I2C Guide: ADC

by Brian Fraser
Last update: Jan 24, 2025

This document guides the user through:

1. Understanding I2C
2. Using I2C from Linux command-line to read an analog to digital converter (ADC).
3. C code to access I2C and drive the display.

Guide has been tested on

BeagleBone (Target):	**Debian 12.8**
PC OS (host):	**Debian 12.8**

Table of Contents

Formatting

1. Commands for the host Linux's console are show as:
   `(host)$ echo "Hello PC world!"`
2. Commands for the target (BeagleBone) Linux's console are shown as:
   `(byai)$ echo "Hello embedded world!"`
3. Almost all commands are case sensitive.

Revision History

- Jan 24, 2025: Changed to BYAI; added ADC.

1. I2C Basics

The I^2C (Inter-Integrated-Circuit, pronounced “I squared C”, and often written I2C) protocol is for synchronously communicating between a master and slave devices using two pins: a data (SDA) and a clock (SCL). Often the microprocessor is the master device which controls communication with one or more slave devices on the bus.

On the BeagleY-AI, the hardware supports numerous I2C buses:

HW Bus	**Linux Device**	**Pins**	**Devices Attached**
I2C1	/dev/i2c-1	GPIO2 (pin 3) = SDA GPIO3 (pin 5) = SCL	Address 0x18: Audio (may not show up) Address 0x19: Accelerometer Address 0x48: ADC Chip
I2C2	/dev/i2c-2	GPIO0 (pin 27) = SDA GPIO1 (pin 28) = SCL	Used for I2C communication to an EEPROM, if present, on a Hat. Prefer I2C1.
I2C3	/dev/i2c-3		
I2C4	/dev/i2c-4	GPIO25 (pin 22) = SDA GPIO22 (pin 15) = SCL	Additional I2C if needed
I2C5	/dev/i2c-5		

Each chip connected to an I2C bus has a unique address which is hard-wired into the chip. (Sometimes the hardware designer can select one of a few possible addresses for a chip.) In the simplest case, when the master wants to initiate a read or write to a device, it communicates over the appropriate I2C bus and indicates the address of the device it wishes to interact with.

Each device exposes a set of registers in a small address space. Each register has a special purpose.

Note that there three things one must specify when interacting with a device:

1. Which bus a device is on (hard-wired).
2. What I2C address that device has (hard-wired).
3. What register address to read/write from (from data-sheet).

References

- BeagleY-AI Pinout: https://pinout.beagleboard.io/pinout/i2c
- Using I2C ADC: https://docs.beagleboard.org/boards/beagley/ai/demos/beagley-ai-using-i2c-adc.html

2. I2C via Linux Command Line[1]

This section walks through controlling a 4-channel analog to digital converter (ADC): a Texas Instruments TLA2024. By controlling this I2C device, we can read the voltages output by an analog device (such as an analog joystick). This chip is found on the Zen Hat, and has a joystick connected to channels 0 and 1.

2.1 Seeing the I2C Bus

All I2C buses are controlled through the Linux kernel. Let's see what's on the bus.

1. Install the I2C tools (if not already installed; requires internet access):
   ```
   (byai)$ sudo apt-get install i2c-tools
   ```

2. Determine which I2C bus the device is on.
   - Check the hardware schematic to determine which device you are accessing.
   - If you are wiring in a new I2C devices to the BeagleY-AI, you should use the I2C1 bus.

3. Display which I2C buses Linux currently has enabled (argument is "minus lower-case L"):
   ```
   (byai)$ i2cdetect -l
   i2c-1 i2c          OMAP I2C adapter                        I2C adapter
   i2c-2 i2c          OMAP I2C adapter                        I2C adapter
   i2c-3 i2c          OMAP I2C adapter                        I2C adapter
   i2c-4 i2c          OMAP I2C adapter                        I2C adapter
   i2c-5 i2c          OMAP I2C adapter                        I2C adapter
   ```

4. Display I2C devices on the chosen I2C bus (here is I2C1):
   ```
   (byai)$ i2cdetect -y -r 1
        0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
   00:                         -- -- -- -- -- -- -- --
   10: -- -- -- -- -- -- -- -- -- 19 -- -- -- -- -- --
   20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
   30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
   40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
   50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
   60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
   70: -- -- -- -- -- -- -- --
   ```
   - Where `1` refers to the Linux device `/dev/i2c-1`
   - "--" means no device found.
     "##" (like "19") means a device was detected at address ## (hex).
     "UU" (for example on `/dev/i2c-2)` means in use by a kernel driver.
   - You may also see "18" in your table.

5. Troubleshooting
   - If you run
     ```
     (byai)$ i2cdetect -y -r 1
     ```
     and it takes a long time (seconds per address), and does not find anything, then it might mean the pins are not configured for I2C use. Check `/boot/firmware/extlinux/extlinux.conf` to see what overlays are loaded.

1 Steps referenced from Exploring BeagleBone by Derek Molloy, 2015, chapter 8.

2.2 Working with a Device from Command Line

1. Display the internal memory of an I2C device

```
(byai)$ i2cdump -y 1 0x48 w
     0,8  1,9  2,a  3,b  4,c  5,d  6,e  7,f
00: 0000 8385 0080 ff7f 0000 8385 0080 XXXX
08: 0000 8385 0080 ff7f 0000 8385 0080 XXXX
10: 0000 8385 0080 ff7f 0000 8385 0080 XXXX
...
```

   - This shows the internal memory for the device at address 0x48 (ADC converter) on `/dev/i2c-1`. ***Output may differ for you.***
   - The \`w\` at the end specifies to view the data as words (2 bytes).
   - Consult the data sheet for your I2C device to identify what each register address means. In this output, note that the data repeats.
   - You can also read a single byte of memory, if desired:

     ```
     (byai)$ i2cget -y 1 0x48 0x01 w
     0x8385
     ```

     Arguments explanation:
     - `-y`: Disable "are you sure" confirmation prompt
     - `1`: I2C bus `/dev/i2c-1`
     - `0x48`: Address of device on bus
     - `0x01`: Register address to read.
     - w: Format of data (read a word)

     Value printed depends on state of device and may be different for you.

2. Write to the I2C device using `i2cset` command:

```
(byai)$ i2cset -y 1 0x48 0x01 0x83C2 w
```

   - This commands control device with address 0x48 on `/dev/i2c-1`: into register 0x01 it writes 0x83C2.
   - Doing this on the ADC device sets the device to continuously samples its analog input channel 0 and be ready to for the controller (host) to read it out with a later command.

3. Display device internal memory:

```
(byai)$ i2cdump -y 1 0x48 w
     0,8  1,9  2,a  3,b  4,c  5,d  6,e  7,f
00: b035 8342 0080 ff7f a035 8342 0080 XXXX
08: b035 8342 0080 ff7f a035 8342 0080 XXXX
10: b035 8342 0080 ff7f a035 8342 0080 XXXX
```

   <… omitted...>

   - Notice that the value in the 3rd column (under the "1,9" has changed. It now shows 0x8342 which is off by one bit from the requested 0x83C2. This change is likely due to a specific meaning of the bit.
   - Output may differ.

2.3 Steps for Zen Hat's DAC

Here are the steps to read the joystick on the Zen Hat's DAC

1. Set the DAC mode:
   - Continuously sampling channel 0 (Joystick Y):
     `(byai)$ i2cset -y 1 0x48 1 0x83C2 w`
   - Continuously sampling channel 1 (Joystick X):
     `(byai)$ i2cset -y 1 0x48 1 0x83D2 w`
   - Continuously sampling channel 2 (LED Receive):
     `(byai)$ i2cset -y 1 0x48 1 0x83E2 w`
   - Continuously sampling channel 3 (ADC Header, pin 2:
     `(byai)$ i2cset -y 1 0x48 1 0x83F2 w`
   - Reference: TLA2024 datasheet section 8.6.2, Configuration Register bits 14:12 Input Multiplexer Configuration.

2. Read the voltage:
   `(byai)$ i2cget -y 1 0x48 0x00 w`
   - The returned value gives the least-significant byte first. So, if the return is: **0xAB12** the actual value is **0x12AB**
   - Value will be 12-bits (it's a 12-bit ADC), with the value left-aligned, as shown in the table showing the bits for this register. The bottom 4 bits will be 0s. To make the value easiest to work with, it is a good idea to shift the value so that the bits are right-alaign.

8.6.1 Conversion Data Register (RP = 00h) [reset = 0000h]

The 16-bit conversion data register contains the result of the last conversion in binary two's-complement format. Following power-up, the conversion data register clears to 0, and remains at 0 until the first conversion is complete.

Figure 16. Conversion Data Register

15	14	13	12	11	10	9	8
D11	D10	D9	D8	D7	D6	D5	D4
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h
7	6	5	4	3	2	1	0
D3	D2	D1	D0	RESERVED			
R-0h	R-0h	R-0h	R-0h	R-0h			

Source: TLA2024 datasheet

3. I2C via C Code

3.1 Initialization

The following function initializes the I2C device, passing in the device path like "`/dev/i2c-1`".

```
static int init_i2c_bus(char* bus, int address)
{
    int i2c_file_desc = open(bus, O_RDWR);
    if (i2c_file_desc == -1) {
        printf("I2C DRV: Unable to open bus for read/write (%s)\n", bus);
        perror("Error is:");
        exit(EXIT_FAILURE);
    }

    if (ioctl(i2c_file_desc, I2C_SLAVE, address) == -1) {
        perror("Unable to set I2C device to slave address.");
        exit(EXIT_FAILURE);
    }
    return i2c_file_desc;
}
```

3.2 Writing a Register

The following function allows the program to write to an I2C device's register:

```
static void write_i2c_reg16(int i2c_file_desc, uint8_t reg_addr, uint16_t value)
{
    int tx_size = 1 + sizeof(value);
    uint8_t buff[tx_size];
    buff[0] = reg_addr;
    buff[1] = (value & 0xFF);
    buff[2] = (value & 0xFF00) >> 8;
    int bytes_written = write(i2c_file_desc, buff, tx_size);
    if (bytes_written != tx_size) {
        perror("Unable to write i2c register");
        exit(EXIT_FAILURE);
    }
}
```

3.3 Reading a Register

The following function allows the program to read from an I2C device's register:

```
static uint16_t read_i2c_reg16(int i2c_file_desc, uint8_t reg_addr)
{
    // To read a register, must first write the address
    int bytes_written = write(i2c_file_desc, &reg_addr, sizeof(reg_addr));
    if (bytes_written != sizeof(reg_addr)) {
        perror("Unable to write i2c register.");
        exit(EXIT_FAILURE);
    }

    // Now read the value and return it
    uint16_t value = 0;
    int bytes_read = read(i2c_file_desc, &value, sizeof(value));
    if (bytes_read != sizeof(value)) {
        perror("Unable to read i2c register");
        exit(EXIT_FAILURE);
    }
    return value;
}
```

3.4 Main program

This continuously reads the joystick's Y position. Note that it reads the raw bytes: these bytes must be swapped and shifted to be meaningful.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <linux/i2c.h>
#include <linux/i2c-dev.h>
#include <stdint.h>
#include <stdbool.h>

// Device bus & address
#define I2CDRV_LINUX_BUS "/dev/i2c-1"
#define I2C_DEVICE_ADDRESS 0x48

// Register in TLA2024
#define REG_CONFIGURATION 0x01
#define REG_DATA 0x00

// Configuration reg contents for continuously sampling different channels
#define TLA2024_CHANNEL_CONF_0 0x83C2


int main()
{
    printf("Read TLA2024 ADC\n");

    int i2c_file_desc = init_i2c_bus(I2CDRV_LINUX_BUS, I2C_DEVICE_ADDRESS);

    // Select the channel
    write_i2c_reg16(i2c_file_desc, REG_CONFIGURATION, TLA2024_CHANNEL_CONF_0);

    while(true) {

        // Read a register:
        uint16_t raw_read = read_i2c_reg16(i2c_file_desc, REG_DATA);
        printf("Raw reading: 0x%04x\n", raw_read);

        // sleep(1);
    }

    // Cleanup I2C access;
    close(i2c_file_desc);
    return 0;
}
```