Interface with Electrical Devices Over I2C

Written By Sage Wu ‘27
Updated by DaHee Kim ‘27
Updated as of Aug 1, 2024

For the purposes of this tutorial, I am describing how to interface with the LT8491 over I2C. You can follow this tutorial with any I2C-compatible device, not just the LT8491! The Microcontroller Unit (MCU) board I am using is the Feather M0 Adalogger, but any MCU board that can run CircuitPython and use I2C works.

Hardware Setup

1. Connect the LT8491 I2C pins.
    a. Connect SDA and SCL pins on the LT8491 to the corresponding SDA and SCL pins on your MCU board.
    b. Ensure that both your LT8491 and your MCU share a common GND connection.
    c. If necessary, use pull-up resistors on the SDA and SCL lines (typical values range from 4.7kΩ to 10kΩ)
2. Power the LT8491 as specified in its datasheet.

Software Setup

Now that you have the hardware setup, it's time to write some CircuitPython  code! This code runs on your MCU.

I. Setup your environment

1. Import the libraries you need

```
import board

import busio
```

2. Initialize I2C object

```
i2c = busio.I2C(board.SCL, board.SDA)
```

3. Declare the LT8491's I2C (slave) address. Check the data sheet to find this address. This address is used to identify the LT8491 on the I2C bus. For the LT8491 specifically, this address is set by the resistor(s). To achieve this specific address, we used a 100k.

```
LT8491_ADDR = 0x10
```

4. Declare the LT8491's I2C device registers.

   Device registers are small storage locations inside the LT8491 that store configuration settings, status information, or any other data the LT8491 needs to operate. Device registers can be read from/written to by the MCU, allowing your code to control the LT8491's functions.

   You can find all the registers on the LT8491 using its data sheet! You won't need all of them, though. The ones I'm using are:

```
TELE_TBAT = 0x00              # Battery temperature

TELE_PIN = 0x02               # Input power

TELE_POUT = 0X04              # Output power
```

```
STAT_CHARGER = 0x12          # Charger status
```

II. Configure registers

5. Before writing to/reading from registers, you need to configure your registers. Configuring your registers changes your hardware from its default settings to the specific operational settings you need – for instance, whether a device should amplify signals, filter them, or pass them through directly depends on your configuration settings.
6. Find the configuration settings, try and find the I2C register configuration section of your datasheet. For the LT8491's datasheet, this is on page 42, called 'I2C Register Descriptions'.
7. Follow the instructions for the registers you need. I will walkthrough one of the LT8491's register configurations for you here.

    First, get the register address we want to write to

    `CFG_RSENSE1 = 0x28`

    If we think of each register as a slot in a bookshelf (each slot being 1 byte), this register requires 2 slots (2 bytes). So `CFG_RSENSE1` is actually `0x28` and `0x29`. So we are going to write these 2 bytes separately.

    In order to do this, we need to separate the bytes. When you convert a number to hexadecimal, you get something that looks like this:

    `0x2710`

    There is a high byte and a low byte. You can make the program calculate this for you.

    `0x27 = high_byte = (value >> 8) & 0xFF`

```
0x10 = low_byte = value & 0xFF
```

Once you get them, you need to write to the registers. The registers in which you write in are dependent on what system you are working with. The LT8491 is low-endian, meaning the first byte written in the register will be the low byte. The other option is high-endian, meaning the high byte is written first.

```
out_buffer = bytearray(2)

out_buffer[0] = register_address

# in order to write to the register address, it must be the first thing in the array.

out_buffer[1] = low_byte

i2c.writeto(LT8491_ADDR, out_buffer)

out_buffer[1] = high_byte

i2c.writeto(LT8491_ADDR, out_buffer)
```

And then you are done! You have configured the register with a specific value.

III. Enter the main loop of code

8. Add a try_lock() statement.

    This is necessary because I2C is mutually exclusive (MutEx) — only one thread of code can use I2C at a time, so we need to lock the I2C bus so that no other code can infiltrate the bus.

    If `i2c.try_lock()` returns True, then the bus is available and we can use it!

    ```
    while not i2c.try_lock():
    ```

```
            pass
```

9. Once the I2C bus can be used, we try to send commands to the device register in a `try` block.

```
    try:
        out_buffer = bytearray(1)
        out_buffer[0] = TELE_TBAT
        in_buffer = bytearray(RESPONSE_LENGTH)
        i2c.writeto_then_readfrom(LT8491_ADDR,
out_buffer, in_buffer)
```

`out_buffer` is a byte array of length 1 that holds the commands we want to run. In this case, we define the first (and only) element of the array to be the command `TELE_TBAT`

`in_buffer` is a byte array of RESPONSE_LENGTH (check the data sheet for the exact response length) which holds the result from the LT8491

i. The function `i2c.writeto_then_readfrom()` writes the `out_buffer[0]` to LT8491, then reads the result into in_buffer

10. Release the lock on the I2C.

```
    finally:
        i2c.unlock()
```

11. Finally, initialize the hardware state, effectively

Example for LT8491 MPPT

- This guide is for porting Linux drivers to CircuitPython. We do this so we can use a feather to communicate with a device over I2C.
- https://github.com/craigpeacock/LT8491/tree/master
    - This person (craigpeacock) wrote a linux driver (in C)
    - Basically craigpeacock wanted to talk to this device (LT8491) on a linux computer
    - But the feathers we run aren't linux computers, so we need to write circuit python code instead to talk to the device
- https://docs.circuitpython.org/en/latest/shared-bindings/busio/

Write code for the feather

1. Create a code.py file in an IDE of your choice. Eventually you will port this to the feather. code.py runs before everything
2. Do a print("test") to make sure the feather works
3. Import busio & board libraries
    a. Import busio
    b. Import board
4. Define device address
    a. MPPT_ADDR =
5. Define device's command codes (hexadecimal)
    a. LT8491_TELE_TBAT = 0x00
    b. LT8491_STAT_VERSION = 0x1A
        i. The hex is a code which represents an instruction/command that the LT8491 device can perform.
        ii. You can find all the commands in LT8491/lt8491.h
        iii. Eventually you will need to turn all of these into CircuitPython constants.
6. Initialise I2C object and write I2C scan code
    a. I2c = busio.I2C(board.SCL, board.SDA)
    b. i2c.try_lock()
        i. Required because the I2C bus is mutually exclusive (mutex)
        ii. ie. only one thread of code can use i2c at a time, so we need to lock the i2c bus so that no other code can infiltrate the bus.
    c. out_buffer = bytearray(1)
        i. Make a byte array of length 1
        ii. The byte array will hold the commands we want to run. We send it to the LT8491 device over I2C.
    d. in_buffer = bytearray(response_length)

        i. Empty byte array of length response_length, where response_length is different for each of LT8491's commands.
        ii. The byte array will be populated by the LT8491's response to our out_buffer
    e. Out_buffer[0] = LT8491_TELE_TBAT
        i. Command [0] in the byte array, which is the command
    f. i2c.write_then_radfrom(MPPT_ADDR, out_buffer, in_buffer)

LT8491/i2c.c

- Linux kernel communication