only. You can define the LED value directly or through fields. The first 7 bits are for the output current, which scale from 1.5 mA to 200 mA for 0x01 to 0x7F. The last bit is for the output channel. Setting this bit selects channel B while clearing selects channel A.



```
/**
 * @enum adpd410x_led_output_opt
 * @brief LED output option
 */
enum adpd410x_led_output_opt {
    /** Option A */
    ADPD410X_OUTPUT_A,
    /** Option B */
    ADPD410X_OUTPUT_B
};

/**
 * @struct _adpd410x_led_control
 * @brief Structure mapping LED output option and LED strength to one byte
 */
struct _adpd410x_led_control {
    /** LED output strength */
    uint8_t let_current_select : 7;
    /** LED option */
    enum adpd410x_led_output_opt led_output_select : 1;
};

/**
 * @union adpd410x_led_control
 * @brief Union of the LED mapping and value so they can be accessed both ways
 */
union adpd410x_led_control {
    /** LED control mapping */
    struct _adpd410x_led_control fields;
    /** LED control value */
    uint8_t value;
};
```

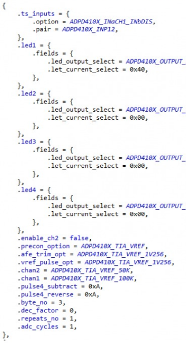
- **ADC Cycles**

  This sets the number of integration cycles per ADC conversion. This value can range from 0x01 to 0xFF.
- **Number of Repeats**

  This sets the number of repeat ADC conversions\\. This value can range from 0x01 to 0xFF\\.

  ```
  total number of pulses = ADC cycles X Number of Repeats
  ```

Here is an example timeslot setting used in the pre-built hex files:



```
{
    .ts_inputs = {
        .option = ADPD410X_INaCH1_INbDIS,
        .pair = ADPD410X_INP12,
    },
    .led1 = {
        .fields = {
            .led_output_select = ADPD410X_OUTPUT_A,
            .let_current_select = 0x40,
        },
    },
    .led2 = {
        .fields = {
            .led_output_select = ADPD410X_OUTPUT_B,
            .let_current_select = 0x00,
        },
    },
    .led3 = {
        .fields = {
            .led_output_select = ADPD410X_OUTPUT_B,
            .let_current_select = 0x00,
        },
    },
    .led4 = {
        .fields = {
            .led_output_select = ADPD410X_OUTPUT_B,
            .let_current_select = 0x00,
        },
    },
    .enable_ch2 = false,
    .precon_option = ADPD410X_TIA_VREF,
    .afe_trim_opt = ADPD410X_TIA_VREF_1V256,
    .vref_pulse_opt = ADPD410X_TIA_VREF_1V256,
    .chan2 = ADPD410X_TIA_VREF_50K,
    .chan1 = ADPD410X_TIA_VREF_100K,
    .pulse4_subtract = 0xA,
    .pulse4_reverse = 0xA,
    .byte_no = 3,
    .dec_factor = 0,
    .repeats_no = 1,
    .adc_cycles = 1,
},
```

Software Setup

Installation

To communicate with the device from the PC or laptop using IIO commands, install the Libiio package by following the guide in the repository: libiio/releases. The method is different for Windows or Linux operating systems.

Connection

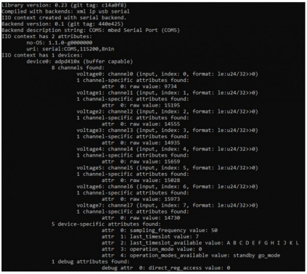
The device must be able to create a context. Context creation in the software depends on the backend used to connect the device. This guide covers the device communication using the currently supported platform, a serial backend through a USB-UART connection. A simple way of checking, if the device is connected, is through the **iio_info** command. Specifically, it reports all IIO attributes of a detected device and context. To do this, simply enter the below command to a terminal or command line.

```
iio_info -u serial:<serial port>
```

Examples:

- In a Windows machine, you can check the port of your ADICUP3029 via Device Manager in the Ports (COM & LPT) section. If your device is in COM4, use **serial:COM4** as your URI (e.g., *iio_info -u serial:COM4*).
- In a Unix-based machine, you will see it under the /dev/ directory in this format “ttyUSBn”, where n is a number depending on how many serial USB devices are attached. If you see that your device is ttyUSB0, use **serial:/dev/ttyUSB0** as your URI. (e.g., *iio_info -u serial:/dev/ttyUSB0*).

An example output of this command should look like the one below:



```
Library version: 0.23 (git tag: c14a0f8)
Compiled with backends: xml ip usb serial
IIO context created with serial backend.
Backend version: 0.1 (git tag: 440e425)
Backend description string: COM5: mbed Serial Port (COM5)
IIO context has 2 attributes:
        no-OS: 1.1.0-g0000000
        uri: serial:COM5,115200,8n1n
IIO context has 1 devices:
        device0: adpd410x (buffer capable)
                8 channels found:
                        voltage0: channel0 (input, index: 0, format: le:u24/32>>0)
                        1 channel-specific attributes found:
                                attr  0: raw value: 9734
                        voltage1: channel1 (input, index: 1, format: le:u24/32>>0)
                        1 channel-specific attributes found:
                                attr  0: raw value: 15195
                        voltage2: channel2 (input, index: 2, format: le:u24/32>>0)
                        1 channel-specific attributes found:
                                attr  0: raw value: 14555
                        voltage3: channel3 (input, index: 3, format: le:u24/32>>0)
                        1 channel-specific attributes found:
                                attr  0: raw value: 14935
                        voltage4: channel4 (input, index: 4, format: le:u24/32>>0)
                        1 channel-specific attributes found:
                                attr  0: raw value: 15659
                        voltage5: channel5 (input, index: 5, format: le:u24/32>>0)
                        1 channel-specific attributes found:
                                attr  0: raw value: 15028
                        voltage6: channel6 (input, index: 6, format: le:u24/32>>0)
                        1 channel-specific attributes found:
                                attr  0: raw value: 15973
                        voltage7: channel7 (input, index: 7, format: le:u24/32>>0)
                        1 channel-specific attributes found:
                                attr  0: raw value: 14730
                5 device-specific attributes found:
                                attr  0: sampling_frequency value: 50
                                attr  1: last_timeslot value: 7
                                attr  2: last_timeslot_available value: A B C D E F G H I J K L
                                attr  3: operation_mode value: 0
                                attr  4: operation_modes_available value: standby go_mode
                1 debug attributes found:
                                debug attr  0: direct_reg_access value: 0
```

Reading and Configuring the Device from Command Line

Using the **iio_attr** command from the Libiio package, you can read and configure the device.

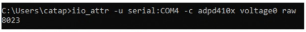
Reading Raw Channel Outputs from the Device

The -c option of the **iio_attr** command allows reading of the individual raw channel outputs. Specifically, this reads the raw attribute of the specified channel of the specified device in the context of the specified URI. The command follows the format shown below.

```
iio_attr -u <URI> -c <DEVICENAME> <CHANNELNAME> <ATTRIBUTE>
E.g., for a Windows Machine
iio_attr -u serial:COM4  -c adpd410x voltage0 raw
```

- <URI> specifies the URI similar to the one used in Connection.
- <DEVICENAME> specifies the device name, which is **adpd410x**.
- <CHANNELNAME> specifies the channel name, which are formatted as voltageX where X can be from 0 to 7.
- <ATTRIBUTE> specifies the name of the channel attribute, which is raw. This is a read-only attribute.

An example output of this command should look like the one below:



```
C:\Users\catap>iio_attr -u serial:COM4 -c adpd410x voltage0 raw
8023
```

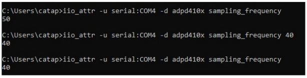
Reading and Writing Device Attributes

The -d option of the **iio_attr** command allows reading or writing (only for specific items) of device attributes. For example, the *sampling_frequency* attribute is readable and writeable. The command to read and write to this attribute is shown below:

```
For reading, use iio_attr -u <URI> -d <DEVICENAME> <ATTRIBUTE>
For writing, use iio_attr -u <URI> -d <DEVICENAME> <ATTRIBUTE> <VALUE>
e.g., for a Windows Machine
iio_attr -u serial:COM4 -d adpd410x sampling_frequency 40
```

- <URI> specifies the URI similar to the one used in Connection.
- <DEVICENAME> specifies the device name which is **adpd410x**.
- <ATTRIBUTE> specifies the name of the device attribute which is **sampling_frequency**.

An example output of this command should look like the one below:

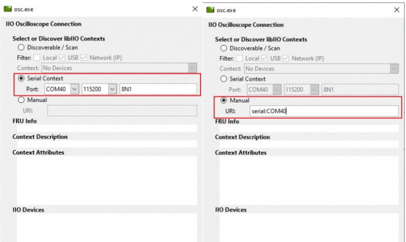


```
C:\Users\catap>iio_attr -u serial:COM4 -d adpd410x sampling_frequency
50

C:\Users\catap>iio_attr -u serial:COM4 -d adpd410x sampling_frequency 40
40

C:\Users\catap>iio_attr -u serial:COM4 -d adpd410x sampling_frequency
40
```

IIO Oscilloscope

Make sure to download/update to the latest version of IIO-Oscilloscope found on this link https://github.com/analogdevicesinc/iio-oscilloscope/releases.

1. Install and start IIO-Oscilloscope. There are two options you can use to select IIO contexts. First, you can use the Serial option and input the correct port settings of the board from the Device Manager. Another way is by manually entering the URI used in Connection.



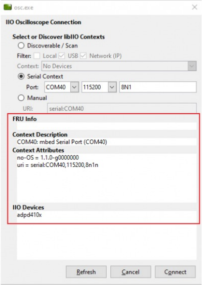
● Serial Context
Port: COM40 115200 8N1

● Manual
URI: serial:COM40

osc.exe
IIO Oscilloscope Connection
Select or Discover libIIO Contexts
○ Discoverable / Scan
Filter: ☐ Local ☑ USB ☑ Network (IP)
Context: No Devices
○ Serial Context
○ Manual
URI:
FRU Info
Context Description
Context Attributes
IIO Devices

Refresh
Cancel
Connect
Refresh
Cancel
Connect

2. Press Refresh to display available IIO Devices, and once adpd410x is detected, press Connect. It may take several presses to Connect before the software proceeds and opens the Debug Panel and Waveform Panel\\.


osc.exe
IIO Oscilloscope Connection
Select or Discover libIIO Contexts
Refresh
Cancel
Connect

○ Discoverable / Scan
Filter: ☐ Local ☑ USB ☑ Network (IP)
Context: No Devices
● Serial Context
Port: COM40 115200 8N1
○ Manual
URI: serial:COM40

FRU Info

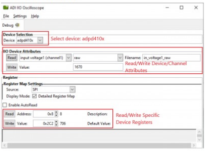
Context Description
COM40: mbed Serial Port (COM40)

Context Attributes
no-OS = 1.1.0-g0000000
uri = serial:COM40,115200,8n1n

IIO Devices
adpd410x

Debug Panel

In the Debug Panel, you can directly access the device/channel attributes and even the device registers. Remember to select first the adpd410x in the Device Selection section.


ADI IIO Oscilloscope
File
Settings
Help
Debug
Select device: adpd410x
Read/Write Device/Channel Attributes
Read/Write Specific Device Registers
Read
Write
Read
Write

Device Selection
Device: adpd410x

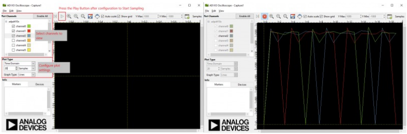
IIO Device Attributes
input voltage1 (channel1) raw Filename: in_voltage1_raw
Value: 1670

Register
Register Map Settings
Source: SPI
Display Mode: ☑ Detailed Register Map
☐ Enable AutoRead
Address: 0x8 8 Description:
Value: 0x2C2 706 Default Value:

Waveform Panel

The Waveform panel, also known as the Capture window, displays the real-time waveform of selected photodiode channels of the ADPD410x. Select the desired channels to display in the upper left section. You can also edit the plot settings in the left section.

> You cannot use the Debug Panel and the Waveform Panel simultaneously. Using the Waveform Panel will freeze the Debug Panel.


ADI IIO Oscilloscope - Capture1
File
Edit
View
Plot Channels
Enable All
Press the Play Button after configuration to Start Sampling
Auto scale
Show grid
Y Max
Y Min
Samples
adpd410x
channel0
channel1
channel2
channel3
channel4
channel5
Select channels to view
Plot Type
Time Domain
20
Samples
Graph Type: Lines
Configure plot settings
Info
Markers
Devices
ANALOG DEVICES
ADI IIO Oscilloscope - Capture1
ANALOG DEVICES

> For best device operation and waveform captured, it is recommended to use a time domain plot with 20 samples.

Python and PyADI-IIO

PyADI-IIO is a python abstraction module for ADI hardware with IIO drivers to make them easier to use. This module provides device-specific APIs built on top of the current libIIO python bindings. These interfaces try to match the driver naming as much as possible without the need to understand the complexities of libIIO and IIO.

Installing the Packages

> PyADI-IIO requires a Python installed on your computer. It is recommended to install Python 3.7 or higher.

Install PyADI-IIO using one of the methods in PyADI-IIO\\.

There are two example scripts found in the examples folder in PyADI-IIO. To run both examples, the following packages are required: pyqtgraph, scipy, PyQt5, matplotlib.

If you are using pip, you can install all of the PyADI-IIO, as well as the example script dependencies, by following these steps:

1. Clone or download the pyadi-iio repository pyadi-iio/
2. Open command prompt or terminal and navigate to the *pyadi-iio* directory.
3. Enter the following commands:

```
...\pyadi-iio\>pip install -r requirements.txt
...\pyadi-iio\>pip install -r examples/requirements_adiplot.txt
...\pyadi-iio\>python setup.py install
```

> One of the packages in *requirements_adiplot.txt* is the PyQt5. If you already have a pre-installed PyQt5 prior to the installation of the packages, it is suggested that you uninstall the said package in the virtual environment. Duplicate installations may sometimes cause errors that inhibit the system from displaying the real-time plot. This can easily be done by inputting the command: *pip uninstall PyQt5* while the virtual environment is active.

Running the Examples

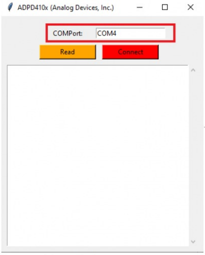
There are three example scripts for the ADPD410x found in pyadi-iio/. The first simply reads from the photodiode channels, the second plots specified photodiode channels, and the third tests the board separately, with its onboard LED and photodiode and with a test setup built from the simple example circuit from Prototyping Connectors.

For example 1, follow these steps:

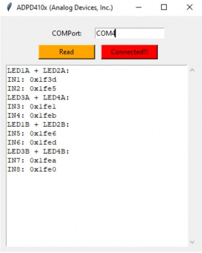
1. Connect the EVAL-ADPD4100-ARDZ or EVAL-ADPD4101-ARDZ to the EVAL-ADICUP3029.
2. Connect the EVAL-ADICUP3029 to the PC using the micro-USB cable and note the serial port from the Device Manager as in Connection.
3. Open command prompt or terminal and navigate to the examples folder inside the downloaded or cloned *pyadi-iio* directory.
4. Run the example using the command:

   ```
   ...\pyadi-iio\examples>python adpd410x_example.py
   ```

5. Input the noted serial port and press *Connect*

   
   ADPD410x (Analog Devices, Inc.)
   COMPort:
   COM4
   Read
   Connect

6. Once connected, press *Read*

   
   ADPD410x (Analog Devices, Inc.)
   COMPort:
   COM4
   Read
   Connected!!!

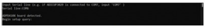
   ```
   LED1A + LED2A:
   IN1: 0x1f3d
   IN2: 0x1fe5
   LED3A + LED4A:
   IN3: 0x1fe1
   IN4: 0x1feb
   LED1B + LED2B:
   IN5: 0x1fe6
   IN6: 0x1fed
   LED3B + LED4B:
   IN7: 0x1fea
   IN8: 0x1fe0
   ```

For example 2, follow these steps:

1. Connect the EVAL-ADPD4100-ARDZ or EVAL-ADPD4101-ARDZ to the EVAL-ADICUP3029.
2. Connect the EVAL-ADICUP3029 to the PC using the micro-USB cable and note the serial port from the Device Manager as in Connection.
3. Open command prompt or terminal and navigate to the examples folder inside the downloaded or cloned *pyadi-iio* directory.
4. Run the example script using the command:

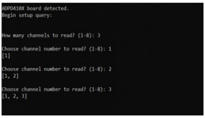
   ```
   ...\pyadi-iio\examples>python adpd410x_plot.py
   ```

5. The script will ask for a serial port. Input the noted serial port and press Enter. In cases when the board is not found, press the reset button (S1) on the EVAL-ADPD4100-ARDZ or EVAL-ADPD4101-ARDZ and input the noted serial port again.

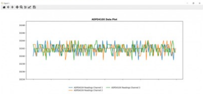
   ```
   Input Serial line (e.g. if ADICUP3029 is connected to COM7, input 'COM7' )
   Serial line:COM6

   ADPD410X board detected.
   Begin setup query:
   ```

6. When the board is detected, you will be asked to specify the number of channels (1 to 8) you want to read. Then, you need to specify the desired channel numbers (1 to 8).

   ```
   ADPD410X board detected.
   Begin setup query:


   How many channels to read? (1-8): 3

   Choose channel number to read? (1-8): 1
   [1]

   Choose channel number to read? (1-8): 2
   [1, 2]

   Choose channel number to read? (1-8): 3
   [1, 2, 3]
   ```

7. A plot will appear showing the specified channels. You have the option to save a copy of the displayed waveform at any point in time using the matplotlib controls at the top.

   
   ADPD410X Data Plot
   ADPD410X Readings Channel 1
   ADPD410X Readings Channel 2
   ADPD410X Readings Channel 3

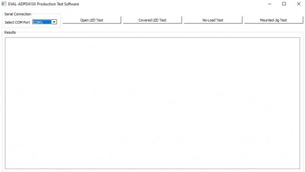
For example 3, follow these steps:

1. Connect the EVAL-ADPD4100-ARDZ or EVAL-ADPD4101-ARDZ to the EVAL-ADICUP3029.
2. Connect the EVAL-ADICUP3029 to the pc using the micro-USB cable and note the serial port from the Device Manager as in Connection.
3. Open command prompt or terminal and navigate to the examples folder inside the downloaded or cloned *pyadi-iio* directory.

4. Run the example script using the command:

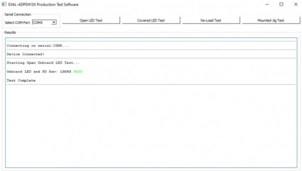
```
...\pyadi-iio\examples>python adpd410x_test.py
```

5. A GUI window will appear, as shown below. There are four buttons at the top right for each test namely, open onboard LED test, covered onboard LED test, no load test, and mounted jig test. **Before pressing a button to start the test, select the noted COM port on the dropdown list at the top left.**


EVAL-ADPD410X Production Test Software
Serial Connection
Select COM Port
COM1
Open LED Test
Covered LED Test
No-Load Test
Mounted-Jig Test
Results

- **Open Onboard LED Test**

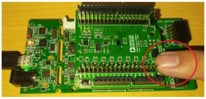
  The test samples raw ADC values from the onboard photodiode and checks whether it is consistent with the standard. Make sure that all shunts on jumper header P10 are present and connected. A sample passing result is shown below


EVAL-ADPD410X Production Test Software
Serial Connection
Select COM Port
COM5
Open LED Test
Covered LED Test
No-Load Test
Mounted-Jig Test
Results

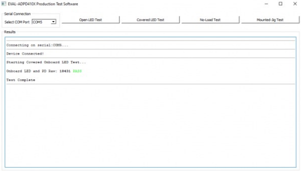
```
Connecting on serial:COM5...
Device Connected!
Starting Open Onboard LED Test...
Onboard LED and PD Raw: 13093 PASS
Test Complete
```

- **Covered Onboard LED Test**

  Place your finger above the onboard photodiode and LED.



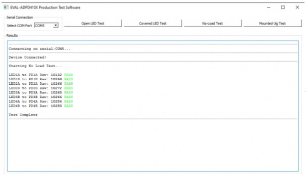
  This test checks if the sampled raw ADC value has significantly increased from the standard uncovered value. A sample passing result is shown below.


EVAL-ADPD410X Production Test Software
Serial Connection
Select COM Port
COM5
Open LED Test
Covered LED Test
No-Load Test
Mounted-Jig Test
Results

```
Connecting on serial:COM5...
Device Connected!
Starting Covered Onboard LED Test...
Onboard LED and PD Raw: 18431 PASS
Test Complete
```

- **No Load Test**

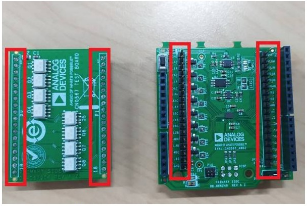
  Remove all shunts on jumper header P10. This test checks the photodiode input when no external sensor is connected. A sample passing result is shown below.


EVAL-ADPD410X Production Test Software
Serial Connection
Select COM Port
COM5
Open LED Test
Covered LED Test
No-Load Test
Mounted-Jig Test
Results

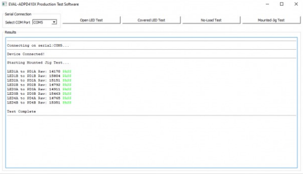
```
Connecting on serial:COM5...
Device Connected!
Starting No Load Test...

LED1A to PD1A Raw: 10132 PASS
LED1B to PD1B Raw: 10268 PASS
LED2A to PD2A Raw: 10266 PASS
LED2B to PD2B Raw: 10272 PASS
LED3A to PD3A Raw: 10265 PASS
LED3B to PD3B Raw: 10266 PASS
LED4A to PD4A Raw: 10256 PASS
LED4B to PD4B Raw: 10250 PASS

Test Complete
```

- **Mounted Jig Test**

  Using the simple test schematic shown in Connection, a test board was fabricated using MOC207M optocouplers. Remove all shunts on jumper header P10 and connect the test jig to the EVAL-ADPD4100-ARDZ or EVAL-ADPD4101-ARDZ, as shown below. **(The test jig shown below was fabricated with female headers for easy mounting)**


ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™
CN0567 TEST BOARD
P1
P9
ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™
EVAL-CN0567_ARDZ
PRIMARY SIDE

A sample passing result is shown below.


EVAL-ADPD410X Production Test Software
Serial Connection
Select COM Port
COM5
Open LED Test
Covered LED Test
No-Load Test
Mounted-Jig Test
Results

```
Connecting on serial:COM5...
Device Connected!
Starting Mounted Jig Test...

LED1A to PD1A Raw: 14170 PASS
LED1B to PD1B Raw: 15804 PASS
LED2A to PD2A Raw: 15151 PASS
LED2B to PD2B Raw: 14792 PASS
LED3A to PD3A Raw: 14911 PASS
LED3B to PD3B Raw: 15463 PASS
LED4A to PD4A Raw: 14765 PASS
LED4B to PD4B Raw: 15351 PASS

Test Complete
```

Schematic, PCB Layout, Bill of Materials

EVAL-AD410X-ARDZ Design & Integration Files

- Schematics
- PCB Layout
- Bill of Materials
- Allegro Project

Additional Information and Useful Links

- ADPD4100 Product Page
- ADPD4101 Product Page
- Fluorescence Measurement Demo

End of Document

resources/eval/user-guides/circuits-from-the-lab/eval-adpd410x.txt · Last modified: 27 Jun 2022 22:48 by Brandon Bushey

©1995 - 2022 Analog Devices, Inc. All Rights Reserved

Downloaded from Arrow.com.